Probabilistic Termination and Composability of Cryptographic Protocols

Ran Cohen[1(✉)], Sandro Coretti[2], Juan Garay[3], and Vassilis Zikas[4]

[1] Department of Computer Science, Bar-Ilan University, Ramat Gan, Israel
cohenrb@cs.biu.ac.il
[2] Department of Computer Science, ETH Zurich, Zürich, Switzerland
corettis@inf.ethz.ch
[3] Yahoo Research, Sunnyvale, USA
garay@yahoo-inc.com
[4] Department of Computer Science, RPI, Troy, USA
vzikas@cs.rpi.edu

Abstract. When analyzing the round complexity of multi-party computation (MPC), one often overlooks the fact that underlying resources, such as a broadcast channel, can by themselves be expensive to implement. For example, it is impossible to implement a broadcast channel by a (deterministic) protocol in a sub-linear (in the number of corrupted parties) number of rounds. The seminal works of Rabin and Ben-Or from the early 80's demonstrated that limitations as the above can be overcome by allowing parties to terminate in different rounds, igniting the study of protocols with probabilistic termination. However, absent a rigorous simulation-based definition, the suggested protocols are proven secure in a property-based manner, guaranteeing limited composability. In this work, we define MPC with probabilistic termination in the UC framework. We further prove a special universal composition theorem for probabilistic-termination protocols, which allows to compile a protocol using deterministic-termination hybrids into a protocol that uses expected-constant-round protocols for emulating these hybrids, preserving the expected round complexity of the calling protocol.

We showcase our definitions and compiler by providing the first composable protocols (with simulation-based security proofs) for the following primitives, relying on point-to-point channels: (1) expected-constant-round perfect Byzantine agreement, (2) expected-constant-round perfect parallel broadcast, and (3) perfectly secure MPC with round complexity independent of the number of parties.

The full version of this paper can be found at the *IACR Cryptology ePrint Archive* [16].

R. Cohen—Work supported by a grant from the Israel Ministry of Science, Technology and Space (grant 3-10883) and by the National Cyber Bureau of Israel.

S. Coretti—Work supported by the Swiss NSF project no. 200020-132794.

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

V. Zikas—Work supported in part by the Swiss NSF Ambizione grant PZ00P2_142549.

M. Robshaw and J. Katz (Eds.): CRYPTO 2016, Part III, LNCS 9816, pp. 240–269, 2016.
DOI: 10.1007/978-3-662-53015-3_9

1 Introduction

In secure multi-party computation (MPC) [27,49] n parties $P_1, \ldots, P_n$ wish to jointly perform a computation on their private inputs in a secure way, so that no coalition of cheating parties can learn more information than their outputs (privacy) or affect the outputs of the computation any more than by choosing their own inputs (correctness).

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

There are several advantages in proving a protocol secure in this way. For starters, the definition of the functionality captures all security properties the protocol is supposed to have, and therefore its design process along with the security proof often exposes potential design flaws or issues that have been overlooked in the protocol design. A very important feature of many simulation-based security definitions is composability, which ensures that a protocol can be composed with other protocols without compromising its security. Intuitively, composability ensures that if a protocol $\pi^{\mathcal{G}}$ which uses a "hybrid" $\mathcal{G}$ (a broadcast channel, for example) securely realizes functionality $\mathcal{F}$, and protocol ρ securely realizes the functionality $\mathcal{G}$, then the protocol $\pi^{\rho/\mathcal{G}}$, which results by replacing in π calls to $\mathcal{G}$ by invocations of ρ, securely realizes $\mathcal{F}$. In fact, simulation-based security is the one and only way known to ensure that a protocol can be generically used to implement its specification within an arbitrary environment.

Round Complexity. The prevalent model for the design of MPC protocols is the synchronous model, where the protocol proceeds in rounds and all messages sent in any given round are received by the beginning of the next round. In fact, most if not all implemented and highly optimized MPC protocols (e.g., [15,18,20,37,43]) are in this model. When executing such synchronous protocols over large networks, one needs to impose a long round duration in order to account for potential delay at the network level, since if the duration of the rounds is too short, then it is likely that some of the messages that arrive late will be ignored or, worse, assigned to a later round. Thus, the round complexity, i.e., the number of rounds it takes for a protocol to deliver outputs, is an important efficiency metric for such protocols and, depending on the network parameters, can play a dominant role in the protocol's running time.

An issue often overlooked in the analysis of the round complexity of protocols is that the relation between a protocol's round complexity and its actual running time is sensitive to the "hybrids" (e.g., network primitives) that the protocol is assumed to have access to. For example, starting with the seminal MPC works [6,14,27,47,49], a common assumption is that the parties have access to a broadcast channel, which they invoke in every round. In reality, however, such a broadcast channel might not be available and would have to be implemented by a broadcast protocol designed for a point-to-point network. Using a standard (deterministic) broadcast protocol for this purpose incurs a linear (in n, the number of parties[1]) blow-up on the round complexity of the MPC protocol, as no deterministic broadcast protocol can tolerate a linear number of corruptions and terminate in a sublinear number of rounds [22,24]. Thus, even though the round complexity of these protocols is usually considered to be linear in the multiplicative depth d of the computed circuit, in reality their running time could become linear in nd (which can be improved to $O(n + d)$ [34]) when executed over point-to-point channels.[2]

In fact, all so-called constant-round multi-party protocols (e.g., [1,3,17,25,30,32,38,44]) rely on broadcast rounds—rounds in which parties make calls to a broadcast channel—and therefore their running time when broadcast is implemented by a standard protocol would explode to be linear in n instead of constant.[3] As the results from [22,24] imply, this is not a consequence of the specific choice of protocol but a limitation of any protocol in which there is a round such that all parties are guaranteed to have received their output; consistently with the literature on fault-tolerant distributed computing, we shall refer to protocols satisfying this property as *deterministic-termination* protocols. In fact, to the best of our knowledge, even if we allow a negligible chance for the broadcast to fail, the fastest known solutions tolerating a constant fraction of corruptions follow the paradigm from [23] (see below), which requires a poly-logarithmic (in n) number of rounds.[4]

Protocols with Probabilistic Termination. A major breakthrough in fault-tolerant distributed algorithms (recently honored with the 2015 Dijkstra Prize in Distributed Computing), was the introduction of randomization to the field by Ben-Or [4] and Rabin [46], which, effectively, showed how to circumvent the above limitation by using randomization. Most relevant to this submission, Rabin [46] showed that

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

[2] Throughout this work we will consider protocols in which all parties receive their output. If one relaxes this requirement (i.e., allows that some parties may not receive their output and give up on fairness) then the techniques of Goldwasser and Lindell [29] allow for replacing broadcast with a constant-round multi-cast primitive.

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

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

linearly resilient *Byzantine agreement* protocols [40,45] (BA, related to broadcast, possibility- and impossibility-wise) in expected *constant* rounds were possible, provided that all parties have access to a "common coin" (i.e., a common source of randomness).[5] This line of research culminated with the work of Feldman and Micali [23], who showed how to obtain a shared random coin with constant probability from "scratch," yielding a probabilistic BA protocol tolerating the maximum number of misbehaving parties ($t < n/3$) that runs in expected constant number of rounds. The randomized BA protocol in [23] works in the information-theoretic setting; these results were later extended to the computational setting by Katz and Koo [33], who showed that assuming digital signatures there exists an (expected-)constant-round protocol for BA tolerating $t < n/2$ corruptions. The speed-up on the running time in all these protocols, however, comes at the cost of uncertainty, as now they need to give up on guaranteed (eventual) termination (no fixed upper bound on their running time[6]) as well as on *simultaneous* termination (a party that terminates cannot be sure that other parties have also terminated[7]) [21]. These issues make the simulation-based proof of these protocols a very delicate task, which is the motivation for the current work.

What made the simulation-based approach a more accessible technique in security proofs was the introduction simulation-based security frameworks. The ones that stand out in this development—and are most often used in the literature—are Canetti's modular composition (aka stand-alone security) [9] and the universal composition (UC) frameworks [10,11]. The former defines security of synchronous protocols executed in isolation (i.e., only a single protocol is run at a time, and whenever a subroutine-protocol is called, it is run until its completion); the latter allows protocols to be executed alongside arbitrary (other) protocols and be interleaved in an arbitrary manner. We remark that although the UC framework is inherently asynchronous, several mechanisms have been proposed to allow for a synchronous execution within it (e.g., [11,12,36,39]).

Despite the wide-spread use of the simulation-based paradigm to prove security of protocols with deterministic termination, the situation has been quite different when probabilistic-termination protocols are considered. Here, despite the existence of round-efficient BA protocols as mentioned above [23,33], to our knowledge, no formal treatment of the problem in a simulation-based model exists, which would allow us to apply the ingenious ideas of Rabin and Ben-Or in order to speed up cryptographic protocols. We note that Katz and Koo [33] even provided an expected-constant-round MPC protocol using their fast BA protocol as a subroutine, employing several techniques to ensure proper use of randomized BA. In lack, however, of a formal treatment, existing constructions

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

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

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

are usually proved secure in a property-based manner or rely on *ad hoc*, less studied security frameworks [42].[8]

A simulation-based and composable treatment of such probabilistic-termination (PT for short) protocols would naturally allow, for example, to replace the commonly used broadcast channel with a broadcast protocol, so that the expected running time of the resulting protocol is the same as the one of the original (broadcast-hybrid) protocol. A closer look at this replacement, however, exposes several issues that have to do not only with the lack of simulation-based security but also with other inherent limitations. Concretely, it is usually the case in an MPC protocol that the broadcast channel is accessed by several (in many cases by all) parties in the same (broadcast) round in parallel. Ben-Or and El-Yaniv [5] observed that if one naïvely replaces each such invocation by a PT broadcast protocol with expected constant running-time, then the expected number of rounds until *all* broadcasts terminate is no longer constant; in fact, it is not hard to see that in the case of [23], the expected round complexity would be logarithmic in the number of instances (and therefore also in the player-set size). Nevertheless, in [5] a mechanism was proposed for implementing such parallel calls to broadcast so that the total number of rounds remains constant.

The difficulties arising with generic parallel composition are not the only issue with PT protocols. As observed by Lindell et al. [42], composing such protocols in sequence is also problematic. The main issue here is that, as already mentioned, PT protocols do not have simultaneous termination and therefore a party cannot be sure how long after he receives his output from a call to such a PT protocol he can safely carry on with the execution of the calling protocol. Although PT protocols usually guarantee a constant "slack" of rounds (say, c) in the output of any two honest parties, the naïve approach of using this property to synchronize the parties—i.e., wait c rounds after the first call, $2c$ rounds after the second call, and so on—imposes an exponential blow-up on the round complexity of the calling protocol. To resolve this, [42] proposed using fixed points in time at which a re-synchronization subroutine is executed, allowing the parties to ensure that they never get too far out-of-sync. Alternative approaches for solving this issue was also proposed in [8,33] but, again, with a restricted (property-based) proof.

Despite their novel aspects, the aforementioned results on composition of PT protocols do not use simulation-based security, and therefore it is unclear how (or if) they could be used to, for example, instantiate broadcast within a higher-level cryptographic protocol. In addition, they do not deal with other important features of modern security definitions, such as adaptive security and strict polynomial time execution. In fact, this lack of a formal cryptographic treatment places some of their claims at odds with the state-of-the-art cryptographic definitions—somewhat pointedly, [5] claims adaptive security, which, although

[8] As we discuss below, the protocol of Katz and Koo has an additional issue with adaptive security in the rushing adversary model, as defined in the UC framework, similar to the issue exploited in [31].

it can be shown to hold in a property-based definition, is not achieved by the specified construction when simulation-based security is considered (cf. Sect. 5).

Our Contributions. In this paper we provide the first formal simulation-based (and composable) treatment of MPC with probabilistic termination. Our treatment builds on Canetti's universal composition (UC) framework [10,11]. In order to take advantage of the fast termination of PT protocols, parties typically proceed at different paces and therefore protocols might need to be run in an interleaved manner—e.g., in an MPC protocol a party might initiate the protocol for broadcasting his r-round message before other parties have received output from the broadcasting of messages for round $r-1$. This inherent concurrency along with its support for synchrony makes the UC framework the natural candidate for our treatment.

Our motivating goal, which we achieve, is to provide a generic compiler that allows us to transform any UC protocol π making calls to deterministic-termination UC protocols ρ_i in a "stand-alone fashion" (similar to [9], i.e., the protocols ρ_i are invoked sequentially and in each round exactly one protocol is being executed by all the parties) into a protocol in which each ρ_i is replaced by a (faster) PT protocol ρ_i'. The compiled protocol achieves the same security as π and has (expected) round complexity proportional to $\sum_i d_i r_i$, where d_i is the expected number of calls π makes to ρ_i and r_i is the expected round complexity of ρ_i.

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

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

We then showcase our definition and composition theorem by providing simulation-based (therefore composable) probabilistic-termination protocols

and security proofs for several primitives relying on point-to-point channels: expected-constant-round perfect Byzantine agreement, expected-constant-round perfect parallel broadcast, and perfectly secure MPC with round complexity independent of the number of parties. Not surprisingly, the simulation-based treatment reveals several issues, both at the formal and at the intuitive levels, that are not present in a property-based analysis, and which we discuss along the way. We now elaborate on each application in turn. Regarding Byzantine agreement, we present a protocol that perfectly securely UC-implements the probabilistic-termination Byzantine agreement functionality for $t < n/3$ in an expected-constant number of rounds. (We will use RBA to denote probabilistic-termination BA, as it is often referred to as "randomized BA."[9]) Our protocol follows the structure of the protocol in [23], with a modification inspired by Goldreich and Petrank [28] to make it strict polynomial time (see the discussion below), and in a sense it can be viewed as the analogue for RBA of the well-known "CLOS" protocol for MPC [13]. Indeed, similarly to how [13] converted (and proved) the "GMW" protocol [26] from statically secure in the stand-alone setting into an adaptively secure UC version, our work transforms the broadcast and BA protocols from [23] into adaptively UC-secure randomized broadcast and RBA protocols.[10]

Our first construction above serves as a good showcase of the power of our composition theorem, demonstrating how UC-secure RBA is built in a modular manner: First, we de-compose the sub-routines that are invoked in [23] and describe simple(r) (SFE-like) functionalities corresponding to these sub-routines; this provides us with a simple "backbone" of the protocol in [23] making calls to these hybrids, which can be easily proved to implement expected-constant-round RBA. Next, we feed this simplified protocol to our compiler which outputs a protocol that implements RBA from point-to-point secure channels; our composition theorem ensures that the resulting protocol is also expected constant round.

There is a sticky issue here that we need to resolve for the above to work: the protocol in [23] does not have guaranteed termination and therefore the distribution of the terminating round is not sampleable by a strict probabilistic polynomial-time (PPT) machine.[11] A way around this issue would be to modify the UC model of execution so that the corresponding ITMs are expected PPTs. Such a modification, however, would impact the UC model of computation, and would therefore require a new proof of the composition theorem—a trickier task than one might expect, as the shift to expected polynomial-time simulation is known to introduce additional conceptual and technical difficulties (cf. [35]),

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

[10] As we show, the protocol in [23] does not satisfy input independence, and therefore is not adaptively secure in a simulation-based manner (cf. [31]).

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

whose resolution is beyond the scope of this work. Instead, here we take a different approach which preserves full compatibility with the UC framework: We adapt the protocol from [23] using ideas from [28] so that it implements a functionality which samples the terminating round with almost the same probability distribution as in [23], but from a finite (linear-size) domain; as we show, this distribution is sampleable in strict polynomial time and can therefore be used by a standard UC functionality.

Next, we use our composition theorem to derive the first simulation-based and adaptively (UC) secure parallel broadcast protocol, which guarantees that all broadcast values are received within an expected constant number of rounds. This extends the results from [5,33] in several ways: first, our protocol is perfectly UC-secure which means that we can now use it within a UC-secure SFE protocol to implement secure channels, and second, it is adaptively secure against a rushing adversary.[12]

Finally, by applying once again our compiler to replace calls to the broadcast channel in the SFE protocol by Ben-Or, Goldwasser, and Wigderson [6] (which, recall, is perfectly secure against $t < n/3$ corruptions in the broadcast-hybrid model [2]) by invocations to our adaptively secure UC parallel broadcast protocol, we obtain the first UC-secure PT MPC protocol in the point-to-point secure channels model with (expected) round complexity $O(d)$, independently of the number of parties, where d is the multiplicative depth of the circuit being computed. As with RBA, this result can be seen as the first analogue of the UC compiler by Canetti et al. [13] for SFE protocols with probabilistic termination.

We stress that the use of perfect security to showcase our composition theorem is just our choice and not a restriction of our composition theorem. In fact, our theorem can be also applied to statistically or computationally secure protocols. Moreover, if one is interested in achieving better constants in the (expected) round complexity then one can use SFE protocols that attempt to minimize the use of the broadcast channel (e.g., [34]). Our composition theorem will give a direct methodology for this replacement and will, as before, eliminate the dependency of the round complexity from the number of parties.[13]

2 Model

We consider n parties $P_1, \ldots, P_n$ and an adaptive *t-adversary*, i.e., the adversary corrupts up to t parties during the protocol execution.[14] We work in the UC model and assume the reader has some familiarity with its basics. To capture synchronous protocols in UC we use the framework of Katz et al. [36]. Concretely,

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

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

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

the assumption that parties are synchronized is captured by assuming that the protocol has access to a "clock" functionality $\mathcal{F}_{\text{CLOCK}}$. The functionality $\mathcal{F}_{\text{CLOCK}}$ maintains an indicator bit which is switched once *all honest parties* request the functionality to do it. At any given round, a party asks $\mathcal{F}_{\text{CLOCK}}$ to turn the bit on only after having finished with all operations for the current round. Thus, this bit's value can be used to detect when every party has completed his round, in which case they can proceed to the next round. As a result, this mechanism ensures that no party sends his messages for round $r + 1$ before every party has completed round r. For clarity, we retain from writing this clock functionality in our theorem statement; however, all our results assume access to such a clock functionality.

In the communication network of [36], parties have access to bounded-delay secure channels. These channels work in a so-called "fetch" mode, i.e., in order to receive his output the receiver issues a `fetch-output` command. This allows to capture the property of a channel between a sender P_s and a receiver P_r, delaying the delivery of a message by an amount δ: as soon as the sender P_s submits an input y (message to be sent to the receiver) the channel functionality starts counting how many times the receiver requests it.[15] The first $\delta - 1$ such `fetch-output` requests (plus all such requests that are sent before the sender submits input) are ignored (and the adversary is notified about them); the δth `fetch-output` request following a submitted input y from the sender results in the channel sending $(\texttt{output}, y)$ to P_r. In this work we take an alternative approach and model secure channels as special simple SFE functionalities. These SFEs also work in a fetch mode[16] and provide the same guarantee as the bounded-delay channels.

There are two important considerations in proving the security of a synchronous UC protocol: (1) The simulator needs to keep track of the protocol's current round, and (2) because parties proceed at the same pace, they can synchronize their reaction to the environment; most fully synchronous protocols, for example, deliver output exactly after a given number of rounds. In [36] this property is captured as follows: The functionality keeps track of which round the protocol would be in by counting the number of activations it receives from honest parties. Thus, if the protocol has a regular structure, where every party advances the round after receiving a fixed number μ of activations from its environment (all protocols described herein will be in this form), the functionality can easily simulate how rounds in the protocol advance by incrementing its round index whenever it receives μ messages from all honest parties; we shall refer to such a functionality as a *synchronous functionality*. Without loss of generality, in this work we will describe all functionalities for $\mu = 1$, i.e., once a functionality receives a message from every party it proceeds to the simulation of the next protocol round. We stress that this is done to simplify the description, and the

[15] Following the simplifying approach of [36], we assume that communication channels are single use, thus each message transmission uses an independent instance of the channel.

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

in an actual evaluation, as in the synchronous setting of [36], in order to give the simulator sufficiently many activations to perform its simulation, functionalities typically have to wait for $\mu > 1$ messages from each party where the last $\mu - 1$ of these messages are typically "dummy" activations (usually of the type `fetch-output`).

To further simplify the description of our functionalities, we introduce the following terminology. We say that *a synchronous functionality $\mathcal{F}$is in round ρ* if the current value of the above internal round counter in $\mathcal{F}$ is $r = \rho$. All synchronous functionalities considered in this work have the following format: They treat the first message they receive from any party P_i as P_i's input[17]—if this message is not of the right form $(\texttt{input}, \cdot)$ then a default value is taken as P_i input; as soon as an honest party sends its first message, any future message by this party is treated as a `fetch-output` message.

3 Secure Computation with Probabilistic Termination

The work of Katz et al. [36] addresses (synchronous) cryptographic protocols that terminate in a fixed number of rounds for all honest parties. However, as mentioned in Sect. 1, Ben-Or [4] and Rabin [46] showed that in some cases, great asymptotic improvements on the *expected* termination of protocols can be achieved through the use of randomization. Recall, for example, that in the case of BA, even though a lower bound of $O(n)$ on the round complexity of any deterministic BA protocol tolerating $t = \Omega(n)$ corruptions exists [22,24], Rabin's global-coin technique (fully realized later on in [23]) yields an expected-constant-round protocol. This speed-up, however, comes at a price, namely, of relinquishing both *fixed* and *simultaneous* termination [21]: the round complexity of the corresponding protocols may depend on random choices made during the execution, and parties may obtain output from the protocol in different rounds.

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

3.1 Canonical Synchronous Functionalities

At a high level, $\mathcal{F}_{\text{CSF}}$ corresponds to a generalization of the UC secure function evaluation (SFE) functionality to allow for potential leakage on the inputs to the

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

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

$\mathcal{F}_{\text{CSF}}$ proceeds in two rounds: in the first round all the parties hand $\mathcal{F}_{\text{CSF}}$ their input values, and in the second round each party receives its output. This is very similar to the standard (UC) SFE functionality; the difference here is that whenever some input is submitted to $\mathcal{F}_{\text{CSF}}$, the adversary is handed some leakage function of this input—similarly, for example, to how UC secure channels leak the message length to the adversary. The adversary can use this leakage when deciding the inputs of corrupted parties. Additionally, he is allowed to input an extra message, which—depending on the function f—might affect the output(s). The detailed description of $\mathcal{F}_{\text{CSF}}$ is given in Fig. 1.

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

$\mathcal{F}_{\text{CSF}}$ proceeds as follows, parametrized by a function $f\colon (\{0,1\}^* \cup \{\perp\})^{n+1} \to (\{0,1\}^*)^n$ and a leakage function $l\colon (\{0,1\}^* \cup \{\perp\})^n \to \{0,1\}^*$, 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 $\perp$.
- 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 $y_1 = \ldots = y_n = \perp$, set $a \leftarrow v$. Otherwise, discard the message.
  - Upon receiving $(\texttt{fetch-output}, \mathsf{sid})$ from some party $P_i \in \mathcal{P}$, if $y_1 = \ldots = y_n = \perp$ compute $(y_1, \ldots, y_n) = f(x_1, \ldots, x_n, a)$. Next, send $(\texttt{output}, \mathsf{sid}, y_i)$ to P_i and $(\texttt{fetch-output}, \mathsf{sid}, P_i)$ to the adversary.

Fig. 1. The canonical synchronous functionality

Next, we point out a few technical issues about the description of $\mathcal{F}_{\text{CSF}}$. Following the simplifications from Sect. 2, $\mathcal{F}_{\text{CSF}}$ advances its round as soon as it receives $\mu = 1$ message from each honest party. This ensures that the adversary cannot make the functionality stall indefinitely. Thus, formally speaking, the functionality $\mathcal{F}_{\text{CSF}}$ is not well-formed (cf. [13]), as its behavior depends on the identities of the corrupted parties.[19] We emphasize that the non-well-formedness

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

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

relates only to advancing the rounds, and is unavoidable if we want to restrict the adversary not to block the evaluation indefinitely (cf. [36]).

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

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

3.2 Probabilistic Termination in UC

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

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

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

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

To avoid this issue and allow for an efficient simulation, we have $\mathcal{F}$ output all the coins that were used for sampling round ρ_{term} to $\mathcal{S}$. Because $\mathcal{S}$ knows the round sampler algorithm, it can reproduce the entire computation of the sampler and use it in its simulation. In fact, as we discuss below, it suffices for our proofs to have $\mathcal{F}$ output a *trace* of its choices to the simulator instead of all the coins that were used

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

to sample this trace. In the remainder of this section, we motivate and formally describe our formulation of such traces. The formal description of the wrappers, which in particular sample traces, can then be found at the end of this section.

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

Given the above, the round sampler of a functionality needs to keep track of how the functionality was realized. This can be achieved via the notion of *trace*. A trace basically records which hybrids were called by a protocol, and in a recursive way, for each hybrid, which hybrids would have been called by a protocol realizing that hybrid. The recursion ends with the hybrids that are "assumed" by the model, called *atomic* functionalities.[21]

Building on our running illustration above, suppose protocol π' (realizing $\mathcal{G}$) makes ideal hybrid calls to $\mathcal{F}$ and to some atomic functionality $\mathcal{H}$. Assume that in an example execution, π' happens to make (sequential) calls to instances of $\mathcal{H}$ and $\mathcal{F}$ in the following order: $\mathcal{F}$, then $\mathcal{H}$, and finally $\mathcal{F}$ again. Moreover, assume that $\mathcal{F}$ is replaced by protocol π (realizing $\mathcal{F}$) and that π happens to make two (sequential) calls to $\mathcal{H}$ upon the first invocation by π', and three (sequential) calls to $\mathcal{H}$ the second time. Then, this would result in the trace depicted in Fig. 2.

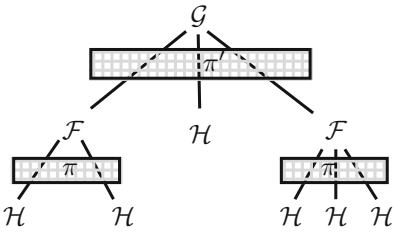

Fig. 2. Example of an execution trace

Assume that π is a probabilistic-termination protocol and π' a deterministic-termination protocol. Consequently, this means that $\mathcal{F}$ is in fact a flexibly wrapped functionality of some CSF $\mathcal{F}'$, i.e., $\mathcal{F} = \mathcal{W}_{\text{flex}}^{D_{\mathcal{F}}}(\mathcal{F}')$, where the distribution $D_{\mathcal{F}}$ samples (from a distribution induced by π) depth-1 traces with root $\mathcal{W}_{\text{flex}}^{D_{\mathcal{F}}}(\mathcal{F}')$ and leaves $\mathcal{H}$.[22] Similarly, $\mathcal{G}$ is a strictly wrapped functionality

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

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

of some CSF $\mathcal{G}'$, i.e., $\mathcal{G} = \mathcal{W}_{\text{strict}}^{D_{\mathcal{G}}}(\mathcal{G}')$, where the distribution $D_{\mathcal{G}}$ first samples (from a distribution induced by π') a depth-1 trace with root $\mathcal{W}_{\text{strict}}^{D_{\mathcal{G}}}(\mathcal{G}')$ and leaves $\mathcal{W}_{\text{flex}}^{D_{\mathcal{F}}}(\mathcal{F}')$ as well as $\mathcal{H}$. Then, each leaf node $\mathcal{W}_{\text{flex}}^{D_{\mathcal{F}}}(\mathcal{F}')$ is replaced by a trace (independently) sampled from $D_{\mathcal{F}}$. Thus, the example trace from Fig. 2 would look as in Fig. 3.

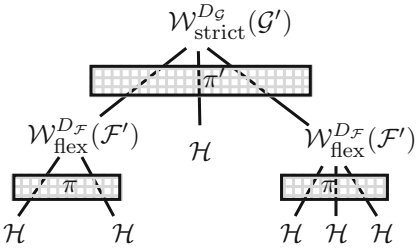

Fig. 3. An execution trace with probabilistic-termination and deterministic-termination protocols

Formally, a trace is defined as follows:

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

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

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

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

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

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

Fig. 4. The strict-wrapper functionality

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.

Fig. 5. The flexible-wrapper functionality

Flexible-Wrapper Functionality. The *flexible-wrapper functionality*, defined in Fig. 5, follows in similar lines to the strict wrapper. The difference is that the adversary is allowed to instruct the wrapper to deliver the output to each party at any round. In order to accomplish this, the wrapper assigns a termination indicator term_i, initially set to 0, to each party. Once the wrapper receives an `early-output` request from the adversary for P_i, it sets $\mathsf{term}_i \leftarrow 1$. Now, when a party P_i sends a `fetch-output` request, the wrapper checks if $\mathsf{term}_i = 1$, and lets the party receive its output in this case (by forwarding the `fetch-output` request to $\mathcal{F}$). When the guaranteed-termination round ρ_{term} arrives, the wrapper provides the output to all parties that didn't receive it yet.

4 (Fast) Composition of PT Protocols

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

This "slack" can be reduced, however, only to a difference of one round, unless one is willing to pay a linear blow-up in round complexity [22,24]. Hence, all protocols must be modified to deal with a non-simultaneous start of (at least) one round, and protocols that introduce slack must be followed by a slack-reduction procedure. Since this is a tedious, yet systematic task, in this section we provide a generic compiler that transforms protocols designed in a simpler "stand-alone" setting, where all parties remain synchronized throughout the protocol (and no slack and round-complexity issues arise) into UC protocols that deal with these issues while maintaining their security.

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

Definition 2 (SNF). *Let $\mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities. A synchronous protocol π in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model is in* synchronous normal form (SNF) *if in every round exactly one ideal functionality $\mathcal{F}_i$ is invoked by all honest parties, and in addition, no honest party hands inputs to other CSFs before this instance halts.*

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

SNF protocols are designed as an intermediate step, since the hybrid functionalities $\mathcal{F}_i$ are two-round CSFs, and can, in general, not be realized by real-world protocols. To that end, we define a protocol compiler that transforms SNF protocols into (non-SNF) protocols making calls to wrapped hybrids that *can*

be realized in the real world, while maintaining their security and asymptotic (expected) round complexity. At the same time, the compiler takes care of any potential slack that is introduced by the protocol and ensures that the protocol can be executed even if the parties do not start the protocol simultaneously.

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

4.1 Composition with Deterministic Termination

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

Slack-Tolerant Strict Wrapper. The *slack-tolerant strict wrapper* $\mathcal{W}^{D,c}_{\text{sl-strict}}$, formally defined in Fig. 6, is parametrized by an integer $c \geq 0$, which denotes the

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

$\mathcal{W}^{D,c}_{\text{sl-strict}}$, 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 $(\mathtt{trace}, \mathsf{sid}, T)$ to the adversary.[a] Initialize slack indicators $c_1, \ldots, c_n \leftarrow 0$.
- At all times, forward $(\mathtt{adv\text{-}input}, \mathsf{sid}, \cdot)$ messages from the adversary to $\mathcal{F}$.
- In rounds $\rho = 1, \ldots, 2c + 1$: Upon receiving a message from some party $P_i \in \mathcal{P}$, proceed as follows:
  - If the message is $(\mathtt{input}, \mathsf{sid}, \cdot)$, forward it to $\mathcal{F}$, forward the $(\mathtt{leakage}, \mathsf{sid}, \cdot)$ message $\mathcal{F}$ subsequently outputs to the adversary, and set P_i's local slack $c_i \leftarrow \rho - 1$.
  - Else, send $(\mathtt{fetch\text{-}output}, \mathsf{sid}, P_i)$ to the adversary.
- In rounds $\rho > 2c + 1$: Upon receiving a message $(\mathtt{fetch\text{-}output}, \mathsf{sid})$ from some party $P_i \in \mathcal{P}$, proceed as follows:
  - If $\rho = \rho_{\mathsf{term}} + c_i$, send the message to $\mathcal{F}$, and the output $(\mathtt{output}, \mathsf{sid}, y_i)$ to P_i.
  - Else, send $(\mathtt{fetch\text{-}output}, \mathsf{sid}, P_i)$ to the adversary.

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

Fig. 6. The slack-tolerant strict wrapper functionality

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

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

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

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

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

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

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

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

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 in $\mathcal{W}_{\text{sl-strict}}^{D^{\text{full}},c}$.

The proof of Theorem 1 can be found in the full version [16].

4.2 Composition with Probabilistic Termination

The composition theorem in Sect. 4.1 does not work if the protocol π itself introduces slack (e.g., the fast broadcast protocol by Feldman and Micali [23]) or if one of the hybrids needs to be replaced by a slack-introducing protocol (e.g., instantiating the broadcast hybrids with fast broadcast protocols in BGW [6]).

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

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

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

where the blow-up factor B_c is as explained in Sect. 4.1, and the additional factor of 2 results from the termination protocol described below for every flexibly wrapped CSF, which increases the round complexity by at most two additional rounds (recall that $\mathsf{flex}_{\mathsf{tr}}(T)$ denotes the number of such CSFs), and c is due to the potential slack. $\mathcal{W}_{\text{sl-flex}}$ allows the adversary to deliver output at any round prior to 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 above termination protocol.

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

$\mathcal{W}^{D,c}_{\text{sl-flex}}$, 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 $\rho_{\mathsf{term}} \leftarrow B_c \cdot c_{\mathsf{tr}}(T) + B' \cdot \mathsf{flex}_{\mathsf{tr}}(T) + c$, where $B_c := 3c + 1$ and $B' = 2$. 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 $(\texttt{adv-input}, \cdot)$ 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.
  - 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.
    - 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 copmuted 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.

Fig. 7. The slack-tolerant flexible wrapper functionality

Probabilistic-Termination Compilers. 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 $\mathcal{W}^{D}_{\text{flex}}(\mathcal{F})$, for some distribution D, in the $(\mathcal{F}_1, \ldots, \mathcal{F}_m)$-hybrid model, assuming all parties start at the same round. Define the following compiler $\mathsf{Comp}_{\text{PTR}}$, parametrized by a slack parameter $c \geq 0$. It 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 $\mathcal{W}_{\text{sl-strict}}$; it replaces every call to a CSF $\mathcal{F}_i$ with a call to the wrapped CSF $\mathcal{W}^{D_i,c}_{\text{sl-flex}}(\mathcal{F}_i)$ if $i \in I$ or to $\mathcal{W}^{D_i,c}_{\text{sl-strict}}(\mathcal{F}_i)$ if $i \notin I$.

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

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

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

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

The following theorem states that the compiled protocol π' UC-realizes the wrapped functionality $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$, again for an adapted distribution D^{full}. Consequently, the compiled protocol π' can handle a slack of up to c rounds, while using hybrids that are realizable themselves, and ensuring that the output slack is at most one round (as opposed to π). Calling the wrapped hybrids instead of the CSFs affects the trace corresponding to $\mathcal{F}$ in exactly the same way as in the case with deterministic termination (cf. Sect. 4.1).[24]

Theorem 2. *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, and let π an SNF protocol that UC-realizes $\mathcal{W}^{D}_{\text{flex}}(\mathcal{F})$, for some 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. 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, protocol $\mathsf{Comp}^{c}_{\text{PTR}}(\pi, D_1, \ldots, D_m, I)$ UC-realizes $\mathcal{W}^{D^{\mathsf{full}},c}_{\text{sl-flex}}(\mathcal{F})$ in the $(\mathcal{W}(\mathcal{F}_1), \ldots, \mathcal{W}(\mathcal{F}_m))$-hybrid model, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds, 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 $i \notin I$.

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.

The proof of Theorem 2 can be found in the full version [16].

Consider now the scenario where SNF protocol π realizes a strictly wrapped functionality, yet soem of the CSF hybrids are to be wrapped by flexible wrappers. The corresponding compiler $\mathsf{Comp}_{\text{PT}}$ works as $\mathsf{Comp}_{\text{PTR}}$ except that it does not perform the slack-reduction protocol in the end. The proof of the following theorem follows that of Theorem 2.

Theorem 3. *Let $\mathcal{F}, \mathcal{F}_1, \ldots, \mathcal{F}_m$ be canonical synchronous functionalities, and let π an SNF protocol that UC-realizes $\mathcal{W}^{D}_{\text{strict}}(\mathcal{F})$ for some 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. Let $I \subseteq [m]$ be the subset (of indices) of functionalities to*

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

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* $\mathcal{F}_i$ *for every* $i \in I$ *is a public-output functionalities.*

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

The expected round complexity of the compiled protocol 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.*

4.3 Wrapping Secure Channels

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

Parallel SMT. The *parallel secure message transmission functionality* $\mathcal{F}_{\text{PSMT}}$ is a CSF for the following functions f_{PSMT} and l_{PSMT} Each party P_i has a vector of input values $(x^i_1, \ldots, x^i_n)$ such that x^i_j is sent from P_i to P_j. That is, the function to compute is $f_{\text{PSMT}}((x^1_1, \ldots, x^1_n), \ldots, (x^n_1, \ldots, x^n_n), a) = ((x^1_1, \ldots, x^1_1), \ldots, (x^1_n, \ldots, x^n_n))$. As we consider rushing adversaries, that can determine the messages sent by the corrupted parties *after* receiving the messages sent by the honest parties, the leakage function should leak the messages that are to be delivered from honest parties to corrupted parties. Therefore, the leakage function is $l_{\text{PSMT}}((x^1_1, \ldots, x^1_n), \ldots, (x^n_1, \ldots, x^n_n)) = (y^1_1, y^1_2, \ldots, y^n_{n-1}, y^n_n)$, where $y^i_j = |x^i_j|$ in case P_j is honest and $y^i_j = x^i_j$ in case P_j is corrupted.

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

A standard solution to overcome asynchrony by a constant number of rounds $c \geq 0$, introduced by Lindell et al. [41] and used by Katz and Koo [33], is to expand each communication round to $2c + 1$ rounds. Each party listens for

messages throughout all $2c + 1$ rounds, and sends its own messages in round $c + 1$. It is straight-forward to verify that if the slack is c, i.e., the parties start within $c + 1$ rounds from each other, round r-messages (in the original protocol, without round-expansion) are sent, and delivered, before round $(r + 1)$-messages and after round $(r - 1)$-messages.

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

As a result, in protocol π_{PSMT}, each round is expanded to $3c+1$ rounds, where during the final c rounds, parties are simply idle and ignore any messages they receive. Denote by D_{PSMT} the deterministic distribution that outputs a depth-1 trace consisting of a single leaf $\mathcal{F}_{\text{PSMT}}$. In the full version [16] of this paper, we prove the following lemma.

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

5 Applications of Our Fast Composition Theorem

In this section we demonstrate the power of our framework by providing some concrete applications. All of the protocols we present in this section enjoy perfect security facing adaptive adversaries corrupting less than a third of the parties.

5.1 Fast and Perfectly Secure Byzantine Agreement

We start by describing the binary and multi-valued randomized Byzantine agreement protocols (the definition of $\mathcal{F}_{\text{BA}}$ appears in Sect. 3.1). These protocols are based on techniques due to Feldman and Micali [23] and Turpin and Coan [48], with modifications to work in the UC framework. We provide simulation-based proofs for these protocols.

At a high level, protocol π_{RBA} proceeds as follows. Initially, each party sends its input to all other parties over a point-to-pint channel using $\mathcal{F}_{\text{PSMT}}$, and sets its

vote to be its input bit. Next, the parties proceed in phases, where each phase consists of invoking the oblivious coin functionality $\mathcal{F}_{\text{OC}}$ (see the full version) followed by a voting process consisting of three rounds of sending messages via $\mathcal{F}_{\text{PSMT}}$. The voting ensures that (1) if all honest parties agree on their votes at the beginning of the phase, they will terminate at the end of the phase, (2) in each phase, all honest parties will agree on their votes at the end of each phase with probability at least p, and (3) if an honest party terminates in some phase then all honest parties will terminate with the same value by the end of the next phase. In the negligible event that the parties do not terminate after $\tau = \log^{1.5}(k) + 1$ phases, the parties use the Byzantine agreement functionality $\mathcal{F}_{\text{BA}}$ in order to ensure termination.

In the full version [16] we prove the following theorem.

Theorem 4. *Let $c \geq 0$ and $t < n/3$. There exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}_{\text{BA}}^{\{0,1\}})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

5.2 Fast and Perfectly Secure Parallel Broadcast

As discussed in Sect. 1 composing protocols with probabilistic termination naïvely does not retain expected round complexity. Ben-Or and El-Yaniv [5] constructed an elegant protocol for probabilistic-termination parallel broadcast[25] with a constant round complexity in expectation, albeit under a property-based security definition. In this section we adapt the [5] protocol to the UC framework and show that it does *not* realize the parallel broadcast functionality, but rather a weaker variant which we call *unfair parallel broadcast*. Next, we show how to use unfair parallel broadcast in order to compute (fair) parallel broadcast in constant excepted number of rounds.

In a standard broadcast functionality (cf. Sect. 3.1), the sender provides a message to the functionality which delivers it to the parties. Hirt and Zikas [31] defined the *unfair* version of the broadcast functionality, in which the functionality informs the adversary which message it received, and allows the adversary, based on this information, to corrupt the sender and replace the message. Following the spirit of [31], we now define the unfair parallel broadcast functionality, using the language of CSF.

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

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

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}.

In the full version [16] we present an adaptation of the [5] protocol, show that it perfectly UC-realizes (a wrapped version of) $\mathcal{F}_{\text{UPBC}}$, and prove the following result.

Theorem 5. *Let $c \geq 0$ and $t < n/3$. There exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}_{\text{UPBC}})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within $c + 1$ consecutive rounds.*

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

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

Unfortunately, the unfair parallel broadcast protocol π_{UPBC} (see the full version [16]) fails to realize (a wrapped version of) the standard parallel broadcast functionality $\mathcal{F}_{\text{PBC}}$. The reason is similar to the argument presented in [31]: in the first round of the protocol, each party distributes its input, and since we consider a rushing adversary, the adversary learns the messages *before* the honest parties do. It follows that the adversary can corrupt a party *before* the honest parties receive the message and replace the message to be delivered. This attack cannot be simulated in the ideal world where the parties interact with $\mathcal{F}_{\text{PBC}}$, since by the time the simulator learns the broadcast message in the ideal world, the functionality does not allow to change it.

Although protocol π_{UPBC} does not realize $\mathcal{F}_{\text{PBC}}$, it can be used in order to construct a protocol that does. Each party commits to its input value before any party learns any new information, as follows. Each party, in parallel, first secret shares its input using a t-out-of-n secret-sharing protocol.[26] In the second step, every party, in parallel, broadcast a vector with all the shares he received, by use of the above unfair parallel broadcast functionality $\mathcal{F}_{\text{UPBC}}$, and each share is reconstructed based on the announced values. The reason this modification achieves fair broadcast is the following: If a sender P_i is not corrupted until he distributes his shares, then a t-adversary has no way of modifying the reconstructed output of P_i's input, since he can at most affect $t < n/3$ shares. Thus, the only

[26] In [31] verifiable secret sharing (VSS) is used; however, as we argue, this is not necessary.

way the adversary can affect any of the broadcast messages is by corrupting the sender independently of his input, an attack which is easily simulated. We describe this protocol, denoted π_{PBC}, in Fig. 8.

Protocol π_{PBC}

1. In the first round, upon receiving $(\texttt{input}, \mathsf{sid}, x_i)$ with $x_i \in V$ from the environment, P_i secret shares x_i using a t-out-of-n secret sharing scheme, denoted by $(x_i^1, \ldots, x_i^n)$. Next, P_i sends for every party P_j its share (sid, x_i^j) (via $\mathcal{F}_{\text{PSMT}}$). Denote by x_j^i the value received from P_j.
2. In the second round, P_i broadcasts the values $\boldsymbol{x}_i = (x_1^i, \ldots, x_n^i)$ using the unfair parallel broadcast functionality, i.e., P_i sends $(\texttt{input}, \mathsf{sid}, \boldsymbol{x}_i)$ to $\mathcal{F}_{\text{UPBC}}$. Denote by $\boldsymbol{y}_j = (y_1^j, \ldots, y_n^j)$ the value received from P_j. Now, P_i reconstructs all the input values, i.e., for every $j \in [n]$ reconstructs y_j from the shares $(y_j^1, \ldots, y_j^n)$, and outputs $(\texttt{output}, \mathsf{sid}, (y_1, \ldots, y_n))$.

Fig. 8. The parallel broadcast protocol, in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{UPBC}})$-hybrid model

We conclude with the following theorem, see the full version [16] for the proof.

Theorem 6. *Let $c \geq 0$ and $t < n/3$. There exists an efficiently sampleable distribution D such that the functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}_{\text{PBC}})$ has an expected constant round complexity, and can be UC-realized in the $\mathcal{F}_{\text{SMT}}$-hybrid model, with perfect security, in the presence of an adaptive malicious t-adversary, assuming that all honest parties receive their inputs within $c+1$ consecutive rounds.*

5.3 Fast and Perfectly Secure SFE

We conclude this section by showing how to construct a perfectly UC-secure SFE protocol which computes a given circuit in expected $O(d)$ rounds, independently of the number of parties, in the point-to-point channels model. The protocol is obtained by taking the protocol from [6],[27] denoted π_{BGW}. This protocol relies on (parallel) broadcast and (parallel) point-to-point channels, and therefore it can be described in the $(\mathcal{F}_{\text{PSMT}}, \mathcal{F}_{\text{PBC}})$-hybrid model. It follows from Theorem 3, that the compiled protocol $\mathsf{Comp}^c_{\text{PT}}(\pi_{\text{BGW}})$ UC-realizes the corresponding wrapped functionality $\mathcal{W}^{D,c}_{\text{sl-flex}}(\mathcal{F}_{\text{SFE}})$ (for an appropriate distribution D), in the $(\mathcal{W}^{D^{\text{full}}_{\text{PSMT}},c}_{\text{sl-strict}}(\mathcal{F}_{\text{PSMT}}), \mathcal{W}^{D^{\text{full}}_{\text{PBC}},c}_{\text{sl-flex}}(\mathcal{F}_{\text{PBC}}))$-hybrid model, resulting in the following.

Theorem 7. *Let f be an n-party function, C an arithmetic circuit with multiplicative depth d computing f, and $t < n/3$. Then there exists an efficiently*

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

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

References

1. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012)
2. Asharov, G., Lindell, Y.: A full proof of the BGW protocol for perfectly-secure multiparty computation. Electron. Colloquium Comput. Complex. (ECCC) **18**, 36 (2011)
3. Beaver, D., Micali, S., Rogaway, P.: The round complexity of secure protocols (extended abstract). In: 22nd ACM STOC, pp. 503–513. ACM Press, May 1990
4. Ben-Or, M.: Another advantage of free choice: completely asynchronous agreement protocols (extended abstract). In: Probert, R.L., Lynch, N.A., Santoro, N. (eds.) 2nd ACM PODC, pp. 27–30. ACM Press, August 1983
5. Ben-O, M., El-Yaniv, R.: Resilient-optimal interactive consistency in constant time. Distrib. Comput. **16**(4), 249–262 (2003)
6. Ben-Or, M., Goldwasser, S., Wigderson, A.: Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). In: 20th ACM STOC, pp. 1–10. ACM Press, May 1988
7. Bracha, G.: An asynchronou [(n − 1)/3]-resilient consensus protocol. In: Probert, R.L., Lynch, N.A., Santoro, N. (eds.) 3rd ACM PODC, pp. 154–162. ACM Press, August 1984
8. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Kilian, J. (ed.) CRYPTO 2001. LNCS, vol. 2139, pp. 524–541. Springer, Heidelberg (2001)
9. Canetti, R.: Security and composition of multiparty cryptographic protocols. J. Cryptology **13**(1), 143–202 (2000)
10. Canetti, R.: Universally composable security: a new paradigm for cryptographic protocols. In: 42nd FOCS, pp. 136–145. IEEE Computer Society Press, October 2001
11. Canetti, R.: Universally composable signatures, certification and authentication. Cryptology ePrint Archive, Report 2003/239 (2003). http://eprint.iacr.org/2003/239
12. Canetti, R., Cohen, A., Lindell, Y.: A simpler variant of universally composable security for standard multiparty computation. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 3–22. Springer, Heidelberg (2015)
13. Canetti, R., Lindell, Y., Ostrovsky, R., Sahai, A.: Universally composable two-party and multi-party secure computation. In: 34th ACM STOC, pp. 494–503. ACM Press, May 2002
14. Chaum, D., Crépeau, C., Damgård, I.: Multiparty unconditionally secure protocols (extended abstract). In: 20th ACM STOC, pp. 11–19. ACM Press, May 1988
15. Choi, S.G., Katz, J., Malozemoff, A.J., Zikas, V.: Efficient three-party computation from cut-and-choose. In: Garay, J.A., Gennaro, R. (eds.) CRYPTO 2014, Part II. LNCS, vol. 8617, pp. 513–530. Springer, Heidelberg (2014)

16. Cohen, R., Coretti, S., Garay, J.A., Zikas, V.: Probabilistic termination and composability of cryptographic protocols. Cryptology ePrint Archive, Report 2016/350 (2016). http://eprint.iacr.org/
17. Damgård, I.B., Ishai, Y.: Constant-round multiparty computation using a black-box pseudorandom generator. In: Shoup, V. (ed.) CRYPTO 2005. LNCS, vol. 3621, pp. 378–394. Springer, Heidelberg (2005)
18. Damgård, I., Keller, M., Larraia, E., Pastro, V., Scholl, P., Smart, N.P.: Practical covertly secure MPC for dishonest majority – or: breaking the SPDZ limits. In: Crampton, J., Jajodia, S., Mayes, K. (eds.) ESORICS 2013. LNCS, vol. 8134, pp. 1–18. Springer, Heidelberg (2013)
19. Damgård, I., Nielsen, J.B.: Adaptive versus static security in the UC model. In: Chow, S.S.M., Liu, J.K., Hui, L.C.K., Yiu, S.M. (eds.) ProvSec 2014. LNCS, vol. 8782, pp. 10–28. Springer, Heidelberg (2014)
20. Damgård, I., Pastro, V., Smart, N., Zakarias, S.: Multiparty computation from somewhat homomorphic encryption. In: Safavi-Naini, R., Canetti, R. (eds.) CRYPTO 2012. LNCS, vol. 7417, pp. 643–662. Springer, Heidelberg (2012)
21. Dolev, D., Reischuk, R., Raymond Strong, H.: Early stopping in Byzantine agreement. J. ACM **37**(4), 720–741 (1990)
22. Dolev, D., Raymond Strong, H.: Authenticated algorithms for Byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983)
23. Feldman, P., Micali, S.: An optimal probabilistic protocol for synchronous Byzantine agreement. SIAM J. Comput. **26**(4), 873–933 (1997)
24. Fischer, M.J., Lynch, N.A.: A lower bound for the time to assure interactive consistency. Inf. Process. Lett. **14**(4), 183–186 (1982)
25. Garg, S., Gentry, C., Halevi, S., Raykova, M.: Two-round secure MPC from indistinguishability obfuscation. In: TCC, pp. 74–94 (2014)
26. Goldreich, O., Micali, S., Wigderson, A.: Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). In: 27th FOCS, pp. 174–187. IEEE Computer Society Press, October 1986
27. Goldreich, O., Micali, S., Wigderson, A.: How to play any mental game or a completeness theorem for protocols with honest majority. In: Aho, A. (ed.) 19th ACM STOC, pp. 218–229. ACM Press, May 1987
28. Goldreich, O., Petrank, E.: The best of both worlds: guaranteeing termination in fast randomized Byzantine agreement protocols. Inf. Process. Lett. **36**(1), 45–49 (1990)
29. Goldwasser, S., Lindell, Y.: Secure multi-party computation without agreement. J. Cryptology **18**(3), 247–287 (2005)
30. Dov Gordon, S., Liu, F.-H., Shi, E.: Constant-round MPC with fairness and guarantee of output delivery. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 63–82. Springer, Heidelberg (2015)
31. Hirt, M., Zikas, V.: Adaptively secure broadcast. In: Gilbert, H. (ed.) EUROCRYPT 2010. LNCS, vol. 6110, pp. 466–485. Springer, Heidelberg (2010)
32. Ishai, Y., Prabhakaran, M., Sahai, A.: Founding cryptography on oblivious transfer– efficiently. In: Wagner, D. (ed.) CRYPTO 2008. LNCS, vol. 5157, pp. 572–591. Springer, Heidelberg (2008)
33. Katz, J., Koo, C.-Y.: On expected constant-round protocols for Byzantine agreement. In: Dwork, C. (ed.) CRYPTO 2006. LNCS, vol. 4117, pp. 445–462. Springer, Heidelberg (2006)
34. Katz, J., Koo, C.-Y.: Round-efficient secure computation in point-to-point networks. In: Naor, M. (ed.) EUROCRYPT 2007. LNCS, vol. 4515, pp. 311–328. Springer, Heidelberg (2007)

35. Katz, J., Lindell, Y.: Handling expected polynomial-time strategies in simulation-based security proofs. In: Kilian, J. (ed.) TCC 2005. LNCS, vol. 3378, pp. 128–149. Springer, Heidelberg (2005)
36. Katz, J., Maurer, U., Tackmann, B., Zikas, V.: Universally composable synchronous computation. In: Sahai, A. (ed.) TCC 2013. LNCS, vol. 7785, pp. 477–498. Springer, Heidelberg (2013)
37. Keller, M., Scholl, P., Smart, N.P.: An architecture for practical actively secure MPC with dishonest majority. In: Sadeghi, A.-R., Gligor, V.D., Yung, M. (eds.) ACM CCS 2013, pp. 549–560. ACM Press, November 2013
38. Kilian, J.: Founding cryptography on oblivious transfer. In: 20th ACM STOC, pp. 20–31. ACM Press, May 1988
39. Kushilevitz, E., Lindell, Y., Rabin, T.: Information-theoretically secure protocols and security under composition. In: Kleinberg, J.M. (ed.) 38th ACM STOC, pp. 109–118. ACM Press, May 2006
40. Lamport, L., Shostak, R.E., Pease, M.C.: The Byzantine generals problem. ACM Trans. Program. Lang. Syst. **4**(3), 382–401 (1982)
41. Lindell, Y., Lysyanskaya, A., Rabin, T.: On the composition of authenticated Byzantine agreement. In: 34th ACM STOC, pp. 514–523. ACM Press, May 2002
42. Lindell, Y., Lysyanskaya, A., Rabin, T.: Sequential composition of protocols without simultaneous termination. In: Ricciardi, A. (ed.) 21st ACM PODC, pp. 203–212. ACM Press, July 2002
43. Lindell, Y., Pinkas, B., Smart, N.P., Yanai, A.: Efficient constant round multiparty computation combining BMR and SPDZ. In: Gennaro, R., Robshaw, M. (eds.) CRYPTO 2015. LNCS, vol. 9216, pp. 319–338. Springer, Heidelberg (2015)
44. Mukherjee, P., Wichs, D.: Two round multiparty computation via multi-key FHE. In: Fischlin, M., Coron, J.-S. (eds.) EUROCRYPT 2016. LNCS, vol. 9666, pp. 735–763. Springer, Heidelberg (2016). doi:10.1007/978-3-662-49896-5_26
45. Pease, M.C., Shostak, R.E., Lamport, L.: Reaching agreement in the presence of faults. J. ACM **27**(2), 228–234 (1980)
46. Rabin, M.O.: Randomized Byzantine generals. In: 24th Annual Symposium on Foundations of Computer Science, Tucson, Arizona, USA, 7–9 November 1983, pp. 403–409. IEEE Computer Society (1983)
47. Rabin, T., Ben-Or, M.: Verifiable secret sharing and multiparty protocols with honest majority (extended abstract). In: 21st ACM STOC, pp. 73–85. ACM Press, May 1989
48. Turpin, R., Coan, B.A.: Extending binary Byzantine agreement to multivalued Byzantine agreement. Inf. Process. Lett. **18**(2), 73–76 (1984)
49. Yao, A.C.-C.: Protocols for secure computations (extended abstract). In: 23rd FOCS, pp. 160–164. IEEE Computer Society Press, November 1982