Probabilistic Termination and Composability of Cryptographic Protocols

Ran Cohen* Sandro Coretti† Juan Garay‡ Vassilis Zikas§

April 1, 2016

Abstract

When analyzing the round complexity of multi-party cryptographic protocols, one often overlooks the fact that underlying resources, such as a broadcast channel, can be by themselves expensive to implement. For example, it is well known that it is impossible to implement a broadcast channel by a (deterministic) protocol in a sub-linear (in the number of corrupted parties) number of rounds.

The seminal works of Rabin and Ben-Or from the early 80's demonstrated that limitations as the above can be overcome by using randomization and allowing parties to terminate at different rounds, igniting the study of protocols over point-to-point channels with probabilistic termination and expected *constant* round complexity. However, absent a rigorous simulation-based definition, the suggested protocols are proven secure in a property-based manner or via *ad hoc* simulation-based frameworks, therefore guaranteeing limited, if any, composability.

In this work, we put forth the first simulation-based treatment of multi-party cryptographic protocols with probabilistic termination. We define secure multi-party computation (MPC) with probabilistic termination in the UC framework, and prove a universal composition theorem for probabilistic-termination protocols. Our theorem allows to compile a protocol using deterministic-termination hybrids into a protocol that uses expected-constant-round protocols for emulating these hybrids, preserving the expected round complexity of the calling protocol.

We showcase our definitions and compiler by providing for the first time simulation-based (therefore composable) protocols and 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, and (3) perfectly secure MPC with round complexity independent of the number of parties.

*Department of Computer Science, Bar-Ilan University. E-mail: cohenrb@cs.biu.ac.il.

†Department of Computer Science, ETH Zurich. E-mail: corettis@inf.ethz.ch.

‡Yahoo Labs. E-mail: garay@yahoo-inc.com.

§Department of Computer Science, RPI. E-mail: vzikas@cs.rpi.edu.

Contents

1 Introduction **1**

2 Model **6**

3 Secure Computation with Probabilistic Termination **7**
- 3.1 Canonical Synchronous Functionalities . . . 7
- 3.2 Probabilistic Termination in UC . . . 9

4 (Fast) Composition of Probabilistic-Termination Protocols **13**
- 4.1 The Simple Setting . . . 13
- 4.2 Using Actual Round Complexities . . . 14
- 4.3 Accounting for Slack . . . 14
- 4.4 The Composition Theorems . . . 15

5 Applications of our Fast Composition Theorem **16**
- 5.1 Fast and Perfectly Secure Byzantine Agreement . . . 16
- 5.2 Fast and Perfectly Secure Parallel Broadcast . . . 20
- 5.3 Fast and Perfectly Secure SFE . . . 21

A On Parallel (In)Composability of Protocols with Probabilistic Termination **24**

B The Model (Cont'd) **25**

C Composition Theorem **27**
- C.1 Using Actual Round Complexities . . . 27
- C.2 Accounting for Slack . . . 29
- C.3 The Composition Theorem . . . 35
- C.4 Wrapping SMTs . . . 36

D Applications of our Fast Composition Theorem (Cont'd) **37**
- D.1 Fast and Perfectly Secure Byzantine Agreement (Cont'd) . . . 38
- D.2 Parallel Broadcast with Probabilistic Termination (Cont'd) . . . 41

1 Introduction

In secure multi-party computation (MPC) [48, 26] n parties $P_1, \ldots, P_n$ wish to jointly perform a computation on their private inputs in a secure way, so that no coalition of cheating parties can learn more information than their outputs (privacy), nor can they affect the outputs of the computation any more than by choosing their own inputs (correctness).

While the original security definitions had the above property-based flavor (i.e., the protocols were required to satisfy correctness and privacy—potentially along with other security properties, such as fairness and input independence), it is by now widely accepted that security of multi-party cryptographic protocols should be argued in a simulation-based manner. Informally, in the simulation paradigm for security the protocol execution is compared to an ideal world where the parties have access to a trusted third party (TTP, aka the "ideal functionality") that captures the security properties we want the protocol to achieve. The TTP takes the parties' inputs and performs the computation on their behalf. A protocol is then rendered 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 them 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. In fact, most if not all implemented and highly optimized MPC protocols (e.g., [16, 18, 36, 14, 42]) are in this model. 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—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 which is often overlooked in the analysis of protocols' round complexity is that the relation between a protocol's round complexity and its actual running time is sensitive to the actual "hybrids" (e.g., network primitives) that the protocol is assumed to have access to. For example, starting with the seminal MPC works [48, 26, 6, 13, 46], a common assumption is that the parties have access to a broadcast channel which they invoke 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 (in n, the number of parties[1]) blow-up on the round complexity of the MPC protocol, as no deterministic broadcast protocol can tolerate a linear number of corruptions in

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

a sublinear number of rounds [23, 20]. Thus, even though these protocols' round complexity 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 $O(n+d)$ [33]) when executed over point-to-point channels.[2]

In fact, all so-called constant-round multi-party protocols (e.g., [37, 3, 15, 31, 1, 24, 29, 43]) 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.[3] As the results from [23, 20] 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 constant fraction of corruptions follow the paradigm from [22] (see below), which requires a poly-logarithmic (in n) number of rounds.[4]

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 [4] and Rabin [45], which, effectively, showed how to circumvent the above limitation by using randomization. Most relevant to this submission, Rabin [45] showed that linearly resilient *Byzantine agreement* protocols [44, 39] (BA, related to broadcast, possibility- and impossibility-wise) in expected *constant* rounds were possible, provided that all parties have access to a "common coin" (i.e., a common source of randomness).[5] This line of research culminated with the work of Feldman and Micali [22], who showed how to obtain a shared random coin with constant probability from "scratch," yielding a probabilistic BA protocol tolerating the maximum number of misbehaving parties ($t < n/3$) that runs in expected constant number of rounds. The randomized BA protocol in [22] works in the information-theoretic setting; these results were later extended to the computational setting by Katz and Koo [32] who showed that assuming digital signatures there exists an (expected-) constant-round protocol for BA tolerating $t < n/2$ corruptions. The speed-up 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[6]) as well as on *simultaneous* termination (a party that terminates cannot be sure that other parties have also terminated[7]) [19]. 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 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 secu-

[2]Throughout this work we will consider protocols in which all 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 [28] allow for replacing broadcast with a constant-round multi-cast primitive.

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

[4]Note that this includes even FHE-based protocols, as they also assume a broadcast channel and their security fails if multi-cast over point-to-point channels is used instead.

[5]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.

[6]Throughout this paper we use running time and round complexity interchangeably.

[7]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 $r+c$.

rity) [7] and the universal composition (UC) frameworks [8, 9]. 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., [9, 35, 10, 38]).

Despite the wide-spread 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 protocols as mentioned above [22, 32], 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 [32] 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. In lack, however, of a formal treatment, existing constructions are usually proved secure in a property-based manner or rely on *ad hoc*, less studied security frameworks [41].[8]

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 the same as the one of the original (broadcast-hybrid) protocol. A closer look at this replacement, however, 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. In [5], Ben-Or and El-Yaniv 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 [22] it will be logarithmic in the number of instances (i.e., in the player-set size). (We expand on the reason for this blow-up in the round complexity in Appendix A.) Nevertheless, in [5] a mechanism was proposed for implementing such parallel calls to broadcast so that the total number of rounds remains constant.

The inability of generic parallel composition is not the only composition task that has issues with PT protocols. As observed by Lindell *et al.* [41], the composition of 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 blow-up on the round complexity of the calling protocol. To resolve this, [41] 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 in [32] but, again, with a restricted (property-based) proof.

Now, 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) we would be able to use them to, for example, instantiate broadcast within a higher-level cryptographic protocol. In addition,

[8] As we discuss below, the protocol of Katz and Koo has an additional issue with adaptive security in the rushing adversary model, as defined in the UC framework, similar to the issue exploited in [30].

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, [5] claims 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. Section 5).

Our contributions. In this paper we provide the first formal simulation-based (and composable) treatment of MPC with probabilistic termination. Our treatment builds on Canetti's universal composition (UC) framework [8, 9]. 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 making calls to a (deterministic termination) r-round UC protocol ρ into a (probabilistic-termination) protocol in which ρ is replaced by a PT protocol ρ' achieving the same security as π so that the following holds: if the (expected) round complexity of ρ' is $g(k)$,[9] then the expected round complexity of the compiled protocol is (only) a factor of $\frac{g(k)}{r}$ larger than the round complexity of π.

Towards 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 protocol ρ, 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 ρ' 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 receives (locally) its output from one emulation, it proceeds to the next one. This yields an (at most) multiplicative blow-up on the expected round complexity as discussed above. In particular, if the protocols used to emulate the SFE's are expected constant-round, then the expected round complexity of ρ' is the same (asymptotically) as that of ρ.

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 perfectly secure MPC 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 in turn. Regarding Byzan-

[9] k is the security parameter. For perfect security statements, we will take $k = n$.

tine agreement, we present a protocol that perfectly securely UC-implements the probabilistic-termination Byzantine agreement functionality for $t < n/3$ in an expected-constant number of rounds. (We will use RBA to denote probabilistic-termination BA, as it is often referred to as "randomized BA."[10]) Our protocol follows the structure of the protocol in [22], with a modification inspired by Goldreich and Petrank [27] to make it strict polynomial time (see the discussion below), and in a sense it can be viewed as the analogue for RBA of the well-known "CLOS" protocol for MPC [11]. Indeed, similarly to how [11] converted (and proved) the "GMW" protocol [25] from statically secure in the stand-alone setting into an adaptively secure UC version, our work transforms the broadcast and BA protocols from [22] into adaptively UC-secure randomized broadcast and RBA protocols.[11]

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 sub-routines that are invoked in [22] and describe simple(r) (SFE-like) functionalities corresponding to these sub-routines; this provides us with a simple "backbone" of the protocol in [22] 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 compilation theorem ensures that the resulting protocol is also expected-constant-round.

There is a sticky issue here that we need to resolve for the above to work: the protocol in [22] does not have guaranteed termination and therefore the distribution of the terminating round is not sampleable by a strict probabilistic polynomial-time (PPT) machine.[12] A way around this issue would be to modify the UC model of execution so that the corresponding ITMs are expected PPTs. Such a modification, however, would impact the UC model of computation, and would therefore require a new proof of the composition theorem—a tricker task than one might expect, as the shift to expected polynomial-time simulation is known to introduce additional conceptual and technical difficulties (cf. [34]), 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 [22] using ideas from [27] so that it implements a functionality which samples the terminating round with almost the same probability distribution as in [22], 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, we use our composition theorem to derive the first simulation-based and adaptively (UC) secure parallel broadcast protocol, which guarantees that all broadcast values are received within an expected-constant number of rounds. This extends the results from [5, 32] in several ways: first, our protocol is perfectly UC-secure which means that we can now use it within a UC-secure SFE protocol to implement secure channels, and second, it is adaptively secure against a rushing adversary.[13]

Finally, by applying once again our compiler to replace calls to the broadcast channel in the SFE protocol by Ben-Or, Goldwasser, and Wigderson [6] (which, recall, is perfectly secure against $t < n/3$ corruptions in the broadcast-hybrid model [2]) by invocations to our adaptively secure UC

[10] 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.

[11] As we show, the protocol in [22] does not satisfy input independence, and therefore is not adaptively secure in a simulation-based manner (cf. [30]).

[12] 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.

[13] Although security against a "dynamic" adversary is also claimed in [5], the protocol does not implement the natural parallel broadcast functionality in the presence of an adaptive adversary (see Section 5).

parallel broadcast protocol , we obtain the first UC-secure PT 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.* [11] for SFE protocols with probabilistic termination.

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., [33]). 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.[14]

2 Model

We consider n parties $P_1, \ldots, P_n$ and an adaptive t-*adversary*, i.e., the adversary corrupts up to t parties during the protocol execution.[15] We work in the UC model and assume the reader has some familiarity with its basics. To capture synchronous protocols in UC we use the framework of Katz *et al.* [35]. Concretely, the assumption that parties are synchronized is captured by assuming that the protocol has access to a "clock" functionality $\mathcal{F}_{\text{CLOCK}}$. The functionality $\mathcal{F}_{\text{CLOCK}}$ maintains an indicator bit which is switched once *all honest parties* request the functionality to do it. At any given round, a party asks $\mathcal{F}_{\text{CLOCK}}$ to turn the bit on only after having finished with all operations for the current round. Thus, this bit's value can be used to detect when every party has completed his round, in which case they can proceed to the next round. As a result, this mechanism ensures that no party sends his messages for round $r+1$ before every party has completed round r. For clarity, we retain 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 [35], 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.[16] 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 δth `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 functionalities. These SFEs also work in a fetch mode[17] and provide the same guarantee as the bounded-delay channels.

There are two important considerations in proving the security of a synchronous UC protocol: (1) The simulator needs to keep track of the protocol's current round, and (2) because parties proceed at the same pace, they can synchronize their reaction to the environment; most fully-synchronous protocols, for example, deliver output exactly after a given number of rounds. In [35] this property is captured as follows: The functionality keeps track of which round the protocol

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

[15] In contrast, a *static* adversary chooses the set of corrupted parties at the onset of the computation.

[16] Following the simplifying approach of [35], we assume that communication channels are single use, thus each message transmission uses an independent instance of the channel (cf. Appendix B).

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

would be in by counting the number of activations it receives from honest parties. Thus, if the protocol has a regular structure, where every party advances the round after receiving a fixed number μ of activations from its environment (all protocols described herein will be in this form), the functionality can easily simulate how rounds in the protocol advance by incrementing its round index whenever it receives μ messages from all honest parties; we shall refer to such a functionality as a *synchronous functionality*. Without loss of generality, in this work we will describe all 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 the in an actual evaluation, as in the synchronous setting of [35], 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 `fetch-output`).

To further 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 round counter in $\mathcal{F}$ is $r = \rho$. All synchronous functionalities considered in this work have the following format: They treat the first message they receive from any party P_i as P_i's input[18]—if this message is not of the right form $(\texttt{input}, \cdot)$ then a default value is taken as P_i input; as soon as an honest party sends its first message, any future message by this party is treated as a `fetch-output` message. Refer to Appendix B for a more detailed overview of [35] and discussion of our model.

3 Secure Computation with Probabilistic Termination

The work of Katz *et al.* [35] addresses (synchronous) cryptographic protocols that terminate in a fixed number of rounds for all honest parties. However, as mentioned in Section 1, Ben-Or [4] and Rabin [45] 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 $O(n)$ on the round complexity of any deterministic BA protocol tolerating $t = O(n)$ corruptions exists [23, 20], Rabin's global-coin technique (fully realized later on in [22]) yields an expected-constant rounds protocol. This speed-up, however, comes at a price, namely, of relinquishing both *fixed* and *simultaneous* termination [19]: 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.

3.1 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 some potential adversarial leakage on the inputs and potential adversarial influence on the outputs.[19] In more detail, $\mathcal{F}_{\text{CSF}}$ has two parameters: (1) a randomized function f that receives $n + 1$ inputs (n inputs from the parties and an additional input from the adversary) and (2) a leakage function l that leaks some information about the input values to the adversary.

[18] Note that this implies that also protocol machines treats its first message as their input.

[19] 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 1.

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

$\mathcal{F}_{\text{CSF}}$ proceeds as follows, parameterized 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 $\mathcal{P} = \{P_1, \ldots, P_n\}$ 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 $(\texttt{input}, \mathsf{sid}, v)$ from some party $P_i \in \mathcal{P}$, set $x_i \leftarrow v$ and send $(\texttt{leakage}, \mathsf{sid}, P_i, l(x_1, \ldots, x_n))$ 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 $P_i \in \mathcal{P}$, 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 $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.

Figure 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 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. [11]), as its behavior depends on the identities of the corrupted parties.[20] 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. [35]).

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:

– 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 $\texttt{input}$ messages from all n parties, we define the two-party task using an n-party function. The function to compute is $f_{\text{SMT}}^{i,j}(x_1, \ldots, x_n, a) = (\lambda, \ldots, x_i, \ldots, \lambda)$ (where x_i is the value of the j'th coordinate) and the leakage function is $l_{\text{SMT}}^{i,j}(x_1, \ldots, x_n) = y$, where $y = |x_i|$ in case P_j is honest and $y = x_i$ in case P_j is corrupted. We

[20] This is, in fact, also the case for the standard UC SFE funtionality.

denote by $\mathcal{F}_{\text{SMT}}^{i,j}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parameterized with the above functions $f_{\text{SMT}}^{i,j}$ and $l_{\text{SMT}}^{i,j}$, 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 $f_{\text{BC}}^{i}(x_1, \ldots, x_n, a) = (x_i, \ldots, x_i)$. The adversary only learns the length of the message x_i before its distribution, i.e., the leakage function is $l_{\text{BC}}^{i}(x_1, \ldots, x_n) = |x_i|$. 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 $\mathcal{F}_{\text{BC}}^{i}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parameterized with the above functions f_{BC}^{i} and l_{BC}^{i}, for sender P_i.
- Secure Function Evaluation. In the *secure function evaluation* functionality, the parties compute a randomized function $g(x_1, \ldots, x_n)$, i.e., the function to compute is $f_{\text{SFE}}^{g}(x_1, \ldots, x_n, a) = g(x_1, \ldots, x_n)$. 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 $\mathcal{F}_{\text{SFE}}^{g}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parameterized with the above functions f_{SFE}^{g} 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 $\mathcal{F}_{\text{BA}}^{V}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parameterized with the above functions f_{BA} and l_{BA}, defined for the set V.

3.2 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 parameterized by an efficient probabilistic algorithm D, termed the *round sampler*, that samples a round number r_{out} 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 at any time before round r_{out}.

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.[21]

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 output. 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.

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

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 r_{out}. 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 efficient simulation, we have $\mathcal{F}$ output all the coins that were used for sampling round r_{out} 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 output. However, this upper bound now not only depends 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.[22]

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. Then, this would result in the trace depicted in Figure 2.

[22] In this work, atomic functionalities are always CSFs.

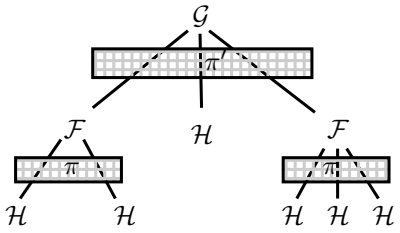

Figure 2: Example of an execution trace

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 $\mathcal{H}$.[23] 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 2 would look as in Figure 3.

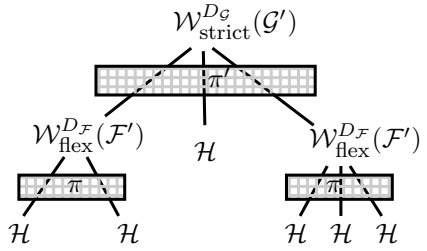

Figure 3: An execution trace with probabilistic-termination and deterministic-termination protocols

Formally, a trace is defined as follows:

Definition 3.1. *A* trace *is a rooted tree 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.*

Remark. The actual trace of a protocol may depend on the input values and the behavior of the adversary. For example, in the setting of Byzantine agreement, the honest parties may get the output faster in case they all have the same input, which results in a different trace. However, the wrappers defined below sample traces independently of the inputs. All protocols considered in this work can be shown to realize useful ideal functionalities in spite of this restriction.

Strict wrapper functionality. We now proceed to give the formal descriptions of the wrappers. The *strict wrapper*, defined in Figure 4, is parameterized 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

[23] 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.

from D; this trace is given to the adversary once the first honest party has provided its input. The trace T is used by the wrapper to define the termination round $r_{\mathsf{out}} \leftarrow 2 \cdot c_{\mathsf{tr}}(T)$, where $c_{\mathsf{tr}}(T)$, called *trace complexity*, is the number of leaves in T (the additional factor of 2 stems from the fact that CSFs always take two rounds to complete). 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 wrappers essentially waits until round r_{out}, 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 r_{out} arrives, the wrapper provides the output generated by $\mathcal{F}$ to all parties.

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

$\mathcal{W}_{\text{strict}}$, parameterized 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 $r_{\mathsf{out}} \leftarrow 2 \cdot c_{\mathsf{tr}}(T)$. Send $(\texttt{trace}, \mathsf{sid}, T)$ to the adversary.[a]
- In every round, 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 $P_i \in \mathcal{P}$ to $\mathcal{F}$. In addition, forward $(\texttt{leakage}, \mathsf{sid}, \cdot)$ messages from $\mathcal{F}$ to the adversary.
- In rounds $\rho = 2, \ldots, r_{\mathsf{out}} - 1$: Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some party $P_i \in \mathcal{P}$, send $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.
- In round $\rho = r_{\mathsf{out}}$: Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some party $P_i \in \mathcal{P}$, forward it to $\mathcal{F}$ and the response from $\mathcal{F}$ to P_i.

Figure 4: The strict wrapper functionality

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

Flexible wrapper functionality. The *flexible wrapper*, defined in Figure 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 ronud. In order to accomplish this, the wrapper assign 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 check 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 r_{out} arrives, the wrapper provides the output to all parties that didn't receive it yet.

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

$\mathcal{W}_{\text{flex}}$, parameterized 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 $r_{\mathsf{out}} \leftarrow 2 \cdot c_{\mathsf{tr}}(T)$. Send $(\texttt{trace}, \mathsf{sid}, T)$ to the adversary.[a] In addition, initialize the termination indicators $\mathsf{term}_1, \ldots, \mathsf{term}_n \leftarrow 0$.
- In every round, 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 $P_i \in \mathcal{P}$ to $\mathcal{F}$. In addition, forward $(\texttt{leakage}, \mathsf{sid}, \cdot)$ messages from $\mathcal{F}$ to the adversary.
- In rounds $\rho = 2, \ldots, r_{\mathsf{out}} - 1$:
  - Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some party $P_i \in \mathcal{P}$, proceed as follows:
    - If $\mathsf{term}_i = 1$, forward the message to $\mathcal{F}$ and the response to P_i.
    - Otherwise, send $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.
  - Upon receiving $(\texttt{early-output}, \mathsf{sid}, P_i)$ from the adversary, set $\mathsf{term}_i \leftarrow 1$.
- In round $\rho = r_{\mathsf{out}}$: Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some party, forward the message to $\mathcal{F}$ and the response to P_i.

Figure 5: The flexible wrapper functionality

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

4 (Fast) Composition of Probabilistic-Termination Protocols

Flexibly-wrapped CSFs (cf. Section 3.2), which 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, but—unless one is willing to pay a linear blow-up in round complexity [23, 20]—only to a difference of one round. Hence, all protocols must be modified to deal with a slack of one round, and protocols that introduce slack must be followed by a slack-reduction phase. Since this is a tedious, yet systematic task, in this section we provide a compiler (with a corresponding theorem) that generically transforms protocols designed in a simple setting without slack and round-complexity issues into protocols that deal with these issues while maintaining their security.

In the following high-level overview, we first introduce the "simple setting." In this setting, protocols only make calls to (unwrapped) CSF hybrids, which allows to design them without worrying about round complexity and slack (Section 4.1). Then, in a first step, we describe how to take into account round complexity (Section 4.2). In a second step, we show how to deal with potential slack (Section 4.3). These two steps define a protocol compiler that transforms protocols for the simple setting into protocols for the "realistic" setting where round complexity and slack is taken into account. Finally, we state the composition theorem (Section 4.4), which deals with the security of the compiler. Details can be found in Appendix C.

4.1 The Simple Setting

The protocols that we feed to our compiler need to be in a special *synchronous normal form (SNF)*, which, intuitively, means that they can only make strictly sequential calls to CSF hybrids.

Definition 4.1. *A protocol is in SNF if at any point during its execution, parties have invoked only a single instance of a single CSF and no honest party hands inputs to other CSFs before this instance halts.*

The starting point of every application of the composition theorem (cf. Section 4.4) is a proof that an SNF protocol π realizes a strictly-wrapped CSF $\mathcal{W}^D_{\text{strict}}(\mathcal{F})$ (resp. a flexibly-wrapped CSF $\mathcal{W}^D_{\text{flex}}(\mathcal{F})$) using some (easy-to-handle) CSF hybrids $\mathcal{F}_i$ for some distribution D over *depth*-1 traces (the traces have depth-1 since the hybrids used are CSFs and not wrapped CSFs), which hides the round complexities and potentially non-simultaneous termination properties of the hybrids.

Note that the simple structure (and deterministic-termination property) of the CSF functionality make designing SNF protocols and proving their security and correctness a far simpler task than designing protocols that call more complicated (probabilistic-termination) hybrids. It is the job of our compiler to map the SNF protocol into one that can work with probabilistic hybrids. Moreover, while considering only SNF protocols may at a first glance seem to be restrictive, most of the known synchronous protocols can be phrased in such a way.

4.2 Using Actual Round Complexities

In a first step, the actual round complexities of (the protocols realizing) the hybrids $\mathcal{F}_i$ are taken into account (while still ignoring the potential slack). Suppose these round complexities are given by distributions D_i (over traces). In Appendix C.1 we prove that the protocol π' that calls hybrids $\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i)$ (instead of the unwrapped CSFs $\mathcal{F}_i$) still realizes $\mathcal{W}^{D'}_{\text{strict}}(\mathcal{F})$ (resp. $\mathcal{W}^{D'}_{\text{flex}}(\mathcal{F})$) but for a distribution D' that is based on D and may change depending on the distributions D_i. In particular, D' outputs *full* traces, in which each depth-1 node $\mathcal{F}_i$ of a trace sampled by D is replaced by a trace (independently) sampled from D_i. In the following, the adapted distribution is denoted by $D' = \mathsf{full\text{-}trace}(D, \{D_i\}_i)$. Full traces are formally defined in Section C.1.

4.3 Accounting for Slack

In the second generic step, we take care of slack introduced by hybrids realized by protocols without simultaneous termination.

Let π be an arbitrary (SNF) protocol that is to be "hardened" such that it can be safely executed even if parties potentially initiate the protocol up to $c \geq 1$ rounds apart.[24] A possible solution to this problem, inspired by [40], is to expand each round r into $3c+1$ rounds as follows: During the first $2c+1$ rounds, each party listens for round-r messages sent by other parties, and in round $c+1$ the party sends its own round-r messages. During the last c rounds, it simply waits (without listening).[25] It can be easily verified (on an informal level) that this results in all round-r messages being properly exchanged, even if parties start this process up to c rounds apart.

Furthermore, if π is not a slack-introducing protocol, that is, if each party runs exactly the same number of rounds, then the protocol resulting from the above transformation is slack-preserving, i.e., parties finish with the same slack with which they initiated the protocol.

In contrast, if π induces slack, then, in addition to the above expansion technique, π needs to be enhanced by a slack-reduction phase. We adopt an approach originally suggested by [5], which works for public-output protocols:

- 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 $(\mathtt{end}, y)$ for the same value y (whichever happens first), it sends $(\mathtt{end}, \mathsf{sid}, y)$ to all parties.

[24] In the context of this work, the slack will always be $c = 1$.

[25] The reason for this is explained in Appendix C.4.

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

In Appendix C.2, we show that the above steps can be described by considering two additional wrappers, $\mathcal{W}^c_{\text{ST}}$ and $\mathcal{W}^c_{\text{STR}}$, where $\mathcal{W}^c_{\text{ST}}$ is used for hybrids realized by non-slack-inducing protocols and $\mathcal{W}^c_{\text{STR}}$ for those realized by slack-inducing protocols.

Denote by $\pi'' := \mathsf{Comp}^c_{\text{PT}}(\pi)$ the "compiled" protocol resulting in the first case above (resp. by $\pi'' := \mathsf{Comp}^c_{\text{DT}}(\pi)$ the "compiled" protocol π'' resulting in the second case).

We prove that

- in the case where π realizes a strictly-wrapped $\mathcal{F}$, then the protocol π'' making calls to hybrids $\mathcal{W}^c_{\text{ST}}(\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i))$ realizes $\mathcal{W}^c_{\text{ST}}(\mathcal{W}^{D'}_{\text{strict}}(\mathcal{F}))$, and
- in the case where π realizes a flexibly-wrapped $\mathcal{F}$, then the protocol π'' making calls to hybrids $\mathcal{W}^c_{\text{ST}}(\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i))$ or $\mathcal{W}^c_{\text{STR}}(\mathcal{W}^{D_i}_{\text{flex}}(\mathcal{F}_i))$ realizes $\mathcal{W}^c_{\text{STR}}(\mathcal{W}^{D'}_{\text{flex}}(\mathcal{F}))$.

4.4 The Composition Theorems

Denote by $\mathcal{W}^D_{\text{DT}}(\mathcal{F}) := \mathcal{W}_{\text{ST}}(\mathcal{W}^D_{\text{strict}}(\mathcal{F}))$ and $\mathcal{W}^D_{\text{PT}}(\mathcal{F}) := \mathcal{W}_{\text{STR}}(\mathcal{W}^D_{\text{flex}}(\mathcal{F}))$. Summarizing, we prove the two theorems below. Recall that c denotes the amount of slack tolerance and $c_{\mathsf{tr}}(T)$ denote the number of leaves in a trace T. The derivations of the round complexities can be found in Section C.3.

Theorem 4.2. *Let $\{\mathcal{F}_i\}_i$ and $\mathcal{F}$ be arbitrary CSFs, D and $\{D_i\}_i$ arbitrary distributions over traces, and $D' = \mathsf{full\text{-}trace}(D, \{D_i\}_i)$. If an SNF protocol π securely realizes $\mathcal{W}^D_{\text{strict}}(\mathcal{F})$ in the $\{\mathcal{F}_i\}_i$-hybrid model, then $\mathsf{Comp}^c_{\text{DT}}(\pi)$ securely realizes $\mathcal{W}^{D'}_{\text{DT}}(\mathcal{F})$ in the $\{\mathcal{W}^{D_i}_{\text{DT}}(\mathcal{F}_i)\}_i$-hybrid model.*

The expected round complexity of the compiled protocol $\mathsf{Comp}^c_{\text{DT}}(\pi)$ is

$$2(3c+1)\sum_i d_i \cdot E[c_{\mathsf{tr}}(T_i)],$$

where d_i is the expected number of calls to hybrid $\mathcal{F}_i$ and T_i is a trace sampled from D_i.

Theorem 4.3. *Let $\{\mathcal{F}_i\}_i$ and $\mathcal{F}$ be arbitrary CSFs, D and $\{D_i\}_i$ arbitrary distributions over traces, and $D' = \mathsf{full\text{-}trace}(D, \{D_i\}_i)$. Moreover, let I be a set indexing a subset of the CSFs $\mathcal{F}_i$. If an SNF protocol π securely realizes $\mathcal{W}^D_{\text{flex}}(\mathcal{F})$ in the $\{\mathcal{F}_i\}_i$-hybrid model, then $\mathsf{Comp}^c_{\text{PT}}(\pi)$ securely realizes $\mathcal{W}^{D'}_{\text{PT}}(\mathcal{F})$ in the $(\{\mathcal{W}^{D_i}_{\text{DT}}(\mathcal{F}_i)\}_{i\notin I}, \{\mathcal{W}^{D_i}_{\text{PT}}(\mathcal{F}_i)\}_{i\in I})$-hybrid model.*

The expected round complexity of the compiled protocol $\mathsf{Comp}^c_{\text{PT}}(\pi)$ is

$$2(3c+1)\sum_i d_i \cdot E[c_{\mathsf{tr}}(T_i)] + \mathcal{O}(1),$$

where d_i is the expected number of calls to hybrid $\mathcal{F}_i$ and T_i is a trace sampled from D_i.

Remark. Note that (recursively) the round complexity of a protocol realizing $\mathcal{W}^{D_i}_{\text{DT}}(\mathcal{F}_i)$ already contains the factor $2(3c+1)$, and, therefore, these factors do not accumulate when one replaces hybrids by their realizing protocols! Moreover, in either case, if π and all the protocols realizing the hybrids are expected constant-round protocols, so is the compiled protocol.

Wrapping SMTs. Using the above composition theorems, one can get protocols realizing wrapped functionalities $\mathcal{W}_{\text{DT}}(\mathcal{F})$ or $\mathcal{W}_{\text{PT}}(\mathcal{F})$ from wrapped functionalities $\mathcal{W}_{\text{DT}}(\mathcal{F}_i)$ and $\mathcal{W}_{\text{PT}}(\mathcal{F}_i)$. To ultimately get a protocol that uses only the parallel secure message transmission CSFs $\mathcal{F}_{\text{PSMT}}$ (defined in Section 5), we show in Appendix C.4 how to realize $\mathcal{W}_{\text{DT}}(\mathcal{F}_{\text{PSMT}})$ from $\mathcal{F}_{\text{PSMT}}$.

5 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 5.1 by presenting expected-constant-round protocols for Byzantine agreement. Next, in Section 5.2 we present an expected-constant-round protocol for parallel broadcast. Finally, in Section 5.3 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.

Our aim is to construct protocols in the point-to-point channels network model, and so it is tempting to use the secure channel functionality $\mathcal{F}_{\text{SMT}}$ (defined in Section 3.1) as our atomic functionality. However, since we consider SNF protocols that can call a single hybrid in any given round, we actually need the *parallel* version of $\mathcal{F}_{\text{SMT}}$, where each pair of parties can communicate privately at the same round.

- Parallel Secure Message Transmission. In the *parallel secure message transmission* functionality, 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 $y_j^i = \left|x_j^i\right|$ in case P_j is honest and $y_j^i = x_j^i$ in case P_j is corrupted. We denote by $\mathcal{F}_{\text{PSMT}}$ the CSF functionality parameterized with the above functions f_{PSMT} and l_{PSMT}.

5.1 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 3.1). These protocols are based on techniques due to Feldman and Micali [22] and Turpin and Coan [47], 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 [22]. 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 $\mathcal{F}_{\text{CSF}}$.

Oblivious common coin. In the *oblivious common coin* ideal functionality (introduced in [22]) 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 parameterized 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 parameterized with the above functions f_{oc} and l_{oc}. In [22, Thm. 3], Feldman and Micali showed a constant-round oblivious common coin protocol for $p = 0.35$.

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 5.1), denoted $\mathcal{F}_{\text{RBA}}$ for short. The protocol π_{RBA}, described in Figure 6, is based on the protocol from [22] modified using the "best-of-both-worlds" technique due to Goldreich and Petrank [27]. Recall that following Theorem 4.3, 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.

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.

Protocol π_{RBA}

Each party $P_i \in \mathcal{P} = \{P_1, \ldots, P_n\}$ 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 $P_j \in \mathcal{P}$ set a value $B_j \leftarrow 0$ for storing the last bit value received from P_j. In addition, denote $\tau = \log^{1.5}(k) + 1$.
- In the first round, upon receiving $(\texttt{input}, \mathsf{sid}, v)$ 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 $\mathcal{F}_{\text{PSMT}}$).
- In the second round, 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 $(\texttt{input}, \mathsf{sid}_\alpha, \lambda)$ to $\mathcal{F}_{\text{OC}}$.
  2. Send $(\texttt{fetch-output}, \mathsf{sid}_\alpha)$ to $\mathcal{F}_{\text{OC}}$.
     Let $(\texttt{output}, \mathsf{sid}_\alpha, \beta)$, with $\beta \in \{0,1\}$, be the output received from $\mathcal{F}_{\text{OC}}$.
  3. 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 $b_j \leftarrow B_j$.
  4. 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 $b_j \leftarrow B_j$.
  5. 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 $(\texttt{output}, \mathsf{sid}, b_i)$ and halt.
- Else (i.e., if $\mathsf{term} = 0$ or $\mathsf{term} = \tau$), proceed as follows:
  1. Send $(\texttt{input}, \mathsf{sid}, b_i)$ to $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ (note that b_i is the value that was set in phase τ).
  2. Send $(\texttt{fetch-output}, \mathsf{sid})$ to $\mathcal{F}_{\text{BA}}^{\{0,1\}}$. Upon receving $(\texttt{output}, \mathsf{sid}, b)$, with $b \in \{0,1\}$, if $\mathsf{term} = 0$ output $(\texttt{output}, \mathsf{sid}, b)$ and halt. Else, if $\mathsf{term} = \tau$, output $(\texttt{output}, \mathsf{sid}, b_i)$ and halt.

Figure 6: The binary randomized Byzantine agreement protocol, 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 [30]).

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 $\tau = \log^{1.5}(k)+1$ phases, the parties use the Byzantine agreement functionality $\mathcal{F}_{\text{BA}}$ in order to ensure termination.

In Appendix D.1 we prove the following lemma.

Lemma 5.1. *Let $p = 0.35$ and $t < n/3$. Denote by D_{RBA} the distribution that outputs a depth-1 trace, where the root is $\mathcal{W}_{\text{flex}}^{D_{\text{RBA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$, and the leaves are set as follows: initially sample an integer r from the geometric distribution with parameter p 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.*

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.

We now use Theorem 4.3 to derive the main result of this section.

Theorem 5.2. *Let $c \geq 0$ and $t < n/3$. There exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}_{\text{PT}}^{D}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$ has an expected-constant-round complexity, and can be UC-realized in the $\mathcal{F}_{\text{PSMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within c consecutive rounds.*

Proof (sketch). Let D_{RBA} be as in Lemma 5.1. Denote by D_{PSMT} the deterministic distribution that outputs a depth-1 trace consisting of a single leaf $\mathcal{F}_{\text{PSMT}}$, denote by D_{OC} the deterministic distribution that outputs a depth-1 trace consisting of a 36 leaves $\mathcal{F}_{\text{PSMT}}$ (recall that the $\mathcal{F}_{\text{OC}}$ protocol in [22, Claim T4-4] requires 36 rounds), and denote by D_{BA} the deterministic distribution that outputs a depth-1 trace consisting of $O(n)$ leaves $\mathcal{F}_{\text{PSMT}}$. Let $D'_{\text{RBA}}, D'_{\text{PSMT}}, D'_{\text{OC}}, D'_{\text{BA}}$ be the translated distributions, respectively.

Then, following Theorem 4.3 and Lemma 5.1, the compiled protocol $\mathsf{Comp}_{\text{PT}}^{c}(\pi_{\text{RBA}})$ UC-realizes $\mathcal{W}_{\text{PT}}^{D'_{\text{RBA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$, in the $(\mathcal{W}_{\text{DT}}^{D'_{\text{PSMT}}}(\mathcal{F}_{\text{PSMT}}), \mathcal{W}_{\text{DT}}^{D'_{\text{OC}}}(\mathcal{F}_{\text{OC}}), \mathcal{W}_{\text{DT}}^{D'_{\text{BA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}}))$-hybrid model, with perfect security, in the presence of adaptive malicious adversaries corrupting at most $t < n/3$ of the parties.

The proof follows since each of the functionalities $(\mathcal{W}_{\text{DT}}^{D'_{\text{PSMT}}}(\mathcal{F}_{\text{PSMT}}), \mathcal{W}_{\text{DT}}^{D'_{\text{OC}}}(\mathcal{F}_{\text{OC}}), \mathcal{W}_{\text{DT}}^{D'_{\text{BA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}}))$ can be UC-realized in the $\mathcal{F}_{\text{PSMT}}$-hybrid model. This follows from Lemma C.4, combined with the protocols from [22] and [30]. □

Multi-Valued Byzantine agreement protocol. In Appendix D.1 we present an analogue of the multi-valued Byzantine agreement protocol due to Turpin and Coan [47] for the UC framework, and prove the following.

Theorem 5.3. *Let $c \geq 0$, $t < n/3$ and $V \subseteq \{0,1\}^*$. There exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}_{\text{PT}}^{D}(\mathcal{F}_{\text{BA}}^{V})$ has an expected-constant-round complexity, and can be UC-realized in the $\mathcal{F}_{\text{PSMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within c consecutive rounds.*

5.2 Fast and Perfectly Secure Parallel Broadcast

As discussed in Section 1 (and Appendix A), composing protocols with probabilistic termination naïvely does not retain expected round complexity. In [5], Ben-Or and El-Yaniv constructed an elegant protocol for probabilistic-termination parallel broadcast[26] with a constant round complexity in expectation, albeit under a property-based security definition. In this section we adapt the [5] 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.

In a standard broadcast functionality (cf. Section 3.1), the sender provides a message to the functionality which delivers it to the parties. In [30], Hirt and Zikas 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 [30], 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 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 parameterized with the above functions f_{UPBC} and l_{UPBC}.

In Appendix D.2.1 we present an adaptation of the [5] protocol and show that it perfectly UC-realizes (a wrapped version of) $\mathcal{F}_{\text{UPBC}}$ (see Figure 12).

- 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 parameterized with the above functions f_{PBC} and l_{PBC}.

Unfortunately, the unfair parallel broadcast protocol π_{UPBC} (cf. Figure 12) 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 [30]: 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-out-of-n secret sharing protocol.[27] In the second step, every party, in parallel, broadcast a vector with all the shares

[26] In [5] the problem is referred to as "interactive consistency."

[27] In [30] verifiable secret sharing (VSS) is used; however, as we argue, this is not necessary.

he received, by 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. 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. We describe this protocol, denoted π_{PBC}, in Figure 7.

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 a t-out-of-n secret sharing scheme, denoted by $(x_i^1, \ldots, x_i^n)$. 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.
2. Next, 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. Now, P_i reconstructs all the input values, i.e., for every $j \in [n]$ reconstructs y_j from the shares $(y_j^1, \ldots, y_j^n)$, and outputs $(\texttt{output}, \mathsf{sid}, (y_1, \ldots, y_n))$.

Figure 7: The parallel broadcast protocol, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{UPBC}})$-hybrid model

Theorem 5.4. *Let $c \geq 0$ and $t < n/3$. There exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}_{\text{PT}}^{D}(\mathcal{F}_{\text{PBC}})$ has an expected-constant-round complexity, and can be UC-realized in the $\mathcal{F}_{\text{PSMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within c 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 simulation to simply hands the adversary random shares for all corrupted parties and for corrupted he follows the adversary's instructions. If during this step the adversary asks to corrupt new senders, the simulator learns their output 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. 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 (Theorem D.3). □

5.3 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 [6],[28] 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 4.3,

[28] A full simulation proof of the protocol with a black-box straight-line simulation was recently given by [2] and [17].

that the compiled protocol $\mathsf{Comp}_{\mathrm{PT}}(\pi_{\mathrm{BGW}})$ UC realizes the corresponding wrapped functionality $\mathcal{W}^{D}_{\mathrm{PT}}(\mathcal{F}_{\mathrm{SFE}})$ (for an appropriate distribution D), in the $(\mathcal{W}^{D'_{\mathrm{PSMT}}}_{\mathrm{DT}}(\mathcal{F}_{\mathrm{PSMT}}), \mathcal{W}^{D'_{\mathrm{PBC}}}_{\mathrm{PT}}(\mathcal{F}_{\mathrm{PBC}}))$-hybrid model, resulting in the following.

Theorem 5.5. *Let f be an n-party function, C an arithmetic circuit with multiplicative depth d computing f, and $t < n/3$. Then there exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}^{D}_{\mathrm{PT}}(\mathcal{F}^{f}_{\mathrm{SFE}})$ has round complexity $O(d)$ in expectation, and can be UC-realized in the $\mathcal{F}_{\mathrm{PSMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within c consecutive rounds.*

References

[1] 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 *EUROCRYPT*, pages 483–501, 2012.

[2] Gilad Asharov and Yehuda Lindell. A full proof of the BGW protocol for perfectly-secure multiparty computation. *Electronic Colloquium on Computational Complexity (ECCC)*, 18:36, 2011.

[3] Donald Beaver, Silvio Micali, and Phillip Rogaway. The round complexity of secure protocols (extended abstract). In *STOC*, pages 503–513. ACM, 1990.

[4] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *Proceedings of the Second Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing, Montreal, Quebec, Canada, August 17-19, 1983*, pages 27–30. ACM, 1983.

[5] Michael Ben-Or and Ran El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16(4):249–262, 2003.

[6] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In *20th ACM STOC*, pages 1–10. ACM Press, May 1988.

[7] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[8] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. In *42nd FOCS*, pages 136–145. IEEE Computer Society Press, October 2001.

[9] Ran Canetti. Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239, 2003. http://eprint.iacr.org/2003/239.

[10] Ran Canetti, Asaf Cohen, and Yehuda Lindell. A simpler variant of universally composable security for standard multiparty computation. In Rosario Gennaro and Matthew Robshaw, editors, *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2015.

[11] Ran Canetti, Yehuda Lindell, Rafail Ostrovsky, and Amit Sahai. Universally composable two-party and multi-party secure computation. In *34th ACM STOC*, pages 494–503. ACM Press, May 2002.

[12] Ran Canetti and Tal Rabin. Universal composition with joint state. In Dan Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, August 2003.

[13] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (extended abstract). In *20th ACM STOC*, pages 11–19. ACM Press, May 1988.

[14] Seung Geol Choi, Jonathan Katz, Alex J. Malozemoff, and Vassilis Zikas. Efficient three-party computation from cut-and-choose. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 513–530. Springer, August 2014.

[15] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 378–394. Springer, 2005.

[16] Ivan Damgård, Marcel Keller, Enrique Larraia, Valerio Pastro, Peter Scholl, and Nigel P. Smart. Practical covertly secure MPC for dishonest majority - or: Breaking the SPDZ limits. In Jason Crampton, Sushil Jajodia, and Keith Mayes, editors, *ESORICS 2013*, volume 8134 of *LNCS*, pages 1–18. Springer, September 2013.

[17] Ivan Damgård and Jesper Buus Nielsen. Adaptive versus static security in the UC model. In *ProvSec*, pages 10–28, 2014.

[18] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 643–662. Springer, August 2012.

[19] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, 1990.

[20] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

[21] Bennett Eisenberg. On the expectation of the maximum of IID geometric random variables. *Statistics & Probability Letters*, 78(2):135–143, 2008.

[22] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

[23] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186, 1982.

[24] Sanjam Garg, Craig Gentry, Shai Halevi, and Mariana Raykova. Two-round secure MPC from indistinguishability obfuscation. In *TCC*, pages 74–94, 2014.

[25] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In *27th FOCS*, pages 174–187. IEEE Computer Society Press, October 1986.

[26] Oded Goldreich, Silvio Micali, and Avi Wigderson. How to play any mental game or A completeness theorem for protocols with honest majority. In Alfred Aho, editor, *19th ACM STOC*, pages 218–229. ACM Press, May 1987.

[27] Oded Goldreich and Erez Petrank. The best of both worlds: Guaranteeing termination in fast randomized byzantine agreement protocols. *Inf. Process. Lett.*, 36(1):45–49, 1990.

[28] Shafi Goldwasser and Yehuda Lindell. Secure multi-party computation without agreement. *J. Cryptology*, 18(3):247–287, 2005.

[29] S. Dov Gordon, Feng-Hao Liu, and Elaine Shi. Constant-round MPC with fairness and guarantee of output delivery. In *CRYPTO*, pages 63–82, 2015.

[30] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In Henri Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 466–485. Springer, May 2010.

[31] Yuval Ishai, Manoj Prabhakaran, and Amit Sahai. Founding cryptography on oblivious transfer - efficiently. In David Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, August 2008.

[32] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In Cynthia Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 445–462. Springer, August 2006.

[33] Jonathan Katz and Chiu-Yuen Koo. Round-efficient secure computation in point-to-point networks. In Moni Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 311–328. Springer, May 2007.

[34] Jonathan Katz and Yehuda Lindell. Handling expected polynomial-time strategies in simulation-based security proofs. In Joe Kilian, editor, *TCC 2005*, volume 3378 of *LNCS*, pages 128–149. Springer, February 2005.

[35] Jonathan Katz, Ueli Maurer, Björn Tackmann, and Vassilis Zikas. Universally composable synchronous computation. In Amit Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, March 2013.

[36] Marcel Keller, Peter Scholl, and Nigel P. Smart. An architecture for practical actively secure MPC with dishonest majority. In Ahmad-Reza Sadeghi, Virgil D. Gligor, and Moti Yung, editors, *ACM CCS 13*, pages 549–560. ACM Press, November 2013.

[37] Joe Kilian. Founding cryptography on oblivious transfer. In *20th ACM STOC*, pages 20–31. ACM Press, May 1988.

[38] Eyal Kushilevitz, Yehuda Lindell, and Tal Rabin. Information-theoretically secure protocols and security under composition. In Jon M. Kleinberg, editor, *38th ACM STOC*, pages 109–118. ACM Press, May 2006.

[39] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[40] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. In *34th ACM STOC*, pages 514–523. ACM Press, May 2002.

[41] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. Sequential composition of protocols without simultaneous termination. In Aleta Ricciardi, editor, *21st ACM PODC*, pages 203–212. ACM Press, July 2002.

[42] Yehuda Lindell, Benny Pinkas, Nigel P. Smart, and Avishay Yanai. Efficient constant round multi-party computation combining BMR and SPDZ. In *CRYPTO*, pages 319–338, 2015.

[43] Pratyay Mukherjee and Daniel Wichs. Two round MPC from LWE via multi-key FHE. Cryptology ePrint Archive, Report 2015/345, 2015. http://eprint.iacr.org/.

[44] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980.

[45] Michael O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7-9 November 1983*, pages 403–409. IEEE Computer Society, 1983.

[46] Tal Rabin and Michael Ben-Or. Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In *21st ACM STOC*, pages 73–85. ACM Press, May 1989.

[47] Russell Turpin and Brian A. Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Inf. Process. Lett.*, 18(2):73–76, 1984.

[48] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, November 1982.

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

Ben-Or and El-Yaniv [5] observed that when executing randomized protocols with probabilistic termination in parallel, then the expected running time of the composed protocol (i.e., the rounds its takes for all protocols to give output to any party) is not, in general, 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 [22].) However, as

implied by the following lemma, if these protocols are run in parallel in a straight-forward 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 Feldman-Micali protocol [22] results in a protocol that in expectation takes $\Theta(\log m)$ phases (and thus rounds) to complete.

Lemma A.1. *Let $X_1, \ldots, X_m$ be geometrically distributed 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 [21],

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

from which the statement of the lemma follows immediately by observing that $\sum_{k=1}^{m} \frac{1}{k} = H_m = \Theta(\log m)$. □

B The Model (Cont'd)

In this section we give complementary material to Section 2 and in particular we include a high-level overview of the formulation of synchronous UC from [35]. More concretely, Katz et al. [35] 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 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 to 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. Recall that, as mentioned in Section 2, for the sake of clarity, we do not explicitly mention $\mathcal{F}_{\text{CLOCK}}$ in our constructions.

In [35], for each message that is to be sent in the protocol, the sender and the receiver are given access to an independent single-use channel.[30] We point out, that instead of the bounded-delay

[29] As argued in [35], bounded-delay channels are essential as they allow parties to detect whether or not a message was sent within round.

[30] As pointed out in [35], an alternative approach would be to have a multi-use communication channel; as modelling the actual communication network is out of the scope of the current work, we will use the more standard and formally treated model of single-use channels from [35].

channels, in this work we will assume very simple SFEs[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. 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 (cf. Section 2) using the methodology from [35] 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.[32] 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 P_i receives μ additional activations. Note that P_i uses these activations to execute his current round instructions; since μ is a bound to the number of hybrids used in any round by any party, μ are enough activations 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.) 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. described a way of capturing in UC the property that a protocol is guaranteed to terminate in a given number of rounds. The idea is that a synchronous protocol in regular form which terminates after r rounds realizes the following functionality $\mathcal{F}$. $\mathcal{F}$ 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 $P_i \in \mathcal{P}$; 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:[33] if $\lambda_i = 1$ for all honest P_i then increase $\lambda := \lambda + 1$ and reset $\lambda_i = 0$ for all $P_i \in \mathcal{P}$. 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, $\mathcal{F}$ 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 $\lambda = \rho$.

We note that protocols in the synchronous model of [35] 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. make use of the composition with joint-state (JUC) [12]. The idea is the parties use an $\mathcal{F}_{\text{CLOCK}}$-hybrid protocol $\hat{\pi}$ that emulates towards each of the protocols, sub-clocks and assigns to each sub-clock a unique

[31] In fact, in Section 3 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 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.

[33] 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.

sub-session ID (ssid). Each of these sub-clocks 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 sub-clocks 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 [35, 12].

C Composition Theorem

This section contains the complementing definitions and proofs for Section 4.

Notations. In this section, let $\mathcal{F}_{\text{CSF}}^{f_1,l_1}, \ldots, \mathcal{F}_{\text{CSF}}^{f_m,l_m}$ and $\mathcal{F}_{\text{CSF}}^{f,l}$ be CSFs and denote $\mathcal{F}_i := \mathcal{F}_{\text{CSF}}^{f_i,l_i}$ as well as $\mathcal{F} := \mathcal{F}_{\text{CSF}}^{f,l}$ for brevity. Moreover, let $D_1 \ldots, D_m$ and D be arbitrary distributions over traces. Finally, let π be an SNF protocol that UC-realizes the strictly-wrapped functionality $\mathcal{W}_{\text{strict}}^{D}(\mathcal{F})$ (resp. the flexibly-wrapped functionality $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F})$) in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrids model.

C.1 Using Actual Round Complexities

Each CSF $\mathcal{F}_i$ takes exactly two rounds, and as such, cannot be implemented in general by any real-world protocol. The first step towards having the protocol π be realizable is to increase the round complexity of each hybrid $\mathcal{F}_i$ according to the distributions D_i. We denote by π' be the protocol that replaces each call to a CSF hybrid $\mathcal{F}_i$ by a call to $\mathcal{W}_{\text{strict}}^{D_i}(\mathcal{F}_i)$. We show in Lemma C.1 that this modified protocol still UC-realizes the wrapped functionality $\mathcal{W}_{\text{strict}}^{D'}(\mathcal{F})$ (resp. $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$), for a distribution D' that corresponds to the output of the following procedure:

1. Sample a trace $T \leftarrow \mathsf{Samp}(D)$. The output T is a depth-1 tree with root $\mathcal{W}_{\text{strict}}^{D'}(\mathcal{F})$ and leaves from the set $\{\mathcal{F}_i\}_i$.
2. For each leaf node $\mathcal{F}' = \mathcal{F}_i$ for some i, sample a trace $T_i \leftarrow \mathsf{Samp}(D_i)$ and replace node $\mathcal{F}'$ by T_i.
3. Output the resulting trace T'.

The distribution D' is also denoted by $\mathsf{full\text{-}trace}(D, \{D_i\}_i)$.

Lemma C.1. *Protocol π' UC-realizes $\mathcal{W}_{\text{strict}}^{D'}(\mathcal{F})$ (resp. $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$) in the $\{\mathcal{W}_{\text{strict}}^{D_i}(\mathcal{F}_i)\}_i$-hybrid model.*

Proof. Assume that π UC-realizes the flexibly-wrapped functionality $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$ (the proof for the case where π UC-realizes the strictly-wrapped functionality is similar).

Let $\mathcal{S}$ be the simulator for protocol π running with the dummy adversary (recall that proving security with respect to the dummy adversary [8, Claim 10]). Consider the following simulator $\mathcal{S}'$ for π':

- Internally run a copy of $\mathcal{S}$.
- In round $\rho = 1$, forward all $(\texttt{leakage}, \mathsf{sid}, \cdot)$ messages from $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$ to $\mathcal{S}$. Moreover, $\mathcal{S}'$ receive a trace T' from $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$. At depth 1, T' has wrapped hybrids, each of which is the root of a subtree T_i. Replace each such subtree by its unwrapped root node to obtain a depth-1 tree T. Pass T to $\mathcal{S}$.
- Simulate the execution of all wrapped hybrids $\mathcal{H}_i$ in the order they appear in T' as follows[34] (the first such hybrid must be simulated as early as in round $\rho = 1$):

[34] Recall that the children at each node in a trace are ordered.

- Let ρ_i be the (simulated) round counter of $\mathcal{H}_i$ and let T_i be the corresponding subtree in T'.
- In any round forward the messages $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ (that are directed to $\mathcal{H}_i$) from the environment to $\mathcal{S}$.
- In round $\rho_i = 1$, forward the simulated leakage messages from $\mathcal{S}$ to the environment and add $(\texttt{trace}, \mathsf{sid}, T_i)$ to the first such message. Moreover, forward the $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ commands from $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$ to $\mathcal{S}$ and from $\mathcal{S}$ to the environment. Note that by sending these messages $\mathcal{S}$ eventually advances to simulate the second round of the (unwrapped) CSF hybrid.
- In rounds $\rho_i = 2, \ldots, (2 \cdot c_{\mathsf{tr}}(T_i) - 1)$, forward all $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages that are received from $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$ to the environment (thus bypassing $\mathcal{S}$).
- In round $\rho_i = 2 \cdot c_{\mathsf{tr}}(T_i)$, forward all $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$ to $\mathcal{S}$ and from $\mathcal{S}$ to the environment.

- At any time forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages and $(\texttt{early-output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{S}$ to $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$; note that by the above simulation strategy for hybrids $\mathcal{H}_i$, this only occurs in the first and last round of the simulated execution of a $\mathcal{H}_i$.

Let $\mathcal{Z}'$ be an environment distinguishing between an execution of π' in the $\{\mathcal{W}_{\text{strict}}^{D_i}(\mathcal{F}_i)\}_i$-hybrid model and the ideal model with $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$ and $\mathcal{S}'$. We construct the following environment $\mathcal{Z}$ distinguishing between an execution of π in the $\{\mathcal{F}_i\}_i$-hybrid model and the ideal model with $\mathcal{F}$ and $\mathcal{S}$:

- Internally run a copy of $\mathcal{Z}'$. Denote by $\mathcal{G}$ the system of ITMs $\mathcal{Z}$ interacts with.
- For each two-round CSF hybrid $\mathcal{F}_i$ executed by $\mathcal{G}$, proceed as follows to simulate an execution of $\mathcal{W}_{\text{strict}}^{D_i}(\mathcal{F}_i)$ to $\mathcal{Z}'$:
  - Initialize a round counter $\rho_i \leftarrow 1$.
  - Sample a trace T_i from D_i.
  - At any time, forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from $\mathcal{Z}'$ to $\mathcal{G}$.
  - In round $\rho_i = 1$, forward $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{Z}'$ to $\mathcal{G}$ (to the corresponding party). Moreover, forward $(\texttt{leakage}, \mathsf{sid}, \cdot)$ messages from $\mathcal{G}$ to $\mathcal{Z}'$. Add $(\texttt{trace}, \mathsf{sid}, T_i)$ to the first such message.
  - In rounds $\rho_i = 2, \ldots, (2 \cdot c_{\mathsf{tr}}(T_i) - 1)$, when $\mathcal{Z}'$ sends a $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ message to a party P_i, discard it and pass $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to $\mathcal{Z}'$ on behalf of the adversary (thus bypassing $\mathcal{G}$).
  - In round $\rho_i = 2 \cdot c_{\mathsf{tr}}(T_i)$, forward $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{Z}'$ to $\mathcal{G}$ (to the corresponding party). Moreover, forward $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{G}$ to $\mathcal{Z}'$.
- At any time, forward $(\texttt{output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{G}$ to $\mathcal{Z}'$; note that by the above simulation strategy for hybrids, this occurs only during the last round of such a simulated execution.
- 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 $\mathcal{W}_{\text{strict}}^{D_i}(\mathcal{F}_i)$, and
- when $\mathcal{Z}$ interacts with an ideal-world execution of $\mathcal{F}$ with simulator $\mathcal{S}$, the view of $\mathcal{Z}'$ is exactly the view it would have when interacting with an ideal-world execution of $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$ with simulator $\mathcal{S}'$.

□

C.2 Accounting for Slack

Note that with SNF protocols one "communication round" consists of one call to an CSF, whose evaluation takes two rounds. For convenience, for the remainder of this section, the term *round* will refer to such communication rounds.

This section formalizes the intuition provided in Section 4 and shows how protocols can be modified in order to function properly even if parties initiate them in different rounds and how slack induced by protocols can be reduced to a single round. Two cases are distinguished:

- *Strictly-wrapped case.* In this case protocol π realizes a strictly-wrapped CSF $\mathcal{W}_{\text{strict}}^{D}(\mathcal{F})$, and all hybrids are strictly wrapped, i.e., they are all realized by non-slack-inducing protocols.
- *Flexibly-wrapped case.* In this case protocol π realizes a flexibly-wrapped CSF $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F})$, and some hybrids are strictly wrapped whereas others are flexibly wrapped.

The approaches to adding slack tolerance and slack reduction described in Section 4 are captured by considering two wrappers $\mathcal{W}_{\text{ST}}$ and $\mathcal{W}_{\text{STR}}$, where the former takes care of *slack tolerance (ST)* and the latter adds both *slack tolerance and reduction (STR)*. Looking ahead, the general idea is that strictly-wrapped CSFs are additionally wrapped with $\mathcal{W}_{\text{ST}}$ while flexibly-wrapped ones are wrapped with $\mathcal{W}_{\text{STR}}$.

Strictly-wrapped case. The *slack-tolerance wrapper*, formally defined in Figure 8, is parameterized by a non-negative integer c, which denotes the amount of slack tolerance that is added. The wrapper ensures that all inputs that are provided within c rounds of the first input will be considered in the computation. Wrapper $\mathcal{W}_{\text{ST}}$ then essentially increases the termination round by a factor of $(3c + 1)$, which is due to the slack tolerance technique used (cf. Section 4). In the end, $\mathcal{W}_{\text{ST}}$ ensures that parties obtain output with the same amount of slack they had initially. Note that the notation $(y_1, \ldots, y_n) \leftarrow \mathcal{F}$ stands for advancing the internally run copy of $\mathcal{F}$ until it produces output.

Wrapper Functionality $\mathcal{W}^c_{\text{ST}}(\mathcal{F})$

$\mathcal{W}^c_{\text{ST}}$, parameterized by a non-negative integer c, internally runs a copy of $\mathcal{F}$ and proceeds as follows:

- At all times, forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from the adversary to $\mathcal{F}$.
- Record trace T when it is output by $\mathcal{F}$.
- In rounds $1, \ldots, 2c+1$: Upon receiving a message from some party $P_i \in \mathcal{P}$, proceed as follows:
  - Upon receiving the first message from P_i, record it and set P_i's local slack $c_i \leftarrow \rho - 1$, where ρ is the current round.
  - If $\rho = (c+1) + c_i$, forward the message initially recorded for P_i to $\mathcal{F}$. Forward the $(\texttt{leakage}, \mathsf{sid}, \cdot)$ message $\mathcal{F}$ subsequently outputs to the adversary.
  - Else, send $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.
- In rounds $\rho > 2c+1$: Upon receiving a message from some party $P_i \in \mathcal{P}$, proceed as follows:
  - If $\rho = (3c+1)c_{\mathsf{tr}}(T) + c_i$, compute $(y_1, \ldots, y_n) \leftarrow \mathcal{F}$ (if not already done so), and output $(\texttt{output}, \mathsf{sid}, y_i)$ to P_i.
  - Else, output $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.

Figure 8: The slack-tolerance wrapper functionality

Let π'' be the protocol obtained from π' by replacing each call to a hybrid $\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i)$ by a call to $\mathcal{W}^c_{\text{ST}}(\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i))$. In Lemma C.2 we show that this modified protocol UC-realizes $\mathcal{W}^c_{\text{ST}}(\mathcal{W}^{D'}_{\text{strict}}(\mathcal{F}))$.

Lemma C.2. *Protocol π'' securely realizes $\mathcal{W}^c_{\text{ST}}(\mathcal{W}^{D'}_{\text{strict}}(\mathcal{F}))$ in the $\{\mathcal{W}^c_{\text{ST}}(\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i))\}_i$-hybrid model.*

Proof. Let $\mathcal{S}'$ be the simulator for protocol π' (running with the dummy adversary). Denote by $\mathcal{W}^{D'}_{\text{DT}}(\mathcal{F})$ the wrapped functionality $\mathcal{W}^c_{\text{ST}}(\mathcal{W}^{D'}_{\text{strict}}(\mathcal{F}))$ for brevity. Consider the following simulator $\mathcal{S}''$ for π'':

- Internally run a copy of $\mathcal{S}'$ and set slack indicators $c_1, \ldots, c_n \leftarrow 0$.
- During rounds $\rho = 1, \ldots, c+1$: Upon receiving the first $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ from $\mathcal{W}^{D'}_{\text{DT}}(\mathcal{F})$, set $c_j \leftarrow \rho - 1$.
- In rounds $\rho = (c+1) + c_j$ for each P_j, forward all leakage messages from $\mathcal{W}^{D'}_{\text{DT}}(\mathcal{F})$ to $\mathcal{S}'$.
- Simulate the execution of all wrapped hybrids $\mathcal{H}_i$ in the order they appear in trace T'[35] (output by $\mathcal{W}^{D'}_{\text{DT}}(\mathcal{F})$ with the first leakage command) as follows (the first such hybrid must be simulated as early as in round $\rho = 1$):
  - Let ρ' be the (simulated) round counter of $\mathcal{H}_i$.
  - At any time forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from the environment to $\mathcal{S}$.
  - In rounds $\rho' = 1, 2, \ldots$: Upon receiving $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ from $\mathcal{W}^{D'}_{\text{DT}}(\mathcal{F})$:
    - If $\rho' \equiv (c+1) + c_j \pmod{3c+1}$, input $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ to $\mathcal{S}'$ and output to the environment whatever $\mathcal{S}'$ outputs.[36]

[35] Recall that the children at each node in a trace are ordered.

[36] Note that for the very first hybrid simulated in this way, this instruction is executed upon receiving $(\texttt{leakage}, \mathsf{sid}, P_j, \cdot)$ from $\mathcal{W}^{D'}_{\text{DT}}(\mathcal{F})$.

∗ Otherwise, output $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ to the environment (thus bypassing $\mathcal{S}'$).

- At any time forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages and $(\texttt{early-output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{S}'$ to $\mathcal{W}_{\text{strict}}^{D'}(\mathcal{F})$.

Let $\mathcal{Z}''$ be an environment distinguishing between an execution of π'' in the $\{\mathcal{W}_{\text{DT}}^{D_i}(\mathcal{F}_i)\}_i$-hybrid model and the ideal model with $\mathcal{W}_{\text{DT}}^{D'}(\mathcal{F}))$ and $\mathcal{S}''$. Consider the following environment $\mathcal{Z}'$ distinguishing between an execution of π' in the $\{\mathcal{W}_{\text{strict}}^{D_i}(\mathcal{F}_i)\}_i$-hybrid model and the ideal model with $\mathcal{W}_{\text{strict}}^{D'}(\mathcal{F})$ and $\mathcal{S}$:

- Internally run a copy of $\mathcal{Z}''$. Denote by $\mathcal{G}'$ the system of ITMs $\mathcal{Z}'$ interacts with .
- Set slack indicators $c_1, \ldots, c_n \leftarrow 0$.
- During rounds $\rho = 1, \ldots, c+1$: Upon $\mathcal{Z}''$ outputting the first $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ for party P_j, set $c_j \leftarrow \rho - 1$.
- For each hybrid $\mathcal{W}_{\text{strict}}^{D_i}(\mathcal{F}_i)$ executed by $\mathcal{G}'$, proceed as follows to simulate an execution of $\mathcal{W}_{\text{ST}}^{c}(\mathcal{W}_{\text{strict}}^{D_i}(\mathcal{F}_i))$ to $\mathcal{Z}''$:
  - Initialize a round counter $\rho' \leftarrow 1$.
  - At any time, forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from $\mathcal{Z}''$ to $\mathcal{G}'$.
  - In rounds $\rho' = 1, 2, \ldots$: Upon $\mathcal{Z}''$ outputting $(\texttt{fetch-output}, \mathsf{sid}, P_j)$:
    - If $\rho' \equiv (c+1) + c_j \pmod{3c+1}$, input $(\texttt{fetch-output}, \mathsf{sid})$ to $\mathcal{G}'$ for P_j and output to the $\mathcal{Z}''$ whatever $\mathcal{G}'$ outputs on behalf of the adversary.
    - Otherwise, output $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ to $\mathcal{Z}''$ on behalf of the adversary.[37]
- At any time, forward $(\texttt{output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{G}'$ to $\mathcal{Z}'$.

It can be seen by inspection that

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

□

Flexibly-wrapped case. The *slack-tolerance and reduction wrapper*, formally defined in Figure 9, is parameterized by a non-negative integer c, which as before denotes the amount of slack tolerance that is added. The wrapper ensures that all inputs that are provided within c rounds of the first input will be considered in the computation. After the initial phase, wrapper $\mathcal{W}_{\text{STR}}$ proceeds differently than $\mathcal{W}_{\text{ST}}$. Firstly, it computes the termination round as

$$\rho_{\mathsf{term}} \leftarrow (3c+1)c_{\mathsf{tr}}(T) + (c+1) + 2\nu,$$

[37] Note that for the very first hybrid simulated in this way, this instruction is executed upon $\mathcal{G}'$ outputting $(\texttt{input}, \mathsf{sid}, \cdot)$.

where ν is the number of flexibly-wrapped CSFs in the trace T initially output by the functionality being wrapped. The term 2ν is due to the fact that after each invocation of such a hybrid, the slack-reduction protocol described in Section 4 must be executed, which lasts for two rounds; the term $(c+1)$ ensures that there are enough rounds for the slowest parties to terminate. Note that the notation $y \leftarrow \mathcal{G}$ stands for advancing the internally run copy of $\mathcal{G}$ until it produces output.

Wrapper Functionality $\mathcal{W}_{\text{STR}}(\mathcal{F})$

$\mathcal{W}^c_{\text{STR}}$, parameterized by a non-negative integer c, internally runs a copy of $\mathcal{F}$ (where $\mathcal{F}$ is a public-output functionality) and proceeds as follows:

- Initially, set termination indicators $\mathsf{term}_1, \ldots, \mathsf{term}_n \leftarrow 0$.
- At all times, forward `adv-input`-messages from the adversary to $\mathcal{F}$.
- Upon receiving the first message from P_i, record it and set P_i's local slack $c_i \leftarrow \rho - 1$, where ρ is the current round.
- Record trace T when it is output by $\mathcal{F}$. Set $\rho_{\mathsf{term}} \leftarrow (3c+1)c_{\mathsf{tr}}(T) + (c+1) + 2\nu$, where ν is the number of flexibly-wrapped functionalities in T.
- In rounds $1, \ldots, 2c+1$: Upon receiving a message from some party $P_i \in \mathcal{P}$, proceed as follows:
  - If $\rho = (c+1) + c_i$, forward the message initially recorded for P_i to $\mathcal{F}$. Forward the `leakage`-message $\mathcal{F}$ subsequently outputs to the adversary.
  - Else, output $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.
- In rounds $\rho > 2c+1$:
  - Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some party $P_i \in \mathcal{P}$, proceed as follows:
    - If $\mathsf{term}_i = 1$, output $(\texttt{output}, \mathsf{sid}, y)$ to P_i.
    - Otherwise, send $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.
  - Upon receiving $(\texttt{early-output}, \mathsf{sid}, P_i)$ from the adversary, set $\mathsf{term}_i \leftarrow 1$ and $\rho_{\mathsf{term}} \leftarrow \min\{\rho_{\mathsf{term}}, \rho+1\}$, compute $y \leftarrow \mathcal{F}$, and output $(\texttt{output}, \mathsf{sid}, y)$ to the adversary.
- In round $\rho = \rho_{\mathsf{term}}$: Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some party $P_i \in \mathcal{P}$, compute $y \leftarrow \mathcal{F}$ (if not already done so) and output $(\texttt{output}, \mathsf{sid}, y)$ to P_i.

Figure 9: The slack-tolerance and reduction wrapper functionality

Recall that in this case some of the hybrids $\mathcal{F}_i$ need to be strictly wrapped and others flexibly. Let I be a set indexing the hybrids that need to be wrapped flexibly. Let π'' be the protocol obtained from π' by

- replacing each call to a hybrid $\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i)$ with $i \notin I$ by a call to $\mathcal{W}^c_{\text{ST}}(\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i))$,
- replacing each call to a hybrid $\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i)$ with $i \in I$ by a call to $\mathcal{W}^c_{\text{STR}}(\mathcal{W}^{D_i}_{\text{flex}}(\mathcal{F}_i))$, and
- by executing the termination protocol described in Section 4.

This modified protocol realizes $\mathcal{W}^c_{\text{ST}}(\mathcal{W}^{D'}_{\text{flex}}(\mathcal{F}))$, as shown below.

Lemma C.3. *Protocol π'' securely UC-realizes $\mathcal{W}^c_{\text{STR}}(\mathcal{W}^{D'}_{\text{flex}}(\mathcal{F}))$ in the $(\{\mathcal{W}^c_{\text{ST}}(\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i))\}_{i\notin I}, \{\mathcal{W}^c_{\text{STR}}(\mathcal{W}^{D_i}_{\text{flex}}(\mathcal{F}_i))\}_{i\in I})$-hybrid model.*

Proof. Let $\mathcal{S}'$ be the simulator for protocol π' (running with the dummy adversary). Denote by $\mathcal{W}_{\text{PT}}^{D'}(\mathcal{F})$ the wrapped functionality $\mathcal{W}_{\text{STR}}^{c}(\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F}))$ for brevity. Consider the following simulator $\mathcal{S}''$ for π'':

- Internally run a copy of $\mathcal{S}'$. Set slack indicators $c_1, \ldots, c_n \leftarrow 0$.
- During rounds $\rho = 1, \ldots, c+1$: Upon receiving the first $(\texttt{fetch-output}, \cdot, P_j)$ from $\mathcal{W}_{\text{PT}}^{D'}(\mathcal{F})$, set $c_j \leftarrow \rho - 1$.
- In rounds $\rho = (c+1) + c_j$ for each P_j, forward all leakage messages from $\mathcal{W}_{\text{PT}}^{D'}(\mathcal{F})$ to $\mathcal{S}'$.
- Simulate the execution of all wrapped hybrids $\mathcal{H}$ in the order they appear in trace T'[38] (output by $\mathcal{W}_{\text{PT}}^{D'}(\mathcal{F})$ with the first leakage command; the first such hybrid must be simulated as early as in round $\rho = 1$). If the hybrid is *not* indexed by I, proceed as follows:
  - Let ρ' be the (simulated) round counter of $\mathcal{H}$.
  - At any time forward $\texttt{adv-input}$-messages from the environment to $\mathcal{S}$.
  - In rounds $\rho' = 1, 2, \ldots$: Upon receiving $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ from $\mathcal{W}_{\text{PT}}^{D'}(\mathcal{F})$:
    - If $\rho' \equiv (c+1) + c_j \pmod{3c+1}$, input $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ to $\mathcal{S}'$ and output to the environment whatever $\mathcal{S}'$ outputs.[39]
    - Otherwise, output $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ to the environment (thus bypassing $\mathcal{S}'$).

  If the hybrid *is* indexed by I, proceed as follows:
  - Let ρ' be the (simulated) round counter of $\mathcal{H}$.
  - At any time forward $\texttt{adv-input}$-messages from the environment to $\mathcal{S}$.
  - In rounds $\rho' = 1, 2, \ldots$: Upon receiving $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ from $\mathcal{W}_{\text{PT}}^{D'}(\mathcal{F})$:
    - If $\rho' \equiv (c+1) + c_j \pmod{3c+1}$, input $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ to $\mathcal{S}'$ and output to the environment whatever $\mathcal{S}'$ outputs.[40] Record the trace $T_{\mathcal{H}}$ output by $\mathcal{S}'$ the first time this instruction is executed and compute $\rho'_{\mathsf{term}} = (3c+1)c_{\mathsf{tr}}(T) + (c+1) + 2\nu$.
    - Otherwise, output $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ to the environment (thus bypassing $\mathcal{S}'$).
  - If the environment issues $\texttt{early-output}$ commands for certain parties before round ρ'_{term}, set $c_i \leftarrow 0$ for these players and $c_i \leftarrow 1$ for the others and end the simulation of $\mathcal{H}$ one round later. If the environment does not issue such a command for any players, set $c_i \leftarrow 0$ for all players and end the simulation of $\mathcal{H}$ in round ρ'_{term}.
- At any time forward $\texttt{adv-input}$-messages from $\mathcal{S}'$ to $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$.
- When $\mathcal{S}'$ wants to output $(\texttt{early-output}, \mathsf{sid}, P_j)$ to $\mathcal{W}_{\text{PT}}^{D'}(\mathcal{F})$, proceed as follows:
  - Forward $(\texttt{early-output}, \mathsf{sid}, P_j)$ to $\mathcal{W}_{\text{PT}}^{D'}(\mathcal{F})$ and obtain $(\texttt{output}, \mathsf{sid}, y)$. Record y (the first time).
  - Simulate P_j sending $(\texttt{end}, y)$ to all parties.

[38] Recall that the children at each node in a trace are ordered.

[39] Note that for the very first hybrid simulated in this way, this instruction is executed upon receiving $(\texttt{leakage}, \mathsf{sid}, P_j, \cdot)$ from $\mathcal{W}_{\text{PT}}^{D'}(\mathcal{F})$.

[40] Note that for the very first hybrid simulated in this way, this instruction is executed upon receiving $(\texttt{leakage}, \mathsf{sid}, P_j, \cdot)$ from $\mathcal{W}_{\text{PT}}^{D'}(\mathcal{F})$.

– For every party P_j, keep track of how many `end`-messages it has received (including those by corrupted parties).
  * When a party receives $t+1$ such messages (for the same y) and hasn't sent any of its own, it sends them at this point.
  * When a party $party_j$ receives $n-t$ such messages (for the same y), $\mathcal{S}''$ outputs $(\texttt{early-output}, \mathsf{sid}, P_j)$ to $\mathcal{W}_{\text{PT}}^{D'}(\mathcal{F})$.

Let $\mathcal{Z}''$ be an environment distinguishing between an execution of π'' in the $\{\mathcal{W}_{\text{STR}}^{c}(\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_i))\}_{i\in I}$-hybrid model and the ideal model with $\mathcal{W}_{\text{STR}}(\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F}))$ and $\mathcal{S}''$. Consider the following environment $\mathcal{Z}'$ distinguishing between an execution of π' in the $\{\mathcal{W}_{\text{strict}}^{D_i}(\mathcal{F}_i)\}_i$-hybrid model and the ideal model with $\mathcal{W}_{\text{flex}}^{D'}(\mathcal{F})$ and $\mathcal{S}$:

- Internally run a copy of $\mathcal{Z}''$. Denote by $\mathcal{G}'$ the system of ITMs $\mathcal{Z}'$ interacts with.
- Set slack indicators $c_1, \ldots, c_n \leftarrow 0$.
- During rounds $\rho = 1, \ldots, c+1$: Upon $\mathcal{Z}''$ outputting the first $(\texttt{fetch-output}, \cdot, P_i)$ for party P_j, set $c_j \leftarrow \rho - 1$.
- For each hybrid $\mathcal{W}_{\text{strict}}^{D_i}(\mathcal{F}_i)$ with $i \notin I$ executed by $\mathcal{G}'$, proceed as follows to simulate an execution of $\mathcal{W}_{\text{ST}}^{c}(\mathcal{W}_{\text{strict}}^{D_i}(\mathcal{F}_i))$ to $\mathcal{Z}''$:
  – Initialize a round counter $\rho' \leftarrow 1$.
  – At any time, forward `adv-input`-messages from $\mathcal{Z}''$ to $\mathcal{G}'$.
  – In rounds $\rho' = 1, 2, \ldots$: Upon $\mathcal{Z}''$ outputting $(\texttt{fetch-output}, \mathsf{sid}, P_j)$:
    * If $\rho' \equiv (c+1) + c_j \pmod{3c+1}$, input $(\texttt{fetch-output}, \mathsf{sid})$ to $\mathcal{G}'$ for P_j and output to the $\mathcal{Z}''$ whatever $\mathcal{G}'$ outputs on behalf of the adversary.
    * Otherwise, output $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ to $\mathcal{Z}''$ on behalf of the adversary.[41]
- For each hybrid $\mathcal{W}_{\text{strict}}^{D_i}(\mathcal{F}_i)$ with $i \in I$ executed by $\mathcal{G}'$, proceed as follows to simulate an execution of $\mathcal{W}_{\text{STR}}^{c}(\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_i))$ to $\mathcal{Z}''$:
  – Initialize a round counter $\rho' \leftarrow 1$.
  – At any time, forward `adv-input`-messages from $\mathcal{Z}''$ to $\mathcal{G}'$.
  – In rounds $\rho' = 1, 2, \ldots$: Upon $\mathcal{Z}''$ outputting $(\texttt{fetch-output}, \mathsf{sid}, P_j)$:
    * If $\rho' \equiv (c+1) + c_j \pmod{3c+1}$, input $(\texttt{fetch-output}, \mathsf{sid})$ to $\mathcal{G}'$ for P_j and output to the $\mathcal{Z}''$ whatever $\mathcal{G}'$ outputs on behalf of the adversary. Record the trace $T_{\mathcal{H}}$ output by $\mathcal{S}'$ the first time this instruction is executed and compute $\rho'_{\mathsf{term}} = (3c+1)c_{\mathsf{tr}}(T) + (c+1) + 2\nu$.
    * Otherwise, output $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ to $\mathcal{Z}''$ on behalf of the adversary.[42]
  – If $\mathcal{Z}''$ issues `early-output` commands for certain parties before round ρ'_{term}, set $c_i \leftarrow 0$ for these players and $c_i \leftarrow 1$ for the others and end the simulation of $\mathcal{H}$ one round later (by sending `fetch-output`-messages on behalf of the parties to $\mathcal{G}'$). If the environment does not issue such a command for any players, set $c_i \leftarrow 0$ for all players and end the simulation of $\mathcal{H}$ in round ρ'_{term}.

[41] Note that for the very first hybrid simulated in this way, this instruction is executed upon $\mathcal{G}'$ outputting $(\texttt{input}, \mathsf{sid}, \cdot)$.

[42] Note that for the very first hybrid simulated in this way, this instruction is executed upon $\mathcal{G}'$ outputting $(\texttt{input}, \mathsf{sid}, \cdot)$.

- When $\mathcal{W}^{D'}_{\text{PT}}(\mathcal{F})$ outputs $(\texttt{output}, \mathsf{sid}, y)$ to a party P_j, proceed as follows:
  - Simulate to $\mathcal{Z}''$ that P_j sends $(\texttt{end}, y)$ to all parties.
  - For every party P_j, keep track of how many $\texttt{end}$-messages it has received (including those by corrupted parties).
    * When a party receives $t+1$ such messages (for the same y) and hasn't sent any of its own, it sends them at this point.
    * When a party P_j receives $n-t$ such messages (for the same y), $\mathcal{Z}'$ outputs $(\texttt{output}, \mathsf{sid}, y)$ to $\mathcal{Z}''$ (for party P_i).

It can be seen by inspection that

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

□

C.3 The Composition Theorem

Set $\mathcal{W}^{D}_{\text{DT}}(\cdot) := \mathcal{W}_{\text{ST}}(\mathcal{W}^{D}_{\text{strict}}(\cdot))$ and $\mathcal{W}^{D}_{\text{PT}}(\cdot) := \mathcal{W}_{\text{STR}}(\mathcal{W}^{D}_{\text{flex}}(\cdot))$. Moreover, denote by $\mathsf{Comp}_{\text{DT}}(\pi)$ the "compiled" protocol π'' resulting in the first case and by $\mathsf{Comp}_{\text{PT}}(\pi)$ the "compiled" protocol π'' resulting in the second case above.

Round complexity. Consider first the strictly-wrapped case. The expected round complexity of the compiled protocol $\mathsf{Comp}_{\text{DT}}(\pi)$ is (cf. Section C.1)

$$\sum_i d_i \cdot (3c+1) \cdot 2 \cdot E[c_{\mathsf{tr}}(T_i)],$$

where d_i is the expected number of calls to hybrid $\mathcal{F}_i$ and T_i is a trace sampled from D_i. This follows from the fact that each hybrid called by π is replaced by $\mathcal{W}^{D_i}_{\text{DT}}(\mathcal{F}_i)$, which increases the round complexity by a factor of $(3c+1)$.

Note that (recursively) the round complexity of a protocol realizing $\mathcal{W}^{D_i}_{\text{DT}}(\mathcal{F}_i)$ already contains the factor $2(3c+1)$, and, therefore, these factors do not accumulate when one replaces hybrids by their realizing protocols!

In the flexibly-wrapped case, the expected round complexity of the compiled protocol $\mathsf{Comp}_{\text{PT}}(\pi)$ is (cf. Section C.2)

$$\sum_i d_i \cdot (3c+1) \cdot 2 \cdot E[c_{\mathsf{tr}}(T_i)] + (c+1) + 2\nu,$$

where d_i is the expected number of calls to hybrid $\mathcal{F}_i$, T_i is a trace sampled from D_i, and where ν is the expected number of flexibly-wrapped CSF in T'.

In summary, note in particular that in either case, if π and all the protocols realizing the hybrids are expected-constant-round protocols, so is the compiled protocol.

Therefore, overall we have proven the following two theorems:

Theorem 4.2. *Let $\{\mathcal{F}_i\}_i$ and $\mathcal{F}$ be arbitrary CSFs, D and $\{D_i\}_i$ arbitrary distributions over traces, and $D' = \mathsf{full\text{-}trace}(D, \{D_i\}_i)$. If an SNF protocol π securely realizes $\mathcal{W}^D_{\text{strict}}(\mathcal{F})$ in the $\{\mathcal{F}_i\}_i$-hybrid model, then $\mathsf{Comp}^c_{\text{DT}}(\pi)$ securely realizes $\mathcal{W}^{D'}_{\text{DT}}(\mathcal{F})$ in the $\{\mathcal{W}^{D_i}_{\text{DT}}(\mathcal{F}_i)\}_i$-hybrid model.*

The expected round complexity of the compiled protocol $\mathsf{Comp}^c_{\text{DT}}(\pi)$ is

$$2(3c+1)\sum_i d_i \cdot E[c_{\mathsf{tr}}(T_i)],$$

where d_i is the expected number of calls to hybrid $\mathcal{F}_i$ and T_i is a trace sampled from D_i.

Theorem 4.3. *Let $\{\mathcal{F}_i\}_i$ and $\mathcal{F}$ be arbitrary CSFs, D and $\{D_i\}_i$ arbitrary distributions over traces, and $D' = \mathsf{full\text{-}trace}(D, \{D_i\}_i)$. Moreover, let I be a set indexing a subset of the CSFs $\mathcal{F}_i$. If an SNF protocol π securely realizes $\mathcal{W}^D_{\mathsf{flex}}(\mathcal{F})$ in the $\{\mathcal{F}_i\}_i$-hybrid model, then $\mathsf{Comp}^c_{\text{PT}}(\pi)$ securely realizes $\mathcal{W}^{D'}_{\text{PT}}(\mathcal{F})$ in the $(\{\mathcal{W}^{D_i}_{\text{DT}}(\mathcal{F}_i)\}_{i\notin I}, \{\mathcal{W}^{D_i}_{\text{PT}}(\mathcal{F}_i)\}_{i\in I})$-hybrid model.*

The expected round complexity of the compiled protocol $\mathsf{Comp}^c_{\text{PT}}(\pi)$ is

$$2(3c+1)\sum_i d_i \cdot E[c_{\mathsf{tr}}(T_i)] + \mathcal{O}(1),$$

where d_i is the expected number of calls to hybrid $\mathcal{F}_i$ and T_i is a trace sampled from D_i.

C.4 Wrapping SMTs

Note that with SNF protocols one "communication round" consists of one call to an CSF, whose evaluation takes two rounds. For convenience, for the remainder of this section, the term *round* will refer to such communication rounds.

This section shows how from (unwrapped) parallel-SMF CSFs $\mathcal{F}_{\text{PSMT}}$, one can realize $\mathcal{W}^D_{\text{DT}}(\mathcal{F}_{\text{PSMT}})$,[43] where D is the distribution that always samples the trace with root $\mathcal{W}^D_{\text{strict}}(\mathcal{F}_{\text{PSMT}})$ and a single child $\mathcal{F}_{\text{PSMT}}$.

Output-round wrapping. The dummy protocol realizes $\mathcal{W}^D_{\text{strict}}(\mathcal{F}_{\text{PSMT}})$ using $\mathcal{F}_{\text{PSMT}}$ as hybrid. It is easily verified that the simulator that hides the trace output by $\mathcal{W}^D_{\text{strict}}(\mathcal{F}_{\text{PSMT}})$ but otherwise forwards between $\mathcal{W}^D_{\text{strict}}(\mathcal{F}_{\text{PSMT}})$ and the environment is adequate.

Adding slack tolerance. The protocol in Figure 10 in the $\mathcal{W}^D_{\text{strict}}(\mathcal{F}_{\text{PSMT}})$-hybrid model is based on the intuition provided in Section 4: Every round is expanded into $2c+1$ rounds, during all of which a party listens for round-r messages from other parties while it sends its own round-r messages in round $c+1$. In addition, to ensure that once some honest party has produced output, no party receives adversarial messages, every party waits (without listening) for an additional c rounds in the end before outputting the values received during the first $2c+1$ rounds.

[43] Recall that $\mathcal{W}^D_{\text{DT}}(\cdot) = \mathcal{W}_{\text{ST}}(\mathcal{W}^D_{\text{strict}}(\cdot))$.

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

Each party $P_i \in \mathcal{P} = \{P_1, \ldots, P_n\}$ proceeds as follows:

- Initially, obtain input $(\texttt{input}, \mathsf{sid}, (x_1^{(i)}, \ldots, x_n^{(i)}))$ from the environment. Set $y_1, \ldots, y_n \leftarrow \perp$.
- In every round $\rho = 1, \ldots, c$:[a] Send $(\texttt{input}, \mathsf{sid}, \perp)$ to (a fresh instance of) $\mathcal{F}_{\text{PSMT}}$. Obtain output $(\texttt{output}, \mathsf{sid}, (u_1, \ldots, u_n))$ 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 $(\texttt{input}, \mathsf{sid}, (x_1^{(i)}, \ldots, x_n^{(i)}))$ to $\mathcal{F}_{\text{PSMT}}$. Obtain output $(\texttt{output}, \mathsf{sid}, (u_1, \ldots, u_n))$ 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 $(\texttt{input}, \mathsf{sid}, \perp)$ to $\mathcal{F}_{\text{PSMT}}$. Obtain output $(\texttt{output}, \mathsf{sid}, (u_1, \ldots, u_n))$ 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 $(\texttt{output}, \mathsf{sid}, (y_1, \ldots, y_n))$.

Figure 10: The wrapped parallel secure message transmission protocol, in the $\mathcal{F}_{\text{PSMT}}$-hybrid model

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

Lemma C.4. *Protocol π UC-realizes $\mathcal{W}_{\text{ST}}(\mathcal{W}^D_{\text{strict}}(\mathcal{F}_{\text{PSMT}}))$ in the $\mathcal{W}^D_{\text{strict}}(\mathcal{F}_{\text{PSMT}})$-hybrid model.*

Proof. Set $\mathcal{W}^D_{\text{DT}}(\mathcal{F}_{\text{PSMT}}) := \mathcal{W}_{\text{ST}}(\mathcal{W}^D_{\text{strict}}(\mathcal{F}_{\text{PSMT}}))$ for brevity. Consider the following simulator (running with the dummy adversary):

- Initialize variables $u_{ij} \leftarrow \perp$ for $i, j = 1, \ldots, n$ and slack variables $c_1, \ldots, c_n \leftarrow 0$.
- Upon receiving $(\texttt{leakage}, \mathsf{sid}, P_i, l_i)$ from $\mathcal{W}^D_{\text{DT}}(\mathcal{F}_{\text{PSMT}})$, record l_i, set $c_i \leftarrow \rho - 1$, where ρ is the current round number.
- Simulate $\rho = 3c + 1$ sequential instances of $\mathcal{F}_{\text{PSMT}}$. In particular:
  - If the environment instructs a corrupted party P_i to send a message m to a party P_j, set $u_{ij} \leftarrow m$.
  - For each party P_i, starting in round c_i and ending in round $(2c + 1) + c_i$, output $(\texttt{leakage}, \mathsf{sid}, P_i, l)$, where $l = l_i$ if the current round is $\rho = (c + 1) + c_i$ and $l = 0$ otherwise.
- After completion of the above simulation, use the adversary input of $\mathcal{W}^D_{\text{DT}}(\mathcal{F}_{\text{PSMT}})$ to set the values sent by corrupted players according to the variables u_{ij}.

By inspection it can be seen that a real-world execution of π is indistinguishable from an execution of $\mathcal{W}^D_{\text{DT}}(\mathcal{F}_{\text{PSMT}})$ with the above simulator. □

D Applications of our Fast Composition Theorem (Cont'd)

This section includes complementary material to Section 5.

D.1 Fast and Perfectly Secure Byzantine Agreement (Cont'd)

In Section 5.1 we presented the randomized binary Byzantine agreement protocol π_{RBA}, we now proceed to prove Lemma 5.1.

Lemma 5.1. *Let $p = 0.35$ and $t < n/3$. Denote by D_{RBA} the distribution that outputs a depth-1 trace, where the root is $\mathcal{W}_{\text{flex}}^{D_{\text{RBA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$, and the leaves are set as follows: initially sample an integer r from the geometric distribution with parameter p 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.*

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 (sketch). 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 [22, 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 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 [22].
- 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 $\mathcal{F}_{\text{BA}}$ and output the result, hence, correctness follows from the $\mathcal{F}_{\text{BA}}$ functionality.
- In case all honest parties receive their output in phase τ (i.e., $\mathsf{term} = \tau$ for all honest parties), then by [22] they receive the same value. In this case, this is the value they will output after calling $\mathcal{F}_{\text{BA}}$ and so correctness is satisfied.
- In case some honest parties receive their output 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 $\mathcal{F}_{\text{BA}}$, and correctness is satisfied.
- In case some honest parties receive their output in phase $\tau - 1$ (i.e., $\mathsf{term} = \tau - 1$) and they do not send any input to $\mathcal{F}_{\text{BA}}$. However, the remaining honest parties will receive the same output in phase τ (i.e., $\mathsf{term} = 0$), and will output this value, *regardless* of the output they receive from $\mathcal{F}_{\text{BA}}$. Therefore, correctness is satisfied.

Regarding termination, [22, 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 $\tau = \log^{1.5}(k) + 1$ 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 $\mathcal{F}_{\text{BA}}$.

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

- $\mathcal{S}$ 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 $(\texttt{leakage}, \mathsf{sid}, P_i, (x_1, \ldots, x_n))$. 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 $\mathcal{F}_{\text{BA}}$).
- 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 $P_i \in \mathcal{P}$, $\mathcal{S}$ 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 $(\texttt{early-output}, \mathsf{sid}, P_i)$ for every $P_i \in \mathcal{P}'$. In addition, $\mathcal{S}$ proceeds based on the following cases:
  - In case $r < \tau$, $\mathcal{S}$ sends $(\texttt{early-output}, \mathsf{sid}, P_i)$ for every $P_i \in \mathcal{P} \setminus \mathcal{P}'$ 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 $\mathcal{F}_{\text{BA}}$, $\mathcal{S}$ 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 $\mathcal{F}_{\text{BA}}$ would, and sends to the adversary the output values. (Recall that the output value from $\mathcal{F}_{\text{BA}}$ is not being used by the honset 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 $\mathcal{F}_{\text{BA}}$ 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 $\mathcal{F}_{\text{BA}}$ to the adversary. Initially, $\mathcal{S}$ simulates all honest parties sending their local intermediate value as their input to $\mathcal{F}_{\text{BA}}$, 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 $\mathcal{F}_{\text{BA}}$, 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 `adv-input` message sent by the adversary.

It follows using a standard hybrid argument that for every environment $\mathcal{Z}$ it holds that

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

□

D.1.1 Multi-Valued Byzantine Agreement Protocol

As presented above, π_{RBA} is a binary BA protocol. Using a transformation due to Turpin and Coan [47], the decision domain can be extended without increasing the expected running time. See Lemma D.1.

Protocol $\pi_{\text{MV-BA}}$

The protocol $\pi_{\text{MV-BA}}$ is parameterized by the set V. Each party $P_i \in \mathcal{P} = \{P_1, \ldots, P_n\}$ proceeds as follows:

- Initially, P_i sets the values $y \leftarrow \perp$, $z \leftarrow \perp$ and $vote \leftarrow 0$.
- In round $\rho = 1$: upon receiving $(\texttt{input}, \mathsf{sid}, v_i)$ from the environment, P_i sends (sid, v_i) to all the parties (via $\mathcal{F}_{\text{PSMT}}$).
- In round $\rho = 2$: send $(\texttt{fetch-output}, \mathsf{sid})$ to $\mathcal{F}_{\text{PSMT}}$ and denote by v_j the value received from P_j is the this round.
- In round $\rho = 3$: if there exists a value $v \in V$ that appears more than $n - t$ times in the set $\{v_1, \ldots, v_n\}$ then set $y \leftarrow v$. Send (sid, y) to all the parties (via $\mathcal{F}_{\text{PSMT}}$).
- In round $\rho = 4$: send $(\texttt{fetch-output}, \mathsf{sid})$ to $\mathcal{F}_{\text{PSMT}}$ and denote by y_j the value received from P_j is this round.
- In round $\rho = 5$: if there exists a value $v \in V$ that appears more than $n - t$ times in the set $\{y_1, \ldots, y_n\}$ then set $vote \leftarrow 1$. In addition, set z to be the value that appears the most in $\{y_1, \ldots, y_n\}$. Send $(\texttt{input}, \mathsf{sid}, vote)$ to $\mathcal{F}_{\text{BA}}^{\{0,1\}}$.
- In round $\rho = 6$: send $(\texttt{fetch-output}, \mathsf{sid})$ to $\mathcal{F}_{\text{BA}}^{\{0,1\}}$, and receive $(\texttt{output}, \mathsf{sid}, b)$ with $b \in \{0, 1\}$. If $b = 1$ then output $(\texttt{output}, \mathsf{sid}, z)$, otherwise output $(\texttt{output}, \mathsf{sid}, v_0)$ for some default $v_0 \in V$.

Figure 11: The multi-valued Byzantine agreement protocol, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model

The following lemma follows from Turpin and Coan [47].

Lemma D.1. *Let $t < n/3$ and $V \subseteq \{0,1\}^*$. Denote by $D_{\text{MV-BA}}$ the distribution that outputs a depth-1 trace, with three leaves $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$.*

Then, assuming all honest parties receive their inputs at the same round, the protocol $\pi_{\text{MV-BA}}$ UC-realizes $\mathcal{W}_{\text{flex}}^{D_{\text{MV-BA}}}(\mathcal{F}_{\text{BA}}^{V})$, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model, with perfect security, in the presence of adaptive malicious t-adversaries.

The proof of the Lemma is straight-forward, it will appear in the full version of the paper.

Theorem 5.3. *Let $c \geq 0$, $t < n/3$ and $V \subseteq \{0,1\}^*$. There exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}_{\text{PT}}^{D}(\mathcal{F}_{\text{BA}}^{V})$ has an expected-constant-round complexity, and can be UC-realized in the $\mathcal{F}_{\text{PSMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within c consecutive rounds.*

Proof (sketch). Let $D_{\text{MV-BA}}$ be as in Lemma D.1 and let D_{PSMT} and D_{RBA} as in Theorem 5.2. Let $D'_{\text{MV-BA}}, D'_{\text{PSMT}}$ and D'_{RBA} be the translated distributions (respectively).

Then, following Theorem 4.3 and Lemma D.1, the compiled protocol $\mathsf{Comp}_{\text{PT}}^{c}(\pi_{\text{MV-BA}})$ UC-realizes $\mathcal{W}_{\text{PT}}^{D'_{\text{MV-BA}}}(\mathcal{F}_{\text{BA}}^{V})$, in the $(\mathcal{W}_{\text{DT}}^{D'_{\text{PSMT}}}(\mathcal{F}_{\text{PSMT}}), \mathcal{W}_{\text{PT}}^{D'_{\text{RBA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}}))$-hybrid model, with perfect security, in the presence of adaptive malicious adversaries corrupting at most $t < n/3$ of the parties.

The proof follows since, following Lemma C.4 the functionality $\mathcal{W}_{\text{DT}}^{D'_{\text{PSMT}}}(\mathcal{F}_{\text{PSMT}})$ can be UC-realized in the $\mathcal{F}_{\text{PSMT}}$-hybrid model, and following Theorem 5.2 the functionality $\mathcal{W}_{\text{PT}}^{D'_{\text{RBA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$ can be UC-realized in the $\mathcal{F}_{\text{PSMT}}$-hybrid model. □

D.2 Parallel Broadcast with Probabilistic Termination (Cont'd)

Our construction proceeds in two steps. In a first step we show how to adapt the protocol from Ben-Or and El-Yaniv [5] to obtain a probabilistic-termination (expected-constant-round) version of *unfair* parallel broadcast with perfect security. In step two, we use (and improve on) an idea due to Hirt and Zikas [30] to transform our unfair protocol into a fair parallel broadcast protocol.

D.2.1 The Unfair Parallel Broadcast Protocol

In this section we adjust the interactive-consistency protocol of Ben-Or and El-Yaniv [5] (with minor adjustments) to the UC framework. The protocol π_{UPBC} (see Figure 12 for a detailed description) is parameterized 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 if 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 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 if 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 3.1), where the parties compute the function $g_{\mathsf{le}}(\lambda, \ldots, \lambda) = (k, \ldots, k)$. We denote by $\mathcal{F}_{\text{LE}}$ the functionality $\mathcal{F}_{\text{SFE}}^{g_{\mathsf{le}}}$.

Truncated Byzantine Agreement The *truncated Byzantine agreement* functionality, is parameterized by an efficiently-sampleable distribution D and a non-negative integer d. Each party P_i has input x_i, 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 $f_{\text{T-RBA}}(x_1, \ldots, x_n, a) = ((y_1^i, y_2^i), \ldots, (y_1^i, y_2^i))$ 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 [5] augmented with (a simpler version of) the technique from [27], where all hybrids used are CSFs;[44] using Theorem 4.3 we then obtain our result. Recall that the unfair parallel broadcast functionality $\mathcal{F}_{\text{UPBC}}$ is defined in Section 5.2.

[44] Note that although the hybrids are CSFs, the protocol has probabilistic termination.

Protocol π_{UPBC}

Protocol π_{UPBC}, parameterized 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 $\mathcal{F}_{\text{T-RBA}}$ is runs nm instances in parallel, and is parameterized 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 $\tau = \log^{1.5}(k) + 1$.
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 $\mathcal{F}_{\text{T-RBA}}$.
   (b) Send $(\mathtt{fetch\text{-}output}, \mathsf{sid}_1^\alpha)$ to $\mathcal{F}_{\text{T-RBA}}$ and receive $(\mathtt{output}, \mathsf{sid}_1^\alpha, \boldsymbol{v})$, where $\boldsymbol{v}$ is a vector of nm pairs $((v_1^1, v_2^1), \ldots, (v_1^{nm}, v_2^{nm}))$ with $v_1^j, v_2^j \in V \cup \{\perp\}$.
   (c) 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).
   (d) If $S_1^j \neq \emptyset$ 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 $\boldsymbol{c}_j = \emptyset$.
   (e) Send $(\mathtt{input}, \mathsf{sid}_2^\alpha)$ to $\mathcal{F}_{\text{LE}}$. Next, send $(\mathtt{fetch\text{-}output}, \mathsf{sid}_2^\alpha)$ to $\mathcal{F}_{\text{LE}}$ and receive $(\mathtt{output}, \mathsf{sid}_2^\alpha, k)$ with $k \in [n]$.
   (f) Send $(\mathtt{input}, \mathsf{sid}_3^\alpha, \boldsymbol{c}_k)$ to $\mathcal{F}_{\text{BA}}$, parameterized by the set $V^n \cup \{\emptyset\}$.
   Send $(\mathtt{fetch\text{-}output}, \mathsf{sid}_3^\alpha)$ to $\mathcal{F}_{\text{BA}}$ and receive $(\mathtt{output}, \mathsf{sid}_3^\alpha, \boldsymbol{c})$ (with $\boldsymbol{c} = (c_1, \ldots, c_n) \in V^n$ or $\boldsymbol{c} = \emptyset$).
   (g) 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 $b \leftarrow 0$.
   (h) Send $(\mathtt{input}, \mathsf{sid}_4^\alpha, b)$ to $\mathcal{F}_{\text{BA}}$, parameterized by the set $\{0, 1\}$.
   Send $(\mathtt{fetch\text{-}output}, \mathsf{sid}_4^\alpha)$ to $\mathcal{F}_{\text{BA}}$ and receive $(\mathtt{output}, \mathsf{sid}_4^\alpha, \beta)$ with $\beta \in \{0, 1\}$. 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, proceed as follows:
   (a) 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 $\mathcal{F}_{\text{UPBC}}$.
   (b) Send $(\mathtt{fetch\text{-}output}, \mathsf{sid})$ to $\mathcal{F}_{\text{UPBC}}$ and receive $(\mathtt{output}, \mathsf{sid}, \boldsymbol{c})$.
   Output $(\mathtt{output}, \mathsf{sid}, \boldsymbol{c})$ and halt.

Figure 12: 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

Lemma D.2. *Let $d \geq 5$ and $m = \log(n)$. 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 values $(r_1, \ldots, r_{nm})$ 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, 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 $\mathcal{F}_{\text{UPBC}}$.

Then, assuming all honest parties receive their inputs at the same round, 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 adaptive malicious adversaries corrupting at most $t < n/3$ of the parties.

Proof (sketch). 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 [5] 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 [5, Thm. 5], the Ben-Or and El-Yaniv 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 [5].
- 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 agree receive their output 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 $\mathcal{F}_{\text{PT-UPBC}}$. Let $\mathcal{A}$ be the dummy adversary. We construct a simulator $\mathcal{S}$ that simulates the honest parties in π_{UPBC}, the environment and the ideal functionalities $\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}, \mathcal{F}_{\text{LE}}, \mathcal{F}_{\text{T-PRBA}}$ and $\mathcal{F}_{\text{UPBC}}$ to $\mathcal{A}$, as follows.

- $\mathcal{S}$ 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 $(\texttt{leakage}, \mathsf{sid}, P_i, (x_1, \ldots, x_n))$. 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-terminating 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.
- Whenever $\mathcal{A}$ requests to corrupt some $P_i \in \mathcal{P}$, $\mathcal{S}$ 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 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 $k \in [n]$, $\mathcal{S}$ samples k such that in case $\mathcal{F}_{\text{T-RBA}}$ was succsessful (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 honset 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 $(\texttt{early-output}, \mathsf{sid}, P_i)$ for every $P_i \in \mathcal{P}$.

- 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 provide the output $(x_1, \ldots, x_n)$ to each party.

It follows using a standard hybrid argument that for every environment $\mathcal{Z}$ it holds that

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

□

Using Theorem 4.3 we obtain the following as a result.

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

Proof (sketch). Let $D_{\text{UPBC}}, D_{\text{PSMT}}, D_{\text{RBA}}, D_{\text{MV-BA}}$ be as in Lemma D.2, Theorem 5.2 and Theorem 5.3. Denote by $D_{\text{T-RBA}}$ the deterministic distribution that outputs a trace consisting of a constant number of leaves $\mathcal{F}_{\text{PSMT}}$ (corresponding to d phases of π_{RBA}). Denote by D_{LE} the distribution that outputs a trace consisting of an expected constant number of leaves $\mathcal{F}_{\text{PSMT}}$ (this follows from [22]). Denote by $D_{\text{DT-UPBC}}$ the deterministic distribution that outputs a trace consisting of $O(n)$ leaves $\mathcal{F}_{\text{PSMT}}$. Let $D'_{\text{UPBC}}, D'_{\text{PSMT}}, D'_{\text{RBA}}, D'_{\text{MV-BA}}, D'_{\text{T-RBA}}, D'_{\text{T-RBA}}, D'_{\text{LE}}, D'_{\text{DT-UPBC}}$ be the translated distributions, respectively.

Then, following Theorem 4.3 and Lemma D.2, the compiled protocol $\mathsf{Comp}^{c}_{\text{PT}}(\pi_{\text{UPBC}})$ UC-realizes $\mathcal{W}^{D'_{\text{UPBC}}}_{\text{PT}}(\mathcal{F}_{\text{UPBC}})$, in the $(\mathcal{W}^{D_{\text{PSMT}}}_{\text{DT}}(\mathcal{F}_{\text{PSMT}}), \mathcal{W}^{D'_{\text{RBA}}}_{\text{PT}}(\mathcal{F}^{\{0,1\}}_{\text{BA}}), \mathcal{W}^{D'_{\text{MV-BA}}}_{\text{PT}}(\mathcal{F}^{V}_{\text{BA}}), \mathcal{W}^{D'_{\text{T-RBA}}}_{\text{DT}}(\mathcal{F}_{\text{T-RBA}}), \mathcal{W}^{D'_{\text{DT-UPBC}}}_{\text{DT}}(\mathcal{F}_{\text{UPBC}}))$-hybrid model,

with perfect security, in the presence of adaptive malicious adversaries corrupting at most $t < n/3$ of the parties.

The proof follows since each of the functionalities $\mathcal{W}_{\text{PT}}^{D'_{\text{UPBC}}}(\mathcal{F}_{\text{UPBC}})$, in the $(\mathcal{W}_{\text{DT}}^{D_{\text{PSMT}}}(\mathcal{F}_{\text{PSMT}}), \mathcal{W}_{\text{PT}}^{D'_{\text{RBA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}}), \mathcal{W}_{\text{PT}}^{D'_{\text{MV-BA}}}(\mathcal{F}_{\text{BA}}^{V}), \mathcal{W}_{\text{DT}}^{D'_{\text{T-RBA}}}(\mathcal{F}_{\text{T-RBA}}), \mathcal{W}_{\text{DT}}^{D'_{\text{DT-UPBC}}}(\mathcal{F}_{\text{UPBC}}))$ can be UC-realized in the $\mathcal{F}_{\text{PSMT}}$-hybrid model. □