Round-Preserving Parallel Composition of Probabilistic-Termination Cryptographic Protocols*

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

April 24, 2017

Abstract

An important benchmark for multi-party computation protocols (MPC) is their *round complexity*. For several important MPC tasks, (tight) lower bounds on the round complexity are known. However, for some of these tasks, such as broadcast, the lower bounds can be circumvented when the termination round of every party is not a priori known, and simultaneous termination is not guaranteed. Protocols with this property are called *probabilistic-termination (PT)* protocols.

Running PT protocols in parallel affects the round complexity of the resulting protocol in somewhat unexpected ways. For instance, an execution of m protocols with constant expected round complexity might take $O(\log m)$ rounds to complete. In a seminal work, Ben-Or and El-Yaniv (Distributed Computing '03) developed a technique for parallel execution of arbitrarily many broadcast protocols, while preserving expected round complexity. More recently, Cohen *et al.* (CRYPTO '16) devised a framework for universal composition of PT protocols, and provided the first composable parallel-broadcast protocol with a simulation-based proof. These constructions crucially rely on the fact that broadcast is "privacy free," and do not generalize to arbitrary protocols in a straightforward way. This raises the question of whether it is possible to execute *arbitrary* PT protocols in parallel, without increasing the round complexity.

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

To prove our results, we utilize the language and results by Cohen *et al.*, which we extend to capture parallel composition and reactive functionalities, and to handle the case of an honest majority.

*An extended abstract of this work appeared at *ICALP 2017, Track A*.

†School of Computer Science, Tel Aviv University. E-mail: cohenran@tauex.tau.ac.il. Research supported by ERC starting grant 638121.

‡Courant Institute of Mathematical Sciences, New York University. E-mail: corettis@nyu.edu.

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

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

Contents

1 Introduction

Secure multi-party computation (MPC) [56, 30] allows a set of parties to jointly perform a computation on their inputs, in such a way that no coalition of cheating parties can learn any information beyond what is revealed by their outputs (privacy) or affect the outputs of the computation in any way other than by choosing their own inputs (correctness). Since the first seminal works on MPC [56, 30, 6, 13, 51], it has been studied in a variety of different settings and for numerous security notions: there exist protocols secure against passively corrupted (aka semi-honest) parties and against actively corrupted (aka malicious) parties; the underlying network can be synchronous or asynchronous; and the required security guarantees can be information-theoretic or computational—to name but a few of the axes along which the MPC task can be evaluated.

The prevalent model for the design of MPC protocols is the synchronous model, where the protocol proceeds in rounds. In this setting, the round complexity, i.e., the number of rounds it takes for a protocol to deliver outputs, is arguably the most important efficiency metric. Tight lower bounds are known on the round complexity of several MPC tasks. For example, for the well-known problems of Byzantine agreement (BA) and broadcast [48, 45], it is known that any protocol against an active attacker corrupting a linear fraction of the parties has linear round complexity [25, 22]. This result has quite far-reaching consequences as, starting with the seminal MPC works mentioned above, a common assumption in the design of secure protocols has been that the parties have access to a broadcast channel, which they potentially invoke in every round. In reality, 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. It follows that even though the round complexity of many MPC protocols is linear in the multiplicative depth of the circuit being computed, their actual running time depends on the number of parties, when executed over point-to-point channels.

The above lower bound on the number rounds for BA holds when all honest parties are required to complete the protocol together, at the same round [23]. Indeed, *randomized* BA protocols that circumvent this lower bound and run in *expected constant* number of rounds (cf. [4, 50, 24, 26, 41]) do not provide simultaneous termination, i.e., once a party completes the protocol's execution it cannot know whether all honest parties have also terminated or if some honest parties are still running the protocol; in particular, the termination round of each party is not a priori known. A protocol with this property is said to have *probabilistic termination (PT)*.

As pointed out by Ben-Or and El-Yaniv [5], when several such PT protocols are executed in parallel, the expected round complexity of the combined execution might no longer be constant (specifically, might not be equal to the maximum of the expected running times of the individual protocols). Indeed, when m protocols, whose termination round is geometrically distributed (and so, have constant expected round complexity), are run in parallel, the expected number of rounds that elapse before all of them terminate is $\Theta(\log m)$ [16]. While an elegant mechanism was proposed in [5] for implementing parallel calls to broadcast such that the total expected number of rounds remains constant, it did not provide any guarantees to remain secure under composition, raising questions about its usability in a higher-level protocol (such as the MPC setting described above). Such a shortcoming was recently addressed by Cohen et al. [16] who provided a framework for universal composition of PT protocols (building upon the universal-composition framework of [8]). An application of their result was the first composable protocol for parallel broadcast (with a simulation-based proof) that can be used for securely replacing broadcast channels in arbitrary protocols, and whose round complexity is constant in expectation.

Indeed, an immediate application of the composable parallel-broadcast protocol from [16] is plugging it into broadcast-model MPC protocols in order to obtain point-to-point protocols with

a round complexity that is independent of the number of parties. In the information-theoretic setting, this approach yields protocols whose round complexity depends on the depth of the circuit computing the function [6, 13, 51, 18], whereas in the computational setting, assuming standard cryptographic assumptions, this approach yields expected-constant-round protocols [44, 3, 20, 38, 2, 29, 31, 46]. However, the resulting point-to-point protocols have probabilistic-termination on their own. The techniques used for composing PT broadcast protocols in parallel crucially rely on the fact that broadcast is a privacy-free functionality, and a naïve generalization of this approach to arbitrary PT protocols fails to be secure. This raises the question of whether it is possible to execute *arbitrary* PT protocols in parallel, without increasing the round complexity.

We remark that circumventing lower bounds on round complexity is just one of the areas where such PT protocols have been successfully used. Indeed, randomizing the termination round has been proven to be a very useful technique in circumventing impossibilities and improving efficiency for many cryptographic protocols. Notable examples include *non-committing encryption* [21], cryptographic protocols designed for rational parties [34, 27, 47, 1, 32, 28], concurrent zero-knowledge protocols [10, 14] and parallel repetition of interactive arguments [33, 35]. The rich literature on such protocols motivates a thorough investigation of their security and composability. As mentioned above, in [16] the initial foundations were laid out for such an investigation, but what was proven for arbitrary PT protocols was a round-preserving *sequential* composition theorem, leaving parallel composition as an open question.

Our contributions. In this work, we investigate the issue of parallel composition for *arbitrary* protocols with probabilistic termination. In particular, we develop a compiler such that given functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_{\mathtt{M}}$ and protocols $\pi_1, \ldots, \pi_{\mathtt{M}}$, where for every $i \in [\mathtt{M}]$, protocol π_i realizes $\mathcal{F}_i$ (possibly using correlated randomness as setup[1]), then the compiled protocol realizes the parallel composition of the functionalities, denoted $(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$.

Our compiler uses the underlying protocols in a black-box manner[2], is robust (i.e., secure without abort) and resilient against a computationally unbounded active adversary, adaptively corrupting up to $t < n/2$ parties (which is optimal [51]). Moreover, our compiler is round-preserving, meaning that if the maximal (expected) round complexity of each protocol is μ, then the expected round complexity of the compiled protocol is $O(\mu)$. For example, if each protocol π_i has constant expected round complexity, then so does the compiled protocol. Recall that this task is quite complicated even for the simple case of BA (cf. [5, 16]). For arbitrary functionalities it is even more involved, since as we show, the approach from [5] cannot be applied in a functionally black-box way in this case. Thus, effectively, our result is the first round-preserving parallel composition result for arbitrary multi-party protocols/tasks with probabilistic termination.

We now describe the ideas underlying our compiler. In [5] (see also [16]), a round-preserving parallel-broadcast protocol was constructed by iteratively running, for a constant number of rounds, *multiple* instances of BA protocols (each instance is executed multiple times in parallel, in a batch), hoping that at least one execution of every BA instance will complete. By choosing the multiplicity suitably, this would occur with constant probability, and the process need therefore be repeated a constant expected number of times only.

At first sight it might seem that this idea can be applied to arbitrary tasks, but this is not the case. Intuitively, the reason is that if the tasks that we want to compose in parallel have privacy requirements, then making the parties run them in (parallel) "batches" with the same input might

[1] A trusted setup phase is needed for implementing broadcast in the honest-majority setting. As shown in [15, 17] some functions can be computed without such a setup phase.

[2] Following [37], by a black-box access to a protocol we mean a black-box usage of a semi-honest MPC protocol computing its next-message function.

compromise privacy, since the adversary will be able to use different inputs and learn multiple outputs of the function(s). This issue is not relevant for broadcast, because it is a "privacy-free" functionality; the adversary may learn the result of multiple computations using the same inputs for honest parties, without compromising security.

To cope with the above issue, our parallel-composition compiler generalizes the approach of [5] in a privacy-preserving manner. At a high level, it wraps the batching technique by an MPC protocol which restricts the parties to use the same input in all protocols for the same function. In particular, the compiler is defined in the *Setup-Commit-then-Prove* hybrid model [11, 39], which allows each party to commit to its input values and later execute multiple instances of every protocol, each time proving that the same input value is used in all executions.

The constructions in [11, 39] for realizing the Setup-Commit-then-Prove functionality are designed for the dishonest-majority setting and therefore allow for a premature abort. Since we assume an honest majority, we require security without abort. A possible way around would be, as is common in the MPC literature, to restart the protocol upon discovering some cheating or add for each abort a recovery round; this, however, would induce a linear overhead (in the number of parties) on the round complexity of the protocol.

Instead, in order to recover from a misbehavior by corrupted parties, we modify the Setup-Commit-then-Prove functionality and secret-share every committed random string between all the parties, using an error-correcting secret-sharing scheme (aka robust secret sharing). In case a party is identified as cheating, every party broadcasts the share of the committed randomness corresponding to that party, reconstructs the correlated randomness for that party, and locally computes the messages corresponding to this party in every instance of every protocol. We also prove that the modified Setup-Commit-then-Prove functionality can be realized in a constant number of rounds, thus yielding no (asymptotic) overhead on the round complexity of the compiler.

Next, given that using only black-box access to the protocols π_i, it is possible to compile them into a protocol that implements the parallel composition $(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$ of the functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_{\mathtt{M}}$ realized by protocols $\pi_1, \ldots, \pi_{\mathtt{M}}$, we investigate the question of whether there exists a protocol that securely realizes $(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$ given only black-box access to the *functionalities* $\mathcal{F}_1, \ldots, \mathcal{F}_{\mathtt{M}}$,[3] but not to protocols realizing them. This question is only sensible if asked for an entire *class* of functionalities (cf. [52]), since otherwise a protocol may always ignore the functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_{\mathtt{M}}$ and implement $(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$ from scratch.

On the one hand, we prove that against semi-honest corruptions, there indeed exists a protocol for parallel composition of arbitrary functionalities $\mathcal{F}_i$ in a functionally black-box manner. On the other hand, in the case of active corruptions, we devise a class of functionalities for which, when using a generalization of the "batching" technique from [5], such a black-box transformation is not possible in the presence of a single active corrupted party. More precisely, (1) a naïve call to each of the ideal functionalities $\mathcal{F}_i$ until termination will not be round-preserving, (2) it is impossible to compute the parallel composition without calling every ideal functionality, and (3) using the same input value in more than one call to any of the ideal functionalities will break privacy. This negative result validates our choice of a *protocol* compiler, and is evidence that such a task, if at all possible, would require entirely new techniques.

We phrase our results using the framework for composition of protocols with probabilistic termination [16], extending it (a side result of independent interest) to include parallel composition, reactive functionalities (to capture the Setup-Commit-then-Prove functionality), and the higher corruption threshold of $t < n/2$.

[3] Loosely speaking, a functionally black-box protocol, as defined in [52], is a protocol that can compute a function f without knowing the code of f, i.e., given an oracle access to the function f. Note that in this model, each ideal functionality $\mathcal{F}_i$ has an oracle access to the function f_i it computes.

Organization of the paper. The rest of the paper is organized as follows. In Section 2 we describe the network model, the basics of the probabilistic-termination framework by Cohen et al. [16], and other tools that are used throughout the paper. In Section 3 we extend the framework to the honest-majority setting, and in Section 4 define parallel composition of PT protocols. Section 5 presents the protocol that achieves round-preserving parallel composition for arbitrary functionalities in a functionally black-box manner against semi-honest adversaries. Section 6 is dedicated to active corruptions; first, the round-preserving protocol-black-box construction, followed by the negative result on round-preserving functionally black-box composition in the case of active corruptions. For ease of exposition, finer details of the model and PT framework are presented in the appendix.

2 Model and Preliminaries

In the following we introduce some necessary notation and terminology. We use calligraphic letters to denote sets, uppercase for random variables, lowercase for values and boldface for vectors. For $n \in \mathbb{N}$, let $[n] = \{1, \cdots, n\}$. We denote by κ the security parameter. Let poly denote the set all positive polynomials and let PPT denote a probabilistic algorithm that runs in *strictly* polynomial time. A function $\nu \colon \mathbb{N} \to [0,1]$ is *negligible* if $\nu(\kappa) < 1/p(\kappa)$ for every $p \in \mathsf{poly}$ and large enough κ. Given a random variable X, we write $x \leftarrow X$ to indicate that x is selected according to X.

2.1 Synchronous Protocols in UC

We consider synchronous protocols in the model of Katz et al. [43], which is designed on top of the universal composability framework of Canetti [8]. More specifically, we consider n parties $P_1, \ldots, P_n$ and a computationally unbounded, adaptive t-*adversary* that can dynamically corrupt up to t parties during the protocol execution. Synchronous protocols in [43] are protocols that run in a hybrid model where parties have access to a simple "clock" functionality $\mathcal{F}_{\text{CLOCK}}$. This functionality keeps an indicator bit, which is switched once *all honest parties* request the functionality to do so, i.e., once all honest parties have completed their operations for the current round. In addition, all communication is done over bounded-delay secure channels, where each party requests the channel to fetch messages that are sent to him, such that the adversary is allowed to delay the message delivery by a bounded and a priori known number of fetch requests. Stated differently, once the sender has sent some message, it is guaranteed that the message will be delivered within a known number of activations of the receiver. For simplicity, we assume that every message is delivered within a single fetch request. A more detailed overview of [43] can be found in Appendix B.

2.2 The Probabilistic-Termination Framework

Cohen et al. [16] extended the UC framework to capture protocols with probabilistic termination, i.e., protocols without a fixed output round and without simultaneous termination. This section outlines their techniques; additional details can be found in Appendix C.

Canonical synchronous functionalities. The main idea behind modeling probabilistic termination is to separate the functionality to be computed from the round complexity that is required for the computation. The atomic building block in [16] is a functionality template called a *canonical synchronous functionality (CSF)*, which is a simple two-round functionality with explicit (one-round) input and (one-round) output phases. The functionality $\mathcal{F}_{\text{CSF}}$ 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 determines what information about the input values is leaked to the adversary.

$\mathcal{F}_{\text{CSF}}$ proceeds in two rounds: in the first (input) round, all the parties hand $\mathcal{F}_{\text{CSF}}$ their input values, and in the second (output) round, each party receives its output. Whenever some input is submitted to $\mathcal{F}_{\text{CSF}}$, the adversary is handed some leakage function of this input; 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 7 in Appendix C.1. As a side contribution, in Definition C.1, we extend the definition of CSF to the reactive setting.

Wrappers and traces. Computation with probabilistic termination is captured by defining *output-round randomizing wrappers*. Such wrappers address the issue that while an ideal functionality abstractly describes a protocol's task, it does not describe its round complexity. Each wrapper is parametrized by a distribution (more precisely, an efficient probabilistic sampling algorithm) D that may depend on a specific protocol implementing the functionality. The wrapper samples a round $\rho_{\mathsf{term}} \leftarrow D$, by which all parties are guaranteed to receive their outputs. Two wrappers are considered: the first, denoted $\mathcal{W}_{\text{strict}}$, ensures in a strict manner that all (honest) parties terminate together in round ρ_{term}; the second, denoted $\mathcal{W}_{\text{flex}}$, is more flexible and allows the adversary to deliver outputs to individual parties at any time before round ρ_{term}. The detailed descriptions of the two wrappers can be found in Appendix C.3.

As pointed out in [16], it is not sufficient to inform the simulator $\mathcal{S}$ about the round ρ_{term}. In many cases, the wrapper should explain to $\mathcal{S}$ *how* this round was sampled; concretely, the wrapper provides $\mathcal{S}$ with the random coins that are used to sample ρ_{term}. In particular, $\mathcal{S}$ learns the entire *trace* of calls to ideal functionalities that are made by the protocol in order to complete by round ρ_{term}. A trace basically records which hybrids were called by a protocol's execution, and in a recursive way, for each hybrid, which hybrids would have been called by a protocol realizing that hybrid. The recursion ends when the base case is reached, i.e., when the protocol is defined using the *atomic* functionalities that are "assumed" by the model.[4] Formally, a trace is defined as follows:

Definition 2.1 (Traces)**.** *A* trace *is a rooted tree of depth at least* 1*, in which all nodes are labeled by functionalities and where every node's children are* ordered*. The root and all internal nodes are labeled by wrapped CSFs (by either of the two wrappers), and the leaves are labeled by unwrapped CSFs. The* trace complexity *of a trace T, denoted $c_{\mathsf{tr}}(T)$, is the number of leaves in T. Moreover, denote by $\mathsf{flex}_{\mathsf{tr}}(T)$ the number nodes labeled by flexibly wrapped CSFs in T.*

Sequential composition of probabilistic-termination protocols. When a set of parties execute a probabilistic-termination protocol, or equivalently, invoke a flexibly wrapped CSF, they might get out-of-sync and start the next protocol in different rounds. The approach in [16] for dealing with sequential composition is to start by designing simpler protocols, that are in a so-called *synchronous normal form*, where the parties remain in-sync throughout the execution, and next, compile these protocols into slack-tolerant protocols.

Definition 2.2 (Synchronous normal form)**.** *Let $\mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities. A synchronous protocol π in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model is in* synchronous normal form

[4] The atomic functionalities considered in this work are the CSFs for the point-to-point communication functionality $\mathcal{F}_{\text{SMT}}$ and the correlated-randomness functionality for broadcast $\mathcal{F}_{\text{CORR-BC}}$.

(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.*

SNF protocols are designed as an intermediate step only, since the hybrid functionalities $\mathcal{F}_1, \ldots, \mathcal{F}_m$ are two-round CSFs, and, in general, cannot be realized by real-world protocols. In order to obtain protocols that can be realized in the real world, [16] introduced *slack-tolerant* variants of both the strict and the flexible wrappers, denoted $\mathcal{W}_{\text{sl-strict}}$ and $\mathcal{W}_{\text{sl-flex}}$. These wrappers are parametrized by a slack parameter $c \geq 0$ and can be used even if parties provide inputs within $c+1$ consecutive rounds (i.e., they tolerate input slack of c rounds); furthermore, the wrappers ensure that all honest parties obtain output within two consecutive rounds (i.e., they reduce the slack to $c = 1$). The detailed definitions of the slack-tolerant wrappers are given in Appendix C.4. In order to convert SNF protocols into protocols that realize functionalities with slack tolerance, [16] constructed a deterministic-termination compiler $\mathsf{Comp}_{\text{DT}}$, a probabilistic-termination compiler $\mathsf{Comp}_{\text{PT}}$ and a probabilistic-termination with slack-reduction compiler $\mathsf{Comp}_{\text{PTR}}$. Loosely speaking, the composition theorems provide the following guarantees:

1. If an SNF protocol π realizes a wrapped CSF $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, then $\mathsf{Comp}^{c}_{\text{DT}}(\pi)$ realizes $\mathcal{W}^{D',c}_{\text{sl-strict}}(\mathcal{F})$ 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 (where D' is defined using $D, D_1, \ldots, D_m$).

2. If an SNF protocol π realizes a wrapped CSF $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, then $\mathsf{Comp}^{c}_{\text{PT}}(\pi)$ realizes $\mathcal{W}^{D',c}_{\text{sl-flex}}(\mathcal{F})$ in the $(\mathcal{W}^{D_1,c}_{\text{sl}}(\mathcal{F}_1), \ldots, \mathcal{W}^{D_m,c}_{\text{sl}}(\mathcal{F}_m))$-hybrid model (where $\mathcal{W}^{D_i,c}_{\text{sl}}(\mathcal{F}_i)$ is $\mathcal{W}^{D_i,c}_{\text{sl-flex}}(\mathcal{F}_i)$ if $i \in I$ and $\mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$ if $i \notin I$, given a subset $I \subseteq [m]$ (of indices) of functionalities to be wrapped using the flexible wrapper).

3. If an SNF protocol π realizes a wrapped CSF $\mathcal{W}^{D}_{\text{flex}}(\mathcal{F})$ in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, then $\mathsf{Comp}^{c}_{\text{PTR}}(\pi)$ realizes $\mathcal{W}^{D',c}_{\text{sl-flex}}(\mathcal{F})$ in the $(\mathcal{W}^{D_1,c}_{\text{sl}}(\mathcal{F}_1), \ldots, \mathcal{W}^{D_m,c}_{\text{sl}}(\mathcal{F}_m))$-hybrid model (where $\mathcal{W}^{D_i,c}_{\text{sl}}(\mathcal{F}_i)$ is $\mathcal{W}^{D_i,c}_{\text{sl-flex}}(\mathcal{F}_i)$ if $i \in I$ and $\mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$ if $i \notin I$, for a subset I as above).

The compilers maintain the security and the asymptotic (expected) round complexity of the original SNF protocols. At the same time, the compilers take care of any potential slack that is introduced by the protocol and ensure that the resulting protocol can be safely executed even if the parties do not start the protocol simultaneously. More precise descriptions of the compilers can be found in Appendix C.5. As a side contribution, we extend this framework to the honest-majority setting in Appendix 3.

Finally, in [16], the authors also provided protocols for realizing wrapped variants of the atomic CSF functionality for secure point-to-point communication. This suggested the following design paradigm for realizing a wrapped functionality $\mathcal{W}_{\text{sl-strict}}(\mathcal{F})$ (resp., $\mathcal{W}_{\text{sl-flex}}(\mathcal{F})$): First, construct an SNF protocol for realizing $\mathcal{W}_{\text{strict}}(\mathcal{F})$ (resp., $\mathcal{W}_{\text{flex}}(\mathcal{F})$) using CSF hybrids $\mathcal{F}_1, \ldots, \mathcal{F}_m$. Next, for each of the non-atomic hybrids $\mathcal{F}_i$, show how to realize $\mathcal{W}_{\text{strict}}(\mathcal{F}_i)$ (resp., $\mathcal{W}_{\text{flex}}(\mathcal{F}_i)$) using CSF hybrids $\mathcal{F}'_1, \ldots, \mathcal{F}'_{m'}$. Proceed in this manner until all CSF hybrids are atomic functionalities. Finally, repeated applications of the composition theorems above yield a protocol for $\mathcal{W}_{\text{sl-strict}}(\mathcal{F})$ (resp., $\mathcal{W}_{\text{sl-flex}}(\mathcal{F})$) using only atomic functionalities as hybrids.

2.3 A Lemma on Termination Probabilities

The following lemma, which will be used in our positive results, provides a constant lower bound on the probability that when running simultaneously (i.e., in parallel) $\mathtt{N}$ copies of $\mathtt{M}$ probabilistic-termination protocols $\pi_1, \ldots, \pi_{\mathtt{M}}$, at least one copy of each π_i will complete after $\mathtt{R}$ rounds, for suitable choices of $\mathtt{N}$ and $\mathtt{R}$.

Lemma 2.3. *Let $\mathtt{M}, \mathtt{N}, \mathtt{R} \in \mathbb{N}$. For $i \in [\mathtt{M}]$ and $j \in [\mathtt{N}]$, let X_{ij} be independent random variables over the natural numbers, such that $X_{i1}, \ldots, X_{i\mathtt{N}}$ are identically distributed with expectation μ_i, for every $i \in [\mathtt{M}]$. Denote $Y_i = \min\{X_{i1}, \ldots, X_{i\mathtt{N}}\}$ and $\mu = \max\{\mu_1, \ldots, \mu_{\mathtt{M}}\}$.*

Then, for any constant $0 < \epsilon < 1$, if $\mathtt{R} > \mu$ and $\mathtt{N} > \frac{\log(\mathtt{M}/\epsilon)}{\log(\mathtt{R}/\mu)}$, then $\Pr[\forall i : Y_i < \mathtt{R}] \geq 1 - \epsilon$.

Proof. First, using Markov's inequality, notice that

$$\begin{aligned}
\Pr[\exists i \colon Y_i \geq \mathtt{R}] &\leq \sum_{i \in [\mathtt{M}]} \Pr[Y_i \geq \mathtt{R}] \\
&\leq \sum_{i \in [\mathtt{M}]} \left(\prod_{j \in [\mathtt{N}]} \Pr[X_{ij} \geq \mathtt{R}] \right) \\
&\leq \sum_{i \in [\mathtt{M}]} \left(\prod_{j \in [\mathtt{N}]} \left(\frac{\mu_j}{\mathtt{R}} \right) \right) \\
&\leq \sum_{i \in [\mathtt{M}]} \left(\prod_{j \in [\mathtt{N}]} \left(\frac{\mu}{\mathtt{R}} \right) \right) \\
&= \mathtt{M} \cdot \left(\frac{\mu}{\mathtt{R}} \right)^{\mathtt{N}},
\end{aligned}$$

Since $\mathtt{R} > \mu$ it holds that $1 > \mu/\mathtt{R}$, therefore, for constant $0 < \epsilon < 1$, it holds that

$$\Pr[\forall i : Y_i < \mathtt{R}] = 1 - \Pr[\exists i \colon Y_i \geq \mathtt{R}] \geq 1 - \mathtt{M} \cdot \left(\frac{\mu}{\mathtt{R}} \right)^{\mathtt{N}} \geq 1 - \epsilon,$$

which holds for

$$\mathtt{N} > \frac{\log(\mathtt{M}/\epsilon)}{\log(\mathtt{R}/\mu)}.$$

□

3 Probabilistic Termination with an Honest Majority

In this section, we extend the probabilistic-termination framework [16] to the honest majority regime.

3.1 Fast Sequential Composition

The composition theorems from [16] are defined for $t < n/3$. When moving to the honest-majority setting, i.e., $t < n/2$, the compilers and composition theorems follow in a straight-forward way. The main difference lies in the usage of the Bracha-termination technique (cf. Theorem 3.2), where the "termination messages" in the compiled protocol $\pi' = \mathsf{Comp}^{c}_{\text{PTR}}(\pi, D_1, \ldots, D_m, I)$ must be authenticated. Therefore, there is an additional hybrid functionality that generates correlated randomness to be used for authenticating messages with information-theoretic security, e.g., correlated randomness for information-theoretic signatures [49].

– Correlated Randomness. The *correlated-randomness* functionality, parametrized by a distribution D, is defined as follows. The function to compute is $f_{\text{CORR}}(\lambda, \ldots, \lambda, a) = (R_1, \ldots, R_n)$, where $(R_1, \ldots, R_n) \leftarrow D$, and the leakage function is $l_{\text{CORR}}(\lambda, \ldots, \lambda) = \perp$. We denote by $\mathcal{F}^{D}_{\text{CORR}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions f_{CORR} and l_{CORR}. We denote

by $\mathcal{F}_{\text{CORR-BC}}$ the functionality $\mathcal{F}_{\text{CORR}}^{D_{\text{BC}}}$, where D_{BC} is the distribution for correlated randomness needed for information-theoretic broadcast [49].

We state without proof the composition theorems for honest majority. The proofs follow in similar lines to [16].

Theorem 3.1. *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, let $t < n/2$ and let π an SNF protocol that UC-realizes $\mathcal{W}_{\text{strict}}^{D}(\mathcal{F})$, with information-theoretic (resp., computational) security, in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, for some depth-1 distribution D, in the presence of an adaptive, malicious t-adversary, and assuming that all honest parties receive their inputs at the same round. Let $D_1, \ldots, D_m$ be arbitrary distributions over traces, $D^{\mathsf{full}} = \mathsf{full\text{-}trace}(D, D_1, \ldots, D_m)$, and $c \geq 0$.*

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

The expected round complexity of the compiled protocol π' is

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

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

Theorem 3.2. *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, let $t < n/2$ and let π an SNF protocol that UC-realizes $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F})$, with information-theoretic (resp., computational) security, in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, for some depth-1 distribution D, in the presence of an adaptive, malicious t-adversary, and assuming that all honest parties receive their inputs at the same round. Let $I \subseteq [m]$ be the subset (of indices) of functionalities to be wrapped using the flexible wrapper, let $D_1, \ldots, D_m$ be arbitrary distributions over traces, denote $D^{\mathsf{full}} = (D, D_1, \ldots, D_m)$ 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 $\pi' = \mathsf{Comp}_{\text{PTR}}^{c}(\pi, D_1, \ldots, D_m, I)$ UC-realizes $\mathcal{W}_{\text{sl-flex}}^{D^{\mathsf{full}},c}(\mathcal{F})$, with information-theoretic (resp., computational) security, in the $(\mathcal{F}_{\text{CORR-BC}}, \mathcal{W}(\mathcal{F}_1), \ldots, \mathcal{W}(\mathcal{F}_m))$-hybrid model, where $\mathcal{W}(\mathcal{F}_i) = \mathcal{W}_{\text{sl-flex}}^{D_i,c}(\mathcal{F}_i)$ if $i \in I$ and $\mathcal{W}(\mathcal{F}_i) = \mathcal{W}_{\text{sl-strict}}^{D_i,c}(\mathcal{F}_i)$ if $i \notin I$, in the presence of an adaptive, malicious t-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.

The expected round complexity of the compiled protocol π' is

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

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

Theorem 3.3. *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, let $t < n/2$ and let π an SNF protocol that UC-realizes $\mathcal{W}_{\text{strict}}^{D}(\mathcal{F})$, with information-theoretic (resp., computational) security, in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, for some depth-1 distribution D, in the presence of an adaptive, malicious t-adversary, and assuming that all honest parties receive their inputs at the same round. Let $I \subseteq [m]$ be the subset (of indices) of functionalities to be wrapped using the flexible wrapper, let $D_1, \ldots, D_m$ be arbitrary distributions over traces, denote $D^{\mathsf{full}} = \mathsf{full\text{-}trace}(D, D_1, \ldots, D_m)$ and let $c \geq 0$. Assume that $\mathcal{F}$ and $\mathcal{F}_i$, for every $i \in I$, are public-output functionalities.*

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

The expected round complexity of the compiled protocol π' *is*

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

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

3.2 Fast Parallel Broadcast

Cohen et al. [16], based on Hirt and Zikas [36], defined the unfair parallel-broadcast functionality, in which the functionality informs the adversary which messages it received, and allows the adversary, based on this information, to corrupt senders and replace their input messages.

- 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_{UPBC} and l_{UPBC}.

The protocol of Katz and Koo [42] realizes (a wrapped version of) $\mathcal{F}_{\text{UPBC}}$ when the parties have correlated-randomness setup. The following result follows.

Theorem 3.4. *Let* $c \geq 0$ *and* $t < n/2$*. 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* $(\mathcal{F}_{\text{SMT}}, \mathcal{F}_{\text{CORR-BC}})$*-hybrid model, with information-theoretic security, in the presence of an adaptive, malicious* t*-adversary, assuming that all honest parties receive their inputs within* $c+1$ *consecutive rounds.*

We next show how to realize the parallel-broadcast functionality $\mathcal{F}_{\text{PBC}}$ in the $\mathcal{F}_{\text{UPBC}}$-hybrid model, in the honest-majority setting. The construction follows [16], where the only difference is that for $t < n/2$, perfectly correct error-correcting secret sharing (cf. Definition A.1) cannot be achieved, and a negligible error probability is introduced. We describe this protocol, denoted π_{PBC}, in Figure 1.

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

Protocol π_{PBC}

1. In the first round, upon receiving $(\texttt{input}, \mathsf{sid}, x_i)$ with $x_i \in V$ from the environment, P_i secret shares x_i using a (t, n) error-correcting secret sharing scheme, $(x_i^1, \ldots, x_i^n) \leftarrow \mathsf{Share}(x_i)$. Next, P_i sends for every party P_j its share (sid, x_i^j). Denote by $\tilde{x}_j^i$ the value received from P_j.
2. In the second round, P_i broadcasts the values $\boldsymbol{x}_i = (\tilde{x}_1^i, \ldots, \tilde{x}_n^i)$ 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. Next, P_i reconstructs all the input values, i.e., for every $j \in [n]$ computes $y_j = \mathsf{Recon}(y_j^1, \ldots, y_j^n)$, and outputs $(\texttt{output}, \mathsf{sid}, (y_1, \ldots, y_n))$.

Figure 1: The parallel-broadcast protocol, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{UPBC}})$-hybrid model

Theorem 3.5. *Let $c \geq 0$ and $t < n/2$. 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 $(\mathcal{F}_{\text{SMT}}, \mathcal{F}_{\text{CORR-BC}})$-hybrid model, with information-theoretic security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within $c + 1$ consecutive rounds.*

The proof of the theorem follows in the same lines of the proof of [16, Thm. 5.5].

3.3 Fast SFE in the Point-to-Point Model

We conclude this section by showing how to construct a 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 of Cramer et al. [18], denoted π_{SFE}. 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.

Theorem 3.6. *Let f be an n-party function, C an arithmetic circuit with multiplicative depth d computing f, $c \geq 0$ and $t < n/2$. Then, there exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}^f_{\text{SFE}})$ has round complexity $O(d)$ in expectation, and can be UC-realized in the $(\mathcal{F}_{\text{SMT}}, \mathcal{F}_{\text{CORR-BC}})$-hybrid model, with information-theoretic security, in the presence of an adaptive, malicious t-adversary, assuming that all honest parties receive their inputs within $c + 1$ consecutive rounds.*

The following result follows using the protocol of Damgård and Ishai [20].

Theorem 3.7. *Let f be an n-party function, $c \geq 0$, $t < n/2$ and assume that one-way functions exist. Then, there exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}^f_{\text{SFE}})$ has round complexity $O(1)$ in expectation, and can be UC-realized in the $(\mathcal{F}_{\text{SMT}}, \mathcal{F}_{\text{CORR-BC}})$-hybrid model, with computational security, in the presence of an adaptive, malicious t-adversary, assuming that all honest parties receive their inputs within $c + 1$ consecutive rounds.*

4 Functionally Black-Box Protocols and Parallel Composition

In this section, we extend the probabilistic-termination framework [16] to capture the notions of functionally black-box protocols [52] and of parallel composition of canonical synchronous functionalities.

Functionally black-box protocols. We formalize the notion of functionally black-box protocols of Rosulek [52] in the language of canonical synchronous functionalities. As in [52], we focus on secure function evaluation. The SFE functionality $\mathcal{F}_{\text{SFE}}^{g}$ (cf. Section C.1), parametrized by an n-party function g, is defined as the CSF $\mathcal{F}_{\text{CSF}}^{f_{\text{SFE}},l_{\text{SFE}}}$, where $f_{\text{SFE}}(x_1,\ldots,x_n,a) = g(x_1,\ldots,x_n)$ (i.e., computes the function g while ignoring the adversary's input a) and the leakage function is $l_{\text{SFE}}(x_1,\ldots,x_n) = (|x_1|,\ldots,|x_n|)$. The following definition explains what we mean by a protocol that realizes the secure function evaluation functionality in a black-box way with respect to the function g.

Definition 4.1. *Let $\mathcal{C} = \{g\colon (\{0,1\}^*)^n \to (\{0,1\}^*)^n\}$ be a class of n-party functions. Denote by $\mathcal{F}_{\text{SFE}}^{\mathcal{C}}$ the CSF, implemented as an (uninstantiated) oracle machine that in order to compute $f_{\text{SFE}}^{\mathcal{C}}(x_1,\ldots,x_n,a)$, queries the oracle with $(x_1,\ldots,x_n)$ and stores the response $(y_1,\ldots,y_n)$. The leakage function $l_{\text{SFE}}(x_1,\ldots,x_n) = (|x_1|,\ldots,|x_n|)$ is unchanged.*

Then, a protocol $\pi = (\pi_1,\ldots,\pi_n)$ is a functionally black-box (FBB) protocol for (a wrapped version of) $\mathcal{F}_{\text{SFE}}^{\mathcal{C}}$, if for every $f \in \mathcal{C}$, the protocol $\pi^f = (\pi_1^f,\ldots,\pi_n^f)$ UC-realizes $\mathcal{F}_{\text{SFE}}^{f}$.

Parallel Composition of CSFs The parallel composition of CSFs is defined in a natural way as the CSF that evaluates the corresponding functions in parallel.

Definition 4.2. *Let $f_1,\ldots,f_M$ be n-input functions. We define the $(n\cdot M)$-input function $(f_1 \parallel \cdots \parallel f_M)$ as follows. Upon input $(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n)$, where each $\boldsymbol{x}_i$ is an M-tuple $(x_i^1,\ldots,x_i^M)$, the output is the M-tuple defined as*

$$(f_1 \parallel \cdots \parallel f_M)(\boldsymbol{x}_1,\ldots,\boldsymbol{x}_n) = \left((y_1^1,\ldots,y_1^M),\ldots,(y_n^1,\ldots,y_n^M)\right),$$

where $(y_1^j,\ldots,y_n^j) = f_j(x_1^j,\ldots,x_n^j)$.

Let $\mathcal{F}_{\text{CSF}}^{f_1,l_1},\ldots,\mathcal{F}_{\text{CSF}}^{f_M,l_M}$ be CSFs, denote $\mathcal{F}_i = \mathcal{F}_{\text{CSF}}^{f_i,l_i}$. The parallel composition of $\mathcal{F}_1,\ldots,\mathcal{F}_M$, denoted as $(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_M)$, is the CSF defined by the function $(f_1 \parallel \cdots \parallel f_M)$ and the leakage function $(l_1 \parallel \cdots \parallel l_M)$.

5 Round-Preserving Parallel Composition: Passive Security

In this section, we show that round-preserving parallel composition is feasible, in a functionally black-box manner, facing semi-honest adversaries. The underlying idea of our protocol π_{PFBB} (standing for parallel functionally black box), formally presented in Figure 2, is based on a simplified form of the parallel-broadcast protocol of Ben-Or and El-Yaniv [5]. The protocol proceeds in iterations, where in each iteration, the parties invoke, in parallel and using the same input values, sufficiently many instances of each (oracle-aided) ideal functionality, but only for a constant number of rounds. If some party received an output in at least one invocation of every ideal functionality, it distributes all output values and the protocol completes; otherwise, the protocol resumes with another iteration. This protocol retains privacy for deterministic functions,[5] since the adversary is semi-honest, and so corrupted parties will provide the same input values to all instances of each ideal functionality.

Intuitively, during the simulation of the protocol, the simulator should imitate every call for every ideal functionality towards the adversary. A subtle issue is that in order to do so, the simulator must know the exact trace that is sampled by each instance of each ideal functionality during the

[5] Although the result holds for deterministic functionalities, we note that using standard techniques every functionality can be transformed to an equivalent deterministic functionality in a black-box way.

execution of the real protocol. Therefore, it is indeed essential for the simulator to receive the *random coins* used to sample the trace for the entire protocol, by the ideal functionality computing the parallel composition (cf. Section 2.2). By defining the trace-distribution sampler in a way that consists of all (potential) sub-traces for every instance of every ideal functionality, the simulator can induce the exact random coins used to sample the correct sub-trace for every ideal functionality that is invoked.

Theorem 5.1. *Let $\mathcal{C}_1, \ldots, \mathcal{C}_M$ be (deterministic-)function classes, let $\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1}, \ldots, \mathcal{F}_{\text{SFE}}^{\mathcal{C}_M}$ be oracle-aided secure function evaluation functionalities, and let $t < n/2$. Let $D_1, \ldots, D_M$ be distributions, such that for every $j \in [M]$, the round complexity of $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_j})$ has expectation μ_j. Denote $\mu = \max\{\mu_1, \ldots, \mu_M\}$.*

Then, there exists a distribution D with expectation $\mu' = O(\mu)$ such that $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \| \cdots \| \mathcal{F}_{\text{SFE}}^{\mathcal{C}_M})$ can be UC-realized by an FBB protocol in the $(\mathcal{F}_{\text{SMT}}, \mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1}), \ldots, \mathcal{W}_{\text{flex}}^{D_M}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_M}))$-hybrid model, with information-theoretic security, in the presence of an adaptive, semi-honest t-adversary, assuming that all honest parties receive their inputs at the same round.

In particular, if for every $j \in [M]$, the expectation μ_j is constant, then μ' is constant.

Protocol $\pi_{\text{PFBB}}(\mathtt{N}, \mathtt{R}, \mathtt{T})$

1. Each party P_i, upon receiving $(\mathtt{input}, \mathsf{sid}, \boldsymbol{x}_i)$ with $\boldsymbol{x}_i = (x_i^1, \ldots, x_i^{\mathtt{M}})$, initializes the iteration index $\alpha \leftarrow 0$.
2. Initiate (in parallel) $\mathtt{N}$ instances of every ideal functionality $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_j})$ as follows:
   (a) For every $j \in [\mathtt{M}]$ and $k \in [\mathtt{N}]$, send $(\mathtt{input}, \mathsf{sid}_{j,\tilde{k}}, x_i^j)$ to $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_j})$ (where $\tilde{k} = \alpha \cdot \mathtt{N} + k$ and $\mathsf{sid}_{j,\tilde{k}} = \mathsf{sid} \circ j \circ \tilde{k}$).
   (b) Execute for rounds $\rho = 1, \ldots, \mathtt{R}$:
      i. Every party P_i sends $(\mathtt{fetch\text{-}output}, \mathsf{sid}_{j,\tilde{k}})$ to $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_j})$.
      ii. If received $(\mathtt{output}, \mathsf{sid}_{j,\tilde{k}}, y_j)$ with $y_j \neq \perp$, add y_j to the (initially empty) set S_i^j.
   (c) If for every $j \in [\mathtt{M}]$, $S_i^j \neq \emptyset$, then set $\boldsymbol{y} = (y_1, \ldots, y_{\mathtt{M}})$, where $y_j \in S_i^j$ is arbitrarily chosen, and send $(\mathsf{sid}, \boldsymbol{y})$ to all parties.
   (d) If P_i received a value $(\mathsf{sid}, \boldsymbol{y})$ from some party, it outputs $(\mathtt{output}, \mathsf{sid}, \boldsymbol{y})$ and halts. Else, set $\alpha \leftarrow \alpha + 1$. If $\alpha < \mathtt{T}$, goto Step 2; else, output $\perp$ and halt.

Figure 2: FBB parallel composition in the $(\mathcal{F}_{\text{SMT}}, \mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1}), \ldots, \mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}}))$-hybrid model

The proof of Theorem 5.1 follows immediately from the following lemma.

Lemma 5.2. *Let $\mathcal{C}_1, \ldots, \mathcal{C}_M$ be (deterministic-)function classes, let $\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1}, \ldots, \mathcal{F}_{\text{SFE}}^{\mathcal{C}_M}$ be oracle-aided secure function evaluation functionalities, and let $t < n/2$. Let $D_1, \ldots, D_M$ be distributions, such that for every $j \in [M]$, the round complexity of $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_j})$ has expectation μ_j. Denote $\mu = \max\{\mu_1, \ldots, \mu_M\}$ and let $0 < \epsilon < 1$.*

Then, for any $R > \mu, N > \frac{\log(M/\epsilon)}{\log(R/\mu)}$ and $T = \mathsf{poly}(\kappa)$, protocol $\pi_{\text{PFBB}}(N, R, T)$ is an FBB protocol for $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pfbb}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \| \cdots \| \mathcal{F}_{\text{SFE}}^{\mathcal{C}_M})$, for a distribution D_{pfbb} with expectation $\mu_{\mathsf{pfbb}} = O(R)$, in the $(\mathcal{F}_{\text{SMT}}, \mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1}), \ldots, \mathcal{W}_{\text{flex}}^{D_M}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_M}))$-hybrid model, with information-theoretic security, in the presence of an adaptive, semi-honest t-adversary, assuming that all honest parties receive their inputs at the same round.

In particular, if for every $j \in [M]$, the expectation μ_j is constant, then μ_{pfbb} is constant.

Proof. We start by defining the sampling algorithm for the distribution D_{pfbb}, parametrized by $\mathtt{N}, \mathtt{R}, \mathtt{T}$ and distributions $D_1, \ldots, D_{\mathtt{M}}$. The sampler initially sets $\alpha \leftarrow 0$ and a trace T with a root labeled by $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pfbb}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$ and no children. Next, independently sample traces $T_j^{\tilde{k}} \leftarrow D_j$, for $j \in [\mathtt{M}]$ and $k \in [\mathtt{N}]$ (where $\tilde{k} = \alpha \cdot \mathtt{N} + k$), and append

$$(\{(\mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1}))^{\mathtt{N}}, \ldots, (\mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}}))^{\mathtt{N}}\}^{\mathtt{R}}, \mathcal{F}_{\text{PSMT}})$$

to the (initially empty) ordered set of leaves of the trace T (i.e., calling $\mathtt{R}$ times, in parallel, to $\mathtt{N}$ instances of $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_j})$, for every $j \in [\mathtt{M}]$, followed by a call to $\mathcal{F}_{\text{PSMT}}$).[6] If for every $j \in [\mathtt{M}]$, there exists $k \in [\mathtt{N}]$, such that $c_{\mathsf{tr}}(T_j^{\tilde{k}}) < \mathtt{R}$, then output T and halt. Else, set $\alpha \leftarrow \alpha + 1$. If $\alpha < \mathtt{T}$, repeat the sampling process; otherwise output T and halt.

Following Lemma 2.3, for $\mathtt{R} > \mu$ and $\mathtt{N} > \frac{\log(\mathtt{M}/c)}{\log(\mathtt{R}/\mu)}$, it holds that in every iteration, at least one invocation of $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_j})$ will produce output, for every $j \in [\mathtt{M}]$, with a constant probability. It follows that the expected number of iterations until all honest parties receive output and the protocol terminates is constant, and since each iteration consists of $O(\mathtt{R})$ rounds, the entire execution completes within $O(\mathtt{R})$ rounds in expectation, as required. The failure probability that the protocol will not terminate within $\mathtt{T} = \mathsf{poly}(\kappa)$ iterations is negligible.

We now construct a simulator $\mathcal{S}$ for the dummy adversary $\mathcal{A}$ in the semi-honest setting. Initially, $\mathcal{S}$ sets the values $\alpha \leftarrow 0$ and $\boldsymbol{y} \leftarrow \perp$, and starts by receiving leakage messages $(\mathtt{leakage}, \mathsf{sid}, P_i, (l_1, \ldots, l_{\mathtt{M}}))$ and a trace message $(\mathtt{trace}, \mathsf{sid}, T)$ from $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pfbb}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$, where T is a depth-1 trace of the form

$$(\{\{(\mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1}))^{\mathtt{N}}, \ldots, (\mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}}))^{\mathtt{N}}\}^{\mathtt{R}}, \mathcal{F}_{\text{PSMT}}\}^{q})$$

(i.e., q iterations of calling $\mathtt{R}$ times, in parallel, to $\mathtt{N}$ instances of $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_j})$, for every $j \in [\mathtt{M}]$, followed by a call to $\mathcal{F}_{\text{PSMT}}$). More precisely, $\mathcal{S}$ receives the coins that were used by the functionality $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pfbb}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$ to sample the trace T and using these coins, $\mathcal{S}$ samples the same traces $T_j^{\tilde{k}}$ that were used to define T.

In order to simulate the α'th iteration, $\mathcal{S}$ sends the message $(\mathtt{leakage}, \mathsf{sid}_{j,\tilde{k}}, P_i, l_j)$ to $\mathcal{A}$, for every $j \in [\mathtt{M}]$, $k \in [\mathtt{N}]$ and honest P_i (where $\tilde{k} = \alpha \cdot \mathtt{N} + k$), and receives $(\mathtt{input}, \mathsf{sid}_{j,\tilde{k}}, x_i^j)$ from $\mathcal{A}$ on behalf of every corrupted party P_i. Since $\mathcal{A}$ is semi-honset, it holds that the same x_i^j is used for each corrupted party P_i in all instances of the functionality $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_j})$. In the first iteration, $\mathcal{S}$ sends to $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pfbb}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$, on behalf of every corrupted party P_i, the message $(\mathtt{input}, \mathsf{sid}, \boldsymbol{x}_i)$, with $\boldsymbol{x}_i = (x_i^1, \ldots, x_i^{\mathtt{M}})$. When $\mathcal{A}$ sends $(\mathtt{fetch\text{-}output}, \mathsf{sid}_{j,\tilde{k}})$ requests in round ρ, $\mathcal{S}$ answers with $\perp$ if $c_{\mathsf{tr}}(T_j^{\tilde{k}}) < \rho$, and with $(\mathtt{output}, \mathsf{sid}_{j,\tilde{k}}, y_j)$ otherwise, where $\boldsymbol{y} = (y_1, \ldots, y_{\mathtt{M}})$; if $\boldsymbol{y} = \perp$ (i.e., on the first time), send $(\mathtt{early\text{-}output}, \mathsf{sid}, P_i)$ to $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pfbb}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$, receive $(\mathtt{output}, \mathsf{sid}, \boldsymbol{y})$ and store $\boldsymbol{y} = (y_1, \ldots, y_{\mathtt{M}})$. In case $\mathcal{A}$ sends $(\mathtt{early\text{-}output}, \mathsf{sid}_{j,\tilde{k}}, P_i)$, for some corrupted P_i, send $(\mathtt{output}, \mathsf{sid}_{j,\tilde{k}}, y_j)$ to $\mathcal{A}$. If the simulated protocol completes during the α'th iteration, $\mathcal{S}$ sends $(\mathtt{early\text{-}output}, \mathsf{sid}, P_i)$ to the functionality $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pfbb}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$ on behalf of every party.

Proving that no environment can distinguish between its view in an execution of π_{PFBB} in the $(\mathcal{F}_{\text{SMT}}, \mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1}), \ldots, \mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}}))$-hybrid model, and its view when interacting with $\mathcal{S}$ in the ideal

[6] In fact, we consider here an augmented definition of traces. Specifically, a trace is augmented with another (potentially empty) layer of nodes, such that each leaf, in the original definition of a trace, may have (unordered) children. In the example above, each iteration corresponds to $\mathtt{R} + 1$ "leaves", where the first $\mathtt{R}$ "leaves" have $\mathtt{M} \cdot \mathtt{N}$ children. The trace complexity is defined as in Definition 2.1, as the number of "leaves". We note that the composition theorems trivially extend to use this augmented definition of a trace.

computation of $\mathcal{W}_{\text{flex}}^{D_{\text{pfbb}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$, follows via a standard hybrid argument. Starting with the execution of π_{PFBB}, the invocations of the ideal functionalities $\mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1}), \ldots, \mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$ are replaced, one-by-one, with the answers of the simulator $\mathcal{S}$. Since $\mathcal{S}$ perfectly emulates each such call using the trace it received from the ideal functionality $\mathcal{W}_{\text{flex}}^{D_{\text{pfbb}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$, it follows immediately that the views of the environment in two neighbouring hybrids are indistinguishable, therefore, the view of the environment in the simulation is indistinguishable from its view in the execution of π_{PFBB}. □

6 Round-Preserving Parallel Composition: Active Security

In this section, we consider security against active adversaries. First, in Section 6.1, we show how to compute the parallel composition of probabilistic-termination functionalities, in a round-preserving manner, using a black-box access to *protocols* realizing the individual functionalities. In Section 6.2, we investigate the question of whether there exists a *functionally* black-box round-preserving malicious protocol for the parallel composition of probabilistic functionalities, and show that for a natural extension of protocols, following the techniques from [5], this is not the case—i.e., there exist functions such that no such protocol with black-box access to them can compute their parallel composition, in a round-preserving manner, tolerating even a single adversarial party.

6.1 Feasibility of Round-Preserving Parallel Composition

In this section, we show how to compile multiple protocols, realizing probabilistic-termination functionalities, into a single protocol that realizes the parallel composition of the functionalities, in a round-preserving manner, and while only using black-box access to the underlying protocols. We start by providing a high-level description of the compiler.

The compiler receives as input protocols $\pi_1, \ldots, \pi_{\mathtt{M}}$, where each protocol π_j is defined in the point-to-point model, in which the parties are given correlated randomness in a secure setup phase, i.e., in the $(\mathcal{F}_{\text{SMT}}, \mathcal{F}_{\text{CORR}}^{D_j^{\text{corr}}})$-hybrid model.[7] It follows that the next-message function for each party in each protocol is a deterministic function that receives the input value, correlated randomness, private randomness and history of incoming messages, and outputs a vector of n messages to be sent in the following round (one message for each party); we denote by $f_{\mathsf{nxt\text{-}msg}}^{\pi_j,i}$ the next-message function for party P_i in protocol π_j. In particular, we note that the entire transcript of the protocol is fixed once the input value, correlated randomness and private randomness of each party are determined.

The underlying ideas of the compiler are inspired by the constructions in [5, 16], where a round-preserving parallel-broadcast protocol was constructed by iteratively running, for a constant number of rounds, multiple instances of BA protocols (each instance is executed multiple times in parallel), until at least one execution of every BA instance is completed. This approach is indeed suitable for computing "privacy-free" functionalities (such as broadcast), where the adversary may learn the result of multiple computations using the same inputs for honest parties, without compromising security. However, when considering the parallel composition of arbitrary functions, running two instances of a protocol *using the same input values* will violate privacy, since the adversary can use different inputs to learn multiple outputs of the function.

The parallel-composition compiler generalizes the above approach in a privacy-preserving manner. The compiler follows the GMW paradigm [30] and is defined in the Setup-Commit-then-Prove

[7] This captures, for example, broadcast-model protocols, where the broadcast channel is realized using an expected-constant-round protocol, cf. Theorems 3.6 and 3.7.

hybrid model [11, 39], which generates committed correlated randomness for the parties and ensures that all parties follow the protocol specification. This mechanism allows each party to commit to its input values and later execute multiple instances of each protocol, while proving that the same input value is used in all executions. For simplicity and without loss of generality, we assume that each function is deterministic and has a public output. In this case it is ensured that if two parties receive output values in two executions of π_j, then they receive the same output value. The private random coins that are used in each execution only affect the termination round, but not the output value. Using this simplification, we can remove the leader-election phase from the output-agreement technique in [5, 16] and directly use the termination technique from Bracha [7].

Another obstacle is to recover from corruptions without increasing the round complexity. Indeed, in case some party misbehaves, e.g., by using different input values in different instances of the same protocol π_j, then the Setup-Commit-then-Prove functionality ensures that all honest parties will identify the cheating party. In this case, the parties cannot recover by, for example, backtracking and simulating the cheating party, as this will yield a round complexity that is linear in the number of parties. Furthermore, the protocol must resume in a way such that all instances of a specific protocol π_j will use the same input value that the identified corrupted party used throughout the protocol's execution until it misbehaved (since the cheating party might have learned an output value in one of the executed protocols).

To this end, we slightly adjust the Setup-Commit-then-Prove functionality and secret-share every committed random string r_i (the correlated randomness for party P_i) among all the parties, using an error-correcting secret-sharing scheme (cf. Section A.1). Note that this can be done information theoretically as we assume an honest majority [51, 19, 12]. In case a party is identified as cheating, every party broadcasts the share of the committed randomness corresponding to that party, reconstructs the correlated randomness for that party, and from that point onwards, locally computes the messages corresponding to this party in every instance of every protocol. Using this approach, every round in the original protocols $\pi_1, \ldots, \pi_{\mathtt{M}}$ is expanded by a constant number of rounds, and the overall round complexity is preserved.

We prove the following theorem.

Theorem 6.1. *Let $\mathcal{F}_1, \ldots, \mathcal{F}_M$ be CSFs, let $t < n/2$, and let $c \geq 1$. Let $\pi_1, \ldots, \pi_M$ be SNF protocols such that for every $j \in [\bar{M}]$, protocol π_j UC-realizes $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_j)$ with expected round complexity μ_j and information-theoretic security, in the $(\mathcal{F}_{\text{SMT}}, \mathcal{F}_{\text{CORR}}^{D_j^{\text{corr}}})$-hybrid model (for a distribution D_j and a distribution D_j^{corr} in NC^0), in the presence of an adaptive, malicious t-adversary, assuming that all honest parties receive their inputs at the same round. Denote $\mu = \max\{\mu_1, \ldots, \mu_M\}$.*

Then, $\mathcal{W}_{\text{sl-flex}}^{D,c}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_M)$, for some distribution D with expectation $\mu' = O(\mu)$, can be UC-realized with information-theoretic security by a protocol π in the $(\mathcal{F}_{\text{SMT}}, \mathcal{F}_{\text{CORR-BC}})$-hybrid model, in the same adversarial setting, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds. In addition, protocol π requires only black-box access to the protocols π_j.

In particular, if for every $j \in [\bar{M}]$, μ_j is constant, then D has constant expectation.

Proof (sketch). Without loss of generality, we assume that every CSF $\mathcal{F}_j$ is deterministic and has public output. The proof for randomized functionalities with private output follows using standard techniques. In Lemma 6.3, we prove that the compiled protocol $\pi_{\text{PBB}} = \mathsf{Comp}(\pi_1, \ldots, \pi_{\mathtt{M}}, \mathtt{N}, \mathtt{R}, \mathtt{T})$, for $\mathtt{R} > \mu, \mathtt{N} > \frac{\log(\mathtt{M}/\epsilon)}{\log(\mathtt{R}/\mu)}$ and $\mathtt{T} = \mathsf{poly}(\kappa)$, UC-realizes $\mathcal{W}_{\text{flex}}^{D_{\text{pbb}}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$ with expected round complexity $O(\mathtt{R})$ and information-theoretic security, in the $(\mathcal{F}_{\text{PBC}}, \mathcal{F}_{\text{SCP}})$-hybrid model. In Lemma 6.2, we show that a wrapped version of $\mathcal{F}_{\text{SCP}}(\mathcal{P}, \mathcal{D}_{\mathsf{parallel}}(\pi_1, \ldots, \pi_{\mathtt{M}}, \mathtt{N} \cdot \mathtt{T}, \mathtt{R}, \ell), \vec{\mathcal{R}}_{\mathsf{parallel}}, \Pi)$ can be implemented, such that every call can be UC-realized in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{PBC}})$-hybrid model with constant round complexity and information-theoretic security. Following Theorem 3.5, $\mathcal{F}_{\text{PBC}}$ can be

UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model with expected constant round complexity and information-theoretic security. The proof follows from the sequential composition theorems, Theorems 3.1, 3.2 and 3.3. □

6.1.1 The Setup-Commit-Then-Prove Functionality

An important building block in our parallel-composition compiler is the Setup-Commit-then-Prove functionality. This functionality is used in order to allow parties to execute multiple instances of a protocol, using the same inputs, while ensuring input consistency. In addition, in case a party misbehaves and tries to deviate from the protocol or to use different inputs in different executions, the functionality allows all parties to identify this misbehaviour and recover, while increasing the round complexity only by a constant factor. This functionality was defined by Ishai et al. [39], based on the Commit-then-Prove functionality of Canetti et al. [11], and was used in order to compile any semi-honest protocol into a protocol that is secure with identifiable abort, facing malicious adversaries, by generating committed setup for the parties and allow them to prove NP-statements in zero knowledge.

Functionality $\mathcal{F}_{\text{SCP}}(\mathcal{P}, \mathcal{D}, \vec{\mathcal{R}} = (\mathcal{R}_1, \ldots, \mathcal{R}_n), \Pi))$

The functionality $\mathcal{F}_{\text{SCP}}$ is a reactive CSF, with vectors $\boldsymbol{f}_{\text{SCP}} = (f^1_{\text{SCP}}, \ldots, f^q_{\text{SCP}})$ and $\boldsymbol{l}_{\text{SCP}} = (l^1_{\text{SCP}}, \ldots, l^q_{\text{SCP}})$ of functions. $\mathcal{F}_{\text{SCP}}$ is parametrized by party-set $\mathcal{P} = \{P_1, \ldots, P_n\}$, a distribution $\mathcal{D}$, a vector $\vec{\mathcal{R}}$ of NP relations and a (t, n) error-correcting secret-sharing scheme $\Pi = (\mathsf{Share}, \mathsf{Recon})$.

Setup-Commit Phase: The function $f^1_{\text{SCP}}(x_{1,1}, \ldots, x_{n,1}, a_1)$ is defined as follows. Initially, sample $(r_1, \ldots, r_n) \leftarrow \mathcal{D}$ and for every $i \in [n]$, compute $(v^1_i, \ldots, v^n_i) \leftarrow \mathsf{Share}(r_i)$. The output for party P_i is $y_{i,1} = (r_i, v^i_1 \ldots, v^i_n)$. The leakage function is $l^1_{\text{SCP}}(x_{1,1}, \ldots, x_{n,1}) = \lambda$.

Prove Phase: For $k > 1$, the function $f^k_{\text{SCP}}(\boldsymbol{s}_{k-1}, x_{1,k}, \ldots, x_{n,k}, a_k)$ is defined as follows. For every $i \in [n]$, check if the relation $\mathcal{R}_i(x_{i,k}, r_i) = 1$. If so, set $b_i = 1$, else set $b_i = 0$. The output for every party P_i is $y_{i,k} = ((x_{1,k}, b_1) \ldots, (x_{n,k}, b_n))$. The leakage function is $l^k_{\text{SCP}}(x_{1,1}, \ldots, x_{n,k}) = (x_{1,k}, \ldots, x_{n,k})$.

Figure 3: The Setup-Commit-then-Prove functionality

The Setup-Commit-then-Prove functionality $\mathcal{F}_{\text{SCP}}$ is a reactive functionality. (The notion of CSF is extended to the reactive setting in Appendix C.2.) The functionality is formally defined in Figure 3 and is parametrized by a party-set $\mathcal{P}$, a distribution $\mathcal{D}$, a vector of n NP-relations $\vec{\mathcal{R}}$ and a (t, n) error-correcting secret-sharing scheme Π. In the first call to the functionality, the parties don't send inputs (more precisely, send λ as input). The functionality samples correlated randomness $(r_1, \ldots, r_n) \leftarrow \mathcal{D}$, hands r_i to P_i and stores r_i as the committed witness for P_i. In addition, the functionality secret shares r_i between all parties. All subsequent calls are used to prove NP-statements on the committed witnesses, i.e., on the k'th call, for $k > 1$, P_i sends as its input a statement $x_{i,k}$; the functionality verifies whether $\mathcal{R}_i(x_{i,k}, r_i) = 1$ and sends $x_{i,k}$ and the result to all parties.

Ishai et al. [39] showed how to realize a slightly reduced version of the Setup-Commit-then-Prove functionality (in which the functionality does not secret share the witnesses), unconditionally, with identifiable abort, facing an arbitrary number of corrupted parties. Here we adjust and simplify the protocol from [39] for the honest-majority setting, and show how to realize $\mathcal{F}_{\text{SCP}}$ without abort.

Lemma 6.2. *Let $\vec{\mathcal{R}} = (\mathcal{R}_1, \ldots, \mathcal{R}_n)$ be a vector of NP-relations, let $\mathcal{D}$ be an efficiently sampleable distribution in NC^0 and let $t < n/2$. Consider the representation of the reactive CSF, $\mathcal{F}_{\text{SCP}}(\mathcal{P}, \mathcal{D}, \vec{\mathcal{R}}, \Pi)$, as a sequence of (non-reactive) CSFs $(\mathcal{F}_{\text{CSF}}^{\tilde{f}^1_{\text{SCP}}, l^1_{\text{SCP}}}, \ldots, \mathcal{F}_{\text{CSF}}^{\tilde{f}^q_{\text{SCP}}, l^q_{\text{SCP}}})$, where $\tilde{f}^j_{\text{SCP}}$ is defined as in Definition C.1. Denote $\mathcal{F}^j_{\text{SCP}} = \mathcal{F}_{\text{CSF}}^{\tilde{f}^j_{\text{SCP}}, l^j_{\text{SCP}}}$.*

Then, there exists a vector of distributions $\boldsymbol{D} = (D_1, \ldots, D_q)$ such that for every $j \in [q]$, the wrapped functionality $\mathcal{W}^{D_j}_{\text{strict}}(\mathcal{F}^j_{\text{SCP}})$ can be UC-realized in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{PBC}})$-hybrid model with constant round complexity and information-theoretic security, in the presence of an adaptive, malicious t-adversary.

Proof (sketch). We start with a high-level description of the protocol in [39], for a single prover that proves a single statement (the extension to the multi-instance version of many provers that prove many statements follows via the JUC theorem [9]). The protocol follows the "MPC-in-the-head" approach [37, 38], where the prover emulates in its head a protocol, where m servers, each has as input a share of the witness ω, compute over a public statement x the function $b = \mathcal{R}(x, \omega)$, and output (x, b). The prover publicly commits to the view of each server, and the verifiers challenge the prover to open the views of some of the servers. The verifiers accept the statement x, if and only if all opened views are consistent.

More specifically, in the setup-commit phase, the parties receive the following correlated randomness:

- The prover receives signed secret shares of the witness $(\omega_i, \sigma(\omega_i))$, for $i \in [n]$, where $(\omega_1, \ldots, \omega_n)$ are shares of the prover's witness ω, and $\sigma(\omega_i)$ is an information-theoretic signature of ω_i.
- The prover also receives random strings $v_1, \ldots, v_m$ along with corresponding signatures $\sigma(v_1), \ldots, \sigma(v_m)$, that will be used for committing to the server's views in the m-party protocol.
- Every party P_i receives a challenge string c_i along with an information-theoretic signature $\sigma(c_i)$.
- Every party P_i receives a verification key for each of the signature values (note that all of the signing keys are hidden from the parties).

The prove phase, consists of three rounds:

1. The prover emulates in its head the protocol computing $b = \mathcal{R}(x, \omega)$ and broadcasts, for every $j \in [m]$, a commitment to the view of the j'th server as $\text{VIEW}_j \oplus v_j$ along with $\sigma(v_j)$.
2. Every party P_i broadcasts the committed random string c_i along with its commitment $\sigma(c_i)$. All parties locally compute $c = \sum c_j$ and use it to choose a subset $\mathcal{J} \subseteq [m]$. In case some party P_i sends invalid values, all parties identify P_i as corrupted and abort.
3. The prover broadcasts VIEW_j and $\sigma(\omega_j)$, for every $j \in \mathcal{J}$, and all parties validate consistency.

Since we consider an honest majority, we can simplify the protocol and achieve security without abort. In the second round, instead of broadcasting committed randomness, the parties jointly compute the XOR function, where each party enters a (locally chosen) random string as its input. Since this functionality can be represented using a constant-depth circuit, we can use the protocol of Cramer et al. [18], whose round complexity is $O(d)$, where d is the depth of the circuit, and provides information-theoretic security in the broadcast-hybrid model. A second modification we

require is to secret share the witness ω between all parties using an error-correcting secret-sharing scheme, which can be easily achieved in the honest-majority setting.

It is left to show that the sampling algorithm for the correlated randomness that is used in [39] can be represented using a constant-depth circuit; the proof will then follow using the protocol from [18]. By assumption, sampling a value from the distribution $\mathcal{D}$ can be done using a constant-depth circuit. In addition, the information-theoretic commitments in [39] are computed using information-theoretic signatures (cf. Section A.2), such that *no party knows the signing key*. This means that the adversary cannot generate signatures on its own. In addition, each signature will only need to verified once (by each party). Using the information-theoretic signatures construction from [55] (cf. Theorem A.3), the degree of the polynomial, that is used to generate the signature, is bounded by the number of signatures the adversary is allowed to see from each honest party, which in our case is constant. We conclude that the randomness-sampling algorithm can be represented using a constant-depth circuit. □

In the following, we will consider the distribution $\mathcal{D}_{\mathsf{parallel}}(\pi_1, \ldots, \pi_{\mathtt{M}}, q, \mathtt{R}, \ell)$ for protocols $\pi_1, \ldots, \pi_{\mathtt{M}}$, where each π_j is defined in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{CORR}}^{D_j^{\mathsf{corr}}})$-hybrid model, that prepares (at most) q executions of each protocol, for exactly $\mathtt{R}$ rounds. For each of the q instances of every protocol π_j, the correlated randomness consists of three parts: first, sample correlated randomness for π_j from the distribution D_j^{corr}; next, sample independent random coins (local for each party) from the corresponding distribution D_{π_j}, suitable for $\mathtt{R}$ rounds; finally, sample random coins that are used to mask all the communication (explained below). The parameter $\ell = \mathsf{poly}(\kappa)$ represents the maximum between the input length and the maximal message length in the protocols $\pi_1, \ldots, \pi_{\mathtt{M}}$. The distribution is formally defined in Figure 4.

Distribution $\mathcal{D}_{\mathsf{parallel}}(\pi_1, \ldots, \pi_{\mathtt{M}}, q, \mathtt{R}, \ell)$

The distribution $\mathcal{D}_{\mathsf{parallel}}$ is parametrized by protocols $\pi_1, \ldots, \pi_{\mathtt{M}}$, where each π_j is defined in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{CORR}}^{D_j^{\mathsf{corr}}})$-hybrid model, and integers $q, \mathtt{R}, \ell$. Denote by $D_{\pi_j}(\mathtt{R})$ the distribution that samples independent random coins for each party for $\mathtt{R}$ rounds of protocol π_j.

1. For every $j \in [\mathtt{M}]$:
   (a) For every $k \in [q]$:
      i. Sample correlated randomness for the k'th instance of protocol π_j, $(r^{\mathsf{corr}}_{1,j,k}, \ldots, r^{\mathsf{corr}}_{n,j,k}) \leftarrow D_j^{\mathsf{corr}}$.
      ii. Sample independent random coins for $\mathtt{R}$ rounds in the k'th instance of protocol π_j, $(r^{\mathsf{prot}}_{1,j,k}, \ldots, r^{\mathsf{prot}}_{n,j,k}) \leftarrow D_{\pi_j}(\mathtt{R})$.
      iii. Sample randomness $(r^{\mathsf{mask}}_{1,j,k}, \ldots, r^{\mathsf{mask}}_{n,j,k})$ to mask the communication in the k'th instance of π_j, where for every $i \in [n]$, $r^{\mathsf{mask}}_{i,j,k}$ is set as follows:
         A. For every $\rho \in [\mathtt{R}]$ and $u \in [n]$, sample $r^{\mathsf{mask}}_{i,j,k,\rho,u} \leftarrow \{0,1\}^\ell$ (used to mask the message from P_i to P_u in the ρ'th round of the k'th execution of π_j),
         B. For every $\rho \in [\mathtt{R}]$, set $r^{\mathsf{mask}}_{i,j,k,\rho}$ as $(r^{\mathsf{mask}}_{i,j,k,\rho,1}, \ldots, r^{\mathsf{mask}}_{i,j,k,\rho,n})$ and $(r^{\mathsf{mask}}_{1,j,k,\rho,i}, \ldots, r^{\mathsf{mask}}_{n,j,k,\rho,i})$.
         C. Set $r^{\mathsf{mask}}_{i,j,k} = (r^{\mathsf{mask}}_{i,j,k,1}, \ldots, r^{\mathsf{mask}}_{i,j,k,\mathtt{R}})$.
   (b) For every $i \in [n]$, denote the random coins as $r^{\mathsf{corr}}_{i,j} = (r^{\mathsf{corr}}_{i,j,1}, \ldots, r^{\mathsf{corr}}_{i,j,q})$, $r^{\mathsf{prot}}_{i,j} = (r^{\mathsf{prot}}_{i,j,1}, \ldots, r^{\mathsf{prot}}_{i,j,q})$ and $r^{\mathsf{mask}}_{i,j} = (r^{\mathsf{mask}}_{i,j,1}, \ldots, r^{\mathsf{mask}}_{i,j,q})$; in addition, denote $\boldsymbol{r}_i^j = (r^{\mathsf{corr}}_{i,j}, r^{\mathsf{prot}}_{i,j}, r^{\mathsf{mask}}_{i,j})$.
2. For every $i \in [n]$, sample randomness $\boldsymbol{r}_i^{\mathsf{input}} \leftarrow (\{0,1\}^\ell)^{\mathtt{M}}$ to mask the input value and denote $\boldsymbol{r}_i = (\boldsymbol{r}_i^{\mathsf{input}}, \boldsymbol{r}_i^1, \ldots, \boldsymbol{r}_i^{\mathtt{M}})$.
3. Return $(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n)$.

Figure 4: The correlated-randomness distribution for parallel composition

The masking of the communication in the ρ'th round of the k'th instance of protocol π_j is performed as follows:

- When P_i wants to send messages $(m_i^1, \ldots, m_i^n)$ (where m_i^u is sent privately to P_u), party P_i sets for every $u \in [n]$, $\tilde{m}_i^u = m_i^u \oplus r^{\mathsf{mask}}_{i,j,k,\rho,u}$ and broadcasts $\tilde{\boldsymbol{m}}_i = (\tilde{m}_i^1, \ldots, \tilde{m}_i^n)$.
- When P_i receives messages $\tilde{\boldsymbol{m}}_u = (\tilde{m}_u^1, \ldots, \tilde{m}_u^n)$ from P_u, P_i computes $m_u^i = \tilde{m}_u^i \oplus r^{\mathsf{mask}}_{u,j,k,\rho,i}$, and uses m_u^i as the message P_u sent him.

The vector of relations $\vec{\mathcal{R}}_{\mathsf{parallel}} = (\mathcal{R}^1_{\mathsf{parallel}}, \ldots, \mathcal{R}^n_{\mathsf{parallel}})$, described below, will be used to verify that every party sends its messages in multiple executions of a protocol according to the specification of the protocol, while using the same input value in all executions. Formally, for every $i \in [n]$, the relation $\mathcal{R}^i_{\mathsf{parallel}}$ consists of pairs $((\alpha, \rho, \tilde{\boldsymbol{m}}, \tilde{\boldsymbol{h}}), \boldsymbol{r}_i)$, satisfying:

- α is an integer representing the iteration number.
- ρ is an integer representing the round number.
- The vector of messages $\tilde{\boldsymbol{m}} = (\tilde{\boldsymbol{m}}_{i,1,\rho}, \ldots, \tilde{\boldsymbol{m}}_{i,\mathtt{M},\rho})$ is structured such that for every $j \in [\mathtt{M}]$, $\tilde{\boldsymbol{m}}_{i,j,\rho} = (\tilde{\boldsymbol{m}}_{i,j,1,\rho}, \ldots, \tilde{\boldsymbol{m}}_{i,j,\mathtt{N},\rho})$, and for every $k \in [\mathtt{N}]$, $\tilde{\boldsymbol{m}}_{i,j,k,\rho} = (\tilde{m}_{i,j,k,\rho,1}, \ldots, \tilde{m}_{i,j,k,\rho,n})$ represents the message P_i broadcasts in the ρ'th round of the k'th execution of protocol π_j, in the α'th iteration.

- The vector of history-messages $\tilde{\boldsymbol{h}} = (\tilde{\boldsymbol{h}}^{\mathsf{input}}, \tilde{\boldsymbol{h}}_1, \ldots, \tilde{\boldsymbol{h}}_{\mathtt{M}})$ is structured such that $\tilde{\boldsymbol{h}}^{\mathsf{input}} = (\tilde{\boldsymbol{m}}_1^{\mathsf{input}}, \ldots, \tilde{\boldsymbol{m}}_n^{\mathsf{input}})$ and for $j \in [\mathtt{M}]$, $\tilde{\boldsymbol{h}}_j = (\tilde{h}_{j,1}, \ldots, \tilde{h}_{j,\mathtt{N}})$, and each $\tilde{h}_{j,k}$ represents the history-transcript until the ρ'th round of the k'th execution of protocol π_j, in the α'th iteration.
- The random coins are $\boldsymbol{r}_i = (\boldsymbol{r}_i^{\mathsf{input}}, \boldsymbol{r}_i^1, \ldots, \boldsymbol{r}_i^{\mathtt{M}})$, with $\boldsymbol{r}_i^j = (r_{i,j}^{\mathsf{corr}}, r_{i,j}^{\mathsf{prot}}, r_{i,j}^{\mathsf{mask}})$, as defined in $\mathcal{D}_{\mathsf{parallel}}$.
- For every $j \in [\mathtt{M}]$ and $k \in [\mathtt{N}]$, denote by $\boldsymbol{m}_{i,j,k,\rho} = (m_{i,j,k,\rho,1}, \ldots, m_{i,j,k,\rho,n})$ the unmasked messages, where for every $u \in [n]$, $m_{i,j,k,\rho,u} = \tilde{m}_{i,j,k,\rho,u} \oplus r_{i,\tilde{k},j,\rho,u}^{\mathsf{mask}}$, with $\tilde{k} = \alpha \cdot \mathtt{N} + k$. Similarly, denote by $h_{i,j,k}$ the unmasked history obtained from $\tilde{h}_{i,j,k}$ using the corresponding randomness in $r_{i,j,\tilde{k}}^{\mathsf{mask}}$. Denote $(x_i^1 \ldots, x_i^{\mathtt{M}}) = \tilde{\boldsymbol{m}}_i^{\mathsf{input}} \oplus \boldsymbol{r}_i^{\mathsf{input}}$. Then, for every j and k, it holds that

$$\boldsymbol{m}_{i,j,k,\rho} = f_{\mathsf{nxt\text{-}msg}}^{\pi_j,i}(x_i^j, r_{i,j,\tilde{k}}^{\mathsf{corr}}, r_{i,j,\tilde{k}}^{\mathsf{prot}}, h_{i,j,k}).$$

  That is, $\boldsymbol{m}_{i,j,k,\rho}$ is the output of the next-message function of P_i in protocol π_j on input x_i^j, correlated randomness $r_{i,j,\tilde{k}}^{\mathsf{corr}}$, private randomness $r_{i,j,\tilde{k}}^{\mathsf{prot}}$ and history $h_{i,j,k}$.

In the sequel, for simplicity, we will denote by $\mathcal{F}_{\mathrm{SCP}}$ the functionality

$$\mathcal{F}_{\mathrm{SCP}}(\mathcal{P}, \mathcal{D}_{\mathsf{parallel}}(\pi_1, \ldots, \pi_{\mathtt{M}}, \mathtt{N} \cdot \mathtt{T}, \mathtt{R}, \ell), \vec{\mathcal{R}}_{\mathsf{parallel}}, \Pi).$$

6.1.2 Round-Preserving Parallel-Composition Compiler

We are now ready to present our protocol-black-box (PBB) parallel-composition compiler, formally described in Figure 5.

Lemma 6.3. *Let $\mathcal{F}_1, \ldots, \mathcal{F}_M$ be deterministic CSFs with public output and let $t < n/2$. Let $\pi_1, \ldots, \pi_M$ be SNF protocols such that for every $j \in [M]$, protocol π_j UC-realizes $\mathcal{W}_{\mathrm{flex}}^{D_j}(\mathcal{F}_j)$ with expected round complexity μ_j and information-theoretic security, in the $(\mathcal{F}_{\mathrm{PSMT}}, \mathcal{F}_{\mathrm{CORR}}^{D_j^{\mathsf{corr}}})$-hybrid model (for some distribution D_j and a distribution D_j^{corr} in NC^0), in the presence of an adaptive, malicious t-adversary, assuming that all honest parties receive their inputs at the same round. Denote $\mu = \max\{\mu_1, \ldots, \mu_M\}$ and let $0 < \epsilon < 1$.*

Then, for parameters $R > \mu, N > \frac{\log(M/\epsilon)}{\log(R/\mu)}$ and $T = \mathsf{poly}(\kappa)$, the compiled protocol $\pi_{\mathrm{PBB}} = \mathsf{Comp}(\pi_1, \ldots, \pi_M, N, R, T)$ UC-realizes $\mathcal{W}_{\mathrm{flex}}^{D_{\mathsf{pbb}}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_M)$ with expected round complexity $\mu_{\mathsf{pbb}} = O(R)$, in the $(\mathcal{F}_{\mathrm{PBC}}, \mathcal{F}_{\mathrm{SCP}})$-hybrid model and same adversarial setting, assuming that all honest parties receive their inputs at the same round. In addition, the compiler requires only black-box access to the protocols π_j.

In particular, if for every $j \in [M]$, the expectation μ_j is constant, then μ_{pbb} is constant.

Protocol $\mathsf{Comp}(\pi_1, \ldots, \pi_{\mathtt{M}}, \mathtt{N}, \mathtt{R}, \mathtt{T})$

1. Each party P_i, upon receiving $(\mathtt{input}, \mathsf{sid}, \boldsymbol{x}_i)$ with $\boldsymbol{x}_i = (x_i^1, \ldots, x_i^{\mathtt{M}})$, sends $(\mathtt{input}, \mathsf{sid}, \lambda)$ to $\mathcal{F}_{\text{SCP}}$ and receives back $(\mathtt{output}, \mathsf{sid}, (\boldsymbol{r}_i, v_1^i \ldots, v_n^i))$, where $\boldsymbol{r}_i = (\boldsymbol{r}_i^{\mathsf{input}}, \boldsymbol{r}_i^1, \ldots, \boldsymbol{r}_i^{\mathtt{M}})$ with $\boldsymbol{r}_i^j = (r_{i,j}^{\mathsf{corr}}, r_{i,j}^{\mathsf{prot}}, r_{i,j}^{\mathsf{mask}})$, for $j \in [\mathtt{M}]$, as defined in $\mathcal{D}_{\mathsf{parallel}}$, and v_u^i is the i'th share of the committed randomness $\boldsymbol{r}_u$ of party P_u. In addition, set the iteration index $\alpha \leftarrow 0$.
2. Every party P_i broadcasts $\tilde{\boldsymbol{m}}_i^{\mathsf{input}} = \boldsymbol{x}_i \oplus \boldsymbol{r}_i^{\mathsf{input}}$.
3. Denote $\tilde{\boldsymbol{m}}^{\mathsf{input}} = (\tilde{\boldsymbol{m}}_1^{\mathsf{input}}, \ldots, \tilde{\boldsymbol{m}}_n^{\mathsf{input}})$. In case some party P_i didn't broadcast a valid value, each party locally sets for P_i a default input value $\tilde{\boldsymbol{x}}_i = (\tilde{x}_1, \ldots, \tilde{x}_n)$ and default random coins $\tilde{\boldsymbol{r}}_i$, and locally computes $\tilde{\boldsymbol{m}}_i^{\mathsf{input}}$.
4. initiate (in parallel) $\mathtt{N}$ instances of every protocol π_j as follows:
   (a) For every $j \in [\mathtt{M}]$ and $k \in [\mathtt{N}]$, initialize the history of each execution, $h_{i,j,k}, \tilde{h}_{i,j,k} \leftarrow \lambda$. Denote the private history of all instances of π_j as $\boldsymbol{h}_i^j = (h_{i,j,1}, \ldots, h_{i,j,\mathtt{N}})$, and the public history as $\tilde{\boldsymbol{h}}_i^j = (\tilde{h}_{i,j,1}, \ldots, \tilde{h}_{i,j,\mathtt{N}})$; finally, denote the public history of the entire protocol as $\tilde{\boldsymbol{h}}_i = (\tilde{\boldsymbol{m}}^{\mathsf{input}}, \tilde{\boldsymbol{h}}_i^1, \ldots, \tilde{\boldsymbol{h}}_i^{\mathtt{M}})$.
   (b) Execute for rounds $\rho = 1, \ldots, \mathtt{R}$:
      i. Every party P_i computes its ρ-round messages for the k'th execution of π_j, for every $j \in [\mathtt{M}]$ and $k \in [\mathtt{N}]$:
      $$\boldsymbol{m}_{i,j,k,\rho} = (m_{i,j,k,\rho,1}, \ldots, m_{i,j,k,\rho,n}) = f_{\mathsf{nxt\text{-}msg}}^{\pi_j,i}(x_i^j, r_{i,j,\tilde{k}}^{\mathsf{corr}}, r_{i,j,\tilde{k}}^{\mathsf{prot}}, h_{i,j,k}),$$
      where $\tilde{k} = \alpha \cdot \mathtt{N} + k$. Next, mask the messages as $\tilde{\boldsymbol{m}}_{i,j,k,\rho} = (\tilde{m}_{i,j,k,\rho,1}, \ldots, \tilde{m}_{i,j,k,\rho,n})$, where for $u \in [n]$, set $\tilde{m}_{i,j,k,\rho,u} = m_{i,j,k,\rho,u} \oplus r_{i,j,\tilde{k},\rho,u}^{\mathsf{mask}}$. Finally, prepare the message $\tilde{\boldsymbol{m}}_{i,\rho} = (\tilde{\boldsymbol{m}}_{i,1,\rho}, \ldots, \tilde{\boldsymbol{m}}_{i,\mathtt{M},\rho})$, where for every $j \in [\mathtt{M}]$, $\tilde{\boldsymbol{m}}_{i,j,\rho} = (\tilde{\boldsymbol{m}}_{i,j,1,\rho}, \ldots, \tilde{\boldsymbol{m}}_{i,j,\mathtt{N},\rho})$.
      ii. Every party P_i sends $(\mathtt{input}, \mathsf{sid}, (\alpha, \rho, \tilde{\boldsymbol{m}}_{i,\rho}, \tilde{\boldsymbol{h}}_i))$ to $\mathcal{F}_{\text{SCP}}$, and receives back $(\mathtt{output}, \mathsf{sid}, (((\alpha, \rho, \tilde{\boldsymbol{m}}_{1,\rho}, \tilde{\boldsymbol{h}}_1), b_1), \ldots, ((\alpha, \rho, \tilde{\boldsymbol{m}}_{n,\rho}, \tilde{\boldsymbol{h}}_n), b_n)))$.
      iii. For every $u \in [n]$ with $b_u = 0$, every P_i broadcasts v_u^i and all parties reconstruct the committed randomness of P_u, $\boldsymbol{r}_u = \mathsf{Recon}(v_u^1, \ldots, v_u^n)$.
      iv. For every $u \in [n]$ with $b_u = 1$, P_i unmasks the message $\tilde{\boldsymbol{m}}_{u,\rho}$ sent by P_u, by computing $m_{u,j,k,\rho,i} = \tilde{m}_{u,j,k,\rho,i} \oplus r_{u,j,\tilde{k},\rho,i}^{\mathsf{mask}}$, for $j \in [\mathtt{M}]$ and $k \in [\mathtt{N}]$.
      v. Every party P_i locally computes the ρ-round messages $\tilde{\boldsymbol{m}}_{u,\rho}$ on behalf of all parties P_u that have been identified as corrupted thus far, using the committed input $\tilde{\boldsymbol{m}}_u^{\mathsf{input}}$ and reconstructed randomness $\boldsymbol{r}_u$.
      vi. Every party P_i appends all unmasked ρ-round messages $(m_{1,j,k,\rho,i}, \ldots, m_{n,j,k,\rho,i})$ to $h_{i,j,k}$, and all masked ρ-round messages $(\tilde{\boldsymbol{m}}_{1,j,k,\rho}, \ldots, \tilde{\boldsymbol{m}}_{n,j,k,\rho})$ to $\tilde{h}_{i,j,k}$.
   (c) For every $j \in [\mathtt{M}]$ and $k \in [\mathtt{N}]$, if party P_i completed the k'th execution of π_j with output, let $y_{i,j,k}$ be the output value ($y_{i,j,k} = \perp$ otherwise). Denote $S_i^j = \{y_{i,j,k} \mid y_{i,j,k} \neq \perp\}$. If for every $j \in [\mathtt{M}]$, $S_i^j \neq \emptyset$, then set $\boldsymbol{y} = (y_1, \ldots, y_{\mathtt{M}})$, where $y_j \in S_i^j$ is arbitrarily chosen, and broadcasts $(\mathsf{sid}, \boldsymbol{y})$.
   (d) If P_i received identical values $(\mathsf{sid}, \boldsymbol{y})$ from (at least) $t+1$ parties, it broadcasts $(\mathsf{sid}, \boldsymbol{y})$.
   (e) If party P_i received identical values $(\mathsf{sid}, \boldsymbol{y})$ from (at least) $n - t$ parties, it outputs $(\mathtt{output}, \mathsf{sid}, \boldsymbol{y})$ and halts. Else, set $\alpha \leftarrow \alpha + 1$. If $\alpha < \mathtt{T}$, goto Step 4; else, output $\perp$ and halt.

Figure 5: The parallel-composition compiler in the $(\mathcal{F}_{\text{PBC}}, \mathcal{F}_{\text{SCP}})$-hybrid model

To prove security of the construction, we construct a simulator for the dummy adversary, which

simulates the functionality $\mathcal{F}_{\text{SCP}}$ and all honest parties. At a high level, the simulation proceeds as follows. Since every protocol π_j realizes $\mathcal{F}_j$, there exists a simulator $\mathcal{S}_j$ for the dummy adversary. In order to simulate the k'th instance of each protocol π_j, the simulator $\mathcal{S}$ invokes an instance of $\mathcal{S}_j$, denoted $\mathcal{S}_j^k$, and receives correlated randomness $\tilde{r}_{i,j,k}^{\mathsf{corr}}$ for every corrupted party P_i. $\mathcal{S}$ samples randomness from the distribution $\mathcal{D}_{\mathsf{parallel}}$, adjusts the correlated randomness for every corrupted party accordingly and hands the adversary $(\boldsymbol{r}_i, v_1^i, \ldots, v_n^i)$ as the answer from the first call to $\mathcal{F}_{\text{SCP}}$, where $(v_i^1, \ldots, v_i^n)$ are shares of zero, for every $i \in [n]$. For the input-commitment message, the simulator broadcasts commitments of zero for the honest parties (i.e., random messages). The k'th instance of π_j is simulated now using $\mathcal{S}_j^k$, where $\mathcal{S}$ masks/unmasks the messages between $\mathcal{A}$ and $\mathcal{S}_j^k$, appropriately. Every message sent by $\mathcal{A}$ on behalf of a corrupted party P_i is validated by $\mathcal{S}$ according to the relation $\mathcal{R}_{\mathsf{parallel}}^i$, and in case it is invalid, $\mathcal{S}$ locally computes the messages for P_i, using its input and correlated randomness. In case party P_i gets corrupted, the simulator corrupts the dummy party in the ideal computation and learns its input; next, $\mathcal{S}$ hands the input to each simulator $\mathcal{S}_j^k$, receives the random coins for P_i in each instance of π_j, updates the random coins $\boldsymbol{r}_i$ accordingly and hands it to $\mathcal{A}$. This is a valid simulation for the dummy adversary, following the security guarantees of each simulator $\mathcal{S}_j$ and of the secret-sharing scheme Π.

Proof. We start by defining the sampling algorithm for the distribution D_{pbb}, parametrized by $\mathtt{N}, \mathtt{R}, \mathtt{T}$ and distributions $D_1, \ldots, D_{\mathtt{M}}$. The sampler initially sets $\alpha \leftarrow 0$ and a trace T with root labeled by $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pbb}}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$ and children $(\mathcal{F}_{\text{SCP}}^1, \mathcal{F}_{\text{PBC}})$. Next, independently sample traces $T_j^{\tilde{k}} \leftarrow D_j$, for $j \in [\mathtt{M}]$ and $k \in [\mathtt{N}]$ (where $\tilde{k} = \alpha \cdot \mathtt{N} + k$), and append

$$(\{\mathcal{F}_{\text{SCP}}^i, \mathcal{F}_{\text{PBC}}\}^{\mathtt{R}}, \mathcal{F}_{\text{PBC}}, \mathcal{F}_{\text{PBC}}, \mathcal{F}_{\text{PBC}})$$

to the leaves of the trace T (i.e., calling $\mathtt{R}$ times, sequentially, to $(\mathcal{F}_{\text{SCP}}^i, \mathcal{F}_{\text{PBC}})$, followed by three sequential calls to $\mathcal{F}_{\text{PBC}}$). If for every $j \in [\mathtt{M}]$, there exists $k \in [\mathtt{N}]$ such that $c_{\mathsf{tr}}(T_j^{\tilde{k}}) < \mathtt{R}$, then output T and halt. Else, set $\alpha \leftarrow \alpha + 1$. If $\alpha < \mathtt{T}$, repeat the sampling process; otherwise output T and halt.

Following Lemma 2.3, for $\mathtt{R} > \mu$ and $\mathtt{N} > \frac{\log(\mathtt{M}/\epsilon)}{\log(\mathtt{R}/\mu)}$, it holds that in every iteration, at least one execution of π_j will produce output, for every $j \in [\mathtt{M}]$, with a constant probability. It follows that the expected number of iterations until all honest parties receive output and the protocol terminates is constant, and since each iteration consists of $\mathtt{R}$ rounds, the entire execution completes within expected $O(\mathtt{R})$ rounds, as required. The failure probability that the protocol will not terminate within $\mathtt{T} = \mathsf{poly}(\kappa)$ iterations is negligible.

We construct a simulator $\mathcal{S}$ for the dummy adversary $\mathcal{A}$. The simulator $\mathcal{S}$ uses, in a black-box way, the simulators $\mathcal{S}_j$, for $j \in [\mathtt{M}]$, where every $\mathcal{S}_j$ simulates the dummy adversary for protocol π_j. The simulators $\mathcal{S}_j$ are guaranteed to exist since every π_j UC-realizes $\mathcal{F}_j$. Each simulator $\mathcal{S}_j$ is invoked (at most) $q = \mathtt{T} \cdot \mathtt{N}$ times; denote by $\mathcal{S}_j^k$ the k'th invocation of $\mathcal{S}_j$.

We consider the representation of the reactive CSF $\mathcal{F}_{\text{SCP}}$ as a sequence of (non-reactive) CSFs $(\mathcal{F}_{\text{CSF}}^{\tilde{f}_{\text{SCP}}^1, l_{\text{SCP}}^1}, \ldots, \mathcal{F}_{\text{CSF}}^{\tilde{f}_{\text{SCP}}^{q'}, l_{\text{SCP}}^{q'}})$, where $\tilde{f}_{\text{SCP}}^j$ is defined as in Definition C.1 and $q' = \mathtt{T} \cdot \mathtt{R} + 1$. Denote $\mathcal{F}_{\text{SCP}}^j = \mathcal{F}_{\text{CSF}}^{\tilde{f}_{\text{SCP}}^j, l_{\text{SCP}}^j}$.

The simulator $\mathcal{S}$ proceeds as follows:

- Initially, set $\boldsymbol{x}_1 = \ldots = \boldsymbol{x}_n = \perp$ and $\boldsymbol{y} = \perp$.
- Simulating the first call to functionality $\mathcal{F}_{\text{SCP}}$:
  1. Sample correlated randomness $(\boldsymbol{r}_1, \ldots, \boldsymbol{r}_n) \leftarrow \mathcal{D}_{\mathsf{parallel}}(\pi_1, \ldots, \pi_{\mathtt{M}}, q, \mathtt{R}, \ell)$, where $q = \mathtt{N} \cdot \mathtt{T}$ and $\boldsymbol{r}_i = (\boldsymbol{r}_i^{\mathsf{input}}, \boldsymbol{r}_i^1, \ldots, \boldsymbol{r}_i^{\mathtt{M}})$ with $\boldsymbol{r}_i^j = (r_{i,j}^{\mathsf{corr}}, r_{i,j}^{\mathsf{prot}}, r_{i,j}^{\mathsf{mask}})$, for $i \in [n]$ and $j \in [\mathtt{M}]$.

2. For every $j \in [\mathtt{M}]$ and $k \in [q]$, invoke $\mathcal{S}_j^k$, request the correlated randomness for every corrupted party P_i and get from $\mathcal{S}_j^k$ the values $\tilde{r}_{i,j,k}^{\mathsf{corr}}$. Set $r_{i,j,k}^{\mathsf{corr}} \leftarrow \tilde{r}_{i,j,k}^{\mathsf{corr}}$ in $\boldsymbol{r}_i$.
3. Receive from $\mathcal{A}$ the message $(\mathtt{input}, \mathsf{sid}, \lambda)$, for a corrupted P_i (that $\mathcal{A}$ sends to $\mathcal{F}_{\text{SCP}}$).
4. For every party P_i, compute $(\tilde{v}_i^1, \ldots, \tilde{v}_i^n) \leftarrow \mathsf{Share}(0^{|\boldsymbol{r}_i|})$.
5. Send the message $(\mathtt{output}, \mathsf{sid}, (\boldsymbol{r}_i, \tilde{v}_1^i \ldots, \tilde{v}_n^i))$ to $\mathcal{A}$ as the response from $\mathcal{F}_{\text{SCP}}$, for every corrupted P_i.

- Simulating the first (input-commitment) message:

1. For every honest party P_i, send a random string $\tilde{\boldsymbol{m}}_i^{\mathsf{input}}$ to $\mathcal{A}$.
2. For every corrupted party P_i, receive the message $\tilde{\boldsymbol{m}}_i^{\mathsf{input}}$ from $\mathcal{A}$ and extract the input value $\boldsymbol{x}_i = \tilde{\boldsymbol{m}}_i^{\mathsf{input}} \oplus \boldsymbol{r}_i^{\mathsf{input}}$. In case $\mathcal{A}$ does not send a message for P_i, set $\boldsymbol{x}_i$ and $\boldsymbol{r}_i$ to predetermined default values and locally compute $\tilde{\boldsymbol{m}}_i^{\mathsf{input}} = \boldsymbol{x}_i \oplus \boldsymbol{r}_i$.
3. Set $\tilde{\boldsymbol{m}}^{\mathsf{input}} = (\tilde{\boldsymbol{m}}_1^{\mathsf{input}}, \ldots, \tilde{\boldsymbol{m}}_n^{\mathsf{input}})$.

- Send inputs to $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pbb}}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$:

1. For every corrupted party P_i, send the message $(\mathtt{input}, \mathsf{sid}, \boldsymbol{x}_i)$ to $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pbb}}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$.
2. Upon receiving message $(\mathtt{leakage}, \mathsf{sid}, P_i, (l_1, \ldots, l_{\mathtt{M}}))$ from $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pbb}}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$, for an honest party P_i, store the leakage information.
3. In addition, $\mathcal{S}$ receives $(\mathtt{trace}, \mathsf{sid}, T)$, from $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pbb}}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$, where T is a depth-1 trace of the form

$$(\mathcal{F}_{\text{SCP}}^1, \mathcal{F}_{\text{PBC}}, \{\{\mathcal{F}_{\text{SCP}}^i, \mathcal{F}_{\text{PBC}}\}^{\mathtt{R}}, \mathcal{F}_{\text{PBC}}, \mathcal{F}_{\text{PBC}}, \mathcal{F}_{\text{PBC}}\}^{q''}),$$

(i.e., initially, calls to $\mathcal{F}_{\text{SCP}}^1$ and $\mathcal{F}_{\text{PBC}}$, followed by q'' iterations of calling $\mathtt{R}$ times $(\mathcal{F}_{\text{SCP}}^i, \mathcal{F}_{\text{PBC}})$ and 3 rounds of $\mathcal{F}_{\text{PBC}}$ in order to agree on termination). More precisely, $\mathcal{S}$ receives the coins that were used by the functionality $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pbb}}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$ to sample the trace T from D_{pbb}. Using these coins, $\mathcal{S}$ samples the same traces $T_j^{\tilde{k}}$ that were used to define T.

- Simulating the α'th iteration:

1. For every $i \in [n]$, $j \in [\mathtt{M}]$ and $k \in [\mathtt{N}]$, set $h_{i,j,k}, \tilde{h}_{j,k} \leftarrow \lambda$. Denote $\boldsymbol{h}_i^j = (h_{i,j,1}, \ldots, h_{i,j,\mathtt{N}})$, $\tilde{\boldsymbol{h}}^j = (\tilde{h}_{j,1}, \ldots, \tilde{h}_{j,\mathtt{N}})$ and $\tilde{\boldsymbol{h}} = (\tilde{h}^1, \ldots, \tilde{h}^{\mathtt{M}})$. For $k \in [\mathtt{N}]$, let $\tilde{k} = \alpha \cdot \mathtt{N} + k$.
2. For $j \in [\mathtt{M}]$ and $k \in [\mathtt{N}]$, send to $\mathcal{S}_j^{\tilde{k}}$, on behalf of every honest party, the leakage messages $(\mathtt{leakage}, \mathsf{sid}, P_i, l_j)$, as well as $(\mathtt{trace}, \mathsf{sid}, T_j^{\tilde{k}})$. In addition, send to $\mathcal{S}_j^{\tilde{k}}$ the message $(\mathtt{input}, \mathsf{sid}, x_i^j)$, for every corrupted party P_i, and receive from $\mathcal{S}_j^{\tilde{k}}$ the messages $(\mathtt{input}, \mathsf{sid}, x_i^j)$ that are sent to $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_j)$, for the corrupted parties.
3. Execute for rounds $\rho = 1, \ldots, \mathtt{R}$:
   (a) For every honest P_i, send $(\alpha, \rho, \tilde{\boldsymbol{m}}_{i,\rho}, \tilde{\boldsymbol{h}})$ to $\mathcal{A}$ as the leakage from $\mathcal{F}_{\text{SCP}}$ for P_i, where $\tilde{\boldsymbol{m}}_{i,\rho}$ is prepared as follows. For every corrupted party P_u, receive from $\mathcal{S}_j^{\tilde{k}}$ the message $m_{i,j,k,\rho,u}$ and set $\tilde{m}_{i,j,k,\rho,u} = m_{i,j,k,\rho,u} \oplus r_{i,j,\tilde{k},\rho,u}^{\mathsf{mask}}$. For every honest P_u, sample a random message $\tilde{m}_{i,j,k,\rho,u}$. Set $\tilde{\boldsymbol{m}}_{i,\rho} = (\tilde{\boldsymbol{m}}_{i,1,\rho}, \ldots, \tilde{\boldsymbol{m}}_{i,\mathtt{M},\rho})$, where $\tilde{\boldsymbol{m}}_{i,j,\rho} = (\tilde{\boldsymbol{m}}_{i,j,1,\rho}, \ldots, \tilde{\boldsymbol{m}}_{i,j,\mathtt{N},\rho})$ with $\tilde{\boldsymbol{m}}_{i,j,k,\rho} = (\tilde{m}_{i,j,k,\rho,1}, \ldots, \tilde{m}_{i,j,k,\rho,n})$.

(b) For every corrupted party P_i, receive from $\mathcal{A}$ the message $(\mathtt{input}, \mathsf{sid}, (\alpha, \rho, \tilde{\boldsymbol{m}}_{i,\rho}, \tilde{\boldsymbol{h}}_i))$ (intendend to $\mathcal{F}_{\text{SCP}}$), and check whether $\tilde{\boldsymbol{h}}_i = \tilde{\boldsymbol{h}}$ and $\mathcal{R}^i_{\mathsf{parallel}}((\alpha, \rho, \tilde{\boldsymbol{m}}_{i,\rho}, \tilde{\boldsymbol{h}}_i), \boldsymbol{r}_i) = 1$. If so, set $b_i = 1$ else $b_i = 0$.

(c) For every corrupted party P_i with $b_i = 1$, it holds that $\tilde{\boldsymbol{m}}_{i,\rho} = (\tilde{\boldsymbol{m}}_{i,1,\rho}, \ldots, \tilde{\boldsymbol{m}}_{i,\mathtt{M},\rho})$, where $\tilde{\boldsymbol{m}}_{i,j,\rho} = (\tilde{\boldsymbol{m}}_{i,j,1,\rho}, \ldots, \tilde{\boldsymbol{m}}_{i,j,\mathtt{N},\rho})$ and $\tilde{\boldsymbol{m}}_{i,j,k,\rho} = (\tilde{\boldsymbol{m}}_{i,j,k,\rho,1}, \ldots, \tilde{\boldsymbol{m}}_{i,j,k,\rho,n})$. Denote, for every honest P_u, $m_{i,j,k,\rho,u} = \tilde{m}_{i,j,k,\rho,u} \oplus r^{\mathsf{mask}}_{i,j,\tilde{k},\rho,u}$.

(d) For every honest party P_i, denote $\tilde{\boldsymbol{h}}_i = \tilde{\boldsymbol{h}}$ and $b_i = 1$, and send to $\mathcal{A}$ $(\mathtt{output}, \mathsf{sid}, (((\alpha, \rho, \tilde{\boldsymbol{m}}_{1,\rho}, \tilde{\boldsymbol{h}}_1), b_1), \ldots, ((\alpha, \rho, \tilde{\boldsymbol{m}}_{n,\rho}, \tilde{\boldsymbol{h}}_n), b_n)))$ as the response from $\mathcal{F}_{\text{SCP}}$, on behalf of every corrupted party.

(e) For every corrupted party P_i with $b_i = 1$ and every honest P_u, forward to $\mathcal{S}^{\tilde{k}}_j$ the message $m_{i,j,k,\rho,u}$.

(f) For every corrupted party P_i with $b_i = 0$, compute $(v^1_i, \ldots, v^n_i) \leftarrow \mathsf{Share}(\boldsymbol{r}_i)$ and send v^u_i to $\mathcal{A}$, on behalf of every honest party P_u. (Note that $\boldsymbol{r}_i$ may change as the simulation proceeds, in case P_i gets corrupted dynamically.)

(g) For every corrupted party that has been previously identified, locally compute the messages $m_{i,j,k,\rho,u}$, for every honest party P_u, using $(\boldsymbol{x}_i, \boldsymbol{r}_i)$ and $h_{i,j,k}$, and send them to $\mathcal{S}^{\tilde{k}}_j$.

(h) Append the messages $(\tilde{\boldsymbol{m}}_{1,\rho}, \ldots, \tilde{\boldsymbol{m}}_{n,\rho})$ to the global history $\tilde{\boldsymbol{h}}$. For every corrupted party P_i, append the messages $(m_{1,j,k,\rho,i}, \ldots, m_{n,j,k,\rho,i})$ to the local history $h_{i,j,k}$.

(i) Upon receiving $(\mathtt{early\text{-}output}, \mathsf{sid}, P_i)$ from $\mathcal{S}^k_j$ for an honest party P_i, simulate the honest party receviing the output message.

(j) Upon receiving $(\mathtt{early\text{-}output}, \mathsf{sid}, P_i)$ from $\mathcal{S}^{\tilde{k}}_j$ for a corrupted party P_i, if $\boldsymbol{y} = \perp$, send $(\mathtt{early\text{-}output}, \mathsf{sid}, P_i)$ to $\mathcal{W}^{D_{\mathsf{pbb}}}_{\text{flex}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$ and receive back $(\mathtt{output}, \mathsf{sid}, \boldsymbol{y})$, with $\boldsymbol{y} = (y_1, \ldots, y_{\mathtt{M}})$. Send $(\mathtt{output}, \mathsf{sid}, y_j)$ to $\mathcal{S}^{\tilde{k}}_j$.

(k) For every $j \in [\mathtt{M}]$ and $k \in [\mathtt{N}]$, check whether $\rho = c_{\mathsf{tr}}(T^{\tilde{k}}_j)$. If so and $\boldsymbol{y} = \perp$, send $(\mathtt{early\text{-}output}, \mathsf{sid}, P_i)$ to $\mathcal{W}^{D_{\mathsf{pbb}}}_{\text{flex}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$ and receive back $(\mathtt{output}, \mathsf{sid}, \boldsymbol{y})$, with $\boldsymbol{y} = (y_1, \ldots, y_{\mathtt{M}})$. Send $(\mathtt{output}, \mathsf{sid}, y_j)$ to $\mathcal{S}^{\tilde{k}}_j$ for every corrupted party.

- Explaining corruption requests: upon a corruption request for P_i, proceed as follows.

  1. Corrupt the dummy party P_i and learn its input $\boldsymbol{x}_i = (x^1_i, \ldots, x^{\mathtt{M}}_i)$.
  2. For every $j \in [\mathtt{M}]$ and $k \in [\mathtt{T} \cdot \mathtt{N}]$:
     (a) Instruct $\mathcal{S}^k_j$ to corrupt P_i and provide it with input x^j_i.
     (b) Receive from $\mathcal{S}^k_j$ the view of P_i, i.e., its internal randomness $\tilde{r}^{\mathsf{prot}}_{i,j,k}$, correlated randomness $\tilde{r}^{\mathsf{corr}}_{i,j,k}$ and history $h_{i,j,k}$.
     (c) For every $u \in [n]$ and every message $m_{u,j,k,\rho,i}$ in $h_{i,j,k}$ (message from P_u to P_i in the ρ'th round of the k'th instance of π_j, for $\rho \in [\mathtt{R}]$), compute $\tilde{r}^{\mathsf{mask}}_{u,j,k,\rho,i} = m_{u,j,k,\rho,i} \oplus \tilde{m}_{u,j,k,\rho,i}$. Similarly, compute $\tilde{r}^{\mathsf{mask}}_{i,j,k,\rho,u} = m_{i,j,k,\rho,u} \oplus \tilde{m}_{i,j,k,\rho,u}$.
     (d) Compute $\tilde{\boldsymbol{r}}^{\mathsf{input}}_i = \tilde{\boldsymbol{m}}^{\mathsf{input}}_i \oplus \boldsymbol{x}_i$.
     (e) Adjust the random coins $r^{\mathsf{corr}}_{i,j,k} \leftarrow \tilde{r}^{\mathsf{corr}}_{i,j,k}$, $r^{\mathsf{prot}}_{i,j,k} \leftarrow \tilde{r}^{\mathsf{prot}}_{i,j,k}$, $r^{\mathsf{mask}}_{i,j,k} \leftarrow \tilde{r}^{\mathsf{mask}}_{i,j,k}$ and $\boldsymbol{r}^{\mathsf{input}}_i \leftarrow \tilde{\boldsymbol{r}}^{\mathsf{input}}_i$ in $\boldsymbol{r}_i$.
  3. Provide the input $\boldsymbol{x}_i$ and the adjusted correlated randomness $\boldsymbol{r}_i$ to $\mathcal{A}$ as the internal state of P_i.

We next prove that no environment can distinguish between interacting with the dummy adversary and the honest parties running protocol π' in the $(\mathcal{F}_{\text{PBC}}, \mathcal{F}_{\text{SCP}})$-hybrid model, from interacting with the simulator $\mathcal{S}$ and the dummy honest parties computing $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pbb}}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$, except for a negligible probability. We prove this using a series of hybrid games. The output of each game is the output of the environment.

The game $\text{HYB}^{1}_{\pi_{\mathtt{PBB}},\mathcal{A},\mathcal{Z}}$. This is exactly the execution of the protocol π_{PBB} in the $(\mathcal{F}_{\text{PBC}}, \mathcal{F}_{\text{SCP}})$-hybrid model with environment $\mathcal{Z}$ and dummy adversary $\mathcal{A}$.

The games $\text{HYB}^{2,i^*}_{\pi_{\mathtt{PBB}},\mathcal{A},\mathcal{Z}}$, for $0 \leq i^* \leq n$. In these games, we modify $\text{HYB}^{1}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}}$ as follows. During the first call to $\mathcal{F}_{\text{SCP}}$, for $i^* < i \leq n$, compute $(v_i^1, \ldots, v_i^n) \leftarrow \mathsf{Share}(\boldsymbol{r}_i)$ (as before) and send $(v_1^i, \ldots, v_n^i)$ as the shares for every party P_i. For every $i \leq i^*$, compute $(v_i^1, \ldots, v_i^n) \leftarrow \mathsf{Share}(\boldsymbol{r}_i)$ and $(\tilde{v}_i^1, \ldots, \tilde{v}_i^n) \leftarrow \mathsf{Share}(0^{|\boldsymbol{r}_i|})$. Next, send to every corrupted P_k the shares $(\tilde{v}_1^k, \ldots, \tilde{v}_n^k)$ and to every honest party the shares $(v_1^i, \ldots, v_n^i)$.

Claim 6.4. $\text{HYB}^{1}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}} \stackrel{\text{s}}{\equiv} \text{HYB}^{2,n}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}}$.

Proof. Note that $\text{HYB}^{1}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}} \equiv \text{HYB}^{2,0}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}}$. For every $0 \leq \tilde{i} < n$, it holds that $\text{HYB}^{2,\tilde{i}}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}} \stackrel{\text{s}}{\equiv} \text{HYB}^{2,\tilde{i}+1}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}}$. This follows from the security of the ECSS scheme and the honest-majority assumption. In particular, the shares $\{v_i^u\}_u$ held by the adversary (for all corrupted P_u), for any honest party P_i, completely hide $\boldsymbol{r}_i$, and are compatible even if P_i gets adaptively corrupted and $\boldsymbol{r}_i$ is revealed. This holds since all honest parties receive valid shares of $\boldsymbol{r}_i$, therefore, $\boldsymbol{r}_i$ will be correctly reconstructed even if all corrupted parties have incorrect shares. The claim follows using a standard hybrid argument. □

The game $\text{HYB}^{3}_{\pi_{\mathtt{PBB}},\mathcal{A},\mathcal{Z}}$. In this game, we modify $\text{HYB}^{2,n}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}}$ as follows. The input-commitment message $\tilde{\boldsymbol{m}}_i^{\mathsf{input}}$, sent by an honest party is uniformly chosen. In addition, in every call to $\mathcal{F}_{\text{SCP}}$ the returned value b_i for every honest party P_i is always set to 1. Finally, the random coins for honest parties that are corrupted adaptively are set as $\boldsymbol{r}_i^{\mathsf{input}} = \tilde{\boldsymbol{m}}_i^{\mathsf{input}} \oplus \boldsymbol{x}_i$.

Claim 6.5. $\text{HYB}^{2,n}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}} \equiv \text{HYB}^{3}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}}$.

Proof. The claim immediately follows since honest parties always send correct messages. □

The games $\text{HYB}^{4,j^*,k^*}_{\pi_{\mathtt{PBB}},\mathcal{A},\mathcal{Z}}$, for $0 \leq j^* \leq \mathtt{M}$ and $0 \leq k^* \leq q = \mathtt{N} \cdot \mathtt{T}$. In these games, we modify $\text{HYB}^{3}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}}$ as follows. For $(j,k) < (j^*,k^*)$ (i.e., for $j < j^*$, or $j = j^*$ and $k < k^*$), the k'th execution of protocol π_j is replaced with the simulated transcript generated by $\mathcal{S}_j^k$.

More specifically, the experiment interacts with the ideal functionality $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pbb}}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$. Initially, it receives the leakage messages $(\mathtt{leakage}, \mathsf{sid}, P_i, (l_1, \ldots, l_{\mathtt{M}}))$ and the coins used to sample a trace T from D_{pbb} (via the message $(\mathtt{trace}, \mathsf{sid}, T)$); using these random coins, sample the traces T_j^k from D_j.

Each simulator $\mathcal{S}_j^k$ (for $(j,k) < (j^*,k^*)$) is invoked and is given the leakage information l_j and the trace T_j^k. $\mathcal{S}_j^k$ provides correlated randomness $\tilde{r}_{i,j,k}^{\mathsf{corr}}$ for every corrupted party; the random coins $\boldsymbol{r}_i$ for corrupted P_i are adjusted accordingly. Once $\mathcal{A}$ sends the input-commitment message $\tilde{\boldsymbol{m}}_i^{\mathsf{input}}$, for a corrupted P_i, extract the input value $\boldsymbol{x}_i = (x_i^1, \ldots, x_i^{\mathtt{M}})$ and hand $\mathcal{S}_j^k$ the value x_i^j as the input value for P_i. The interaction with $\mathcal{S}_j^k$ is done as in the simulation, i.e., validate the messages from

$\mathcal{A}$ and unmask valid messages before forwarding to $\mathcal{S}_j^k$, and mask messages from $\mathcal{S}_j^k$ using $r_{i,j,k}^{\mathsf{corr}}$. Messages from identified corrupted parties are locally computed and sent to $\mathcal{S}_j^k$. When $\mathcal{S}_j^k$ sends $(\texttt{early-output}, \mathsf{sid}, \cdot)$ requests, respond with the output value y_j from $\boldsymbol{y} = (y_1, \ldots, y_j)$, where if $\boldsymbol{y} = \perp$, then forward the message to $\mathcal{W}_{\text{flex}}^{D_{\mathsf{pbb}}}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$.

Claim 6.6. $\text{HYB}^{3}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}} \overset{\text{s}}{\equiv} \text{HYB}^{4,\mathtt{M},q}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}}$.

Proof. Note that $\text{HYB}^{3}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}} \equiv \text{HYB}^{4,0,0}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}}$. For every $0 \leq j < \mathtt{M}$ and $0 \leq k < q$, it holds that $\text{HYB}^{4,j,k}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}} \overset{\text{s}}{\equiv} \text{HYB}^{2,j,k+1}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}}$ and similarly, $\text{HYB}^{4,j,q}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}} \overset{\text{s}}{\equiv} \text{HYB}^{2,j+1,1}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}}$; otherwise, there exists an environment that violates the simulation of $\mathcal{S}_j$ (resp., $\mathcal{S}_{j+1}$). The claim follows using a standard hybrid argument. □

The proof follows since $\text{HYB}^{4,\mathtt{M},q}_{\pi_{\text{PBB}},\mathcal{A},\mathcal{Z}}$ (for $q = \mathtt{N} \cdot \mathtt{T}$) exactly describes the simulation done by $\mathcal{S}$. □

6.2 An Impossibility of FBB Round-Preserving Parallel Composition

In this section, we prove that for a natural class of protocols, following and/or extending in various ways the techniques from Ben-Or and El-Yaniv [5],[8] there exist functions such that no protocol can compute their parallel composition in a round-preserving manner, while accessing the functions in a black-box way, tolerating even a single adversarial party. Although this is not a general impossibility result, it indicates that the batching approach of [5] is limited to semi-honest security (cf. Section 5) and/or functionally white-box transformations.

We observe that this impossibility serves as an additional justification for the optimality of our protocol-black-box parallel composition (cf. Section 6.1). Indeed, on the one hand, it formally confirms the generic observation that the naïve parallel composition of a set of PT functionalities does not preserve their round complexity. On the other hand, and most importantly, it proves that all existing techniques for composing PT functionalities in parallel in the natural (FBB) manner fail in preserving the round complexity. Hence, the only known existing round-preserving composition for such functionalities are the protocol-black-box compiler presented in Section 6.1 or more inefficient non-black-box techniques. The wideness of the class of excluded protocols by our impossibility result justifies our conjecture that there exists no round-preserving FBB protocol for parallel composition of PT functionalities. Proving this conjecture is in our opinion a very interesting research direction.

We first argue informally why the approach of [5], cannot be directly extended to privacy-sensitive functions. The idea in [5] for allowing each of the n parties to broadcast its value is to have each of the n parties participate in $m = O(\log n)$ parallel invocations (hereafter called *batches*, to avoid confusion with the goal of parallel broadcast for different messages) of broadcast as sender with the same input. Each of those batches is executed in parallel for a fixed (constant) number of rounds (for the same broadcast message); this increases the probability that sufficiently many parties receive output from each batch. At the end of each batch execution, the parties check whether they jointly hold the output, and if not, they repeat the computation of the batches. It might seem that this idea can be applied to arbitrary tasks, but this is not the case. The reason is that this idea fails if the functionality has any privacy requirements, is that the adversary can input different values on different calls of the functionality within a batch and learn more information on the input.

[8] To our knowledge, the only known techniques for round-preserving parallel composition are those of Ben-Or and El-Yaniv [5] and are only for the specific case of Byzantine agreement.

Batched parallel composition. The above issue with privacy appears whenever a function is invoked twice in the same round on the same inputs from honest parties. Indeed, in this case the adversary can use different inputs to each invocation and learn information as sketched above. The same attack can be extended to composition-protocols which invoke the function in two different rounds ρ and ρ'; as long as the adversary knows these rounds he can still launch the above attack on privacy. Generalizing the claim even further, for specific classes of functions, it suffices that there are two (possibly different) functions which are evaluated on the same inputs in rounds ρ and ρ'. This excludes protocols that might attempt to avoid using some functionality $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_j)$ by invoking some other $\mathcal{W}_{\text{flex}}^{D_{j'}}(\mathcal{F}_{j'})$.

To capture the above generalization, we define the class of *batched-parallel composition* protocols: A protocol π implementing the PT parallel composition $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$ in the $(\mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_1), \ldots, \mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\mathtt{M}}))$-hybrid model (for some distributions $D, D_1, \ldots, D_{\mathtt{M}}$) is a *batched-parallel composition* protocol if it has the following structure: It proceeds in rounds, where in each round the protocol might initiate (possibly multiple) calls to any number of the hybrid functionalities $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_j)$ and/or continue calls that were initiated in previous rounds. Furthermore, there exist two publicly known protocol rounds ρ and ρ', and indices $j, j', \ell \in [\mathtt{M}]$, such that for the input vector $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ that π gives to $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F}_1 \parallel \cdots \parallel \mathcal{F}_{\mathtt{M}})$ (where $\boldsymbol{x}_i = (x_i^1, \ldots, x_i^{\mathtt{M}})$) the following properties are satisfied:

1. In round ρ the functionality $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_j)$ is called on input $\boldsymbol{x}^\ell = (x_1^\ell, \ldots, x_n^\ell)$ and at least two of its rounds are executed.[9]

2. In round ρ' the functionality $\mathcal{W}_{\text{flex}}^{D_{j'}}(\mathcal{F}_{j'})$ is also called on input $\boldsymbol{x}^\ell$ and at least two of its rounds are executed.

Note that the protocol in [5] (as well as our semi-honest protocol from Section 5) is an example of a batched-parallel composition protocol for $\rho = \rho' = 1$ and for $j = j' = \ell$ being the index of any one of the hybrid functionalities. Indeed, in the first round of these protocols, each functionality is invoked at least twice on the same inputs. In particular, protocols that follow this structure, e.g., even ones that do not call all functionalities in every phase, or those that have variable batch sizes, can be described as such batched-parallel composition protocols.

We next show that the there are classes of functions $\mathcal{C}_1, \ldots, \mathcal{C}_{\mathtt{M}}$ such and for any protocol π that securely computes the parallel composition $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$ while given hybrid access to PT functionalities $\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_i})$ the following properties hold *simultaneously*:

1. π has to call each of the hybrids $\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_i})$ (for at least 2 rounds each).[10]

2. The naïve solution of π calling each of the $\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_i})$'s in parallel until they terminate is not round-preserving (for an appropriate choice of D_i's.)

3. π cannot be a batched-parallel composition protocol.

The above shows that the classes $\mathcal{C}_1, \ldots, \mathcal{C}_{\mathtt{M}}$ not only exclude the existence of a batched-parallel composition protocol, but they also exclude all other known solutions. This implies that for this classes of functions, every known approach—and generalizations thereof—fail to compute the parallel composition of the corresponding functionality in an FBB and round-preserving manner.

[9] Note that any call to a PT (i.e., wrapped) functionality that executes less than two rounds is useless as it can be simulated by the protocol that does nothing (without of course increasing the round complexity). The reason is that, by definition of PT functionalities, any such call gives no output (the first round is an input-only round).

[10] Note that this does not mean that π is not round preserving as the calls might be in parallel.

Theorem 6.7. *Let $\mathtt{M} = O(\kappa)$. There exist n-party function classes $\mathcal{C}_1, \ldots, \mathcal{C}_{\mathtt{M}}$ and distributions $D_1, \ldots, D_{\mathtt{M}}$, such that the following properties hold in the presence of a malicious adversary corrupting any one of the parties:*

1. *The protocol that calls each $\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_i})$ in parallel (once) until it terminates is not round-preserving (i.e., its expected round complexity is asymptotically higher than that of the distributions D_i.)*
2. *Any $(\mathcal{F}_{\text{SMT}}, \mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1}), \ldots, \mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}}))$-hybrid protocol for computing $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$ has to make a meaningful call (i.e., a call that executes at least two rounds) to each PT hybrid $\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_i})$.*
3. *There exists no functionally black-box batched-parallel composition protocol for computing $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$ in the $(\mathcal{F}_{\text{SMT}}, \mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1}), \ldots, \mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}}))$-hybrid model, where D has (asymptotically) the same expectation as $D_1, \ldots, D_{\mathtt{M}}$.*

Proof. Let $D_1 = \ldots = D_{\mathtt{M}}$ be the geometric distribution with parameter $1/2$. (This means that the expected round complexity of each $\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_i})$ is constant.) Then, Property 1 follows immediately from the observation in [5] (see also [16]), which implies that the expectation of the round complexity of the naïve protocol that executes each functionality in parallel until its completion will be $\Theta(\log \mathtt{M})$, which is super-constant.

We next turn to Property 2. Towards proving it we first prove the following useful lemma.

Lemma 6.8. *There exists a family of functions $\mathcal{C} = \{f_\alpha\}_{\alpha \in \{0,1\}^\kappa}$ such that there exists no FBB protocol for computing the family of (oracle-aided) n-party SFE-functionalities $\{\mathcal{F}^{f_\alpha}\}_{f_\alpha \in \mathcal{C}}$, which is private against a semi-honest adversary corrupting any one of the n parties, and is correct with noticeable probability.*

Proof. Let f_α be the function that takes inputs $x_1 \in \{0,1\}^\kappa$ and $x_2 \in \{0,1\}^\kappa$ from P_1 and P_2, respectively (and a default input λ from every P_j with $j \in \{3, \ldots, n\}$) and outputs y_i to each P_i as follows:

- $y_1 = \begin{cases} x_2 & , \text{ if } x_1 \oplus x_2 = \alpha \\ 0^\kappa & , \text{ otherwise.} \end{cases}$
- $y_2 = \begin{cases} x_1 & , \text{ if } x_1 \oplus x_2 = \alpha \\ 0^\kappa & , \text{ otherwise.} \end{cases}$
- For every $j \in \{3, \ldots, n\}$, $y_j = \begin{cases} x_1 \oplus x_2 & , \text{ if } x_1 \oplus x_2 = \alpha \\ 0^\kappa & , \text{ otherwise.} \end{cases}$

The argument that there exists no FBB protocol for the above function family is inspired by [40, Theorem 1]. Concretely, assume towards contradiction that such a protocol π exists[11] and consider the following experiment. Pick x_1, x_2, α uniformly and independently at random and run π^{f_α} with inputs x_1 and x_2 for P_1 and P_2, respectively, and input λ for all other parties. Then we argue that the following events have negligible probability of occurring (where the probability is taken over the choice of x_1, x_2, α and the random coins $\boldsymbol{r} = (r_1, \ldots, r_n)$ of the parties):

[11] Note that by definition of FBB [52] protocols, $\pi = (\pi_1, \ldots, \pi_n)$ and each of the π_i's is an oracle machine which can query the function f_α is trying to compute (cf. Section 4).

(A) Any of the parties (i.e., any of the π_i's) queries its oracle f_α with $(p, q, \lambda, \ldots, \lambda)$ such that $p \oplus q = \alpha$.

(B) Any of the parties queries its oracle f_α with $(p, q, \lambda, \ldots, \lambda)$ such that $p \oplus q = x_1 \oplus x_2$.

The fact that (A) occurs with negligible probability is due to the fact that α is uniformly random.

The fact that (B) occurs with negligible probability follows from the protocol's privacy (and is, in fact, independent of the distribution of α). Indeed, suppose that the probability that P_i makes a query $(p, q, \lambda, \ldots, \lambda)$ such that $p \oplus q = x_1 \oplus x_2$ is noticeable (i.e., not negligible). Consider an adversary corrupting P_i and outputting the list of values $p \oplus q$, for each $(p, q, \lambda, \ldots, \lambda)$ that P_i makes to its oracle. By assumption, this list will include $x_1 \oplus x_2$ with noticeable probability. However, in the ideal-world execution, the simulator, even if he knows α, will be unable to produce a list of values containing $x_1 \oplus x_2$ with noticeable probability, since x_1 and x_2 are chosen uniformly at random, and a simulator corrupting a single party cannot learn both x_1 and x_2. This implies that the above adversary cannot be simulated, which contradicts the protocol's privacy.

Let now R denote the set of all $(\boldsymbol{r}, x_1, x_2, \alpha)$ such that in the above experiment, neither event (A) nor (B) occurs. From the above argument we know that $\Pr[(r, x_1, x_2, \alpha) \in R] > 1 - \nu(\kappa)$, for a negligible function ν.

Next, as in [40, Theorem 1], we consider the coupled experiment in which we use the same $\boldsymbol{r}, x_1, x_2$ as above, but run the protocol $\pi^{f_{\alpha^*}}$ where $\alpha^* = x_1 \oplus x_2$. As in [40, Theorem 1], we can prove that this experiment proceeds identically as the original one (which, recall differs only on the oracle calls); in particular, all oracle queries will be answered by 0^κ to all parties. The reason for this is that an oracle query $(p, q, \lambda, \ldots, \lambda)$ is answered by a value other than 0^κ only if it has the property that $p \oplus q = x_1 \oplus x_2$ which, for the combinations of $(\boldsymbol{r}, x_1, x_2, \alpha) \in R$ does not occur. This in particular implies that the output vector $\boldsymbol{y} = (y_1, \ldots, y_n)$ that the parties $P_1, \ldots, P_n$ receive from the protocol will be identical in both experiments for $(\boldsymbol{r}, x_1, x_2, \alpha) \in R$.

But since $\Pr[(\boldsymbol{r}, x_1, x_2, \alpha) \in R] > 1 - \nu(\kappa)$, the distribution of $\boldsymbol{y}$ in both experiments can be at most negligible apart. However, since the protocol is required to output the correct value with noticeable probability, this means that in the first experiment $\Pr[\boldsymbol{y} = (0^\kappa)^n] = \textit{noticeable}$ and in the second experiment $\Pr[\boldsymbol{y} \neq (0^\kappa)^n] = \textit{noticeable}$ (where the latter holds because of the choice of $\alpha^* = x_1 \oplus x_2$). This creates a noticeable advantage in distinguishing the outputs of the two experiments leading to contradiction. □

Lemma 6.9. *For the above function class $\mathcal{C}$, let $\mathcal{C}_i = \mathcal{C}$ for each $i \in [\mathtt{M}]$. Then any FBB protocol for computing $\mathcal{W}^D_{\text{flex}}(\mathcal{F}^{\mathcal{C}_1}_{\text{SFE}} \| \cdots \| \mathcal{F}^{\mathcal{C}_\mathtt{M}}_{\text{SFE}})$ with hybrid access to all $\mathcal{W}^{D_i}_{\text{flex}}(\mathcal{F}^{\mathcal{C}_i}_{\text{SFE}})$'s needs to invoke at least two rounds of every hybrid $\mathcal{W}^{D_i}_{\text{flex}}(\mathcal{F}^{\mathcal{C}_i}_{\text{SFE}})$.*

Proof. Assume towards contradiction that there exists a protocol π computing $\mathcal{W}^D_{\text{flex}}(\mathcal{F}^{\mathcal{C}_1}_{\text{SFE}} \| \cdots \| \mathcal{F}^{\mathcal{C}_\mathtt{M}}_{\text{SFE}})$ such that for some i, π might execute at most one round of $\mathcal{W}^{D_i}_{\text{flex}}(\mathcal{F}^{\mathcal{C}_i}_{\text{SFE}})$. We will prove that no such protocol can exists for a uniformly random choice of $\vec{\alpha} = (\alpha_1, \ldots, \alpha_\mathtt{M})$.[12] This is sufficient, since by an averaging argument, it implies that there exists no such protocol that is secure for all choices of $\vec{\alpha}$, as required by the definition in [52]. (Indeed, if there exists a protocol that is secure for all choices of $\vec{\alpha}$ (independent of $\vec{\alpha}$) there exists one that is secure for a randomly chosen $\vec{\alpha}$.)

[12] Consistently with [52], we assume that although $\vec{\alpha}$ is chosen uniformly at random, it is known to the environment, the adversary and the simulator in the proof (in particular, we can assume that the environment chooses $\vec{\alpha}$ to be uniformly random in each of the experiments and hands it to the adversary/simulator).

First we observe that any call to a functionality $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_j})$ that executes less than two rounds can be simulated by the protocol that does nothing during this execution (without of course increasing the round complexity). The reason is that, by definition of flexibly wrapped CSFs (cf. Section 2.2), any such call gives no output (the first round is an input-only round). Thus, we can assume without loss of generality that π makes no call to $\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_i})$.

Next, observe that any protocol that computes $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$ can be trivially used to compute $\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_i})$ for any $i \in [\mathtt{M}]$. Indeed, one simply needs to invoke the protocol for $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$ and take its i'th output as the output of $\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_i})$. Hence, π can be trivially turned to a protocol for computing $\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_i})$ (without ever accessing $\mathcal{W}_{\text{flex}}^{D_i}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_i})$). Note that since we assume that π is FBB, the parties are given access to all other functionalities $\mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{CSF}}^{f_{\alpha_1}}), \ldots, \mathcal{W}_{\text{flex}}^{D_{i-1}}(\mathcal{F}_{\text{CSF}}^{f_{\alpha_{i-1}}}), \mathcal{W}_{\text{flex}}^{D_{i+1}}(\mathcal{F}_{\text{CSF}}^{f_{\alpha_{i+1}}}), \mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\text{CSF}}^{f_{\alpha_{\mathtt{M}}}})$, and can make oracle queries to all underlying functions $f_{\alpha_1}, \ldots, f_{\alpha_{\mathtt{M}}}$.

Finally, we observe the argument of Lemma 6.8 can be easily extended to the above scenario where we are aiming to compute $\{\mathcal{F}^{f_{\alpha_i}}\}_{f_{\alpha_i} \in \mathcal{C}}$ in an FBB manner but where the parties have oracle access to f_{α_i} (as required by the definition of FBB protocols [52]) and in addition have oracle access to $f_{\alpha_1}, \ldots, f_{\alpha_{i-1}}, f_{\alpha_{i+1}}, \ldots, f_{\alpha_{\mathtt{M}}}$ and ideal access to $\mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{CSF}}^{f_{\alpha_1}}), \ldots, \mathcal{W}_{\text{flex}}^{D_{i-1}}(\mathcal{F}_{\text{CSF}}^{f_{\alpha_{i-1}}}), \mathcal{W}_{\text{flex}}^{D_{i+1}}(\mathcal{F}_{\text{CSF}}^{f_{\alpha_{i+1}}}), \mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\text{CSF}}^{f_{\alpha_{\mathtt{M}}}})$. The reason is that the behavior of functions $f_{\alpha_1}, \ldots, f_{\alpha_{i-1}}, f_{\alpha_{i+1}}, \ldots, f_{\alpha_{\mathtt{M}}}$ is independent of f_{α_i} and can be therefore trivially simulated by means of an information-theoretic MPC protocol (recall we only have one corruption) that implements a globally accessible oracle to $f_{\alpha_1}, \ldots, f_{\alpha_{i-1}}, f_{\alpha_{i+1}}, \ldots, f_{\alpha_{\mathtt{M}}}$ and ideal functionalities $\mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{CSF}}^{f_{\alpha_1}}), \ldots, \mathcal{W}_{\text{flex}}^{D_{i-1}}(\mathcal{F}_{\text{CSF}}^{f_{\alpha_{i-1}}}), \mathcal{W}_{\text{flex}}^{D_{i+1}}(\mathcal{F}_{\text{CSF}}^{f_{\alpha_{i+1}}}), \mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\text{CSF}}^{f_{\alpha_{\mathtt{M}}}})$. (Since $\vec{\alpha}$ is chosen uniformly at random their role in computing f_{α_i} can be emulated by choosing independent α_j's for $j \in [\mathtt{M}] \setminus \{i\}$.) Thus, if there were a protocol which would compute $\{\mathcal{F}^{f_{\alpha_i}}\}_{f_{\alpha_i} \in \mathcal{C}}$ in the above hybrid world, then it can be trivially converted to a protocol which does not access the hybrids or the oracle calls to the functions other than f_{α_i}, which would contradict Lemma 6.8. Observe that the above extension is independent of rounds, and adding a PT structure to the hybrids does not affect the impossibility. □

We complete the proof of the theorem by proving Property 3 for the above choice of $\mathcal{C}_1, \ldots, \mathcal{C}_{\mathtt{M}}$ and $D_1, \ldots, D_{\mathtt{M}}$.

Lemma 6.10. *There exists no functionally black-box batched-parallel composition protocol for computing $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}})$ in the $(\mathcal{F}_{\text{SMT}}, \mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_1}), \ldots, \mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\text{SFE}}^{\mathcal{C}_{\mathtt{M}}}))$-hybrid model, tolerating an adversary actively corrupting any one of the parties.*

Proof. As before, it suffices to prove that there exists no batched-parallel composition protocol π that is secure for computing $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F}_{\text{SFE}}^{f_{\alpha_1}} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{f_{\alpha_{\mathtt{M}}}})$ in the $(\mathcal{F}_{\text{SMT}}, \mathcal{W}_{\text{flex}}^{D_1}(\mathcal{F}_{\text{SFE}}^{f_{\alpha_1}}), \ldots, \mathcal{W}_{\text{flex}}^{D_{\mathtt{M}}}(\mathcal{F}_{\text{SFE}}^{f_{\alpha_{\mathtt{M}}}}))$-hybrid model, for a uniformly random choice of $\vec{\alpha} = (\alpha_1, \ldots, \alpha_{\mathtt{M}})$.

Assume towards contradiction that such a protocol π exists, which is secure against a malicious (i.e., active) adversary corrupting any one party, and assume without loss of generality that party P_1 is corrupted. Let also $(\rho, j), (\rho', j'), \ell$ denote the values that are assumed to exists by the fact that π is a batched-parallel composition protocol and denote by $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_{\text{SFE}}^{f_{\alpha_j}})$ and $\mathcal{W}_{\text{flex}}^{D_{j'}}(\mathcal{F}_{\text{SFE}}^{f_{\alpha_{j'}}})$ the corresponding functionalities indexed by j and j'. We will denote by $\boldsymbol{x}_1 = (x_1^1, \ldots, x_1^{\mathtt{M}}) \in (\{0,1\}^\kappa)^{\mathtt{M}}$ the input of P_1 and by $\boldsymbol{x}_2 = (x_2^1, \ldots, x_2^{\mathtt{M}}) \in (\{0,1\}^\kappa)^{\mathtt{M}}$ the input of P_2 to $\mathcal{W}_{\text{flex}}^{D}(\mathcal{F}_{\text{SFE}}^{f_{\alpha_1}} \parallel \cdots \parallel \mathcal{F}_{\text{SFE}}^{f_{\alpha_{\mathtt{M}}}})$. Consider an environment that chooses all inputs to the parties uniformly at random but hands its adversary the first $\kappa - 2$ bits of the input x_2^ℓ. (Recall that the batched-parallel composition requires that P_2 inputs x_2^ℓ to $\mathcal{W}_{\text{flex}}^{D_j}(\mathcal{F}_{\text{SFE}}^{f_{\alpha_j}})$ in round ρ and to $\mathcal{W}_{\text{flex}}^{D_{j'}}(\mathcal{F}_{\text{SFE}}^{f_{\alpha_{j'}}})$ in round ρ'.)

Denote the output from the evaluation of $\mathcal{W}^D_{\text{flex}}(\mathcal{F}^{f_{\alpha_1}}_{\text{SFE}} \parallel \cdots \parallel \mathcal{F}^{f_{\alpha_{\mathtt{M}}}}_{\text{SFE}})$ as $(\boldsymbol{y}_1, \ldots, \boldsymbol{y}_n)$, where each $\boldsymbol{y}_i = (y_i^1, \ldots, y_i^{\mathtt{M}})$ is the output of P_i. Because all inputs are independently and uniformly distributed, the simulator gains no information on the missing (i.e., last) two bits of x_2^ℓ neither by using its knowledge of the α_ℓ, nor by the inputs and outputs of any of the parallelly composed $\mathcal{W}^D_{\text{flex}}(\mathcal{F}^{f_{\alpha_1}}_{\text{SFE}} \parallel \cdots \parallel \mathcal{F}^{f_{\alpha_{\mathtt{M}}}}_{\text{SFE}})$ other than its ℓ'th output. In other words, the only way that the simulator might learn additional information on the missing two last bits of the input x_2^ℓ of the honest party P_2 is from the corrupted P_1's ℓ'th output y_1^ℓ of the ideal functionality $\mathcal{W}^D_{\text{flex}}(\mathcal{F}^{f_{\alpha_1}}_{\text{SFE}} \parallel \cdots \parallel \mathcal{F}^{f_{\alpha_{\mathtt{M}}}}_{\text{SFE}})$.

In the analysis below we use the following notation. Given a string $\boldsymbol{x} = (x_1, \ldots, x_m) \in \{0,1\}^m$, denote by $\boldsymbol{x}[\text{i},\ldots,\text{j}]$, for $i < j$, the substring $(x_i, \ldots, x_j)$. Consider the following cases for the input x_1^ℓ that $\mathcal{S}$ hands to the ℓ'th functionality in $\mathcal{W}^D_{\text{flex}}(\mathcal{F}^{f_{\alpha_1}}_{\text{SFE}} \parallel \cdots \parallel \mathcal{F}^{f_{\alpha_{\mathtt{M}}}}_{\text{SFE}})$:

1. $x_1^\ell[1,\ldots,\kappa\text{-}2] \neq x_2^\ell[1,\ldots,\kappa\text{-}2] \oplus \alpha_\ell[1,\ldots,\kappa\text{-}2]$. In this case, by correctness of π, the ℓ'th output y_1^ℓ equals 0^κ independent of the last two bits of x_1^ℓ. Hence, in this case the simulator is able to output the last two bits of x_2^ℓ with probability $1/4$ (i.e., the best he can do is guess).

2. $x_1^\ell[1,\ldots,\kappa\text{-}2] = x_2^\ell[1,\ldots,\kappa\text{-}2] \oplus \alpha_\ell[1,\ldots,\kappa\text{-}2]$. In this case, we consider the following event

$$E_1 \colon x_1^\ell[\kappa\text{-}1,\kappa] = x_2^\ell[\kappa\text{-}1,\kappa] \oplus \alpha_\ell[\kappa\text{-}1,\kappa].$$

   - If E_1 occurs, then the simulator will see that the output $y_1^\ell \neq 0^\kappa$, so can output $x_2^\ell[\kappa\text{-}1,\kappa] = x_1^\ell[\kappa\text{-}1,\kappa] \oplus \alpha_\ell[\kappa\text{-}1,\kappa]$ as his guess for the last two bits of x_2^ℓ, which will always (with probability 1) be the correct guess (by definition of the function). Note that $\Pr[E_1] = 1/4$ since the simulator knows α_ℓ (by the definition of FBB [52]) and has no information on $x_2^\ell[\kappa\text{-}1,\kappa]$.
   - If E_1 does not occur, then $\mathcal{S}$ will see that $y_1^\ell = 0^\kappa$, from which it can deduce that $x_2^\ell[\kappa\text{-}1,\kappa] \neq x_1^\ell[\kappa\text{-}1,\kappa] \oplus \alpha_\ell[\kappa\text{-}1,\kappa]$, but gets no more information on $x_2^\ell[\kappa\text{-}1,\kappa]$. Hence, the probability of outputting $x_2^\ell[\kappa\text{-}1,\kappa]$ in this case is at most $1/3$ (i.e., the probability of guessing amongst the 2-bit strings that are not equal to $x_1^\ell[\kappa\text{-}1,\kappa] \oplus \alpha_\ell[\kappa\text{-}1,\kappa]$).

Hence, the total probability that a simulator outputs a correct guess for $x_2^\ell[\kappa\text{-}1,\kappa]$ is

$$P_{\mathcal{S}} \leq 1/4 + (3/4)(1/3) = 1/2.$$

To complete the proof, we will describe an adversary who outputs the two last bits of x_2^ℓ, i.e., $x_2^\ell[\kappa\text{-}1,\kappa]$, with probability noticeably higher than $1/2$; this implies a noticeable distinguishing advantage between the real world and the ideal world.

The adversary chooses two different random two-bit strings $\boldsymbol{b}, \boldsymbol{b}' \in \{0,1\}^2$ and inputs $(x_2^\ell[1,\ldots,\kappa\text{-}2], \boldsymbol{b}) \oplus \alpha_j$ on the execution of $\mathcal{W}^{D_j}_{\text{flex}}(\mathcal{F}^{f_{\alpha_j}}_{\text{SFE}})$ in round ρ and $(x_2^\ell[1,\ldots,\kappa\text{-}2], \boldsymbol{b}') \oplus \alpha_{j'}$ on the $\mathcal{W}^{D_{j'}}_{\text{flex}}(\mathcal{F}^{f_{\alpha_{j'}}}_{\text{SFE}})$ in round ρ'. Once the adversary receives P_1's outputs from the above two functionalities (denote them as $\hat{y}_{1,j}$ and $\hat{y}_{1,j'}$)[13] he does the following: If $\hat{y}_{1,j} \neq 0^\kappa$ or $\hat{y}_{1,j'} \neq 0^\kappa$ (an event that happens with probability $1/2$ since there are four possible two-bit strings and one of them makes the output $\neq 0^\kappa$) then the adversary outputs $\boldsymbol{b} \oplus \alpha_j[\kappa\text{-}1,\kappa]$ (or $\boldsymbol{b}' \oplus \alpha_{j'}[\kappa\text{-}1,\kappa]$, respectively) as his guess of the last two bits of x_2^ℓ. By correctness of the protocol, except with negligible probability the guess will be correct in this case. Otherwise, the adversary outputs a random string from $T = \{00, 01, 10, 00\} \setminus \{\boldsymbol{b}, \boldsymbol{b}'\}$; the probability of outputting a correct guess in this case is $1/2$ since it has to be one of the strings in T. Hence, the overall probability that this adversary outputs

[13] Recall that, by definition of probabilistic-termination SFE, the adversary is always able to learn the output in the second round.

the right guess for the last two bits of x_2 is $3/4 - \nu$, where ν is a negligible function implied by the error probabilities in the above protocol. Hence, the output of the adversary is distinguishable from the output of the best simulator which contradicts the assumed security of π. □

This completes the proof of the theorem. □

References

[1] G. Asharov and Y. Lindell. Utility dependence in correct and fair rational secret sharing. In S. Halevi, editor, *CRYPTO 2009*, volume 5677 of *LNCS*, pages 559–576. Springer, Aug. 2009.

[2] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs. Multiparty computation with low communication, computation and interaction via threshold FHE. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 483–501. Springer, Apr. 2012.

[3] D. Beaver, S. Micali, and P. Rogaway. The round complexity of secure protocols (extended abstract). In *22nd ACM STOC*, pages 503–513. ACM Press, May 1990.

[4] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In R. L. Probert, N. A. Lynch, and N. Santoro, editors, *2nd ACM PODC*, pages 27–30. ACM Press, Aug. 1983.

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

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

[7] G. Bracha. An asynchronou [(n-1)/3]-resilient consensus protocol. In R. L. Probert, N. A. Lynch, and N. Santoro, editors, *3rd ACM PODC*, pages 154–162. ACM Press, Aug. 1984.

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

[9] R. Canetti and T. Rabin. Universal composition with joint state. In D. Boneh, editor, *CRYPTO 2003*, volume 2729 of *LNCS*, pages 265–281. Springer, Aug. 2003.

[10] R. Canetti, J. Kilian, E. Petrank, and A. Rosen. Black-box concurrent zero-knowledge requires omega (log n) rounds. In *33rd ACM STOC*, pages 570–579. ACM Press, July 2001.

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

[12] A. Cevallos, S. Fehr, R. Ostrovsky, and Y. Rabani. Unconditionally-secure robust secret sharing with compact shares. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 195–208. Springer, Apr. 2012.

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

[14] K.-M. Chung, R. Pass, and W.-L. D. Tseng. The knowledge tightness of parallel zero-knowledge. In R. Cramer, editor, *TCC 2012*, volume 7194 of *LNCS*, pages 512–529. Springer, Mar. 2012.

[15] R. Cohen and Y. Lindell. Fairness versus guaranteed output delivery in secure multiparty computation. In *ASIACRYPT 2014, Part II*, volume 8874 of *LNCS*, pages 466–485. Springer, Dec. 2014.

[16] R. Cohen, S. Coretti, J. A. Garay, and V. Zikas. Probabilistic termination and composability of cryptographic protocols. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part III*, volume 9816 of *LNCS*, pages 240–269. Springer, Aug. 2016.

[17] R. Cohen, I. Haitner, E. Omri, and L. Rotem. Characterization of secure multiparty computation without broadcast. In E. Kushilevitz and T. Malkin, editors, *TCC 2016-A, Part I*, volume 9562 of *LNCS*, pages 596–616. Springer, Jan. 2016.

[18] R. Cramer, I. Damgård, S. Dziembowski, M. Hirt, and T. Rabin. Efficient multiparty computations secure against an adaptive adversary. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 311–326. Springer, May 1999.

[19] R. Cramer, I. Damgård, and S. Fehr. On the cost of reconstructing a secret, or VSS with optimal reconstruction phase. In J. Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 503–523. Springer, Aug. 2001.

[20] I. Damgård and Y. Ishai. Constant-round multiparty computation using a black-box pseudorandom generator. In V. Shoup, editor, *CRYPTO 2005*, volume 3621 of *LNCS*, pages 378–394. Springer, Aug. 2005.

[21] I. Damgård and J. B. Nielsen. Improved non-committing encryption schemes based on a general complexity assumption. In M. Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 432–450. Springer, Aug. 2000.

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

[23] D. Dolev, R. Reischuk, and H. R. Strong. Early stopping in byzantine agreement. *Journal of the ACM*, 37(4):720–741, 1990.

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

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

[26] M. Fitzi and J. A. Garay. Efficient player-optimal protocols for strong and differential consensus. In E. Borowsky and S. Rajsbaum, editors, *22nd ACM PODC*, pages 211–220. ACM Press, July 2003.

[27] G. Fuchsbauer, J. Katz, and D. Naccache. Efficient rational secret sharing in standard communication networks. In D. Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 419–436. Springer, Feb. 2010.

[28] J. A. Garay, J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Rational protocol design: Cryptography against incentive-driven adversaries. In *54th FOCS*, pages 648–657. IEEE Computer Society Press, Oct. 2013.

[29] S. Garg, C. Gentry, S. Halevi, and M. Raykova. Two-round secure MPC from indistinguishability obfuscation. In Y. Lindell, editor, *TCC 2014*, volume 8349 of *LNCS*, pages 74–94. Springer, Feb. 2014. doi: 10.1007/978-3-642-54242-8_4.

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

[31] S. D. Gordon, F. Liu, and E. Shi. Constant-round MPC with fairness and guarantee of output delivery. In R. Gennaro and M. Robshaw, editors, *CRYPTO 2015, Part II*, volume 9216 of *LNCS*, pages 63–82. Springer, Aug. 2015.

[32] A. Groce and J. Katz. Fair computation with rational players. In D. Pointcheval and T. Johansson, editors, *EUROCRYPT 2012*, volume 7237 of *LNCS*, pages 81–98. Springer, Apr. 2012.

[33] I. Haitner. A parallel repetition theorem for any interactive argument. In *50th FOCS*, pages 241–250. IEEE Computer Society Press, Oct. 2009.

[34] J. Y. Halpern and V. Teague. Rational secret sharing and multiparty computation: Extended abstract. In L. Babai, editor, *36th ACM STOC*, pages 623–632. ACM Press, June 2004.

[35] J. Håstad, R. Pass, D. Wikström, and K. Pietrzak. An efficient parallel repetition theorem. In D. Micciancio, editor, *TCC 2010*, volume 5978 of *LNCS*, pages 1–18. Springer, Feb. 2010.

[36] M. Hirt and V. Zikas. Adaptively secure broadcast. In H. Gilbert, editor, *EUROCRYPT 2010*, volume 6110 of *LNCS*, pages 466–485. Springer, May 2010.

[37] Y. Ishai, E. Kushilevitz, R. Ostrovsky, and A. Sahai. Zero-knowledge from secure multiparty computation. In D. S. Johnson and U. Feige, editors, *39th ACM STOC*, pages 21–30. ACM Press, June 2007.

[38] Y. Ishai, M. Prabhakaran, and A. Sahai. Founding cryptography on oblivious transfer - efficiently. In D. Wagner, editor, *CRYPTO 2008*, volume 5157 of *LNCS*, pages 572–591. Springer, Aug. 2008.

[39] Y. Ishai, R. Ostrovsky, and V. Zikas. Secure multi-party computation with identifiable abort. In J. A. Garay and R. Gennaro, editors, *CRYPTO 2014, Part II*, volume 8617 of *LNCS*, pages 369–386. Springer, Aug. 2014.

[40] Y. Ishai, E. Kushilevitz, M. Prabhakaran, A. Sahai, and C. Yu. Secure protocol transformations. In M. Robshaw and J. Katz, editors, *CRYPTO 2016, Part II*, volume 9815 of *LNCS*, pages 430–458. Springer, Aug. 2016.

[41] J. Katz and C.-Y. Koo. On expected constant-round protocols for byzantine agreement. In C. Dwork, editor, *CRYPTO 2006*, volume 4117 of *LNCS*, pages 445–462. Springer, Aug. 2006.

[42] J. Katz and C.-Y. Koo. Round-efficient secure computation in point-to-point networks. In M. Naor, editor, *EUROCRYPT 2007*, volume 4515 of *LNCS*, pages 311–328. Springer, May 2007.

[43] J. Katz, U. Maurer, B. Tackmann, and V. Zikas. Universally composable synchronous computation. In A. Sahai, editor, *TCC 2013*, volume 7785 of *LNCS*, pages 477–498. Springer, Mar. 2013.

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

[45] L. Lamport, R. E. Shostak, and M. C. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[46] P. Mukherjee and D. Wichs. Two round multiparty computation via multi-key FHE. In M. Fischlin and J. Coron, editors, *EUROCRYPT 2016*, volume 9666 of *LNCS*, pages 735–763. Springer, May 2016.

[47] S. J. Ong, D. C. Parkes, A. Rosen, and S. P. Vadhan. Fairness with an honest minority and a rational majority. In O. Reingold, editor, *TCC 2009*, volume 5444 of *LNCS*, pages 36–53. Springer, Mar. 2009.

[48] M. C. Pease, R. E. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[49] B. Pfitzmann and M. Waidner. Unconditional byzantine agreement for any number of faulty processors. In *STACS*, volume 577 of *LNCS*, pages 339–350. Springer, 1992.

[50] M. O. Rabin. Randomized byzantine generals. In *24th FOCS*, pages 403–409, Nov. 1983.

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

[52] M. Rosulek. Must you know the code of f to securely compute f? In R. Safavi-Naini and R. Canetti, editors, *CRYPTO 2012*, volume 7417 of *LNCS*, pages 87–104. Springer, Aug. 2012.

[53] T. Seito, T. Aikawa, J. Shikata, and T. Matsumoto. Information-theoretically secure key-insulated multireceiver authentication codes. In D. J. Bernstein and T. Lange, editors, *AFRICACRYPT 10*, volume 6055 of *LNCS*, pages 148–165. Springer, May 2010.

[54] J. Shikata, G. Hanaoka, Y. Zheng, and H. Imai. Security notions for unconditionally secure signature schemes. In L. R. Knudsen, editor, *EUROCRYPT 2002*, volume 2332 of *LNCS*, pages 434–449. Springer, Apr. / May 2002.

[55] C. Swanson and D. R. Stinson. Unconditionally secure signature schemes revisited. In S. Fehr, editor, *ICITS 11*, volume 6673 of *LNCS*, pages 100–116. Springer, May 2011.

[56] A. C.-C. Yao. Protocols for secure computations (extended abstract). In *23rd FOCS*, pages 160–164. IEEE Computer Society Press, Nov. 1982.

A Preliminaries (Cont'd)

A.1 Error-Correcting Secret Sharing

Definition A.1. *A (t, n)* error-correcting secret sharing (ECSS) scheme *over a message space $\mathcal{M}$ consists of a pair of algorithms $(\mathsf{Share}, \mathsf{Recon})$ satisfying the following properties:*

1. ***t-privacy:** For every $m \in \mathcal{M}$, and every subset $\mathcal{I} \subseteq [n]$ of size $|\mathcal{I}| \leq t$, the distribution of $\{s_i\}_{i\in\mathcal{I}}$ is independent of m, where $(s_1, \ldots, s_n) \leftarrow \mathsf{Share}(m)$.*
2. ***Reconstruction from up to t erroneous shares:** For every $m \in \mathcal{M}$, every $\boldsymbol{s} = (s_1, \ldots, s_n)$, and every $\boldsymbol{s}' = (s'_1, \ldots, s'_n)$ such that $\Pr_{\boldsymbol{S}\leftarrow\mathsf{Share}(m)}[\boldsymbol{S} = \boldsymbol{s}] > 0$ and $|\{i \mid s_i = s'_i\}| \geq n - t$, it holds that $m = \mathsf{Recon}(\boldsymbol{s}')$ (except for a negligible probability).*

ECSS can be constructed information-theoretically, with a negligible positive error probability, when $t < n/2$ [51, 19, 12].

A.2 Information-Theoretic Signatures

Parts of the following section are taken almost verbatim from [39].

$\mathcal{P}$-verifiable Information-Theoretic Signatures We recall the definition and construction of information-theoretic signatures [54, 53] but slightly modify the terminology to what we consider to be more intuitive. The signature scheme (in particular the key-generation algorithm) needs to know the total number of verifiers or alternatively the list $\mathcal{P}$ of their identities. Furthermore, as usually for information-theoretic primitives, the key-length needs to be proportional to the number of times that the key is used. Therefore, the scheme is parameterized by two natural numbers $\ell_{\mathcal{S}}$ and ℓ_V which will be upper bounds on the number of signatures that can be generated and verified, respectively, without violating the security.

A *$\mathcal{P}$-verifiable signature scheme* consists of a triple of randomized algorithms $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$, where:

1. $\mathsf{Gen}(1^\kappa, n, \ell_{\mathcal{S}}, \ell_V)$ outputs a pair $(\mathtt{sk}, \vec{\mathtt{vk}})$, where $\mathtt{sk} \in \{0,1\}^\kappa$ is a signing key, $\vec{\mathtt{vk}} = (\mathtt{vk}_1, \ldots, \mathtt{vk}_n) \in (\{0,1\}^\kappa)^n$ is a verification key-vector consisting of (private) verification sub-keys and $\ell_{\mathcal{S}}, \ell_V \in \mathbb{N}$.

2. $\mathsf{Sign}(m, \mathtt{sk})$ on input a message m and the signing-key $\mathtt{sk}$ outputs a signature $\sigma \in \{0,1\}^{poly(\kappa)}$.

3. $\mathsf{Ver}(m, \sigma, \mathtt{vk}_i)$ on input a message m, a signature σ and a verification sub-key $\mathtt{vk}_i$, outputs a decision-bit $d \in \{0,1\}$.

Key Generation: The algorithm for key generation $\mathsf{Gen}(1^\kappa, n, \ell_{\mathcal{S}})$ is as follows:

1. For $(j,k) \in \{0,\ldots,n-1\} \times \{0,\ldots,\ell_{\mathcal{S}}\}$: choose $a_{ij} \in_R F$ uniformly at random and set the signing key to be (the description of) the multi-variate polynomial

$$\mathtt{sk} := f(y_1, \ldots, y_{n-1}, x) = \sum_{k=0}^{\ell_{\mathcal{S}}} a_{0,k} x^k + \sum_{j=1}^{n-1} \sum_{k=0}^{\ell_{\mathcal{S}}} a_{j,k} y_j x^k.$$

2. For $i \in [n]$, choose vector $\vec{v}_i = (v_{i,1}, \ldots, v_{i,n-1}) \in_R F^{n-1}$ uniformly at random and set the i'th verification key to be

$$\mathtt{vk}_i = (\vec{v}_i, f_{\vec{v}_i}(x)),$$

where $f_{\vec{v}_i}(x) = f(v_{i,1}, \ldots, v_{i,n-1}, x)$.

Signature Generation: The algorithm for signing a message $m \in F$, given the above signing key, is (a description of) the following polynomial

$$\mathsf{Sign}(m, \mathtt{sk}) := g(y_1, \ldots, y_{n-1}) := f(y_1, \ldots, y_{n-1}, m)$$

Signature Verification: The algorithm for verifying a signature $\sigma = g(y_1, \ldots, y_n)$ on a given message m using the i'th verification key is

$$\mathsf{Ver}(m, \sigma, \mathtt{vk}_i) = \begin{cases} 1, & \text{if } g(\vec{v}_i) = f_{\vec{v}_i}(m) \\ 0, & \text{otherwise} \end{cases}$$

Figure 6: Construction of information-theoretic signatures [55]

Definition A.2. *A $\mathcal{P}$-verifiable signature scheme $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ is said to be* information-theoretically $(\ell_{\mathcal{S}}, \ell_V)$-secure *if it satisfies the following properties:*

- *(completeness) A valid signature is accepted from any honest receiver:*

$$\Pr[\mathsf{Gen} \to (\mathtt{sk}, (\mathtt{vk}_1, \ldots, \mathtt{vk}_n)); \quad \text{for } i \in [n] : (\mathsf{Ver}(m, \mathsf{Sign}(m, \mathtt{sk}), \mathtt{vk}_i) = 1)] = 1.$$

- *Let $\mathcal{O}^S_{\mathtt{sk}}$ denote a signing oracle (on input m, $\mathcal{O}^S_{\mathtt{sk}}$ outputs $\sigma = \mathsf{Sign}(m, \mathtt{sk})$) and $\mathcal{O}^V_{\vec{\mathtt{vk}}}$ denote a verification oracle (on input (m, σ, i), $\mathcal{O}^V_{\vec{\mathtt{vk}}}$ outputs $\mathsf{Ver}(m, \sigma, \mathtt{vk}_i)$). Also, let $\mathcal{A}^{\mathcal{O}^S_{\mathtt{sk}}, \mathcal{O}^V_{\vec{\mathtt{vk}}}}$ denote a computationally unbounded adversary that makes at most $\ell_{\mathcal{S}}$ calls to $\mathcal{O}^S_{\mathtt{sk}}$ and at most ℓ_V calls to $\mathcal{O}^V_{\vec{\mathtt{vk}}}$, and gets to see the verification keys indexed by some subset $\mathcal{I} \subsetneq [n]$. The following properties hold:*
  - *(unforgeability) $\mathcal{A}^{\mathcal{O}^S_{\mathtt{sk}}, \mathcal{O}^V_{\vec{\mathtt{vk}}}}$ cannot generate a valid signature on message m' of his choice,*

other than the one he queries $\mathcal{O}^S_{sk}$ on (except with negligible probability). Formally,

$$\Pr\left[\begin{array}{c} \mathsf{Gen} \to (\mathsf{sk}, \vec{\mathsf{vk}});\ \textit{for some } \mathcal{I} \subsetneq [n] \textit{ chosen by } \mathcal{A}^{\mathcal{O}^S_{sk}, \mathcal{O}^V_{\vec{vk}}}: \\ \left(A^{\mathcal{O}^S_{sk}, \mathcal{O}^V_{\vec{vk}}}(\vec{\mathsf{vk}}|_{\mathcal{I}}) \to (m, \sigma, j)\right) \quad \wedge \quad (m \textit{ was not queried to } \mathcal{O}^S_{sk}) \quad \wedge \\ (j \in [n] \setminus \mathcal{I}) \quad \wedge \quad (\mathsf{Ver}(m, \sigma, \mathsf{vk}_j) = 1) \end{array}\right] = \mathsf{negl}.$$

– *(consistency)*[14] $\mathcal{A}^{\mathcal{O}^S_{sk}, \mathcal{O}^V_{\vec{vk}}}$ *cannot create a signature that is accepted by some (honest) party and rejected by some other even after seeing ℓ_S valid signatures and verifying ℓ_V signatures (except with negligible probability). Formally,*

$$\Pr\left[\begin{array}{c} \mathsf{Gen} \to (\mathsf{sk}, \vec{\mathsf{vk}});\ \textit{for some } \mathcal{I} \subsetneq [n] \textit{ chosen by } \mathcal{A}^{\mathcal{O}^S_{sk}, \mathcal{O}^V_{\vec{vk}}}(\mathsf{sk}): \\ A^{\mathcal{O}^S_{sk}, \mathcal{O}^V_{\vec{vk}}}(\mathsf{sk}, \vec{\mathsf{vk}}|_{\mathcal{I}}) \to (m, \sigma) \\ \exists i, j \in [n] \setminus \mathcal{I} \textit{ s.t. } \mathsf{Ver}(m, \sigma, \mathsf{vk}_i) \neq \mathsf{Ver}(m, \sigma, \mathsf{vk}_j) \end{array}\right] = \mathsf{negl}.$$

In [54, 55] a signature scheme satisfying the above notion of security was constructed. These signatures have a deterministic signature generation algorithm Sign. In the following (Figure 6) we describe the construction from [54] (as described by [55] but for a single signer). We point out that the keys and signatures in the described scheme are elements of a sufficiently large finite field F (i.e., $|F| = O(2^{\mathsf{poly}(\kappa)})$; one can easily derive a scheme for strings of length $\ell = \mathsf{poly}(\kappa)$ by applying an appropriate encoding: e.g., map the i'th element of F to the i'th string (in the lexicographic order) and vice versa. We say that a value σ is *a valid signature* on message m (with respect to a given key setup $(sk, \vec{\mathtt{vk}})$), if for every honest P_i it holds that $\mathsf{Ver}(m, \sigma, \mathtt{vk}_i) = 1$.

Theorem A.3 ([55])**.** *Assuming $|F| = \Omega(2^\kappa)$ and $\ell_S = poly(\kappa)$ the above signature scheme (Figure 6) is an information-theoretically $(\ell_S, poly(\kappa))$-secure $\mathcal{P}$-verifiable* signature scheme.

B Synchronous Protocols in UC (Cont'd)

In this section, we give complementary material to Section 2.1 and in particular we include a high-level overview of the formulation of synchronous UC from [43]. More concretely, Katz et al. [43] 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.[15] In a nutshell, the clock-functionality works as follows: It stores a bit b which is initially set to 0 and it accepts from each party two types of messages: CLOCK-UPDATE and CLOCK-READ. The response to CLOCK-READ is the value of the bit b to the requestor. Each CLOCK-UPDATE is forwarded to the adversary, but it is also recorded, and upon receiving such a CLOCK-UPDATE message from all honest parties, the clock functionality updates b to $b \oplus 1$. It then keeps working as above, until it receives again a CLOCK-UPDATE message from all honest parties, in which case it resets b to $b \oplus 1$ and so on.

Such a clock can be used as follows to ensure that honest parties remain synchronized, i.e., no honest party proceeds to the next round before all (honest) parties have finished the current

[14] This property is often referred to as transferability.

[15] As argued in [43], bounded-delay channels are essential as they allow parties to detect whether or not a message was sent within a round.

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. For the sake of clarity, we do not explicitly mention $\mathcal{F}_{\text{CLOCK}}$ in our constructions.

In [43], 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.[16] We point out, that instead of the bounded-delay channels, in this work we will assume very simple CSFs 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 using the methodology from [43] 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.[17] Every party in the protocol uses exactly μ activations in each round. That is, once a party P_i observes that the round has changed, i.e., the indicator-bit b of the clock has being flipped, P_i starts its next round as described above. However, this round finishes only after P_i receives μ additional activations. Note that P_i uses these activations to execute his current round instructions; since μ is a bound to the number of hybrids used in any round by any party, μ activations are enough for the party to complete its round (If P_i finishes the round early, i.e., in less than μ activations, it simply does nothing until the μ activations are received.) Once μ activations are received in the current round, P_i sends CLOCK-UPDATE to the clock and then keeps sending CLOCK-READ message, as described above, until it observes a flip of b indicating that P_i can go to the next round.

In addition to the regular form of protocol execution, Katz et al. [43] described a way of capturing in UC the property that a protocol is guaranteed to terminate in a given number of rounds. The idea is that a synchronous protocol in regular form, which terminates after r rounds, realizes the following functionality $\mathcal{F}$. $\mathcal{F}$ keeps track of the number of times every honest party sends μ activations/messages and delivers output as soon as this has happened r times. More concretely, imitating an r-round synchronous protocol with μ activations per party per round, upon being instantiated, $\mathcal{F}$ initiates a global round-counter $\lambda = 0$ and an indicator variable $\lambda_i := 0$ for each $P_i \in \mathcal{P}$; as soon as some party P_i sends μ messages to $\mathcal{F}$, while the round-counter λ is the same, $\mathcal{F}$ sets $\lambda_i := 1$ and does the following check:[18] 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

[16] As pointed out in [43], 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 [43].

[17] 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.

[18] 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.

round λ by counting how many times every honest party has sent a certain number μ of messages, as a *synchronous functionality.* To simplify the description of our functionalities, we introduce the following terminology. We say that *a synchronous functionality $\mathcal{F}$ is in round ρ* if the current value of the above internal counter in $\mathcal{F}$ is $\lambda = \rho$.

We note that protocols in the synchronous model of [43] 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. [43] make use of the composition with joint-state (JUC) [9]. 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 [43, 9].

C The Probabilistic-Termination Framework (Cont'd)

In this section, we provide supplementary material for Section 2.2.

C.1 Canonical Synchronous Functionalities

The description of the *canonical synchronous functionality (CSF)* is given in Figure 7. 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 a few standard functionalities from the MPC literature as CSFs, we refer the reader to [16] for additional examples.

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

$\mathcal{F}_{\text{CSF}}$ proceeds as follows, parametrized by a function $f\colon (\{0,1\}^* \cup \{\bot\})^{n+1} \times R \to (\{0,1\}^*)^n$ 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 $y_1 = \ldots = y_n = \bot$, then choose $r \leftarrow R$ and compute $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n, a, r)$. Next, send $(\texttt{output}, \mathsf{sid}, y_i)$ to P_i and $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.

Figure 7: The canonical synchronous functionality

- Secure Message Transmission (aka Secure Channel). In the *secure message transmission* (SMT) functionality, a sender P_i with input x_i sends its input to P_j. The function to compute is $f^{i,j}_{\text{SMT}}(x_1, \ldots, x_n, a) = (\lambda, \ldots, x_i, \ldots, \lambda)$ (where x_i is the value of the j'th coordinate) and the leakage function is $l^{i,j}_{\text{SMT}}(x_1, \ldots, x_n) = y$, where $y = |x_i|$ in case P_j is honest and $y = x_i$ in case P_j is corrupted. We denote by $\mathcal{F}^{i,j}_{\text{SMT}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions $f^{i,j}_{\text{SMT}}$ and $l^{i,j}_{\text{SMT}}$, for sender P_i and receiver P_j.
- Broadcast. In the (standard) *broadcast* functionality, a sender P_i with input x_i distributes its input to all the parties, i.e., the function to compute is $f^{i}_{\text{BC}}(x_1, \ldots, x_n, a) = (x_i, \ldots, x_i)$. The adversary only learns the length of the message x_i before its distribution, i.e., the leakage function is $l^{i}_{\text{BC}}(x_1, \ldots, x_n) = |x_i|$. This means that the adversary does not gain new information about the input of an honest sender before the output value for all the parties is determined, and in particular, the adversary *cannot* corrupt an honest sender and change its input *after* learning the input message. We denote by $\mathcal{F}^{i}_{\text{BC}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions f^{i}_{BC} and l^{i}_{BC}, for sender P_i.
- Secure Function Evaluation. In the *secure function evaluation* functionality, the parties compute a randomized function $g(x_1, \ldots, x_n)$, i.e., the function to compute is $f^{g}_{\text{SFE}}(x_1, \ldots, x_n, a) = g(x_1, \ldots, x_n)$. The adversary learns the length of the input values via the leakage function, i.e., the leakage function is $l_{\text{SFE}}(x_1, \ldots, x_n) = (|x_1|, \ldots, |x_n|)$. We denote by $\mathcal{F}^{g}_{\text{SFE}}$ the functionality $\mathcal{F}_{\text{CSF}}$ when parametrized with the above functions f^{g}_{SFE} and l_{SFE}, for computing the n-party function g.

C.2 Reactive CSFs

In this section, we extend the notion of CSF to *reactive* CSFs (RCSFs), i.e., CSFs with multiple input/output phases. Correspondingly, a reactive CSF is parametrized by two vectors of functions $\boldsymbol{f} = (f_1, \ldots, f_q)$ and $\boldsymbol{l} = (l_1, \ldots, l_q)$. The description of reactive CSFs can be found in Figure 8.

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

$\mathcal{F}_{\text{RCSF}}$ proceeds as follows, parametrized by two vectors of functions $\boldsymbol{f} = (f_1, \ldots, f_q)$ and $\boldsymbol{l} = (l_1, \ldots, l_q)$, where $f_k \colon \left((\{0,1\}^* \cup \{\bot\})^{n+1} \times R\right)^k \to (\{0,1\}^*)^n$ and $l_k \colon ((\{0,1\}^* \cup \{\bot\})^n)^k \to \{0,1\}^*$ for each $k \in [q]$, and running with parties $\mathcal{P} = \{P_1, \ldots, P_n\}$ and an adversary $\mathcal{S}$.

- Initially, set the input values $x_{1,1}, \ldots, x_{n,q}$, the output values $y_{1,1}, \ldots, y_{n,q}$ and the adversary's values $a_1, \ldots, a_q$ to $\bot$; in addition, set the state $\boldsymbol{s}_0 = (\bot, \ldots, \bot)$.
- In round $\rho = 2k - 1$, with $k \in [q]$:
  - Upon receiving $(\texttt{adv-input}, \mathsf{sid}, v)$ from the adversary, set $a_k \leftarrow v$.
  - Upon receiving a message $(\texttt{input}, \mathsf{sid}, v)$ from some party $P_i \in \mathcal{P}$, set $x_{i,k} \leftarrow v$ and send $(\texttt{leakage}, \mathsf{sid}, P_i, l_k(x_{1,1}, \ldots, x_{n,k}))$ to the adversary.
- In round $\rho = 2k$, with $k \in [q]$:
  - Upon receiving $(\texttt{adv-input}, \mathsf{sid}, v)$ from the adversary, if $y_{1,k} = \ldots = y_{n,k} = \bot$, set $a_k \leftarrow v$. Otherwise, discard the message.
  - Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some party $P_i \in \mathcal{P}$, if $y_{1,k} = \ldots = y_{n,k} = \bot$, then choose $r_k \leftarrow R$ and compute $(y_{1,k}, \ldots, y_{n,k}) = f(\boldsymbol{s}_{k-1}, x_{1,k}, \ldots, x_{n,k}, a_k, r_k)$. Next, set, $\boldsymbol{s}_k = (\boldsymbol{s}_{k-1}, x_{1,k}, \ldots, x_{n,k}, a_k, r_k)$ send $(\texttt{output}, \mathsf{sid}, y_{i,k})$ to P_i and $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.

Figure 8: The reactive canonical synchronous functionality

Definition C.1. *Let $\mathcal{F}_{\text{RCSF}}^{\boldsymbol{f},\boldsymbol{l}}$ be a reactive CSF, with $\boldsymbol{f} = (f_1, \ldots, f_q)$ and $\boldsymbol{l} = (l_1, \ldots, l_q)$, let $t < n/2$ and let $\Pi = (\mathsf{Share}, \mathsf{Recon})$ be a (t, n) error-correcting secret-sharing scheme. For every $k \in [q]$, denote by $\tilde{f}_k$ the function that on inputs $(\tilde{x}_1, \ldots, \tilde{x}_n, a)$, with $\tilde{x}_i = (x_i, s_i)$, first reconstructs the state $\boldsymbol{s} = \mathsf{Recon}(s_1, \ldots, s_n)$, next samples random coins r and computes $(y_1, \ldots, y_n) = f_k(\boldsymbol{s}, x_1, \ldots, x_n, a, r)$, and finally shares the new state $\boldsymbol{s'} = (\boldsymbol{s}, x_1, \ldots, x_n, a, r)$ as $(s'_1, \ldots, s'_n) \leftarrow \mathsf{Share}(\boldsymbol{s'})$ and outputs $\tilde{y}_i = (y_i, s'_i)$ to P_i.*

A protocol π UC-realizes $\mathcal{W}_{\text{strict}}^{\boldsymbol{D}}(\mathcal{F}_{\text{RCSF}}^{\boldsymbol{f},\boldsymbol{l}})$, for a vector of distributions $\boldsymbol{D} = (D_1, \ldots, D_q)$, if π consists of q sub-protocols $(\pi_1, \ldots, \pi_q)$, such that for every $k \in [q]$, sub-protocol π_k UC-realizes $\mathcal{W}_{\text{strict}}^{D_k}(\mathcal{F}_{\text{CSF}}^{\tilde{f}_k, l_k})$. In addition, each party P_i in π keeps a value s_i, initially set to $\bot$, that is used as the second input for each sub-protocol. Upon completing the execution of each sub-protocol, party P_i updates s_i to be the second output value received.

C.3 Strict and Flexible Wrappers

This section contains the definitions of the strict wrapper $\mathcal{W}_{\text{strict}}$ and of the flexible wrapper $\mathcal{W}_{\text{flex}}$.

Strict-wrapper functionality. The *strict-wrapper functionality*, formally defined in Figure 9, is parametrized by (a sampler that induces) a distribution D over traces, and internally runs a copy of a CSF functionality $\mathcal{F}$. Initially, a trace T is sampled from D; this trace is given to the adversary once the first honest party provides its input. The trace T is used by the wrapper to define the termination round $\rho_{\mathsf{term}} \leftarrow c_{\mathsf{tr}}(T)$. In the first round, the wrapper forwards all the messages from the parties and the adversary to (and from) the functionality $\mathcal{F}$. Next, the wrapper essentially waits until round ρ_{term}, with the exception that the adversary is allowed to send $(\texttt{adv-input}, \mathsf{sid}, \cdot)$

messages and change its input to the function computed by the CSF. Finally, when round ρ_{term} arrives, the wrapper provides the output generated by $\mathcal{F}$ to all parties.

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

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

- Initially, sample a trace $T \leftarrow D$ and compute the output round $\rho_{\mathsf{term}} \leftarrow c_{\mathsf{tr}}(T)$. Send $(\texttt{trace}, \mathsf{sid}, T)$ to the adversary.[a]
- At all times, forward $(\texttt{adv-input}, \mathsf{sid}, \cdot)$ messages from the adversary to $\mathcal{F}$.
- In round $\rho = 1$: Forward $(\texttt{input}, \mathsf{sid}, \cdot)$ messages from each party $P_i \in \mathcal{P}$ 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 9: The strict wrapper functionality

Flexible-wrapper functionality. The *flexible-wrapper functionality*, defined in Figure 10, follows in similar lines to the strict wrapper. The difference is that the adversary is allowed to instruct the wrapper to deliver the output to each party at any round. In order to accomplish this, the wrapper assigns a termination indicator term_i, initially set to 0, to each party. Once the wrapper receives an `early-output` request from the adversary for P_i, it sets $\mathsf{term}_i \leftarrow 1$. Now, when a party P_i sends a `fetch-output` request, the wrapper checks if $\mathsf{term}_i = 1$, and lets the party receive its output in this case (by forwarding the `fetch-output` request to $\mathcal{F}$). When the guaranteed-termination round ρ_{term} arrives, the wrapper provides the output to all parties that didn't receive it yet.

C.4 Slack-Tolerant Wrappers

Slack-tolerant strict wrapper. The *slack-tolerant strict wrapper* $\mathcal{W}^{D,c}_{\text{sl-strict}}$, formally defined in Figure 11, 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 (see [16]).

Slack-tolerant flexible wrapper. The *slack-tolerant flexible wrapper* $\mathcal{W}^{D,c}_{\text{sl-flex}}$, formally defined in Figure 12, 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

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

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

$\mathcal{W}_{\text{flex}}$, 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 10: The flexible wrapper functionality

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

C.5 Compilers and Composition Theorems

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

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 11: The slack-tolerant strict wrapper functionality

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

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

$\mathcal{W}_{\text{sl-flex}}$, 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 $(\texttt{trace}, \mathsf{sid}, T)$ to the adversary.[a] 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 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 $(\texttt{input}, \mathsf{sid}, \cdot)$, send it to $\mathcal{F}$ and forward the $(\texttt{leakage}, \mathsf{sid}, \cdot)$ message $\mathcal{F}$ subsequently outputs to the adversary.
  - Else, send $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.
- In rounds $\rho > 2c + 1$:
  - Upon receiving a message $(\texttt{fetch-output}, \mathsf{sid})$ from some party $P_i \in \mathcal{P}$, proceed as follows:
    - If $\mathsf{term}_i = 1$ or $\rho = \rho_{\mathsf{term}}$, forward the message to $\mathcal{F}$, and the output $(\texttt{output}, \mathsf{sid}, y)$ to P_i. Record the output value y.
    - Else, output $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.
  - Upon receiving $(\texttt{get-output}, \mathsf{sid})$ from the adversary, if the output value y was not recorded yet, send $(\texttt{fetch-output}, \mathsf{sid})$ to $\mathcal{F}$ on behalf of some party P_i. Next, send $(\texttt{output}, \mathsf{sid}, y)$ to the adversary.
  - Upon receiving $(\texttt{early-output}, \mathsf{sid}, P_i)$ from the adversary, set $\mathsf{term}_i \leftarrow 1$ and $\rho_{\mathsf{term}} \leftarrow \min\{\rho_{\mathsf{term}}, \rho + 1\}$.

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

Figure 12: The slack-tolerant flexible wrapper functionality

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

In addition, the compiler adds the 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 $(\texttt{end}, \mathsf{sid}, y)$ for the same value y (whichever happens first), it sends $(\texttt{end}, \mathsf{sid}, y)$ to all parties.
- Upon receiving $n - t$ messages $(\texttt{end}, \mathsf{sid}, y)$ for the same value y, a party outputs y as the result of the computation and halts.

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

The compiled protocol π' UC-realizes the wrapped functionality $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$, for the adapted distribution $D^{\mathsf{full}} = \mathsf{full\text{-}trace}(D, D_1, \ldots, D_m)$. Consequently, the compiled protocol π' can handle a slack of up to c rounds, while using hybrids that are realizable themselves, and ensuring that the output slack is at most one round (as opposed to π). Calling the wrapped hybrids instead of the CSFs affects the trace corresponding to $\mathcal{F}$ in exactly the same way as in the case with deterministic termination.[19]

The probabilistic-termination compiler $\mathsf{Comp}_{\text{PTR}}$ is suitable for SNF protocols that implement a flexibly wrapped functionality, e.g., the (adjusted) protocol of Feldman and Micali [24] that realizes randomized Byzantine agreement. Indeed, such protocols introduce new slack, hence, the slack-reduction technique described above is needed to control the new slack and reduce it to $c = 1$. As pointed out in [16], in some cases the SNF protocol may realize a strictly wrapped functionality, however, some of the hybrids are to be wrapped using the flexible wrapper. An example for the latter type of probabilistic termination protocols is the protocol of Ben-Or et al. [6] that has deterministic termination in the broadcast model, yet, once the broadcast channel is implemented using randomized protocols, the obtained protocol has probabilistic termination. For that reason, a second probabilistic-termination compiler $\mathsf{Comp}_{\text{PT}}$, without the slack-reduction procedure, was introduced in [16].

[19] 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.