# Probabilistic Termination and Composability of Cryptographic Protocols*

Ran Cohen† Sandro Coretti‡ Juan Garay§¶ Vassilis Zikas‖¶

July 21, 2016

**Abstract**

When analyzing the round complexity of multi-party protocols, one often overlooks the fact that underlying resources, such as a broadcast channel, can by themselves be 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 the first composable protocols (with simulation-based security proofs) for the following primitives, relying on point-to-point channels: (1) expected-constant-round perfect Byzantine agreement, (2) expected-constant-round perfect parallel broadcast, and (3) perfectly secure MPC with round complexity independent of the number of parties.

---

*An extended abstract of this work appeared at *CRYPTO 2016*.

†Department of Computer Science, Bar-Ilan University. E-mail: cohenrb@cs.biu.ac.il. Work supported by a grant from the Israel Ministry of Science, Technology and Space (grant 3-10883) and by the National Cyber Bureau of Israel.

‡Department of Computer Science, ETH Zurich. E-mail: corettis@inf.ethz.ch. This author was supported in part by the Swiss NSF project no. 200020-132794.

§Yahoo Research. E-mail: garay@yahoo-inc.com.

¶Work done in part while the author was visiting the Simons Institute for the Theory of Computing, supported by the Simons Foundation and by the DIMACS/Simons Collaboration in Cryptography through NSF grant #CNS-1523467.

‖Department of Computer Science, RPI. E-mail: vzikas@cs.rpi.edu. This author was supported in part by the Swiss NSF Ambizione grant PZ00P2_142549.

# Contents

# 1 Introduction

In secure multi-party computation (MPC) [50, 28] $n$ 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) or 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 (aka the "ideal functionality") that captures the security properties the protocol is required to achieve. The trusted third party takes the parties' inputs and performs the computation on their behalf. A protocol is regarded as secure if for any adversary attacking it, there exists an ideal adversary (the simulator) attacking the execution in the ideal world, such that no external distinguisher (environment) can tell the real and the ideal executions apart.

There are several advantages in proving a protocol secure in this way. For starters, the definition of the functionality captures all security properties the protocol is supposed to have, and therefore its design process along with the security proof often exposes potential design flaws or issues that have been overlooked in the protocol design. A very important feature of many simulation-based security definitions is *composability*, which ensures that a protocol can be composed with other protocols without compromising its security. Intuitively, composability ensures that if a protocol $\pi^{\mathcal{G}}$ which uses a "hybrid" $\mathcal{G}$ (a broadcast channel, for example) securely realizes functionality $\mathcal{F}$, and protocol $\rho$ securely realizes the functionality $\mathcal{G}$, then the protocol $\pi^{\rho/\mathcal{G}}$, which results by replacing in $\pi$ calls to $\mathcal{G}$ by invocations of $\rho$, 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., [18, 20, 38, 16, 44]) 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, i.e., the number of rounds it takes for a protocol to deliver outputs, is an important efficiency metric for such protocols and, depending on the network parameters, can play a dominant role in the protocol's running time.

An issue often overlooked in the analysis of the round complexity of protocols is that the relation between a protocol's round complexity and its actual running time is sensitive to the "hybrids" (e.g., network primitives) that the protocol is assumed to have access to. For example, starting with the seminal MPC works [50, 28, 6, 15, 48], a common assumption is that the parties have access to a broadcast channel, which they invoke possibly in every round. In reality, however, such a broadcast channel might not be available and would have to be implemented by a broadcast protocol designed for a point-to-point network. Using a standard (deterministic) broadcast protocol for this purpose

incurs a linear blow-up (in $n$, the number of parties[1]) on the round complexity of the MPC protocol, as no deterministic broadcast protocol can tolerate a linear number of corruptions and terminate in a sublinear number of rounds [25, 22]. Thus, even though the round complexity of these protocols is usually considered to be linear in the multiplicative depth $d$ of the computed circuit, in reality their running time could become linear in $nd$ (which can be improved to $O(n+d)$ [35]) when executed over point-to-point channels.[2]

In fact, all so-called constant-round multi-party protocols (e.g., [39, 3, 17, 33, 1, 26, 31, 45]) 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 [25, 22] 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 outputs; 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 [24] (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 [47], which, effectively, showed how to circumvent the above limitation by using randomization. Most relevant to this submission, Rabin [47] showed that linearly resilient *Byzantine agreement* protocols [46, 41] (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 [24], 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 [24] works in the information-theoretic setting; these results were later extended to the computational setting by Katz and Koo [34], 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]) [21]. These issues make the simulation-based proof of these protocols a very delicate task, which is the motivation for the current work.

[1] More precisely, in the number of corruptions a protocol can tolerate, which is a constant fraction of $n$.

[2] Throughout this work we will consider protocols in which all parties receive their output. If one relaxes this requirement (i.e., allows that some parties may not receive their outputs and give up on fairness) then the techniques of Goldwasser and Lindell [30] 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 $r+c$.

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 security) [9] and the universal composition (UC) frameworks [10, 11]. 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., [11, 40, 37, 12]).

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 BA protocols as mentioned above [24, 34], 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 [34] even provided an expected-constant-round MPC protocol using their fast BA protocol as a subroutine, employing several techniques to ensure proper use of randomized BA. However, in lack of a formal treatment, existing constructions are usually proved secure in a property-based manner or rely on *ad hoc*, less studied security frameworks [43].[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. Ben-Or and El-Yaniv [5] observed that if one naïvely replaces each such invocation by 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 [24], the expected round complexity would be logarithmic in the number of instances (and therefore also 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 in expectation.

The difficulties arising with generic parallel composition are not the only issue with PT protocols. As observed by Lindell et al. [43], composing such protocols in sequence is also problematic. The main issue here is that, as already mentioned, PT protocols do not have simultaneous termination and therefore a party cannot be sure how long after he receives his output from a call to such a PT protocol he can safely carry on with the execution of the calling protocol. Although PT protocols usually guarantee a constant "slack" of rounds (say, $c$) in the output of any two honest parties, the naïve approach of using this property to synchronize the parties (i.e., wait $c$ rounds after the first call, $2c$ rounds after the second call, and so on) imposes an exponential blow-up on the round complexity of the calling protocol. To resolve this, [43] 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. Alternative approaches for solving this issue was also proposed in [8, 34] but, again, with a restricted (property-based) proof.

[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 [32].

Despite their novel aspects, the aforementioned results on composition of PT protocols do not use simulation-based security, and therefore it is unclear how (or if) they could be used to, for example, instantiate broadcast within a higher-level cryptographic protocol. In addition, they do not deal with other important features of modern security definitions, such as adaptive security and *strict* polynomial time execution. In fact, this lack of a formal cryptographic treatment places some of their claims at odds with the state-of-the-art cryptographic definitions. Somewhat pointedly, [5] claimed adaptive security, which, although it 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 [10, 11]. 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 $\pi$ making calls to deterministic-termination UC protocols $\rho_i$ in a "stand-alone fashion" (similar to [9], i.e., the protocols $\rho_i$ are invoked sequentially and in each round exactly one protocol is being executed by all the parties) into a (probabilistic-termination) protocol $\pi'$ (where the parties are no longer synchronized and the hybrids are invoked concurrently) in which each $\rho_i$ is replaced by a PT protocol $\rho'_i$. The compiled protocol $\pi'$ achieves the same security as $\pi$ and has (expected) round complexity proportional to $\sum_i d_i r_i$, where $d_i$ is the expected number of calls $\pi$ makes to $\rho_i$ and $r_i$ is the expected round complexity of $\rho_i$.

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 "stand-alone" protocol $\pi$, 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 $\pi'$ in which the calls to the SFEs are replaced by probabilistic-termination protocols realizing them. The important feature of our compiler is that in the compiled protocol, the parties do not need to wait for every party to terminate their emulation of each SFE to proceed to the emulation of the next SFE. Rather, shortly after a party (locally) receives its output from one emulation, it proceeds to the next one. This yields an (at most) multiplicative blow-up on the expected round complexity as discussed above. In particular, if the protocols used to emulate the SFEs are expected constant round, then the expected round complexity of $\pi'$ is the same (asymptotically) as that of $\pi$.

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 application in turn. Regarding Byzantine 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."[9]) Our protocol follows the structure of the protocol in [24], with a modification inspired by Goldreich and Petrank [29] 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 [13]. Indeed, similarly to how [13] converted (and proved) the "GMW" protocol [27] from statically secure in the stand-alone setting into an adaptively secure UC version, our work transforms the broadcast and BA protocols from [24] into adaptively UC-secure randomized broadcast and RBA protocols.[10]

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 [24] and describe simple(r) (SFE-like) functionalities corresponding to these sub-routines; this provides us with a simple "backbone" of the protocol in [24] making calls to these hybrids, which can be easily proved to implement expected-constant-round RBA. Next, we feed this simplified protocol to our compiler which outputs a protocol that implements RBA from point-to-point secure channels; our composition theorem ensures that the resulting protocol is also expected constant round.

There is a sticky issue here that we need to resolve for the above to work: the protocol in [24] does not have guaranteed termination and therefore the distribution of the terminating round is not sampleable by a strict probabilistic polynomial-time (PPT) machine.[11] A way around this issue would be to modify the UC model of execution so that the corresponding ITMs are *expected* PPT mach8ines. Such a modification, however, would impact the UC model of computation, and would therefore require a new proof of the composition theorem, a trickier task than one might expect, as the shift to expected polynomial-time simulation is known to introduce additional conceptual and technical difficulties (cf. [36]), 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 [24] using ideas from [29] so that it implements a functionality which samples the terminating round with almost the same probability distribution as in [24], 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, 34] 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.[12]

[9] 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.

[10] As we show, the protocol in [24] does not satisfy input independence, and therefore is not adaptively secure in a simulation-based manner (cf. [32]).

[11] 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.

[12] Although security against a "dynamic" adversary is also claimed in [5], the protocol does not implement the

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 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. [13] 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., [35]). 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.[13]

## 2 The 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.[14] 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. [37]. 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 [37], 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 $\delta$: 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.[15] 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 $\delta$'th `fetch-output` request following a submitted input $y$ from the sender results in the channel sending $(\texttt{output}, y)$ to $P_r$. In this work we take an alternative approach and model secure channels as special simple SFE functionalities. These SFEs also work in a fetch mode[16] and provide the same guarantee as the bounded-delay channels.

natural parallel broadcast functionality in the presence of an adaptive adversary (see Section 5).

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

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

[15] Following the simplifying approach of [37], we assume that communication channels are single use, thus each message transmission uses an independent instance of the channel (cf. Appendix B).

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

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 [37] this property is captured as follows: The functionality keeps track of which round the protocol 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 $\mu$ 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 $\mu$ 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 in an actual evaluation, as in the synchronous setting of [37], 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 $\rho$* 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[17]—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 [37] and discussion of our model.

## 3 Secure Computation with Probabilistic Termination

The work of Katz et al. [37] 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 [47] 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 $t = \Omega(n)$ corruptions exists [25, 22], Rabin's global-coin technique (fully realized later on in [24]) yields an expected constant round protocol. This speed-up, however, comes at a price, namely, of relinquishing both *fixed* and *simultaneous* termination [21]: 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.

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

### 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 potential leakage on the inputs to the adversary and potential adversarial influence on the outputs.[18] In more detail, $\mathcal{F}_{\text{CSF}}$ has two parameters: (1) a (possibly) randomized function $f$ that receives $n+1$ inputs ($n$ inputs from the parties and one additional input from the adversary) and (2) a leakage function $l$ that leaks some information about the input values to the adversary.

$\mathcal{F}_{\text{CSF}}$ 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, parametrized by a function $f\colon (\{0,1\}^* \cup \{\bot\})^{n+1} \to (\{0,1\}^*)^n$ and a leakage function $l\colon (\{0,1\}^* \cup \{\bot\})^n \to \{0,1\}^*$, and running with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ 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 a message $(\texttt{input}, \mathsf{sid}, v)$ 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. [13]), as its behavior depends on the identities of the corrupted parties.[19] 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. [37]).

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

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

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 `input` messages from all $n$ parties, we define the two-party task using an $n$-party function. The function to compute is $f^{i,j}_{\text{SMT}}(x_1,\ldots,x_n,a) = (\lambda,\ldots,x_i,\ldots,\lambda)$ (where $x_i$ is the value of the $j$'th coordinate) and the leakage function is $l^{i,j}_{\text{SMT}}(x_1,\ldots,x_n) = y$, where $y = |x_i|$ in case $P_j$ is honest and $y = x_i$ in case $P_j$ is corrupted. We denote by $\mathcal{F}^{i,j}_{\text{SMT}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions $f^{i,j}_{\text{SMT}}$ and $l^{i,j}_{\text{SMT}}$, for sender $P_i$ and receiver $P_j$.
- Broadcast. In the (standard) *broadcast* functionality, a sender $P_i$ with input $x_i$ distributes its input to all the parties, i.e., the function to compute is $f^{i}_{\text{BC}}(x_1,\ldots,x_n,a) = (x_i,\ldots,x_i)$. The adversary only learns the length of the message $x_i$ before its distribution, i.e., the leakage function is $l^{i}_{\text{BC}}(x_1,\ldots,x_n) = |x_i|$. This means that the adversary does not gain new information about the input of an honest sender before the output value for all the parties is determined, and in particular, the adversary *cannot* corrupt an honest sender and change its input *after* learning the input message. We denote by $\mathcal{F}^{i}_{\text{BC}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions $f^{i}_{\text{BC}}$ and $l^{i}_{\text{BC}}$, 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 $f^{g}_{\text{SFE}}(x_1,\ldots,x_n,a) = g(x_1,\ldots,x_n)$. The adversary learns the length of the input values via the leakage function, i.e., the leakage function is $l_{\text{SFE}}(x_1,\ldots,x_n) = (|x_1|,\ldots,|x_n|)$. We denote by $\mathcal{F}^{g}_{\text{SFE}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions $f^{g}_{\text{SFE}}$ and $l_{\text{SFE}}$, for computing the $n$-party function $g$.
- Byzantine Agreement (aka Consensus). In the *Byzantine agreement* functionality, defined for the set $V$, each party $P_i$ has input $x_i \in V$. The common output is computed such that if $n-t$ of the input values are the same, this will be the output; otherwise the adversary gets to decide on the output. The adversary is allowed to learn the content of each input value from the leakage (and so it can corrupt parties and change their inputs based on this information). The function to compute is $f_{\text{BA}}(x_1,\ldots,x_n,a) = (y,\ldots,y)$ such that $y = x$ if there exists a value $x$ such that $x = x_i$ for at least $n-t$ input values $x_i$; otherwise $y = a$. The leakage function is $l_{\text{BA}}(x_1,\ldots,x_n) = (x_1,\ldots,x_n)$. We denote by $\mathcal{F}^{V}_{\text{BA}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions $f_{\text{BA}}$ and $l_{\text{BA}}$, 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 parametrized by an efficient probabilistic algorithm $D$, termed the *round sampler*, that may depend on a specific protocol implementing the functionality. The round sampler $D$ samples a round number $\rho_{\mathsf{term}}$ by which all parties are guaranteed to receive their outputs no matter what the adversary strategy is. Moreover, since there are protocols in which all parties terminate in the same round and protocols in which they do not, we consider two wrappers: the first, denoted $\mathcal{W}_{\text{strict}}$, ensures in a strict manner that all (honest) parties terminate in the same round, whereas the second, denoted $\mathcal{W}_{\text{flex}}$, is more flexible and allows the adversary to deliver outputs to individual

parties at any time before round $\rho_{\mathsf{term}}$.

A delicate issue that needs to be addressed is the following: While an ideal functionality can be used to abstractly describe a protocol's task, it cannot hide the protocol's round complexity. This phenomenon is inherent in the synchronous communication model: any environment can observe how many rounds the execution of a protocol takes, and, therefore, the execution of the corresponding ideal functionality must take the same number of rounds.[20]

As an illustration of this issue, let $\mathcal{F}$ be an arbitrary functionality realized by some protocol $\pi$. If $\mathcal{F}$ is to provide guaranteed termination (whether probabilistic or not), it must enforce an upper bound on the number of rounds that elapse until all parties receive their outputs. If the termination round of $\pi$ 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 $\pi$.

Thus, in order to simulate correctly, the functionality $\mathcal{F}$ and $\pi$'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 $\rho_{\mathsf{term}}$. Intuitively, the reason is that protocol $\pi$ 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 $\pi$ and the protocols realizing the hybrids called by $\pi$. As $\mathcal{S}$ needs to emulate the entire protocol execution, it is now left with the task of trying to sample the protocol's choices conditioned on the upper bound it receives from $\mathcal{F}$. In general, however, it is unclear whether such a reverse sampling can be performed in (strict) polynomial time.

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

**Execution traces.** As mentioned above, in the synchronous communication model, the execution of the ideal functionality must take the same number of rounds as the protocol.For example, suppose that the functionality $\mathcal{F}$ in our illustration above is used as a hybrid by a higher-level protocol $\pi'$. The functionality $\mathcal{G}$ realized by $\pi'$ must, similarly to $\mathcal{F}$, choose an upper bound on the number of rounds that elapse before parties obtain their outputs. However, this upper bound now not only depends on $\pi'$ itself but also on $\pi$ (in particular, when $\pi$ 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.[21]

Building on our running illustration above, suppose protocol $\pi'$ (realizing $\mathcal{G}$) makes ideal hybrid calls to $\mathcal{F}$ and to some atomic functionality $\mathcal{H}$. Assume that in an example execution, $\pi'$ 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 $\pi$ (realizing $\mathcal{F}$) and that $\pi$ happens to make two (sequential) calls to $\mathcal{H}$ upon the first invocation by $\pi'$, and three (sequential) calls to $\mathcal{H}$

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

[21] In this work, atomic functionalities are always $\mathcal{F}_{\text{PSMT}}$ CSFs.

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

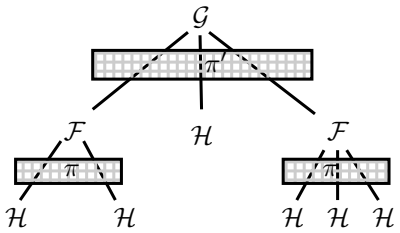


Figure 2: Example of an execution trace

Assume that $\pi$ is a probabilistic-termination protocol and $\pi'$ 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 $\pi$) depth-1 traces with root $\mathcal{W}_{\text{flex}}^{D_{\mathcal{F}}}(\mathcal{F}')$ and leaves $\mathcal{H}$.[22] 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 $\pi'$) 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.

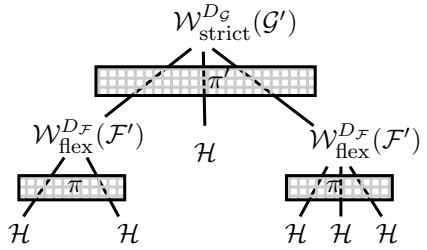


Figure 3: An execution trace with probabilistic-termination and deterministic-termination protocols

Formally, a trace is defined as follows:

**Definition 3.1** (traces)**.** *A* trace *is a rooted tree of depth at least* 1*, in which all nodes are labeled by functionalities and where every node's children are* ordered*. The root and all internal nodes are labeled by wrapped CSFs (by either of the two wrappers), and the leaves are labeled by unwrapped CSFs. The* trace complexity *of a trace $T$, denoted $c_{\mathsf{tr}}(T)$, is the number of leaves in $T$. Moreover, denote by $\mathsf{flex}_{\mathsf{tr}}(T)$ the number of flexibly wrapped CSFs in $T$.*

**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.

[22] 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.

**Strict wrapper functionality.** We now proceed to give the formal descriptions of the wrappers. The *strict wrapper functionality*, defined in Figure 4, is parametrized by (a sampler that induces) a distribution $D$ over traces, and internally runs a copy of a CSF functionality $\mathcal{F}$. Initially, a trace $T$ is sampled from $D$; this trace is given to the adversary once the first honest party provides its input. The trace $T$ is used by the wrapper to define the termination round $\rho_{\mathsf{term}} \leftarrow c_{\mathsf{tr}}(T)$. In the first round, the wrapper forwards all the messages from the parties and the adversary to (and from) the functionality $\mathcal{F}$. Next, the wrapper essentially waits until round $\rho_{\mathsf{term}}$, with the exception that the adversary is allowed to send $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages and change its input to the function computed by the CSF. Finally, when round $\rho_{\mathsf{term}}$ arrives, the wrapper provides the output generated by $\mathcal{F}$ to all parties.

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

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

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

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

Figure 4: The strict-wrapper functionality

**Flexible-wrapper functionality.** The *flexible-wrapper functionality*, 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 round. In order to accomplish this, the wrapper assigns a termination indicator $\mathsf{term}_i$, initially set to 0, to each party. Once the wrapper receives an `early-output` request from the adversary for $P_i$, it sets $\mathsf{term}_i \leftarrow 1$. Now, when a party $P_i$ sends a `fetch-output` request, the wrapper checks if $\mathsf{term}_i = 1$, and lets the party receive its output in this case (by forwarding the `fetch-output` request to $\mathcal{F}$). When the guaranteed-termination round $\rho_{\mathsf{term}}$ arrives, the wrapper provides the output to all parties that didn't receive it yet.

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

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

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

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

Figure 5: The flexible-wrapper functionality

# 4 (Fast) Composition of Probabilistic-Termination Protocols

Canonical synchronous functionalities that are wrapped using the flexible wrapper (cf. Section 3.2), i.e., functionalities that correspond to protocols with non-simultaneous termination, are cumbersome to be used as hybrid functionalities for protocols. The reason is that the adversary can cause parties to finish in different rounds, and, as a result, after the execution of the first such functionality, the parties might be *out of sync*.

This "slack" can be reduced, however, only to a difference of one round, unless one is willing to pay a linear blow-up in round complexity [25, 22]. Hence, all protocols must be modified to deal with a non-simultaneous start of (at least) one round, and protocols that introduce slack must be followed by a slack-reduction procedure. Since this is a tedious, yet systematic task, in this section we provide a generic compiler that transforms protocols designed in a simpler "stand-alone" setting, where all parties remain synchronized throughout the protocol (and no slack and round-complexity issues arise) into UC protocols that deal with these issues while maintaining their security.

Out starting point are protocols that are defined in the "stand-alone" setting. In such protocols all the hybrids are CSFs and are called in a strictly sequential manner.

**Definition 4.1** (SNF)**.** *Let $\mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities. A synchronous protocol $\pi$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model is in* synchronous normal form (SNF) *if in every round exactly one ideal functionality $\mathcal{F}_i$ is invoked by all honest parties, and in addition, no honest party hands inputs to other CSFs before this instance halts.*

Clearly, designing and proving the security of SNF protocols, which only make calls to simple two-round CSFs is a much simpler task than dealing with protocols that invoke more complicated hybrids, potentially with probabilistic termination (see Section 5 for concrete examples).

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

In Section 4.1 we apply this approach to deterministic-termination protocols that use deterministic-termination hybrids, and in Section 4.2 generalize it to the probabilistic-termination setting. Section 4.3 covers the base case of realizing the wrapped $\mathcal{F}_{\text{PSMT}}$ using only $\mathcal{F}_{\text{SMT}}$ functionalities. All proofs can be found in Appendix C.

## 4.1 Composition with Deterministic Termination

We start by defining a slack-tolerant variant of the strict wrapper (cf. Section 3.2), which can be used even when parties operate with a (known) slack. Then, we show how to compile an SNF protocol $\pi$ realizing a strictly wrapped CSF $\mathcal{F}$ into a (non-SNF) protocol $\pi'$ realizing a version of $\mathcal{F}$ wrapped with the slack-tolerant strict wrapper and making calls to wrapped hybrids.

**Slack-tolerant strict wrapper.** The *slack-tolerant strict wrapper* $\mathcal{W}^{D,c}_{\text{sl-strict}}$, formally defined in Figure 6, is parametrized by an integer $c \geq 0$, which denotes the amount of slack tolerance that is added, and a distribution $D$ over traces. The wrapper $\mathcal{W}_{\text{sl-strict}}$ is similar to $\mathcal{W}_{\text{strict}}$ but allows parties to provide input within a window of $2c+1$ rounds and ensures that they obtain output with the same slack they started with. The wrapper essentially increases the termination round by a factor of $B_c = 3c+1$, which is due to the slack-tolerance technique used to implement the wrapped version of the atomic parallel SMT functionality (cf. Section 4.3).

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

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

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

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

Figure 6: The slack-tolerant strict wrapper functionality

**Deterministic-termination compiler.** Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, and let $\pi$ an SNF protocol that UC-realizes the strictly wrapped functionality $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$, for some depth-1 distribution $D$, in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, assuming that all honest parties receive their inputs at the same round. We define a compiler $\mathsf{Comp}^{c}_{\text{DT}}$, parametrized with a slack parameter $c \geq 0$, that receives as input the protocol $\pi$ and distributions $D_1, \ldots, D_m$ over traces and replaces every call to a CSF $\mathcal{F}_i$ with a call to the wrapped CSF $\mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$. We denote the output of the compiler by $\pi' = \mathsf{Comp}^{c}_{\text{DT}}(\pi, D_1, \ldots, D_m)$.[23]

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

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

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

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

**Theorem 4.2.** *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, and let $\pi$ an SNF protocol that UC-realizes $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, for some depth-1 distribution $D$, assuming that all honest parties receive their inputs at the same round. Let $D_1, \ldots, D_m$ be arbitrary distributions over traces, $D^{\mathsf{full}} = \mathsf{full\text{-}trace}(D, D_1, \ldots, D_m)$, and $c \geq 0$. Then, protocol $\pi' = \mathsf{Comp}^{c}_{\text{DT}}(\pi, D_1, \ldots, D_m)$ UC-realizes $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$ in the $(\mathcal{W}^{D_1,c}_{\text{sl-strict}}(\mathcal{F}_1), \ldots, \mathcal{W}^{D_m,c}_{\text{sl-strict}}(\mathcal{F}_m))$-hybrid model, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

*The expected round complexity of the compiled protocol $\pi'$ is*

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

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

The proof of Theorem 4.2 can be found in Appendix C.1.

## 4.2 Composition with Probabilistic Termination

The composition theorem in Section 4.1 does not work if the protocol $\pi$ itself introduces slack (e.g., the fast broadcast protocol by Feldman and Micali [24]) or if one of the hybrids needs to be replaced by a slack-introducing protocol (e.g., instantiating the broadcast hybrids with fast broadcast protocols in BGW [6]).

[23] The distributions $D_i$ depend on the protocols realizing the strictly wrapped functionalities $\mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$. Note, however, that the composition theorems in Sections 4.1 and 4.2 actually work for arbitrary distributions $D_i$.

As in Section 4.1, we start by adjusting the flexible wrapper (cf. Section 3.2) to be slack-tolerant. In addition, the slack-tolerant flexible wrapper ensures that all parties will obtain their outputs within two consecutive rounds. Then, we show how to compile an SNF protocol $\pi$ realizing a CSF $\mathcal{F}$, wrapped with the flexible wrapper, into a (non-SNF) protocol $\pi'$ realizing a version of $\mathcal{F}$ wrapped with slack-tolerant flexible wrapper. The case where $\pi$ implements a strictly wrapped CSF, but some of the hybrids are wrapped with the slack-tolerant flexible wrapper follows along similar lines.

**Slack-tolerant flexible wrapper.** The *slack-tolerant flexible wrapper* $\mathcal{W}_{\text{sl-flex}}^{D,c}$, formally defined in Figure 7, is parametrized by an integer $c \geq 0$, which denotes the amount of slack tolerance that is added, and a distribution $D$ over traces. The wrapper $\mathcal{W}_{\text{sl-flex}}$ is similar to $\mathcal{W}_{\text{flex}}$ but allows parties to provide input within a window of $2c+1$ rounds and ensures that all honest parties will receive their output within two consecutive rounds. The wrapper essentially increases the termination round to

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

where the blow-up factor $B_c$ is as explained in Section 4.1, and the additional factor of 2 results from the termination protocol described below for every flexibly wrapped CSF, which increases the round complexity by at most two additional rounds (recall that $\mathsf{flex}_{\mathsf{tr}}(T)$ denotes the number of such CSFs), and $c$ is due to the potential slack. $\mathcal{W}_{\text{sl-flex}}$ allows the adversary to deliver output at any round prior to $\rho_{\mathsf{term}}$ but ensures that all parties obtain output with a slack of at most one round. Moreover, it allows the adversary to obtain the output using the $(\texttt{get-output}, \mathsf{sid})$ command, which is necessary in order to simulate the termination protocol.

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

In addition, the compiler adds the following termination procedure, based on an approach originally suggested by Bracha [7], which ensures all honest parties will terminate within two consecutive rounds:

- As soon as a party is ready to output a value $y$ (according to the prescribed protocol) or upon receiving at least $t+1$ messages $(\mathsf{end}, \mathsf{sid}, y)$ for the same value $y$ (whichever happens first), it sends $(\mathsf{end}, \mathsf{sid}, y)$ to all parties.
- Upon receiving $n - t$ messages $(\mathsf{end}, \mathsf{sid}, y)$ for the same value $y$, a party outputs $y$ as the result of the computation and halts.

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

The following theorem states that the compiled protocol $\pi'$ UC-realizes the wrapped functionality $\mathcal{W}_{\text{sl-flex}}^{D^{\mathsf{full}},c}(\mathcal{F})$, for the adapted distribution $D^{\mathsf{full}} = \mathsf{full\text{-}trace}(D, D_1, \ldots, D_m)$. Consequently, the compiled protocol $\pi'$ can handle a slack of up to $c$ rounds, while using hybrids that are realizable

themselves, and ensuring that the output slack is at most one round (as opposed to $\pi$). Calling the wrapped hybrids instead of the CSFs affects the trace corresponding to $\mathcal{F}$ in exactly the same way as in the case with deterministic termination (cf. Section 4.1).[24]

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

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

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

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

Figure 7: The slack-tolerant flexible wrapper functionality

**Theorem 4.3.** *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, and let $\pi$ an SNF protocol that UC-realizes $\mathcal{W}^{D}_{\text{flex}}(\mathcal{F})$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, for some depth-1 distribution $D$, assuming that all honest parties receive their inputs at the same round. Let $I \subseteq [m]$ be the subset (of indices) of functionalities to be wrapped using the flexible wrapper, let $D_1, \ldots, D_m$ be arbitrary distributions over traces, denote $D^{\mathsf{full}} = (D, D_1, \ldots, D_m)$ and let $c \geq 0$. Assume that $\mathcal{F}$ and $\mathcal{F}_i$, for every $i \in I$, are public-output functionalities.*

*Then, the compiled protocol $\pi' = \mathsf{Comp}^{c}_{\text{PTR}}(\pi, D_1, \ldots, D_m, I)$ UC-realizes $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$ in the $(\mathcal{W}(\mathcal{F}_1), \ldots, \mathcal{W}(\mathcal{F}_m))$-hybrid model, where $\mathcal{W}(\mathcal{F}_i) = \mathcal{W}^{D_i,c}_{\text{sl-flex}}(\mathcal{F}_i)$ if $i \in I$ and $\mathcal{W}(\mathcal{F}_i) = \mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$ if $i \notin I$, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

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

*The expected round complexity of the compiled protocol $\pi'$ is*

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

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

The proof of Theorem 4.3 can be found in Appendix C.2.

Consider now the scenario where an SNF protocol $\pi$ realizes a strictly wrapped functionality, yet some of the CSF hybrids are to be wrapped by flexible wrappers. The corresponding compiler $\mathsf{Comp}_{\text{PT}}$ works as $\mathsf{Comp}_{\text{PTR}}$ with the exception that the slack-reduction protocol is not performed at the end. The proof of the following theorem follows that of Theorem 4.3.

**Theorem 4.4.** *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, and let $\pi$ an SNF protocol that UC-realizes $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, for some depth-1 distribution $D$, assuming that all honest parties receive their inputs at the same round. Let $I \subseteq [m]$ be the subset (of indices) of functionalities to be wrapped using the flexible wrapper, let $D_1, \ldots, D_m$ be arbitrary distributions over traces, denote $D^{\mathsf{full}} = \mathsf{full\text{-}trace}(D, D_1, \ldots, D_m)$ and let $c \geq 0$. Assume that $\mathcal{F}$ and $\mathcal{F}_i$. for every $i \in I$. is a public-output functionalities.*

*Then, the compiled protocol $\pi' = \mathsf{Comp}^{c}_{\text{PT}}(\pi, D_1, \ldots, D_m, I)$ UC-realizes $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$ in the $(\mathcal{W}(\mathcal{F}_1), \ldots, \mathcal{W}(\mathcal{F}_m))$-hybrid model, where $\mathcal{W}(\mathcal{F}_i) = \mathcal{W}^{D_i,c}_{\text{sl-flex}}(\mathcal{F}_i)$ if $i \in I$ and $\mathcal{W}(\mathcal{F}_i) = \mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$ if $i \notin I$, assuming that all honest parties receive their inputs within $c + 1$ consecutive rounds.*

*The expected round complexity of the compiled protocol $\pi'$ is*

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

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

### 4.3 Wrapping Secure Channels

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

**Parallel SMT.** The *parallel secure message transmission functionality* $\mathcal{F}_{\text{PSMT}}$ is a CSF for the following functions $f_{\text{PSMT}}$ and $l_{\text{PSMT}}$ Each party $P_i$ has a vector of input values $(x^i_1, \ldots, x^i_n)$ such that $x^i_j$ is sent from $P_i$ to $P_j$. That is, the function to compute is $f_{\text{PSMT}}((x^1_1, \ldots, x^1_n), \ldots, (x^1_1, \ldots, x^n_n), a) = ((x^1_1, \ldots, x^n_1), \ldots, (x^1_n, \ldots, x^n_n))$. 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 $l_{\text{PSMT}}((x^1_1, \ldots, x^1_n), \ldots, (x^n_1, \ldots, x^n_n)) = (y^1_1, y^1_2, \ldots, y^n_{n-1}, y^n_n)$, where $y^i_j = |x^i_j|$ in case $P_j$ is honest and $y^i_j = x^i_j$ in case $P_j$ is corrupted.

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

A standard solution to overcome asynchrony by a constant number of rounds $c \geq 0$, introduced by Lindell et al. [42] and used by Katz and Koo [34], is to expand each communication round to $2c+1$ rounds. Each party listens for messages throughout all $2c+1$ rounds, and sends its own messages in round $c+1$. It is straight-forward to verify that if the slack is $c$, i.e., the parties start within $c+1$ rounds from each other, round $r$-messages (in the original protocol, without round expansion) are sent, and delivered, before round $(r+1)$-messages and after round $(r-1)$-messages.

The solution described above does not immediately apply to our case, due to the nature of canonical synchronous functionalities. Recall that in a CSF the adversary can send an `adv-input` message (and affect the output) only before any honest party has received an output from the functionality. If only $2c+1$ rounds are used a subtle problem arises: Assume for simplicity that $c = 1$ and say that $P_1$ is a fast party and $P_2$ is a slow party. Initially, $P_1$ listens for one round. In the second round $P_2$ listens and $P_1$ send its messages to all the parties. In the third round $P_2$ sends its messages and $P_1$ receives its message, produces output and completes the round. Now, $P_2$ listens for an additional round, and the adversary can send it messages on behalf of corrupted parties. In other words, the adversary can choose the value for $P_2$'s output *after* $P_1$ has received its output – such a phenomena cannot be modeled using CSFs. For this reason we add an additional round where each party is idle; if $P_1$ waits one more round (without listening) before it produces its output, then $P_2$ will receive all the messages that determine its output, and so once $P_1$ produces output and completes, the adversary cannot affect the output of $P_2$.

As a result, in the protocol presented in Figure 8, each round is expanded to $3c+1$ rounds, where during the final $c$ rounds, parties are simply idle and ignore any messages they receive.

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

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

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

[a] Note that $\rho$ is the *local* round counter of party $P_i$.

Figure 8: The wrapped parallel SMT protocol, in the $\mathcal{F}_{\text{PSMT}}$-hybrid model

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

**Lemma 4.5.** *Let $c \geq 0$. Protocol $\pi_{\text{PSMT}}$ UC-realizes $\mathcal{W}_{\text{sl-strict}}^{D_{\text{PSMT}},c}(\mathcal{F}_{\text{PSMT}})$ in the $\mathcal{F}_{\text{PSMT}}$-hybrid model, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

*Proof.* For simplicity, denote by $\mathcal{W}(\mathcal{F}_{\text{PSMT}})$ the wrapped functionality $\mathcal{W}_{\text{sl-strict}}^{D_{\text{PSMT}},c}(\mathcal{F}_{\text{PSMT}})$. Let $\mathcal{Z}$ be an environment. We construct the following simulator $\mathcal{S}$ running with $\mathcal{W}(\mathcal{F}_{\text{PSMT}})$ and $\mathcal{Z}$, simulating the dummy adversary.[25] Initially, $\mathcal{S}$ receives the message $(\texttt{trace}, \mathsf{sid}, T)$, where $T$ is a depth-1 trace consisting of a single leaf $\mathcal{F}_{\text{PSMT}}$. Next, $\mathcal{S}$ simulates $3c+1$ sequential instances of $\mathcal{F}_{\text{PSMT}}$, by interacting with $\mathcal{Z}$ and $\mathcal{W}(\mathcal{F}_{\text{PSMT}})$. In every instances of $\mathcal{F}_{\text{PSMT}}$, $\mathcal{S}$ proceeds as follows:

- If $\mathcal{S}$ receives an input message $(\texttt{input}, \mathsf{sid}, x_i)$ from $\mathcal{Z}$, where $x_i \neq \perp$ is a vector of messages to be sent from a corrupted $P_i$, $\mathcal{S}$ forwards the message to $\mathcal{W}(\mathcal{F}_{\text{PSMT}})$.
- If $\mathcal{S}$ receives a leakage message $(\texttt{leakage}, \mathsf{sid}, P_i, l_i)$ from $\mathcal{W}(\mathcal{F}_{\text{PSMT}})$, where $l_i$ is a length-$n$ vector, consisting of the messages sent by $P_i$ to each corrupted party (and the length of messages $P_i$ sends to honest parties), $\mathcal{S}$ forwards the message to $\mathcal{Z}$. If no leakage message arrived during this round, $\mathcal{S}$ sends the message $(\texttt{leakage}, \mathsf{sid}, P_i, \perp)$ to $\mathcal{Z}$, on behalf of every party.
- In the "output" round of this $\mathcal{F}_{\text{PSMT}}$ instance, $\mathcal{S}$ sends $(\texttt{output}, \mathsf{sid}, y_i)$ to $\mathcal{Z}$ for every corrupted $P_i$, where $y_i$ is a vector consisting of the messages sent to $P_i$ in the "input" round (if some party did not send a mesasge to $P_i$ the value $\lambda$ is used).

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

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

**Corollary 4.6.** *Let $c \geq 0$. The functionality $\mathcal{W}_{\text{sl-strict}}^{D_{\text{PSMT}},c}(\mathcal{F}_{\text{PSMT}})$ can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

# 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.

[25] Recall that proving security with respect to the dummy adversary is sufficient (cf. [10, Claim 10]).

### 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 [24] and Turpin and Coan [49], 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 [24]. 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 [24]) every honest party $P_i$ outputs a bit $y_i \in \{0,1\}$ such that the following holds: with probability $p > 0$ all honest parties will agree on a uniformly distributed bit, and with probability $1-p$ the output for each honest party is determined by the adversary. The meaning of *obliviousness* here is that the parties are unaware of whether agreement on the coin is achieved or not.

In more detail, each honest party $P_i$ sends an empty string $x_i = \lambda$ as input, and the leakage function is $l_{\text{oc}}(x_1, \ldots, x_n) = \perp$. The function to compute, $f_{\text{oc}}(x_1, \ldots, x_n, a) = (y_1, \ldots, y_n)$, is parametrized by an efficiently sampleable distribution $D$ over $\{0,1\}$, that outputs 1 with probability $p$ and 0 with probability $1-p$, and works as follows:

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

We denote by $\mathcal{F}_{\text{OC}}$ the CSF functionality parametrized with the above functions $f_{\text{oc}}$ and $l_{\text{oc}}$. Feldman and Micali [24, Thm. 3] 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 $D_{\text{RBA}}$ is formally defined in Lemma 5.1), denoted $\mathcal{F}_{\text{RBA}}$ for short. The protocol $\pi_{\text{RBA}}$, described in Figure 9, is based on the protocol from [24] modified using the "best-of-both-worlds" technique due to Goldreich and Petrank [29]. Recall that following Section 4, it is sufficient to describe the protocol using CSFs as hybrids rather than wrapped CSFs (even though such a description might be overly ideal, and cannot be instantiated in the real world), and the same level of security is automatically achieved in a compiled protocol (that can be instantiated) where the underlying CSFs are properly wrapped. Therefore, the protocol is defined in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{OC}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model.

At first sight, it may seem odd that the binary Byzantine agreement functionality $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ is used in order to implement the randomized binary Byzantine agreement functionality $\mathcal{F}_{\text{RBA}}$. However, the functionality $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ will only be invoked in the event (which occurs with a negligible probability) that the protocol does not terminate within a poly-log number of rounds. Once the protocol is compiled, the CSF functionality $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ will be wrapped using a strict wrapper, such that the wrapped functionality $\mathcal{W}_{\text{strict}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$ can be instantiated using any linear-round deterministic Byzantine agreement protocol (e.g., the protocol in [32]).

At a high level, protocol $\pi_{\text{RBA}}$ 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.

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

**Protocol $\pi_{\text{RBA}}$**

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

Figure 9: The binary randomized Byzantine agreement protocol, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{OC}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model

Denote by $D_{\text{RBA}}$ 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 $p = 0.35$ and support $\{1 \ldots, \tau+1\}$ (representing the phase where $\mathcal{F}_{\text{OC}}$ samples a fairness bit 1, plus the option that $\mathcal{F}_{\text{OC}}$ samples 0 in all $\tau$ 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. In Appendix D.1 we prove the following lemma.

**Lemma 5.1.** *Let $t < n/3$, then, assuming all honest parties receive their inputs at the same round, protocol $\pi_{\text{RBA}}$ 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 $\mathcal{W}_{\text{sl-flex}}^{D,c}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious $t$-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

*Proof (sketch).* Denote by $D_{\text{OC}}$ the deterministic distribution that outputs a depth-1 trace consisting of a root $\mathcal{W}_{\text{strict}}^{D_{\text{OC}}}(\mathcal{F}_{\text{OC}})$ and 36 leaves $\mathcal{F}_{\text{PSMT}}$ (recall that the $\mathcal{F}_{\text{OC}}$ protocol in [24, Claim T4-4] requires 36 rounds), and denote by $D_{\text{BA}}$ the deterministic distribution that outputs a depth-1 trace consisting of a root $\mathcal{W}_{\text{strict}}^{D_{\text{BA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$ and $O(n)$ leaves $\mathcal{F}_{\text{PSMT}}$. Let $D_{\text{RBA}}^{\mathsf{full}} = \mathsf{full\text{-}trace}(D_{\text{RBA}}, D_{\text{OC}}, D_{\text{PSMT}}, D_{\text{BA}})$.

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

From Lemma 5.1, $\pi_{\text{RBA}}$ UC-realizes $\mathcal{W}_{\text{flex}}^{D_{\text{RBA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{OC}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model, in an expected constant number of rounds, assuming all parties receive their inputs at the same round. Following Theorem 4.3, the compiled protocol $\mathsf{Comp}_{\text{PTR}}^{c}(\pi_{\text{RBA}}, D_1, D_2, D_3, I)$ UC-realizes $\mathcal{F}_{\text{BA}}^{\text{PT}}$, in the $(\mathcal{F}_{\text{PSMT}}^{\text{DT}}, \mathcal{F}_{\text{OC}}^{\text{DT}}, \mathcal{F}_{\text{BA}}^{\text{DT}})$-hybrid model, in an expected constant number of rounds, assuming all parties receive their inputs within $c+1$ consecutive rounds.

The proof follows since each of the functionalities $(\mathcal{F}_{\text{PSMT}}^{\text{DT}}, \mathcal{F}_{\text{OC}}^{\text{DT}}, \mathcal{F}_{\text{BA}}^{\text{DT}})$ can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model. This follows from Lemma 4.5, combined with the protocols from [24] and [32]. □

**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 [49] 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 $\mathcal{W}_{\text{sl-flex}}^{D,c}(\mathcal{F}_{\text{BA}}^{V})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious $t$-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

## 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. Ben-Or and El-Yaniv [5] constructed an elegant protocol for probabilistic-termination parallel broadcast[26] with a constant round complexity in

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

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. Hirt and Zikas [32] 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 [32], 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 parametrized with the above functions $f_{\text{UPBC}}$ and $l_{\text{UPBC}}$.

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

**Theorem 5.4.** *Let $c \geq 0$ and $t < n/3$. There exists an efficiently sampleable distribution $D$ such that the functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}_{\text{UPBC}})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious $t$-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

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

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

Unfortunately, the unfair parallel broadcast protocol $\pi_{\text{UPBC}}$ (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 [32]: 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 $\pi_{\text{UPBC}}$ 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

[27] In [32] 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 $\pi_{\text{PBC}}$, in Figure 10.

**Protocol $\pi_{\text{PBC}}$**

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 $(\mathsf{sid}, x_i^j)$ (via $\mathcal{F}_{\text{PSMT}}$). Denote by $x_j^i$ the value received from $P_j$.
2. In the second round, $P_i$ broadcasts the values $\boldsymbol{x}_i = (x_1^i, \ldots, x_n^i)$ using the unfair parallel broadcast functionality, i.e., $P_i$ sends $(\texttt{input}, \mathsf{sid}, \boldsymbol{x}_i)$ to $\mathcal{F}_{\text{UPBC}}$. Denote by $\boldsymbol{y}_j = (y_1^j, \ldots, y_n^j)$ the value received from $P_j$. 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 10: The parallel broadcast protocol, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{UPBC}})$-hybrid model

**Theorem 5.5.** *Let $c \geq 0$ and $t < n/3$. There exists an efficiently sampleable distribution $D$ such that the functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}_{\text{PBC}})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious $t$-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

*Proof (sketch).* The simulator uses the adversary attacking $\pi_{\text{PBC}}$ 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 outputs and can easily complete the sharing to match this output. At the end of this phase, the simulator interacts with its hybrid until it produces output. Once this is the case, he uses this output to continue the simulation with its adversary. Clearly, for any sender $P_i$ who is not corrupted until he distributes his shares, then a $t$-adversary has no way of modifying the reconstructed output of $P_i$'s input, since he can at most affect $t < n/3$ shares. 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 $\pi_{\text{PBC}}$ executes only one round (namely the sharing round) more than the unfair protocol which is expected constant round (cf. Theorem 5.4). □

### 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 $\pi_{\text{BGW}}$. 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 4.4,

[28] A full simulation proof of the protocol with a black-box straight-line simulation was recently given by [2] and [19].

that the compiled protocol $\mathsf{Comp}^{c}_{\mathrm{PT}}(\pi_{\mathrm{BGW}}, D_1, D_2, I)$, for $D_1 = D_{\mathrm{PSMT}}$, $D_2 = D^{\mathsf{full}}_{\mathrm{PBC}}$ and $I = \{2\}$, UC-realizes the corresponding wrapped functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}_{\mathrm{SFE}})$ (for an appropriate distribution $D$), in the $(\mathcal{W}^{D_{\mathrm{PSMT}},c}_{\text{sl-strict}}(\mathcal{F}_{\mathrm{PSMT}}), \mathcal{W}^{D^{\mathsf{full}}_{\mathrm{PBC}},c}_{\text{sl-flex}}(\mathcal{F}_{\mathrm{PBC}}))$-hybrid model, resulting in the following.

**Theorem 5.6.** *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 $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}^{f}_{\mathrm{SFE}})$ has round complexity $O(d)$ in expectation, and can be UC-realized in the $\mathcal{F}_{\mathrm{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious $t$-adversary, assuming that all honest parties receive their inputs within $c + 1$ consecutive rounds.*

# 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 2012*, 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 *22nd ACM STOC*, pages 503–513. ACM Press, May 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, *2nd ACM PODC*, pages 27–30. ACM Press, August 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] Gabriel Bracha. An asynchronou [(n-1)/3]-resilient consensus protocol. In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *3rd ACM PODC*, pages 154–162. ACM Press, August 1984.

[8] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 524–541. Springer, August 2001.

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

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

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

[12] 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, *CRYPTO 2015, Part II*, volume 9216 of *Lecture Notes in Computer Science*, pages 3–22. Springer, 2015.

[13] 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.

[14] 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.

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

[16] 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.

[17] Ivan Damgård and Yuval Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In Victor Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 378–394. Springer, August 2005.

[18] 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.

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

[20] 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.

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

[22] Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

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

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

[25] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Information Processing Letters*, 14(4):183–186, 1982.

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

[27] 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.

[28] 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.

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

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

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

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

[33] 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.

[34] 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.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

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

[40] 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.

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

[42] 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.

[43] 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.

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

[45] Pratyay Mukherjee and Daniel Wichs. Two round multiparty computation via multi-key FHE. In Marc Fischlin and Jean-Sébastien Coron, editors, *EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 735–763. Springer, 2016.

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

[47] 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.

[48] 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.

[49] Russell Turpin and Brian A. Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Information Processing Letters*, 18(2):73–76, 1984.

[50] 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, in general, the expected running time of the composed protocol (i.e., the rounds its takes for all protocols to give output to all parties) is not preserved. We prove a formal example where this is the case. Concretely, consider a protocol realizing a particular ideal functionality such that the probability that all parties have completed the protocol by round $k$ is $p^k$ for some $0 < p < 1$. Then, the expected running time of the protocol is $1/p$ rounds, i.e., constant. (This is essentially the case in most randomized BA protocols starting with Feldman and Micali [24].) However, as implied by the following lemma, if $m$ instances of the protocol 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 protocol of Feldman and Micali [24] 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 independent, identically distributed (IID) geometric random variables, such that for every $i \in [m]$ it holds that $\Pr[X_i = 1] = p$ for some $0 < p < 1$. Then,*

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

*Proof.* As shown in Eisenberg [23], the expected value of the maximum of the random variables satisfies the following inequality:

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

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

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

□

# 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 [37]. More concretely, Katz et al. [37] 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

[29] As argued in [37], bounded-delay channels are essential as they allow parties to detect whether or not a message was sent within round.

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 [37], 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 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 [37] 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 $\mu$ 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 $\mu$ additional activations. Note that $P_i$ uses these activations to execute his current round instructions; since $\mu$ is a bound to the number of hybrids used in any round by any party, $\mu$ activations are enough for the party to complete its round (If $P_i$ finishes the round early, i.e., in less than $\mu$ activations, it simply does nothing until the $\mu$ activations are received.) Once $\mu$ 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 $\mu$ activations/messages and delivers output as soon as this has happened $r$ times. More concretely, imitating an $r$-round synchronous protocol with $\mu$ activations per party per round, upon being

[30] As pointed out in [37], 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 [37].

[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.

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 $\mu$ messages to $\mathcal{F}$, while the round-counter $\lambda$ 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 $\lambda$ by counting how many times every honest party has sent a certain number $\mu$ 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 $\rho$* if the current value of the above internal counter in $\mathcal{F}$ is $\lambda = \rho$.

We note that protocols in the synchronous model of [37] 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. [37] make use of the composition with joint-state (JUC) [14]. 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 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 [37, 14].

# C Composition of Probabilistic-Termination Protocols (Cont'd)

This section contains the proofs for Section 4.

## C.1 Composition with Deterministic Termination (Cont'd)

We start by giving the intuition for the proof of Theorem 4.2. Loosely speaking, the main differences between the SNF protocol $\pi$ implementing the functionality $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ and the compiled protocol $\pi'$ implementing $\mathcal{W}^{D^{\text{full}},c}_{\text{sl-strict}}(\mathcal{F})$ is that the hybrids are that $\pi$ uses CSF as hybrids whereas $\pi'$ uses wrapped CSFs, and in addition, parties might not start at the same round, but with a slack of $c$ rounds. In order to ensure that any potential overlap between concurrent calls to different wrapped hybrids remain secure, the wrappers expand each round to $3c + 1$ rounds.

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

**Theorem 4.2.** *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, and let $\pi$ an SNF protocol that UC-realizes $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, for some depth-1 distribution $D$, assuming that all honest parties receive their inputs at the same round. Let $D_1, \ldots, D_m$ be*

[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.

*arbitrary distributions over traces,* $D^{\mathsf{full}} = \mathsf{full\text{-}trace}(D, D_1, \ldots, D_m)$*, and* $c \geq 0$*. Then, protocol* $\pi' = \mathsf{Comp}^c_{\mathrm{DT}}(\pi, D_1, \ldots, D_m)$ *UC-realizes* $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$ *in the* $(\mathcal{W}^{D_1,c}_{\text{sl-strict}}(\mathcal{F}_1), \ldots, \mathcal{W}^{D_m,c}_{\text{sl-strict}}(\mathcal{F}_m))$*-hybrid model, assuming that all honest parties receive their inputs within* $c+1$ *consecutive rounds.*

*The expected round complexity of the compiled protocol* $\pi'$ *is*

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

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

*Proof.* Let $\mathcal{S}$ be the simulator for protocol $\pi$ running with the dummy adversary.[34] Consider the following simulator $\mathcal{S}'$ for $\pi'$, that internally runs a copy of $\mathcal{S}$. Initially, $\mathcal{S}'$ sets slack counters $c_1, \ldots, c_n \leftarrow 0$ and proceeds as follows.

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

  Along with the very first such message, $\mathcal{S}'$ receives a trace $T^{\mathsf{full}}$ from $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$. $\mathcal{S}'$ constructs a new trace $T$ with the root $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$, where the leaves are set as follows: Each node in the first layer of $T^{\mathsf{full}}$ is a root for a subtree labeled with $\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i)$ (for some $i \in [m]$), $\mathcal{S}'$ adds the leaf $\mathcal{F}_i$ to the first layer in $T$. Finally, $\mathcal{S}'$ passes $T$ to $\mathcal{S}$.
- Simulate the execution of every wrapped hybrid $\mathcal{H}_i = \mathcal{W}^{D_j,c}_{\text{sl-strict}}(\mathcal{F}_j)$ (for some $j \in [m]$) in the order they appear in the first layer in $T^{\mathsf{full}}$ as follows[35] (the first such hybrid must be simulated as early as in round $\rho = 1$. Note that if there is actual slack among the parties, the simulations of consecutive hybrids overlap):
  - Let $\rho_i$ be the (simulated) round counter of $\mathcal{H}_i$ and let $T^{\mathsf{full}}_i$ be the corresponding subtree in $T^{\mathsf{full}}$.
  - In any round $\rho_i$ forward the messages $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ (that are directed to $\mathcal{H}_i$) from the environment to $\mathcal{S}$.
  - For every party $P_j$, in round $\rho_i = c_j + 1$, obtain the simulated leakage for $P_j$ from $\mathcal{S}$ and pass it to the environment;[36] add $(\texttt{trace}, \mathsf{sid}, T^{\mathsf{full}}_i)$ to the first such message.
  - In all other rounds $\rho_i$, simply forward $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$ to the environment (to simulate the advancement of the execution of $\mathcal{H}_i$).
  - The simulation for party $P_j$ ends in round $\rho_i = B_c \cdot c_{\mathsf{tr}}(T^{\mathsf{full}}_i) + c_j$.

Let $\mathcal{Z}'$ be an environment that can distinguish between an execution of protocol $\pi'$ in the $(\mathcal{W}^{D_1,c}_{\text{sl-strict}}(\mathcal{F}_1), \ldots, \mathcal{W}^{D_m,c}_{\text{sl-strict}}(\mathcal{F}_m))$-hybrid model with the dummy adversary and the execution in the ideal model with $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-strict}}(\mathcal{F})$ and $\mathcal{S}'$. We construct the following environment $\mathcal{Z}$ distinguishing between an execution of $\pi$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model with the dummy adversary and the ideal model with $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ and $\mathcal{S}$:

[34] Recall that proving security with respect to the dummy adversary is sufficient (cf. [10, Claim 10]).

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

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

- $\mathcal{Z}$ internally runs a copy of $\mathcal{Z}'$, emulating the parties and the adversary (either in a real execution of $\pi$ or an ideal execution of $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$). Initialize slack counters $c_1, \ldots, c_n \leftarrow 0$ and a simulated round counter $\rho$ (for $\mathcal{Z}'$).
- Whenever $\mathcal{Z}'$ sends a message $(\texttt{input}, \mathsf{sid}, \cdot)$ to $P_j$ in rounds $\rho = 1, \ldots, 2c+1$, $\mathcal{Z}$ forward the message to $P_j$ and records slack $c_j \leftarrow \rho - 1$.
- For each executed (resp. simulated) two-round CSF hybrid $\mathcal{F}_i$, proceed as follows to simulate an execution (resp. simulation) of $\mathcal{W}^{D_i,c}_{\text{sl-flex}}(\mathcal{F}_i)$ to $\mathcal{Z}'$ (the first such simulation takes place as early as in round $\rho = 1$. Note that if there is actual slack among the parties, the simulations of consecutive hybrids overlap):
  - Initialize a round counter $\rho_i \leftarrow 1$ and sample a trace $T_i^{\mathsf{full}}$ from $D_i$.
  - At any round, forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from $\mathcal{Z}'$ to the adversary.
  - For every party $P_j$, in round $\rho_i = c_j + 1$, obtain the leakage for $P_j$ from the adversary and pass it to $\mathcal{Z}'$;[37] add $(\texttt{trace}, \mathsf{sid}, T_i^{\mathsf{full}})$ to the first such message.
  - In all other rounds $\rho_i$, upon receiving $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{Z}'$ for some party $P_j$, pass $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ to $\mathcal{Z}'$ (to simulate the advancement of the execution).
  - The simulation for party $P_j$ ends in round $\rho_i = B_c \cdot c_{\mathsf{tr}}(T_i^{\mathsf{full}}) + c_j$.
- At any round, forward $(\texttt{output}, \mathsf{sid}, \cdot)$ messages from a party to $\mathcal{Z}'$.
- Output whatever decision bit $\mathcal{Z}'$ outputs.

It can be seen by inspection that:

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

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

## C.2 Composition with Probabilistic Termination (Cont'd)

The intuition for proving Theorem 4.3 is similar to that of proving Theorem 4.2. In addition to simply synchronizing between the simulator $\mathcal{S}$ and the ideal functionality and environment, $\mathcal{S}'$ must also address the following issues. First, some CSFs ($\mathcal{F}_i$ for $i \in I$) are wrapped using the flexible wrapper whereas other ($\mathcal{F}_i$ for $i \notin I$) are wrapped using the strict wrapper. Second, $\mathcal{S}'$ must simulate the termination procedure at the end of every flexibly wrapped CSF and at the end of the simulation.

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

**Theorem 4.3.** *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, and let $\pi$ an SNF protocol that UC-realizes $\mathcal{W}^D_{\text{flex}}(\mathcal{F})$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, for some depth-1 distribution $D$, assuming that all honest parties receive their inputs at the same round. Let $I \subseteq [m]$ be the subset (of indices) of functionalities to be wrapped using the flexible wrapper, let $D_1, \ldots, D_m$ be arbitrary distributions over traces, denote $D^{\mathsf{full}} = (D, D_1, \ldots, D_m)$ and let $c \geq 0$. Assume that $\mathcal{F}$ and $\mathcal{F}_i$, for every $i \in I$, are public-output functionalities.*

*Then, the compiled protocol $\pi' = \mathsf{Comp}^c_{\text{PTR}}(\pi, D_1, \ldots, D_m, I)$ UC-realizes $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$ in the $(\mathcal{W}(\mathcal{F}_1), \ldots, \mathcal{W}(\mathcal{F}_m))$-hybrid model, where $\mathcal{W}(\mathcal{F}_i) = \mathcal{W}^{D_i,c}_{\text{sl-flex}}(\mathcal{F}_i)$ if $i \in I$ and $\mathcal{W}(\mathcal{F}_i) = \mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$ if $i \notin I$, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

*The expected round complexity of the compiled protocol $\pi'$ is*

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

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

*Proof.* Let $\mathcal{S}$ be the simulator for protocol $\pi$ running with the dummy adversary.[38] Consider the following simulator $\mathcal{S}'$ for $\pi'$, that internally runs a copy of $\mathcal{S}$. Initially, $\mathcal{S}'$ sets slack counters $c_1, \ldots, c_n \leftarrow 0$ and proceeds as follows.

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

  Along with the very first such message, $\mathcal{S}'$ receives a trace $T^{\mathsf{full}}$ from $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$. $\mathcal{S}'$ constructs a new trace $T$ with the root $\mathcal{W}^D_{\text{flex}}(\mathcal{F})$, where the leaves are set as follows: Each node in the first layer of $T^{\mathsf{full}}$ is a root for a subtree labeled with $\mathcal{W}^{D_i}_{\text{strict}}(\mathcal{F}_i)$ or $\mathcal{W}^{D_i}_{\text{flex}}(\mathcal{F}_i)$ (for some $i \in [m]$), $\mathcal{S}'$ adds the leaf $\mathcal{F}_i$ to the first layer in $T$. Finally, $\mathcal{S}'$ passes $T$ to $\mathcal{S}$.
- Simulate the execution of all wrapped hybrids $\mathcal{H}_i$ in the order they appear in $T^{\mathsf{full}}$[39] (the first such hybrid must be simulated as early as in round $\rho = 1$. Note that if there is actual slack among the parties, the simulations of consecutive hybrids overlap). If the hybrid $\mathcal{H}_i$ is of the form $\mathcal{W}^{D_j,c}_{\text{sl-strict}}(\mathcal{F}_j)$ (for some $j \in [m]$), i.e., if $j \notin I$, proceed as follows:
  - Let $\rho_i$ be the (simulated) round counter of $\mathcal{H}_i$ and let $T^{\mathsf{full}}_i$ be the corresponding subtree in $T^{\mathsf{full}}$.
  - In any round $\rho_i$ forward the messages $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ (that are directed to $\mathcal{H}_i$) from the environment to $\mathcal{S}$.
  - For every party $P_j$, in round $\rho_i = c_j + 1$, obtain the simulated leakage for $P_j$ from $\mathcal{S}$ and pass it to the environment;[40] add $(\texttt{trace}, \mathsf{sid}, T^{\mathsf{full}}_i)$ to the first such message.
  - In all other rounds $\rho_i$, simply forward $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$ to the environment (to simulate the advancement of $\mathcal{H}_i$' execution).
  - The simulation for party $P_j$ ends in round $\rho_i = B_c \cdot c_{\mathsf{tr}}(T^{\mathsf{full}}_i) + c_j$.

[38] Recall that proving security with respect to the dummy adversary is sufficient [10, Claim 10].
[39] Recall that the children at each node in a trace are ordered.
[40] $\mathcal{S}$ can be advanced by suitably sending it $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages.

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

- Let $\rho_i$ be the (simulated) round counter of $\mathcal{H}_i$ and let $T_i^{\mathsf{full}}$ be the corresponding subtree in $T^{\mathsf{full}}$. Set $\rho_{\mathsf{term}} \leftarrow B_c \cdot c_{\mathsf{tr}}(T^{\mathsf{full}}) + 2 \cdot \mathsf{flex}_{\mathsf{tr}}(T^{\mathsf{full}}) + c$.
- In any round $\rho_i$ forward the messages $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ (that are directed to $\mathcal{H}_i$) from the environment to $\mathcal{S}$.
- For every party $P_j$, in round $\rho_i = c_j + 1$, obtain the simulated leakage for $P_j$ from $\mathcal{S}$ and pass it to the environment;[41] add $(\texttt{trace}, \mathsf{sid}, T_i^{\mathsf{full}})$ to the first such message.
- In all other rounds $\rho_i$, simply forward $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{W}_{\text{sl-flex}}^{D^{\mathsf{full}},c}(\mathcal{F})$ to the environment (to simulate the advancement of $\mathcal{H}_i$' execution).
- If the environment issues $(\texttt{early-output}, \mathsf{sid}, \cdot)$ commands for certain parties before round $\rho_{\mathsf{term}}$, set $c_i \leftarrow 0$ for these parties 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 party, set $c_i \leftarrow 0$ for all parties and end the simulation of $\mathcal{H}$ in round $\rho_{\mathsf{term}}$.

- When $\mathcal{S}$ wants to output $(\texttt{early-output}, \mathsf{sid}, P_j)$ to $\mathcal{W}_{\text{sl-flex}}^{D^{\mathsf{full}},c}(\mathcal{F})$, proceed as follows:
  - Pass $(\texttt{get-output}, \mathsf{sid})$ to $\mathcal{W}_{\text{sl-flex}}^{D^{\mathsf{full}},c}(\mathcal{F})$, obtain $(\texttt{output}, \mathsf{sid}, y)$, and record $y$ (the first time).
  - Simulate $P_j$ sending $(\mathsf{end}, \mathsf{sid}, y)$ to all parties.
  - For every party $P_j$, keep track of how many simulated $(\mathsf{end}, \mathsf{sid}, \cdot)$ messages have been received by $P_j$ (including those sent by corrupted parties).
    - When a party receives $t + 1$ such messages (for the same value $y$), simulate that party's sending of such a message of its own (unless already done so).
    - When a party $P_j$ receives $n - t$ such messages (for the same value $y$), send $(\texttt{early-output}, \mathsf{sid}, P_j)$ to $\mathcal{W}_{\text{PT}}^{D^{\mathsf{full}}}(\mathcal{F})$.

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

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

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

- For each executed (resp. simulated) two-round CSF hybrid $\mathcal{F}_i$, simulate an execution (resp. simulation) of $\mathcal{W}_{\text{sl-flex}}^{D_i,c}(\mathcal{F}_i)$ to $\mathcal{Z}'$ (the first such simulation takes place as early as in round $\rho = 1$. Note that if there is actual slack among the parties, the simulations of consecutive hybrids overlap). If the hybrid $\mathcal{H}_i$ is of the form $\mathcal{W}_{\text{sl-strict}}^{D_j,c}(\mathcal{F}_j)$ (for some $j \in [m]$), i.e., if $j \notin I$, proceed as follows:
  - Initialize a round counter $\rho_i \leftarrow 1$ and sample a trace $T_i^{\mathsf{full}}$ from $D_i$.

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

– At any round, forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from $\mathcal{Z}'$ to the adversary.

– For every party $P_j$, in round $\rho_i = c_j + 1$, obtain the leakage for $P_j$ from the adversary and pass it to $\mathcal{Z}'$;[42] add $(\texttt{trace}, \mathsf{sid}, T_i^{\mathsf{full}})$ to the first such message.

– In all other rounds $\rho_i$, upon receiving $(\texttt{fetch-output}, \mathsf{sid}, \cdot)$ messages from $\mathcal{Z}'$ for some party $P_j$, pass $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ to $\mathcal{Z}'$ (to simulate the advancement of the execution).

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

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

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

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

– At any round, forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from $\mathcal{Z}'$ to the adversary.

– For every party $P_j$, in round $\rho_i = c_j + 1$, obtain the leakage for $P_j$ from the adversary and pass it to $\mathcal{Z}'$;[43] add $(\texttt{trace}, \mathsf{sid}, T_i^{\mathsf{full}})$ to the first such message.

– In all other rounds $\rho_i$, upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ messages from $\mathcal{Z}'$ for some party $P_j$, pass $(\texttt{fetch-output}, \mathsf{sid}, P_j)$ to $\mathcal{Z}'$ (to simulate the advancement of the execution).

– If $\mathcal{Z}'$ issues $(\texttt{early-output}, \mathsf{sid}, \cdot)$ commands for certain parties before round $\rho_{\mathsf{term}}$, set $c_i \leftarrow 0$ for these parties and $c_i \leftarrow 1$ for the others, and end the simulation of $\mathcal{H}$ one round later. If $\mathcal{Z}'$ did not issue such a command to any party by round $\rho_{\mathsf{term}}$, set $c_i \leftarrow 0$ for all parties and end the simulation of $\mathcal{H}$ in round $\rho_{\mathsf{term}}$.

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

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

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

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

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

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

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

It can be seen by inspection that:

- When $\mathcal{Z}$ interacts with a real-world execution of $\pi$ 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 $\pi'$ with hybrids $\mathcal{W}(\mathcal{F}_i)$ and the dummy adversary.

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

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

The expected round complexity follows by similar arguments as in Theorem 4.2. □

# 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 $\pi_{\text{RBA}}$, we now proceed to prove Lemma 5.1.

**Lemma 5.1.** *Let $t < n/3$, then, assuming all honest parties receive their inputs at the same round, protocol $\pi_{\text{RBA}}$ UC-realizes $\mathcal{F}_{\text{RBA}} = \mathcal{W}_{\text{flex}}^{D_{\text{RBA}}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{OC}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model, with perfect security, in the presence of an adaptive malicious $t$-adversary.*

*Proof.* We first claim correctness, i.e., that all honest parties output the same value and that if $n-t$ of the inputs are the same, this value will be the common output. The protocol $\pi_{\text{RBA}}$ consists of two parts, the first is running (up to) $\tau$ 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 $\tau$, i.e., has value $\mathsf{term} = \tau$) consists of calling a BA functionality. As shown in [24, 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 $\alpha$ (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 $\alpha$. It follows that:

- In case $n-t$ honest parties (in particular if all honest parties) start with the same input, they will agree on this value as their output and terminate in the first phase. (In all other cases it remains only to show that all honest parties agree on the output.)
- In case the first honest party received output in phase $\alpha < \tau - 1$, it holds that by phase $\alpha + 1 < \tau$ all honest parties will receive the same output (i.e., $0 < \mathsf{term} < \tau$ for all honest parties), and so correctness follows from [24].
- In case no honest party received output in all $\tau$ 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 outputs in phase $\tau$ (i.e., $\mathsf{term} = \tau$ for all honest parties), then by [24] 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 outputs in phase $\tau$ (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 outputs in phase $\tau - 1$ (i.e., $\mathsf{term} = \tau - 1$), they do not send any input to $\mathcal{F}_{\text{BA}}$. However, the remaining honest parties will receive the same output in phase $\tau$ (i.e., $\mathsf{term} = \tau$), and will output this value, *regardless* of the output they receive from $\mathcal{F}_{\text{BA}}$. Therefore, correctness is satisfied.

Regarding termination, [24, 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 $\tau$ phases, termination is guaranteed by $\mathcal{F}_{\text{BA}}$.

We now prove that $\pi_{\text{RBA}}$ UC-realizes $\mathcal{F}_{\text{RBA}}$. Let $\mathcal{A}$ be the dummy adversary and let $\mathcal{Z}$ be an environment. We construct a simulator $\mathcal{S}$ that simulates the honest parties in $\pi_{\text{RBA}}$, the adversary $\mathcal{A}$ and the ideal functionalities $\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{OC}}$ and $\mathcal{F}_{\text{BA}}$ to the environment, 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 $r_{\mathsf{out}}$ 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 $(\mathsf{sid}, b_j)$ 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 $r_{\mathsf{out}}$) 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 honest parties.)
  - Note that the case $r = \tau + 1$ can never happen.

- In case no honest party has terminated prior to phase $r_{\mathsf{out}}$, 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 $r_{\mathsf{out}}$'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 $\tau$ 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 $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ message sent by the adversary.

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

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

□

### D.1.1 Multi-Valued Byzantine Agreement Protocol

As presented above, $\pi_{\text{RBA}}$ is a binary BA protocol. Using a transformation due to Turpin and Coan [49], the decision domain can be extended without increasing the expected running time. Given a set $V \subseteq \{0,1\}^*$, denote by $D_{\text{MV-BA}}$ the deterministic distribution that outputs a depth-1 trace consisting of a root $\mathcal{W}_{\text{strict}}^{D_{\text{MV-BA}}}(\mathcal{F}_{\text{BA}}^{V})$ and three leaves $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$.

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

The proof of the Lemma is straight-forward.

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

The protocol $\pi_{\text{MV-BA}}$ is parametrized 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 $(\mathsf{sid}, v_i)$ to all the parties (via $\mathcal{F}_{\text{PSMT}}$). Denote by $v_j$ the value received from $P_j$ in this round.
- In round $\rho = 2$: 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 $(\mathsf{sid}, y)$ to all the parties (via $\mathcal{F}_{\text{PSMT}}$). Denote by $y_j$ the value received from $P_j$ in this round.
- In round $\rho = 3$: 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 $\mathcal{F}_{\text{BA}}^{\{0,1\}}$ and let $(\texttt{output}, \mathsf{sid}, b)$, with $b \in \{0, 1\}$, be the output from $\mathcal{F}_{\text{BA}}^{\{0,1\}}$. 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

**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 $\mathcal{W}_{\text{sl-flex}}^{D,c}(\mathcal{F}_{\text{BA}}^{V})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious $t$-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

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

From Lemma D.1, $\pi_{\text{MV-BA}}$ UC-realizes $\mathcal{W}_{\text{strict}}^{D_{\text{MV-BA}}}(\mathcal{F}_{\text{BA}}^{V})$, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}^{\{0,1\}})$-hybrid model, in three rounds, assuming all parties receive their inputs at the same round. Following Theorem 4.4, the compiled protocol $\mathsf{Comp}_{\text{PT}}^{c}(\pi_{\text{MV-BA}}, D_1, D_2, I)$ UC-realizes $\mathcal{F}_{\text{BA}}^{\text{PT,V}}$, in the $(\mathcal{F}_{\text{PSMT}}^{\text{DT}}, \mathcal{F}_{\text{BA}}^{\text{PT}})$-hybrid model, in an expected constant number of rounds, assuming all parties receive their inputs within $c + 1$ consecutive rounds.

The proof follows since, following Lemma 4.5 and Theorem 5.2 the functionalities $\mathcal{F}_{\text{PSMT}}^{\text{DT}}$ and $\mathcal{F}_{\text{BA}}^{\text{PT}}$ can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, using expected-constant-round protocols. □

## D.2 Fast and Perfectly Secure Parallel Broadcast (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 [32] 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 $\pi_{\text{UPBC}}$ (see Figure 12 for a detailed description) is parametrized by two integers $d$ and $m$. Initially, each party distributes its input to all other parties. The underlying idea of the protocol is to run $n \cdot m$ instances of the BA protocol $\pi_{\text{RBA}}$ in parallel, such that for each $P_i$, a class of $m$ instances of $\pi_{\text{RBA}}$ 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 a CSF whose function is parametrized 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 $\pi_{\text{UPBC}}$, 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 [29], 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.

**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 $nm$ "terminating phases" $(r_1, \ldots, r_{nm})$ from the distribution $D_{\text{RBA}}$ 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 $\tau$ phases.)*

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

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

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

**Protocol $\pi_{\text{UPBC}}$**

Protocol $\pi_{\text{UPBC}}$, parametrized by positive integers $d$ (number of phases to run the truncated BA functionality) and $m$ (how many instances of truncated BA to compute for each input value). The functionality $\mathcal{F}_{\text{T-RBA}}$ runs $nm$ instances in parallel, and is parametrized by the distribution $D_{\text{RBA}}$ 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 $(\mathsf{sid}, x_i)$ to all the parties (via $\mathcal{F}_{\text{PSMT}}$). Denote by $x_j$ the value received from $P_j$.
3. While $\mathsf{term} = 0$ and $\alpha \leq \tau$, do the following:
   (a) Set $\alpha \leftarrow \alpha + 1$ and send values to $\mathcal{F}_{\text{T-RBA}}$, such that the value $x_j$ is sent to the $m$ instances corresponding to the $j$'th value. Formally, prepare the vector $\boldsymbol{z} = (z_1, \ldots, z_{nm})$ such that for every $j \in [n]$ and every $l \in [m]$ set $z_{(j-1)m+l} = x_j$. Send $(\mathtt{input}, \mathsf{sid}_1^\alpha, \boldsymbol{z})$ to $\mathcal{F}_{\text{T-RBA}}$.
   Let $(\mathtt{output}, \mathsf{sid}_1^\alpha, \boldsymbol{v})$ be the output from $\mathcal{F}_{\text{T-RBA}}$, where $\boldsymbol{v}$ is a vector of $nm$ pairs $((v_1^1, v_2^1), \ldots, (v_1^{nm}, v_2^{nm}))$ with $v_1^j, v_2^j \in V \cup \{\perp\}$.
   (b) For every $j \in [n]$, set $S_1^j \leftarrow \{v_1^{(j-1)m+1}, \ldots, v_1^{jm}\}$ (corresponding to output values before phase $d$) and $S_2^j \leftarrow \{v_2^{(j-1)m+1}, \ldots, v_2^{jm}\}$ (corresponding to output values at phase $d$).
   (c) If $S_1^j \neq \emptyset$ 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 $\boldsymbol{c}_j = \emptyset$.
   (d) Send $(\mathtt{input}, \mathsf{sid}_2^\alpha, \lambda)$ to the functionality $\mathcal{F}_{\text{LE}}$. Let $(\mathtt{output}, \mathsf{sid}_2^\alpha, k)$, with $k \in [n]$, be the output received from $\mathcal{F}_{\text{LE}}$.
   (e) Send $(\mathtt{input}, \mathsf{sid}_3^\alpha, \boldsymbol{c}_k)$ to $\mathcal{F}_{\text{BA}}$, parametrized by the set $V^n \cup \{\emptyset\}$. Let $(\mathtt{output}, \mathsf{sid}_3^\alpha, \boldsymbol{c})$ be the output received from $\mathcal{F}_{\text{BA}}$ (with $\boldsymbol{c} = (c_1, \ldots, c_n) \in V^n$ or $\boldsymbol{c} = \emptyset$).
   (f) If $\boldsymbol{c} \neq \emptyset$ and for every $j \in [n]$, $c_j \in S_1^j \cup S_2^j$ then set $b \leftarrow 1$; otherwise set $b \leftarrow 0$.
   (g) Send $(\mathtt{input}, \mathsf{sid}_4^\alpha, b)$ to $\mathcal{F}_{\text{BA}}$, parametrized by the set $\{0, 1\}$. Let $(\mathtt{output}, \mathsf{sid}_4^\alpha, \beta)$, with $\beta \in \{0, 1\}$, be the output received from $\mathcal{F}_{\text{BA}}$. If $\beta = 1$ then set $\mathsf{term} \leftarrow 1$.
4. If $\mathsf{term} = 1$, then output $(\mathtt{output}, \mathsf{sid}, \boldsymbol{c})$ and halt.
5. Else, set the vector $\boldsymbol{x}_i = (\lambda, \ldots, \lambda, x_i, \lambda, \ldots, \lambda)$ (the vector of length $n$ whose $i$'th coordinate is $x_i$ and all other cooridinates are the empty string $\lambda$) and send $(\mathtt{input}, \mathsf{sid}, \boldsymbol{x}_i)$ to $\mathcal{F}_{\text{UPBC}}$. Let $(\mathtt{output}, \mathsf{sid}, \boldsymbol{c})$ be the output received from $\mathcal{F}_{\text{UPBC}}$. Output $(\mathtt{output}, \mathsf{sid}, \boldsymbol{c})$ and halt.

Figure 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

*Proof.* We first claim correctness. The protocol $\pi_{\text{UPBC}}$ consists of two parts, the first is running (up to) $\tau$ 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 $\tau$ phases (i.e., $\mathsf{term} = 0$ for all honest parties), all honest parties send their initial values to $\mathcal{F}_{\text{UPBC}}$ and output the result, hence, correctness follows from the $\mathcal{F}_{\text{UPBC}}$ functionality.

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

We now prove that $\pi_{\text{UPBC}}$ UC-realizes $\mathcal{F}_{\text{PT-UPBC}}$. Let $\mathcal{A}$ be the dummy adversary and let $\mathcal{Z}$ be an environment. We construct a simulator $\mathcal{S}$ that simulates the honest parties in $\pi_{\text{UPBC}}$, the adversary $\mathcal{A}$ 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 the environment, 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-termination phase $r_{\mathsf{out}}$ 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 $(\mathsf{sid}, x_j)$ on behalf of a corrupted party $P_j$ to some honest party during the first round, $\mathcal{S}$ sends $(\texttt{input}, \mathsf{sid}, x_j)$ to $\mathcal{F}_{\text{PT-UPBC}}$ on behalf of $P_j$. (Note that $x_j$ is in fact a vector.)
- Whenever $\mathcal{A}$ requests to corrupt some $P_i \in \mathcal{P}$, $\mathcal{S}$ 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 $D_{\text{RBA}}$. 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 $r_{\mathsf{out}}$) 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 $r_{\mathsf{out}}$, then $\mathcal{S}$ proceeds as follows:
  - In case $r_{\mathsf{out}} \leq \tau$, when simulating $\mathcal{F}_{\text{T-RBA}}$ in the $r_{\mathsf{out}}$'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 $\tau$ 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 the view of the environment $\mathcal{Z}$ is identically distributed when interacting with a real-world execution of $\pi_{\text{UPBC}}$ and the dummy adversary, and when interacting with the simulator $\mathcal{S}$ and the ideal model computation of $\mathcal{F}_{\text{PT-UPBC}}$, i.e.,

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

□

Using Theorem 4.3 we obtain the following as a result.

**Theorem 5.4.** *Let $c \geq 0$ and $t < n/3$. There exists an efficiently sampleable distribution $D$ such that the functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}_{\text{UPBC}})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious $t$-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

*Proof (sketch).* Denote by $D_{\text{T-RBA}}$ the deterministic distribution that outputs a trace consisting of a root $\mathcal{W}^{D_{\text{T-RBA}}}_{\text{strict}}(\mathcal{F}_{\text{T-RBA}})$ and a constant number of leaves $\mathcal{F}_{\text{PSMT}}$ (corresponding to $d$ phases of $\pi_{\text{RBA}}$). Denote by $D_{\text{LE}}$ the deterministic distribution that outputs a trace consisting of a root $\mathcal{W}^{D_{\text{LE}}}_{\text{strict}}(\mathcal{F}_{\text{LE}})$ and a constant number of leaves $\mathcal{F}_{\text{PSMT}}$ (to be precise 36 leaves, as in the oblivious coin protocol from [24]) followed by $\mathcal{F}_{\text{RBA}}$. Denote by $D_{\text{DT-UPBC}}$ the deterministic distribution that outputs a trace consisting of a root $\mathcal{W}^{D_{\text{DT-UPBC}}}_{\text{strict}}(\mathcal{F}_{\text{UPBC}})$ and $O(n)$ leaves $\mathcal{F}_{\text{PSMT}}$.

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

Following Lemma D.2, $\pi_{\text{UPBC}}$ UC-realizes $\mathcal{W}^{D_{\text{UPBC}}}_{\text{flex}}(\mathcal{F}_{\text{UPBC}})$, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{BA}}, \mathcal{F}_{\text{LE}}, \mathcal{F}_{\text{T-RBA}}, \mathcal{F}_{\text{UPBC}})$–hybrid model, using an expected constant number of rounds, assuming that all the parties receive their inputs at the same round. By applying Theorem 4.3, the compiled protocol $\mathsf{Comp}^{c}_{\text{PTR}}(\pi_{\text{UPBC}}, D_1, D_2, D_3, D_4, D_5, I)$ UC-realizes $\mathcal{F}^{\text{PT}}_{\text{UPBC}}$, in the $(\mathcal{F}^{\text{DT}}_{\text{PSMT}}, \mathcal{F}^{\text{PT}}_{\text{BA}}, \mathcal{F}^{\text{PT}}_{\text{LE}}, f^{\text{DT}}_{\text{T-RBA}}, \mathcal{F}^{\text{DT}}_{\text{UPBC}})$-hybrid model, in an expected constant number of rounds, assuming all parties receive their inputs within $c+1$ consecutive rounds.

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