SoK: A Consensus Taxonomy in the Blockchain Era

Juan Garay* Aggelos Kiayias†

Wednesday 29th August, 2018

Abstract

Consensus (a.k.a. Byzantine agreement) is arguably one of the most fundamental problems in distributed systems, playing also an important role in the area of cryptographic protocols as the enabler of a secure broadcast functionality. While the problem has a long and rich history and has been analyzed from many different perspectives, recently, with the advent of blockchain protocols like Bitcoin, it has experienced renewed interest from a much wider community of researchers and has seen its application expand to various novel settings.

One of the main issues in consensus research is the many different variants of the problem that exist as well as the various ways the problem behaves when different setup, computational assumptions and network models are considered. In this work we perform a systematization of knowledge in the landscape of consensus research starting with the original formulation in the early 1980s up to the present blockchain-based new class of consensus protocols. Our work is a roadmap for studying the consensus problem under its many guises, classifying the way it operates in many settings and highlighting the exciting new applications that have emerged in the blockchain era.

1 Introduction

The consensus problem—reaching agreement distributedly in the presence of faults—has been extensively studied in the literature starting with the seminal work of Shostak, Pease and Lamport [PSL80, LSP82]. The traditional setting of the problem involves parties connected by point-to-point channels, possibly using digital signatures in order to ensure the integrity of the information that is exchanged in the course of the protocol. For a relatively recent overview of the many variants of consensus that are considered in the distributed systems literature see Cachin *et al.* [CGR11]. Tolerating "Byzantine" behavior, i.e., the presence of parties that may behave arbitrarily, possibly in malicious ways, has been one of the hallmark features in the study of the problem.

Bitcoin was introduced by Nakamoto in 2008-2009 [Nak08a, Nak09], with the objective of providing a payment system that is decentralized in the sense of not relying on a central authority that should be trusted for transactions to be considered as final. Expectedly, the fundamental enabling component of the Bitcoin system is a consensus mechanism that facilitates agreement on the history of transactions. Given the conflicting interests of the Bitcoin protocol participants, such a system should be resilient to Byzantine behavior, which brings us to the main contribution of Bitcoin in the context of the consensus problem, namely, a non-traditional and novel approach from the perspective of distributed computing to solve the problem in a setting that until then had not received sufficient attention.

In light of these developments, it is important to rethink the consensus problem in the blockchain era and organize the landscape that is currently being formed, acknowledging all the new directions and novel tools that have become available in the context of consensus protocol design.

*Texas A&M University, garay@cse.tamu.edu.

†University of Edinburgh and IOHK, akiayias@inf.ed.ac.uk. Research partly supported by H2020 Project Priviledge # 780477.

One main aspect of our work is to look into the consensus problem from a modeling perspective providing the definitions needed to understand the problem and the solutions that have been developed over the years both in the traditional and the newer blockchain settings. In the course of this, we provide a taxonomy of protocols and impossibility results that comprehensively outline what is currently known about consensus and which questions continue to remain open. Also important is to "extract" the relevant consensus question that is particular to the Bitcoin protocol, which we term "ledger consesus" (sometimes referred to as Nakamoto consensus), and which is an instance of the state machine replication problem that has been long-studied in distributed systems [Sch90].

Consequently, we provide in this paper precise definitions of the relevant versions of consensus that have been investigated and systematize the existing knowledge about the problem with respect to (i) the network model, (ii) trusted setup assumptions, and (iii) computational assumptions under which, and at what cost in terms of running time and communication overhead, the problem can be solved.

We emphasize that our approach is problem-centric and the results being overviewed conceptual and fundamental in nature, thus complementing the various other enumerative surveys of results and publications on the subect (e.g., [BSA+17, SJS+18]).

Organization of the paper. We start in section 2 by specifying a model of multi-party protocol execution and how protocols' properties will be deemed satisfied, as well as presenting the definition of (variants of) the consensus problem. We then specify the available resources and assumptions mentioned above under which the problem has been studied: Network assumptions (communication primitives, synchrony) in Section 3.1; trusted setup assumptions (no setup, public-state setup, private-state setup) in Section 4; and computational assumptions (none, one-way functions, proofs-of-work, random oracle) in Section 5. We then overview possibility (i.e., constructions) and impossibility results for consensus with respect to number of parties as a function of misbehaving parties (resp., honest vs malicious computational power), trusted setup, running time and communication costs in the traditional (point-to-point communication) setting (Section 6), and in the Bitcoin (peer-to-peer) setting (Section 7). Finally, we dedicate Section 8 to the treatment of ledger consensus, starting with the definition of the problem, followed by the evaluation of existing results through a similar lens as in the case of (standard) consensus in the previous section.

Some of the material (specifically, the ideal specification of some of the resources available to the protocols) is presented in the appendix.

2 Model and Definitions

2.1 Protocol execution

In order to provide a formal description of protocols and their executions it is useful to consider a formal model of computation. We will choose the Interactive Turing Machine (ITM) based resource bounded model put forth by [Gol01, Can00b]. An ITM is like a Turing Machine but with the addition of an incoming and an outgoing communication tape as well as an identity tape and a "subroutine" tape. When an instance of an ITM is generated (we will henceforth call this an ITI, standing for interactive Turing machine instance), the identity tape is initialized to a specific value that remains constant throughout the instance's execution. The ITI may communicate with other ITI's by writing to its outgoing communication tape.

Let us consider a protocol Π that is modeled as an ITM. Ideally, we would like to consider the execution of this protocol in an arbitrary setting, i.e., with an arbitrary set of parties and arbitrary configuration. A common way to model this in cryptography and distributed systems is to consider that a certain program, thought of as an adversary, produces this configuration and therefore the properties of the protocol should hold for any possible choice of that program, potentially with some restrictions that are explicitly defined. The advantage of this particular modeling approach is that

it obviates the need to quantify over all the details that concern the protocol (and substitutes them with a single universal quantification over all such "environments").

Suppose now that we have a protocol Π that is specified as an ITM and we would like to consider all possible executions of this protocol in the presence of an adversary $\mathcal{A}$, that is also modeled as an ITM. We capture this by specifying a pair of ITM's $(\mathcal{Z}, C)$, called the "environment" and the "control program" respectively. The environment $\mathcal{Z}$ is given some input which may be trivial (like a security parameter 1^κ) and is allowed to "spawn" new ITI's using the programs of Π and $\mathcal{A}$. By convention, only a single instance of $\mathcal{A}$ will be allowed. Spawning such new instances is achieved by writing a single message to its outgoing tape which is read by C. The control program is responsible for approving such spawning requests by $\mathcal{Z}$. Subsequently, all communication of the instances that are created will be routed via C, i.e., C will be receiving the instances' outgoing messages and will be approving whether the can be forwarded to the receiving parties' incoming tape. Note that this may be used to simulate the existence of point to point channels, nevertheless we will take a more general approach. Specifically, the control function C, will by definition only permit outgoing messages of running ITI's to be sent to the adversary $\mathcal{A}$ (with instructions for further delivery). This captures the fact that the network cannot be assumed to be safe for the instances that are communicating during the protocol execution. Beyond writing messages that are routed though $\mathcal{A}$, ITIs can also spawn additional ITIs as prescribed by the rules hardcoded in C. This enables instances of a protocol Π to invoke subroutines that can assist in its execution. These subroutines, can be sub-protocols or instances of "ideal functionalities" that may be accessible by more than a single running instance.

Given these features, the above approach provides a comprehensive framework for reasoning about protocol executions. Nevertheless, some care needs to be applied to ensure that the total execution run time of the $(\mathcal{Z}, C)$ system remains polynomial-time: it is easy to see that even if all ITIs are assumed to be polynomially bounded, the total execution run time may not be. By introducing a more refined time keeping capability for ITIs, it was proven in [Can00b] that as long as the ITM $\mathcal{Z}$ is "locally polynomial bounded" and C is a polynomial-time computable function, the execution of $(\mathcal{Z}, C)$ as described above when $\mathcal{Z}$ is given input κ is polynomial-time bounded in κ (cf. Proposition 3 in [Can00b]). From this point on we will assume always that $\mathcal{Z}$ is locally polynomial bounded and, for brevity, we will just state that it is polynomial bounded.

Functionalities. We will next need to specify the "resources" that may be available to the instances running protocol Π. For instance, access to a reliable point-to-point channel or a "diffuse" channel (see below). To allow for the most general way to specify such resources we will follow the approach of describing them as "ideal functionalities" in the terminology of [Can00b]. In simple terms, an ideal functionality is another ITM that may interact with instances running in parallel in the protocol execution. The critical feature of ideal functionalities though is that they can be spawned by ITI's running protocol Π. In such case, the protocol Π is defined with respect to the functionality $\mathcal{F}$. The ideal functionality may interact with the adversary $\mathcal{A}$ as well as other ITI's running the protocol Π. One main advantage of using the concept of an ideal functionality in our setting, is that we can capture various different communication resources that may be available to the participants running the protocol. For instance, a secure channel functionality may be spawned to transmit a message between two instances of Π that will only leak the length of the message to the adversary. As another example, a message passing functionality may ensure that all parties are activated prior to advancing to the next communication round (see below for synchronous vs. asynchronous executions).

Execution of multiparty protocols. When protocol instances are spawned by $\mathcal{Z}$ they will be initialized with an identity which is available to the program's code as well as, possibly, with the identities of other instances that may run in parallel (this is at the discretion of the environment program $\mathcal{Z}$). The identities themselves may be useful to the program instance, as they may be used by the instance to address other instances running in parallel. We will use the notation $\mathsf{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa)$ to denote an *execution* of the protocol Π with an adversary $\mathcal{A}$ and an environment $\mathcal{Z}$. The execution is a string

that is formed by the concatenation of all messages and all ITI states at each step of the execution of the system $(\mathcal{Z}, C)$. The total length of the execution will be polynomial in κ. The parties' inputs are provided by the environment $\mathcal{Z}$ which also receives the parties' outputs. Parties that receive no input from the environment remain inactive. We denote by INPUT() the input tape of each party.

We note that by adopting the resource bounded computation modeling of systems of ITM's by [Can00b] we obviate the need of imposing a strict upper bound on the number of messages that may be transmitted by the adversary in each activation. In our setting, honest parties, at the discretion of the environment, are given sufficient time to process all messages delivered by any communication functionality given to them as a resource. It follows that denial of service attacks cannot be used to the adversary's advantage in the analysis (they are out of scope from our perspective of studying the consensus problem).

Properties of protocols. In our theorems we will be concerned with *properties* of protocols Π. Such properties will be defined as predicates over the random variable $\mathsf{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa)$ by quantifying over all possible adversaries $\mathcal{A}$ and environments $\mathcal{Z}$. Note that protocols may only satisfy some properties with a small probability of error in κ (formally, a function that is *negligible* in κ or $\mathsf{negl}(\kappa)$) as well as in potentially other parameters. The probability space is determined by the private coins of all participants and the functionalities they employ. Formally we have the following.

Definition 1. Given a predicate Q we say that *the protocol* Π *satisfies property* Q provided that for all polynomial-time $\mathcal{Z}, \mathcal{A}$ the probability that $Q(\mathsf{VIEW}_{\Pi,\mathcal{A},\mathcal{Z}}(\kappa))$ is false is $\mathsf{negl}(\kappa)$.

Note that we will only consider properties that are polynomial-time computable predicates. Our notion of execution will capture the single-session setting for the protocol Π hence properties Q will be single-session properties.

Asynchronous vs. synchronous execution. The model is able to capture various flavors of synchrony. This is achieved by abstracting the network communication as a functionality and specifying how the adversary may interfere with message delivery. The functionality may keep track of parties' activations and depending on the case ensure that parties will be given a chance to act as the protocol execution advances.

Static vs. dynamic environments. The model we present capture both static and dynamic environments. Specifically, it is suitable for protocols that run with a fixed number of parties that should be known to all participants in advance but it can also allow protocols for which the number of participants is not known beforehand and, in fact, it may not even be known during the course of the execution. Note that in order to allow for proper ITI intercommunication we will always assume that the total set of parties is known, nevertheless, only a small subset of them may be active in a particular moment during the protocol execution.

Setup assumptions. In a number of protocols, there is a need to have some pre-existing configuration (such as the knowledge of a common reference string (CRS), or a public-key infrastructure (PKI)). Such setup assumptions can be also captured as separate functionalities $\mathcal{F}$ that are available to the running protocol ITI's.

Permissioned vs. permissionless networks. In the context of the consensus problem, this terminology became popular with the advent of blockchain protocols. The Bitcoin blockchain protocol is the prototypical "permissionless" protocol where read access to the ledger is unrestricted and write access (in the form of posting transactions) can be obtained by anyone that possesses bitcoin (which may be acquired, in principle, by anyone that is running the Bitcoin client and invests computational power solving proofs of work). On the other hand, a permissioned protocol imposes more stringent

access control on the read and write operations that are available as well as with respect to who can participate in the protocol. Extrapolating from the terminology as applied in the ledger setting, a permissionless consensus protocol would enable any party to participate and contribute input for consideration of the other parties. With this in mind, the traditional setting of consensus is permissioned, since only specific parties are allowed to participate; on the other hand, consensus in the blockchain setting can be either permissioned or permissionless.

Cryptographic primitives. Some standard cryptographic primitives will be useful in the description of the consensus protocols. We overview them below. A digital signature scheme consists of three PPT algorithms (Gen, Sign, Verify) such that $(vk, sk) \leftarrow \mathsf{Gen}(1^\kappa)$ generates a public-key/secret-key pair, $\sigma \leftarrow \mathsf{Sign}(sk, m)$ signs a message m, $\mathsf{Verify}(vk, m, \sigma)$ returns 1 if and only if σ is a valid signature for m given vk. A digital signature scheme is existentially unforgeable, if for any PPT adversary $\mathscr{A}$, that has access to a $\mathsf{Sign}(sk, \cdot)$ oracle, the event that $\mathscr{A}$ returns some (m, σ) such that $\mathsf{Verify}(vk, m, \sigma) = 1$ has measure $\mathsf{negl}(\kappa)$, where the probability is taken over the coin tosses of the algorithms. A collision resistant (keyed) hash function family $\{H_k\}_{k \in K}$ has the property that $H_k : \{0,1\}^* \to \{0,1\}^\kappa$, it is efficiently computable and the probability to produce $x \neq y$ with $H_k(x) = H_k(y)$ given k is $\mathsf{negl}(\kappa)$.

2.2 The consensus problem

As mentioned earlier, *consensus* (aka *Byzantine agreement*), formulated by Shostak, Pease and Lamport [PSL80, LSP82], is one of the fundamental problems in the areas of fault-tolerant distributing computing and cryptographic protocols, in particular secure multi-party computation [Yao82, GMW86, BGW88, CCD87]. In the consensus problem, n parties attempt to reach agreement on a value from some fixed domain V, despite the malicious behavior of up to t of them. More specifically, every party P_i starts the consensus protocol with an initial value $v \in V$, and every run of the protocol must satisfy (except possibly for some negligible probability) the following conditions:

- *Termination:* All honest parties decide on a value.
- *Agreement:* If two honest parties decide on v and w, respectively, then $v = w$.
- *Validity:* If all honest parties have the same initial value v, then all honest parties decide on v.

The domain V can be arbitrary, but frequently the case $V = \{0,1\}$ is considered given the efficient transformation of binary agreement protocols to the multi-valued case [TC84].

There exist various measures of quality of a consensus protocol: its *resiliency*, expressed as the fraction $(\frac{t}{n})$ of misbehaving parties a protocol can tolerate; its running time—worst number of rounds by which honest parties terminate; and its communication complexity—worst total number of bits/messages communicated during a protocol run.

In the consensus problem, all the parties start with an initial value. A closely related variant is the single-source version of the problem (aka the *Byzantine Generals* problem [LSP82], or simply (reliable or secure) "broadcast"), where only a distinguished party—the *sender*—has an input. In this variant, both the Termination and Agreement conditions remain the same, and Validity becomes:

- *Validity:* If the sender is honest and has initial value v, then all honest parties decide on v.

A stronger, albeit natural, version of the consensus problem requires that the output value be one of the honest parties' inputs, a distinction that is only important in the case of non-binary inputs. In this version, called *strong consensus* [Nei94], the Validity condition becomes:

- *Strong Validity:* If the honest parties decide on v, then v is the input of some honest party.

Note that the distinction with the standard version of the problem is only relevant in the case of non-binary inputs. Further, the resiliency bounds for this version also depend on $|V|$ (see Section 6).

Another way to enhance validity is to require that the output of an honest party conforms to an *external* predicate Q [CKPS01]. In this setting, each input v is accompanied by a proof π and is supposed to satisfy $Q(v,\pi) = 1$ (for instance, π can be a digital signature on v and Q would be verifying its validity). Note that the resulting guarantee is weaker than strong validity (since it could be the case that the decision is made on an input suggested by a corrupted party), but nevertheless it can be suitable in a multi-valued setting where only externally validated inputs are admissible as outputs.

Finally, we point out that, traditionally, consensus problems have been specified as above, in a *property-based* manner. Protocols for the problem are then proven secure/correct by showing how the properties (e.g., the Agreement, Validity and Termination conditions) are met. Nowadays, however, it is widely accepted to formulate the security of a protocol via the "trusted-party paradigm" (cf. [GMW86, Gol01]), where the protocol execution is compared with an ideal process where the outputs are computed by a trusted party that sees all the inputs. A protocol is then said to securely carry out the task if running the protocol with a realistic adversary amounts to "emulating" the ideal process with the appropriate trusted party. One advantage of such a simulation-based approach is that it simultaneously captures *all* the properties that are guaranteed by the ideal world, without having to enumerate some list of desired properties. Simulation-based definitions are also useful for applying *composition theorems* (e.g., [Can00a, Can00b]) that enable proving the security of protocols that use other protocols as sub-routines, which typically would be the case for consensus and/or broadcast protocols.

The above captures the classical definition of the consensus problem. A related and recently extensively studied version of the problem is state-machine replication or "ledger" consensus that we will treat in Section 8.

On the necessity of an honest majority. Regardless of the resources available to the parties in the protocol execution, an upper bound of (less than) $n/2$ can be shown for resiliency (see, for example, [Fit03]). Specifically, consider a set n of parties that are equally divided with respect to their initial values between inputs 0 and 1, and an adversary that with 1/3 probability corrupts no one (case 1), with 1/3 probability corrupts the parties that have input 0 (case 2) and with 1/3 probability corrupts the parties that have input 1 (case 3). In any case, the adversarial parties follow the protocol. Observe that case 1 requires from the honest parties to converge to a common output (due to Agreement), while in the other two cases the honest parties should output 0 (case 2) and 1 (case 3). However, all three cases are perfectly indistinguishable in the view of the honest parties and as a result a logical contradiction ensues.

3 Network Assumptions

3.1 Communication primitives

Consensus protocols are described with respect to a network layer that enables parties to send messages to each other. An important distinction we will make is between point-to-point connectivity vs. message "diffusion" as it manifests in a peer-to-peer communication setting.

Point-to-point channels. In this setting parties are connected with pairwise reliable and authentic channels. We call that resource RMT, for *reliable message transmission*. When a party sends a message it specifies its recipient as well as the message contents and it is guaranteed that the recipient will receive it. The recipient can identify the sender as the source of the message. Refer to Appendix A for the specification of the RMT ideal functionality. In such fixed connectivity setting, all parties are aware of the set of parties running the protocol. Full connectivity has been the standard

communication setting for consensus protocols, see [LSP82], although sparse connectivity has also been considered (cf. [DPPU88, Upf92]). We present the functionality for RMT in the appendix.

In terms of measuring communication costs in this model, it will be simpler for us to use the (maximum) total number of messages in a protocol run, rather than the total number of communicated bits, assuming a suitable message size. See, e.g., [Fit03] (Chapter 3) for a detailed account of the communication complexity of consensus (and broadcast) protocols.

Peer-to-peer diffusion. This setting is motivated by peer-to peer message transmission that happens via "gossiping," i.e., messages received by a party are passed along on to the party's peers. We refer to this basic message passing operation as "Diffuse." Message transmission is not authenticated and it does not preserve the order of messages in the views of different parties. When a message is diffused by an honest party, there is no specific recipient and it is guaranteed that all activated honest parties will receive the same message. Nonetheless, the source of the message may be "spoofed" and thus the recipient may not reliably identify the source of the message,[1] and when the sender is malicious not everyone is guaranteed to receive the same message. Contrary to the point-to-point channels setting, parties may neither be aware of the identities of the parties running the protocol nor their precise number. The ideal functionality capturing the diffuse operation is also presented in Appendix A.

In order to measure the total communication costs of peer-to-peer diffusion, one needs to take into account the underlying network graph. The typical deployment setting will be a sparse constant-degree graph for which it holds that the number of edges equals $O(n)$. In such setting, each invocation of the primitive requires $O(n)$ messages to be transmitted in the network.

Relation between the communication primitives. It is easy to see that given RMT, there is a straightforward, albeit inefficient, protocol that simulates Diffuse; given a message to be diffused, the protocol using RMT, will send the message to each party in the set of parties running the protocol. On the other hand, it is not hard to establish that no protocol can simulate RMT given Diffuse. The argument is as follows, and it works no matter how the protocol using Diffuse may operate. When a party A transmits a message M to party B, it is possible for the adversary in the Diffuse setting to simulate a "fake" party A that sends a message $M' \neq M$ to B concurrently. Invariably, this will result to a setting where B has to decide which is the correct message to output and will have to produce the wrong message with non-negligible probability. It follows that Diffuse is a weaker communication primitive: one would not be able to substitute Diffuse for RMT in a protocol setting.

Other models. The above models may be extended in a number of ways to capture various real world considerations in message passing. For instance, in point to point channels, the communication graph may change over the course of protocol execution with edges being added or removed adversarially something that may also result in temporary network partitions. Another intermediate model between point to point channels and diffusion, formulated by Okun [Oku05a], is to have a diffusion channel with "port awareness," i.e., the setting where messages from the same source are linkable, or without port awareness, but where each party is restricted to sending one message per round (see Section 3.2 for the notion of round) and their total number is known.

3.2 Synchrony

The ability of the parties to synchronize in protocol execution is an important aspect in the design of consensus protocols. Synchrony in message passing can be captured by dividing the protocol execution in rounds where parties are activated in some sequence and each one of them has the opportunity to send messages which are received by the recipients at the onset of the next round.

[1] Note that in contrast to a sender-anonymous channel (cf. [Cha81]), a diffuse channel will leak the identity of the sender to the adversary.

This reflects the fact that in real world networks messages are delivered most of the time in a timely fashion and thus parties can synchronize the protocol execution in discrete rounds.

A first important relaxation to the synchronous model is to allow a "rushing adversary" [Can00a], i.e., allow the adversary to control the activation of parties so that it acts last in each round having access to all messages sent by honest participants before it decides on the actions of the adversarial participants and the ordering of message delivery for the honest parties in the next round. This is captured by the functionalities in Figures 3 and 4. A second relaxation is to impose a time bound on message delivery that is not known to the protocol participants. We shall refer to this as the "partially synchronous setting" [DLS88]. The partial synchronous setting is easy to capture by the functionalities in Figures 3 and 4 as follows: a parameter $\Delta \in \mathbb{N}$ is introduced in each functionality that determines the maximum time a message can remain "in limbo." For each message that is sent, a counter is introduced that is initially 0 and counts the number of rounds that have passed since its transmission. When this counter reaches Δ the message is copied to the $\mathsf{inbox}(\cdot)$ strings for the active participants.

An even weaker setting than partial synchrony is that of message transmission with eventual message delivery, where all messages between honest parties are guaranteed to be delivered but there is no specific time bound that mandates their delivery in the course of the protocol execution. This is the classical model in fault tolerant distributed computing that is referred to as asynchronous [FLP85, Lyn96]. It is easy to adapt Figures 3 and 4 to accomodate eventual delivery, following the recent formalization of this model in [CGHZ16]. Note that it is proven that no deterministic consensus protocol exists in this setting [FLP85], and the impossibility can be overcome by randomization [BO83, Rab83, CD89, FM97].

Finally, in the "fully asynchronous setting" (cf. [Can00b]), where messages may be arbitrarily delayed *or dropped* consensus is trivially impossible.

4 Setup Assumptions

In the context of protocol design, a setup assumption refers to information that can be available at the onset of the protocol to each protocol participant. Consensus protocols are designed with respect to a number of different setup assumptions that we outline below.

4.1 No setup

In this setting we consider protocols that parties do not utilize any setup functionality beyond the existence of the communication functionality. Note that the communication functionality may already provide some information to the participants about the environment of the protocol; nevertheless, this setting is distinguished from other more thorough setup assumptions that are described below. We note that in this setting it may be of interest to consider protocol executions wherein the adversary is allowed a certain amount of precomputation prior to the onset of protocol session that involves the honest parties.

4.2 Public-state setup

A public-state setup is parameterized by a probability ensemble $\mathscr{D}$. For each input size κ, the ensemble $\mathscr{D}$ specifies a probability distribution that is sampled a single time at the onset of the protocol execution to produce a string denoted by s that is of length polynomial in κ. All protocol parties, including adversarial ones, are assumed to have access to s. In this setting, the consensus protocol will be designed for a specific ensemble $\mathscr{D}$.

The concept of a public-state setup can be further relaxed in a model that has been called "sunspots" [CPS07], where the ensemble is further parameterized by an index a. The definition is the same as above but now the protocol execution will be taken for some arbitrary choice of a. Intuitively,

the parameter a can be thought as an adversarial influence in the choice of the public string s. In this setting, the consensus protocol will be designed with respect to the ensemble class $\{\mathscr{D}_a\}_a$.

4.3 Private-state setup

As in the public state case, a private state setup is parameterized by an ensemble $\mathscr{D}$. For each input size κ and number of parties n, $\mathscr{D}$ specifies a probability distribution that is sampled a single time to produce a sequence of values $(s_1, \ldots, s_n)$. The length of each value s_i is polynomial in κ. At the onset of the protocol execution, the ensemble is sampled once and each protocol participant will receive one of the values s_i following some predetermined order. The critical feature of this setting is that each party will have private access to s_i. Observe that, trivially, the setting of private-state setup subsumes the setting of public-state setup.

As in the case of a public-state setup, it is important to consider the relaxation where the ensemble $\mathscr{D}$ is parameterized by string a. As before sampling from $\mathscr{D}_a$ will be performed from some arbitrary choice of a. It is in this sense where private-state setup has been most useful. In particular, we can use it to express the concept of a public-key infrastructure (PKI). In this setting the ensemble $\mathscr{D}$ employs a digital signature algorithm $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Verify})$ and samples a value $(pk_i, sk_i) \leftarrow \mathsf{Gen}(1^\kappa)$ independently for each honest participant. For each participant which is assumed to be adversarial at the onset of the execution, its public and secret key pair is set to a predetermined value that is extracted from a. The private input s_i for the i-th protocol participant will be equal to $(pk_1, \ldots, pk_n, sk_i)$, thus giving access to all parties' public keys and its own private key. Other types of private setup include "correlated randomness" [Bea96], where parties get correlated random strings $(r_1, r_2, ..., r_n)$ drawn from some predetermined distribution, which has been used to implement a random beacon [Rab83].

One may consider more complicated interactive setups, such as for example the adversary choosing a somehow based on public information available about $(s_1, \ldots, s_n)$, but we will refrain from considering those here. An alternative (and subsumed by the above) formulation of a private setup includes the availability of a broadcast channel prior to the protocol execution, which enables participants to exchange shared keys [PW92].

5 Computational Assumptions

The assumptions used to prove the properties of consensus protocols can be divided into two broad categories. In the information-theoretic (aka "unconditional") setting, the adversary is assumed to be unbounded in terms of its computational resources. In the computational setting, on the other hand, a polynomial-time bound is assumed.

5.1 Information-theoretic security

In the information-theoretic setting the adversarial running time is unbounded. It follows that the adversary may take arbitrary time to operate in each invocation. Note that the protocol execution may continue to proceed in synchronous rounds, nevertheless the running time of the adversary within each round will dilate sufficiently to accomodate its complete operation. When proving the consensus properties in this setting we can further consider two variations: perfect and statistical. When a property, Agreement for example, is perfectly satisfied this means that in all possible executions the honest parties never disagree on their outputs. On the other hand, in the statistical variant, there will be certain executions where the honest parties are allowed to disagree. Nevertheless, these executions will have negligible density in a security parameter (in this case, n) among all executions. We observe that the statistical setting is only meaningful for a probabilistic consensus protocol, where the honest parties may be "unlucky" in their choices of coins.

5.2 Computational security

In the computational setting the adversarial running time, and/or the computational model within which the adversary (and the parties running the protocol) are expressed becomes restricted. We distinguish the following variants.

One-way functions. A standard computational assumption is the existence of one-way functions. A one-way function is a function $f : X \to Y$ for which it holds that f is polynomial-time computable, but the probability $\mathscr{A}(1^{|x|}, f(x)) \in f^{-1}(f(x))]$ is negligible in $|x|$ for any polynomial time bounded program $\mathscr{A}$. One-way functions, albeit quite basic, are a powerful primitive that enables the construction of more complex cryptographic algorithms that include symmetric-key encryption, collision-resistant hash functions and digital signatures [NY89]. Depending on the setup, it is possible to use the one-way function assumption in various ways to assist in consensus protocol design.

Proof of Work. A proof of work (POW) [DN92] is a cryptographic primitive that enables a verifier to be convinced that certain amount of computational effort has been invested with respect to a certain context, e.g., a plaintext message or a nonce that the verifier has provided. A number of properties have been identified as important for the application of the primitive including amortization resistance, sampleability, fast verification, hardness against tampering and message attacks, and alnost k-wise independence [GKP17]. Some variants of POWs have been shown to imply one-way functions [BGJ$^+$16].

5.3 The random oracle model

In the previous subsections the level of security described was captured in the standard computational model where all parties are assumed to be Interactive Turing machines. In many cases, including consensus protocol design, it is proven useful to describe properties in the random oracle model, [BR93]. The random oracle model can be captured as an ideal functionality $\mathscr{F}_{\text{RO}}$ (see Appendix A). In a relevant adaptation of the $\mathscr{F}_{\text{RO}}$ model for the consensus setting, the access to the oracle is restricted by a quota of $q \geq 1$ queries per party per round of protocol execution [GKL15]. This bound is also imposed on the adversary who is assumed to control t parties. In case $t < n/2$, the execution will be said to impose honest majority in terms of "computational power."

6 Consensus in the Point-to-Point Setting

In the traditional network model of point-to-point reliable channels between every pair of parties, the problem was formulated in [LSP82] in the two settings described in Section 5: the information-theoretic setting and the computational (also called *cryptographic*, or *authenticated*) setting. As mentioned above, in the former no assumptions are made about the adversary's computational power, while the latter relies on the hardness of computational problems (such as factoring large integers or computing discrete logs), and requires a trusted setup in the form of a PKI. Depending on the setting, some of the bounds on the problems' quality measures differ. Refer to Figure 1 (specifically, the left subtree) as we go through the classification below.

6.1 Number of parties

For the information-theoretic setting, Lamport *et al.* [LSP82] showed that $n > 3t$ is both necessary and sufficient for the problems to have a solution. The necessary condition is presented in [LSP82] for the broadcast problem (see [FLM86] for the consensus version of the impossibility result), as the special case of 3 parties ("generals"), having to agree on two values ('attack', 'retreat'), with one of them being dishonest. As in the information-theoretic setting (with no additional setup) the parties are not able to forward messages in an authenticated manner, it is easily shown that an honest

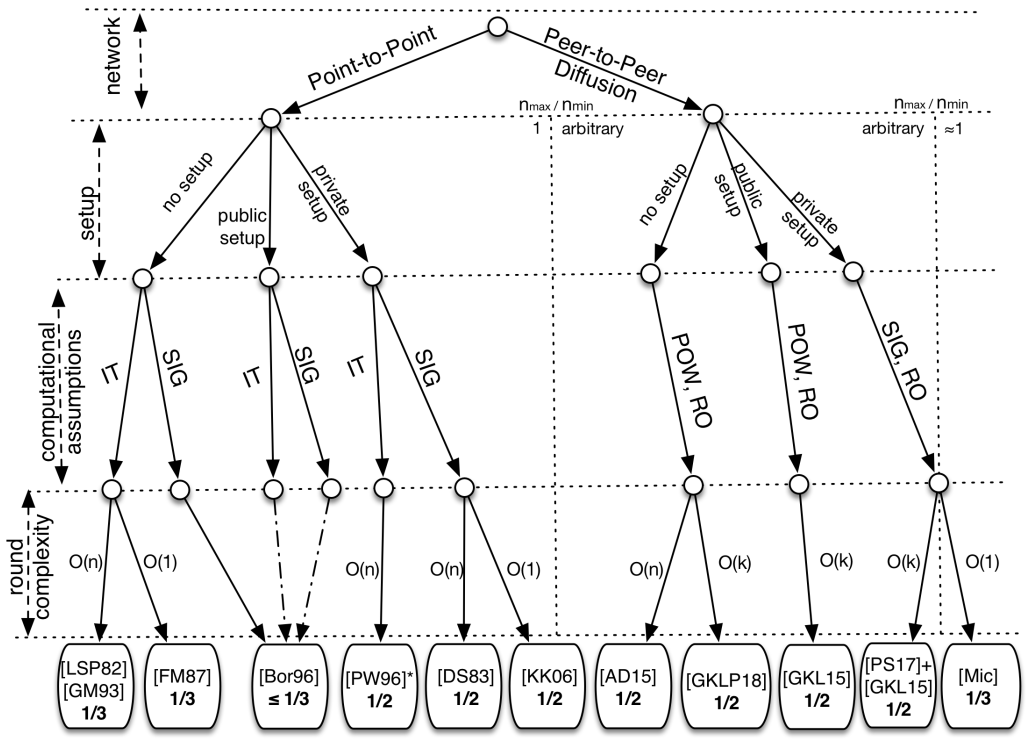

Figure 1: *The taxonomy of consensus protocols and impossibility results in the synchronous setting;* $n_{\max}/n_{\min}$ *refers to* participation tolerance *(cf. Sec 7.1).*

receiver cannot distinguish between a run where the sender is dishonest and sends conflicting messages, and a run where a receiver is dishonest and claims to have received the opposite message, which leads to the violation of the problem's conditions (Agreement and Validity, respectively). The general case (arbitrary values of n) reduces to the 3-party case. The protocol presented in [LSP82] matches this bound ($n > 3t$), and essentially consists in recursively echoeing messages received in a round while excluding the messages' senders. (In the first round, only the sender sends messages.) This is done for $t+1$ rounds, at which point the parties take majority of the values received, and go back up the recursion. $t+1$ rounds were later shown to be optimal (see below), but the protocol requires exponential (in n) computation and communication.

Lamport *et al.* [LSP82] also formulated the problem in the computational setting, where, specifically, there is a trusted private-state setup (of a PKI), and the parties have access to a digital signature scheme. This version of the problem has been referred to as *authenticated* Byzantine agreement. In contrast to the information-theoretic setting, in the computational setting with a trusted setup the bounds for broadcast and consensus differ: $n > t$ [LSP82] and $n > 2t$ (e.g., [Fit03]), respectively. The protocol presented in [LSP82] runs in $t+1$ rounds but, as in the information-theoretic setting, is also exponential; an efficient (polynomial-time) was presented early on by Dolev and Strong [DS83], which we now briefly describe. In this protocol in the first round the sender digitally signs and sends his message to all the other parties, while in subsequent rounds parties append their signatures and forward the result. If any party ever observes valid signatures of the sender on two *different* messages, then that party forwards both signatures to all other parties and disqualifies the sender (and all parties output some default message). This simple protocol is a popular building block in the area of cryptographic protocols (refer, however, to Section 6.6).

In the computational setting, the original formulation of the problem assumes a PKI. In [Bor96], Borderding considered the situation where no PKI is available, which he refers to as "local authentication," meaning that no agreement on the parties' keys is provided, as each party distributes its verification key by itself. Borderding shows that in this case, as in the information-theoretic setting

above, broadcast and consensus are not possible if $n \leq 3t$, even though this setting is strictly stronger, as a dishonest party cannot forge messages sent by honest parties. The gist of the impossibility is that the adversary can always confuse honest parties about the correct protocol outcome and digital signatures cannot help if they are not pre-associated with the parties running the protocol in advance (something only ensured given a private setup).

Regarding the "strong" version of the problem (the decision value must be one of the honest parties' input values), Fitzi and Garay [FG03] showed that the problem has a solution if and only if $n > \max(3, |V|)t$ in the unconditional setting[2], where V is the domain of input/output values, and $n > |V|t$ in the computational setting with a trusted setup, giving resiliency-optimal and polynomial-time protocols that run in $t+1$ rounds.

6.2 Running time

Regarding the running time of consensus protocols, a lower bound of $t+1$ rounds for deterministic protocols was established by Fischer and Lynch [FL82]; the lower bound was shown for the case of benign ("crash") failures, and extended to malicious failures by Dolev and Strong [DS83]. As mentioned above, the original protocols by Lamport *et al.* already achieved this bound, but required exponential computation and communication. In contrast to the computational setting, were a polynomial-time resiliency- and round-optimal was found relatively soon [DS83], in the information-theoretic setting this took quite a bit longer, and was achieved by Garay and Moses [GM98]. In a nutshell, the [GM98] result builds on the "unraveled" version of the original protocol, presented and called *Exponential Information Gathering* by Bar-Noy *et al.* [BDDS92], applying a suite of "early-stopping" (see more on this below) and fault-detection techniques to prune the tree data structure to polynomial size. Regarding strong consensus, the $t+1$-round lower bound also applies to this version of the problem, which the protocols by Fitzi and Garay [FG03] achieve (as well as being polynomial-time and resiliency-optimal).

In the $t+1$-round lower bound for deterministic protocols, t is the maximum number of corruptions that can be tolerated in order to achieve consensus in a given model. Dolev, Reischuk and Strong [DRS90] asked what would the running time be when the *actual* number of corruptions, say, f is smaller than t, and showed a lower bound of $\min\{t+1, f+2\}$ for any consensus protocol, even when only crash failures occur, which is important when f is very small. They called a consensus protocol satisfying this property *early-stopping*. Faster termination, however, comes at a price of non-simultaneous termination, as they also showed that if simultaneous termination is required, then $t+1$ rounds are necessary. (See also [DM90].)

Optimal early stopping for the optimal number of parties (i.e., $n > 3t$) was achieved in the information-theoretic setting by Berman and Garay [BGP92b]; the protocol, however, is inefficient, as it requires exponential communication and computation. Relatively recently, an efficient (polynomial-time) optimal early-stopping consensus protocol was presented by Abraham and Dolev [AD15a].

The above $t+1$-round lower bound applies to deterministic protocols. A major breakthrough in fault-tolerant distributed algorithms was the introduction of randomization to the field by Ben-Or [BO83] and Rabin [Rab83], which, effectively, showed how to circumvent the above limitation by using randomization. Rabin [Rab83], in particular, showed that linearly resilient consensus protocols in expected *constant* rounds were possible, provided that all parties have access to a "common coin" (i.e., a common source of randomness). 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. This line of research culminated with the work of Feldman and Micali [FM97], who showed how to obtain a shared random coin with constant probability from "scratch," yielding a probabilistic consensus protocol tolerating the maximum number of misbehaving parties ($t < n/3$) that runs in expected constant number of rounds.

The [FM97] protocol works in the information-theoretic setting; these results were later extended

[2] The lower bound was in fact shown by Neiger, who formulated this version of the problem [Nei94].

to the computational setting by Katz and Koo [KK09], who showed that assuming a PKI and digital signatures there exists an (expected-)constant-round consensus protocol tolerating $t < n/2$ corruptions. Recall that broadcast protocols in the computational setting with setup tolerate an arbitrary number (i.e., $n > t$) of dishonest parties; in contrast, the protocol in [KK09] assumes $n > 2t$ (as it is based on VSS—*verifiable secret sharing* [CGMA85]). In [GKKO07], Garay *et al.* consider the case of a dishonest majority (i.e., $n \leq 2t$), presenting an expected-constant-round protocol for $t = \frac{n}{2} + O(1)$ dishonest parties (more generally, expected $O(k^2)$ running time when $t = \frac{n}{2} + k$), and showing the impossibility of expected-constant-round broadcast protocols when $n - t = o(n)$.

The speed-up on the running time of probabilistic consensus protocols comes at the cost of uncertainty, as a party that terminates can never be sure that other parties have also terminated—i.e., there cannot be *simultaneous* termination [DRS90], which is an issue when these protocols are invoked from a higher-level protocol, as a party cannot be sure how long after he receives his output from a call to such a *probabilistic termination* (PT) consensus protocol (cf. [CCGZ16]) he can safely carry out with the execution of the calling protocol. The sequential composition of PT consensus protocols was addressed by Lindell *et al.* [LLR06] while the parallel composition of such protocols by Ben-Or and El-Yaniv [BE03]. (The issue in the case of parallel invocations of expected-constant-round PT protocols is that the overall running time of the parallel executions is not necessarily expected constant.) The above results on sequential and parallel composition, however, do not use simulation-based security, and it was therefore unclear how (or if) one would be able to use them to instantiate consensus (and/or broadcast) from a higher-level protocol. Such formal simulation-based (and therefore composable) definition and constructions of consensus protocols with probabilistic termination has been recently presented in [CCGZ16].

6.3 Trusted setup

We already covered this aspect above while describing the protocols achieving the different bounds on the number of parties; here we briefly summarize it. There is no trusted setup in the unconditional setting, although in the case of randomized protocols there is the additional requirement of the point-to-point channels being private in addition to reliable, while the "authenticated" consensus protocols assume a PKI. Related to a trusted setup assumption, we remark that if a *precomputation* phase is allowed in the information-theoretic setting where reliable broadcast is guaranteed, then Pfitzmann and Waidner showed that broadcast and consensus are achievable with the same bounds on the number of parties as in the computational setting, using a tool known as a "pseudo-signatures" [PW92].

6.4 Communication cost

A lower bound of $\Omega(n^2)$ on the number of messages (in fact, $\Omega(nt)$) was shown by Dolev and Reischuk for consensus for both information-theoretic and computational security [DR85]; for the latter, what they showed was that the number of signatures that are required by any protocol is $\Omega(nt)$, resulting in an $\Omega(nt|\sigma|)$ bit complexity (for a constant-size domain), where $|\sigma|$ represents the maximum signature size. The first information-theoretically secure protocols to match this bound were given by Berman *et al.* [BGP92a] and independently by Coan and Welch [CW89]; regarding computational security, the protocol presented by Dolev and Strong [DS83] requires that many messages. By relaxing the model and allowing for a small probability of error, King and Saia [KS16], presented a protocol that circumvents the impossibility result (with message complexity $\tilde{O}(n^{1.5})$).

6.5 Beyond synchrony

The case of partial synchrony, introduced in [DLS88], considers the existence of an unknown bound Δ that determines the maximum delay of a message that is unknown to the protocol participants.[3] As shown in [DLS88], the resiliency bounds presented in the point-to-point subtree of Figure 1 remain unaltered in the no setup and public setup cases, but it degrades to $n/3$ in the private setup case.

In the eventual delivery setting, as mentioned above, deterministic consensus is impossible but it is still feasible to obtain protocols with probabilistic guarantees. Furthermore, note that in this setting it is not possible to account for all of the honest parties' inputs since parties cannot afford to wait for all the parties to engage (since corrupt parties may never transmit their messages and it is impossible to set a correct time-out). In more detail, without a setup in the information-theoretic setting, it is possible to adapt the protocol in [FM97] and achieve $n/4$ resilience [Fel88] (see Figure 1). Moreover, by allowing the protocol not to terminate with negligible probability it is possible to bring the resiliency to $n/3$ [CR93, ADH08].

In the private-setup setting, assuming one-way functions, it is possible to obtain an always-terminating protocol with $n/3$ resiliency (cf. [Fel88]). We note that it is infeasible to go beyond $n/3$ resiliency, as shown in [BCG93, Can96], where this bound is argued for fail-stop failures, and thus the above results are optimal in this sense.

Most protocols mentioned above demonstrate the feasibility of the respective bounds. Much effort has also been dedicated to achieving practical Byzantine fault tolerance (BFT) in the eventual message delivery model. For completeness, here we mention some relevant results, with the work by Castro and Liskov [CL02] as a notable instance, where they focus on a fault-tolerant replicated transactions service in the cryptographic setting with the corresponding Safety and Liveness properties (see Section 8), achieving $n/3$ resiliency. Cachin *et al.* [CKS05] study consensus in the same model, showing an efficient coin tossing protocol assuming a random oracle. Other related works focusing on practical efficiency include the work by Kursawe and Shoup on "asynchronous" atomic broadcast [KS05] (atomicity means that broadcast executions are ordered in such a way that two broadcast requests are received in the same order by any two honest parties), following the "optimistic" approach presented in [CL02] where first only a "Bracha broadcast" protocol [Bra84] is first attempted, reverting to the use of cryptography if things go wrong. Finally, Miller *et al.* [MXC+16] improve on the communication complexity of the protocol in [CKPS01], and guarantees Liveness without any timing assumptions, which was the case in [CL02].

6.6 Property-based vs. simulation-based proofs

As mentioned in Section 2.2, consensus and broadcast protocols have been typically proven secure/-correct following a *property-based* approach. It turns out, as pointed out by Hirt and Zikas [HZ10] (see also [GKKZ11]), that in the case of *adaptive* adversaries who can choose which parties to corrupt dynamically, during the course of the protocol execution (cf. [CFGN96]), most existing broadcast and consensus protocols cannot be proven secure in a simulation-based manner. The reason, at a high level, is that when the adversary (having corrupted a party) receives a message from an honest party, can corrupt that party and make him change his message to other parties. This creates an inconsistency with the ideal process, where the party has already provided his input to the trusted party/ideal functionality that abstracts consensus. To be amenable to a simulation-based proof, instead of sending its initial message "in the clear," the sender in a broadcast protocol sends a *commitment* to the message, allowing the simulator in the ideal process to "equivocate" when the committed value becomes known in case the party has been corrupted and the initial value changed [HZ10, GKKZ11].

[3] In [DLS88] partial synchrony between the clocks of the processors is also considered as a separate relaxation to the model. In the present treatment we only focus on partial synchrony with respect to message passing.

7 Consensus in the Peer-to-Peer Setting

Consensus in the peer-to-peer setting is the consensus problem when the available communication resource is peer-to-peer diffusion (cf. Section 3.1), a weaker communication primitive compared to point-to-point channels. (For this section, refer to the right subtree of Figure 1.) This setting arose with the advent of the Bitcoin blockchain protocol, and was formally studied for the first time in [GKL15]. In a nutshell, it constitutes an unauthenticated model of communication where no correlation of message sources across rounds can be established and the exact number of parties that participate in the protocol may be unknown to the protocol participants. Moreover, since the adversary may inject messages in the network, an honest party cannot infer the number of participants from a message count. Because of this uncertainty, the consensus properties of Termination, Agreement and Validity hold for the honest parties that are participating in the protocol.

We note that in a precursor model, where there is no correlation of message sources, but the point-to-point structure is still in place albeit without authentication, Okun showed that deterministic consensus algorithms are impossible for even a single failure [Oku05b, Oku05a], but that probabilistic consensus is still feasible by suitably adapting the protocols of [BO83, FM97][4]; the protocol, however, takes exponentially many rounds.

The consensus problem in the peer-to-peer setting has mostly been considered in the computational setting utilizing one-way functions and the proof-of-work (POW) primitive (Section 5). The first suggestion for a solution was informally described in [AJK05], where it was suggested that POWs can be used as an "identity assignment" tool, which subsequently can be used to bootstrap a standard consensus protocol like [DS83]. Nevertheless, the viability of this plan was never fully analyzed until an alternative approach to the problem was informally described by Nakamoto in an email exchange [Nak08b], where he argued that the "Byzantine Generals" problem can be solved by a blockchain/POW approach tolerating a number of misbehaving parties strictly below $n/2$. As independently observed in [ML14, GKL14], however, with overwhelming probability the Validity property is not satisfied by Nakamoto's informal suggestion.

The blockchain approach suggests to string POWs together in a hash chain and achieve agreement using a rule that favors higher concentrations of computational effort as reflected in the resulting hash chains. The inputs to the consensus problem are "entangled" within the POWs themselves and the final output results from a processing of the hash chain. The approach was first formalised in [GKL15] where also two constructions were provided that satisfy all properties assuming a public setup.

Without access to a public setup, it is also possible to obtain a construction based on the results of [AD15b], who were the first to formalise the [AJK05] informal approach of using POWs for identity assignment. Moreover, a blockchain-based approach is also possible as shown in [GKLP18]. Using a private setup, it becomes feasible to use primitives such as digital signatures and verifiable random functions and obtain even more efficient constructions such as the consensus sub-protocol of [Mic16].

7.1 Number of parties

One of the most important characteristics of consensus in the peer-to-peer setting is that the actual number of parties that are running the protocol is not assumed to be known in advance. Instead, the actual number of parties becomes a run-time execution parameter and the protocol is supposed to be able to tolerate a range of different of possible choices for the number of parties. We capture this by posing a range of possible operational values $[n_{\min}, n_{\max}]$, and posit that if the actual number of parties falls within the range then the properties will be guaranteed. We call the ratio $n_{\max}/n_{\min}$ for a given protocol a protocol's *participation tolerance*. We note that this notion is somewhat related to models that have been considered in fault-tolerant distributed computing and secure multiparty

[4] Hence, consensus in this setting shares a similar profile with consensus in the asynchronous network model [FLP85].

computation (see, e.g., [GP92] and [HLP11], respectively). In such scenarios the parties are subject to two types of faults, Byzantine and benign, such as "going to sleep," but adversarially scheduled. In the latter type, the parties will cease participating in the protocol execution.

In the convention introduced in [GKL15], each party has a fixed quota of hashing queries that is allowed per round. As a result, the number of parties is directly proportional to the "computational power" that is present in the system and the total number of POWs produced by the honest parties collectively would exceed that of the adversary assuming honest majority with very high probability. Given this it is tempting to imagine a direct translation of computational power to a set of identities [AJK05]. The main problem is that the set of identities as perceived by the honest partipants in the protocol execution might be inconsistent. This was resolved with the protocol of [AD15b] where POWs are used to build a "graded" PKI, where keys have ranks. The graded PKI is an instance of the *graded agreement* problem [FM97], or partial consistency problem [CFF+05], where honest parties do not disagree by much, according to some metric. Subsequently, it is possible to morph this graded consistency to global consistency by running multiple instances of [DS83]. This can be used to provide a consensus protocol with resiliency $n/2$ without a trusted setup.

It is unnecessary though for the parties to reach consensus by establishing identities. In the first consensus protocol presented in [GKL15], the parties build a blockchain where each block contains a value that matches the input of the party that produced the block. The protocol continues for a certain number of rounds that ensures that the blockchain has grown to a certain length. In the final round, the parties remove a k-block suffix from their local blockchain, and output the majority bit from the remaining prefix. Based on the property called "common prefix" in [GKL15], it is shown that with overwhelming probability in the security parameter, the parties terminate with the same output, while using the "chain quality" property, it is shown that if all the honest parties start with the same input, the corrupt parties cannot overturn the majority bit, which corresponds to the honest parties' input. The number of tolerated misbehaving parties in this protocol is strictly below $n/3$, a suboptimal resiliency due to the low chain quality of the underlying blockhain protocol. The maximum resiliency that can be expected is $n/2$, something that can be shown by easily adapting the standard argument for the necessity of honest majority shown in Section 2.

Optimal resiliency can be reached by the second consensus protocol of [GKL15] as follows: The protocol substitutes Bitcoin transactions with a type of transactions that are *themselves* based on POWs, and hence uses POWs in two distinct ways: for the maintenance of the ledger and for the generation of the transactions themselves. The protocol requires special care in the way it employs POWs since the adversary should be incapable of "shifting" work between the two POW tasks that it faces in each round. To solve this problem, a special strategy for POW-based protocol composition is introduced in [GKL15] called "2-for-1 POWs." In the second solution presented in [GKL15] the number of tolerated misbehaving parties is strictly below $n/2$.

We note that all these protocols come with a hard-coded difficulty level for POWs which is assumed to be correlated with the number of parties n. If f is the probability that at least one honest party will produce a POW in a round of protocol execution, it holds that f approaches 0 for small n while it approaches 1 for large n. It follows that the choice of POW difficulty results in an operational range of values $[n_{\min}, n_{\max}]$ and it is possible to set the difficulty for any constant ratio $n_{\max}/n_{\min}$, so the participation tolerance of the protocol can be set to any arbitrary constant. We note that the lower bound $n_{\min}$ can be arbitrarily small as long as we are able to assume that even a single party has sufficient computational power to ensure that finding POWs is not very rare. In case this is not true and $n < n_{\min}$, the protocol cannot be guaranteed to satisfy Validity with high probability, while on the other hand, if $n > n_{\max}$, the protocol cannot be guaranteed to achieve agreement with high probability.

Using digital signatures and verifiable random functions (VRFs) (or just digital signatures and a hash function modeled as a random oracle), it is possible to implement the second consensus protocol in [GKL15] over an underlying blockchain protocol that uses a public-key infrastructure as opposed to POWs, and allows for arbitrary participation tolerance such as [PS17a] for optimal resiliency

of $n/2$. The idea is as follows: one can use VRFs for each participant to enable a random subset of elected transaction issuers in each round. The ledger will then incorporate such transactions within a window of time following the same technique and counting argument as in the second consensus protocol of [GKL15].

7.2 Running time

In order to measure the running time that the protocols require in the peer-to-peer setting assuming POW, one will have to also take into account that periods of silence, i.e., rounds without any message passing, may also be required for ensuring the required properties with high probability in κ, a security parameter. In the consensus protocol derived from the protocol of [AD15b], $O(n)$ rounds are required where n is the number of parties. This can be improved to $O(\kappa)$ by, e.g., using a blockchain-based approach [GKLP18]. In the public-setup setting, assuming that the number of parties fall within the operational range, the protocols of [GKL15] run also in time $O(\kappa)$.

It is worth noting the contrast to the approach used in randomized solutions to the problem in the standard setting (cf. Section 6.2), where achieving consensus is reduced to (the construction of) a shared random coin, and comparable guarantees are obtained after a poly-logarithmic number of rounds in the number of parties. The probabilistic aspect in the blockchain setting stems from the parties' likelihood of being able to provide proofs of work.

In the private setup setting it is possible to improve the running time to expected constant, e.g., by deploying the consensus sub-protocol of Algorand [Mic16] for 1/3 resiliency.

7.3 Trusted setup

The relevant trusted setup assumption in the above protocols include a fresh random string, that can be incorporated as part of a "genesis block" in the blockchain protocol setting, or in general as part of the POWs[5]. The objective of this public setup is to prevent a pre-computation attack by the adversary that will violate the relative superiority of honest parties which would be derived by the honest majority assumption. Note that protocols that require no trusted setup such as [AD15b, GKLP18] take advantage of a special randomness exchange phase prior to POW calculation that guarantees freshness without the need of a common random string.

It is worth to emphasize the fundamental advantage of the POW setting compared to other computational assumptions that have been used for consensus. Specifically, it is known that without a private setup, consensus is not possible with more than $n/3$ corruptions [Bor96] even assuming digital signatures. The $n/3$ impossibility result does not apply here since, essentially, proofs of work can make it infeasible for the adversary to present diverging protocol transcripts without investing effort for distinct POW calculations.

Another observation is that assuming a private setup in the peer-to-peer setting, one can simulate point-to-point connectivity, and thus run any protocol from the previous section; nevertheless; this reduction is not efficient and in the peer-to-peer setting with private-setup one can still obtain protocols that are more efficient (e.g., with subquadratic communication complexity).

7.4 Communication cost

The total number of transmitted messages in the consensus protocols described above is, in expectation, $O(n^2\kappa)$ for the case of [AD15b, GKLP18] counting each invocation of the diffuse channel as costing $O(n)$ messages. For the two protocols of [GKL15] the number of messages is $O(n\kappa)$ in the public setup setting. In the private setup setting it can be possible to reduce this further using techniques from [Mic16].

[5] Alternatively, the protocols would consider as valid any chain that extends the empty chain, and where the adversary is not allowed any pre-computation.

We recall that an important difference with randomized consensus protocols in the standard setting is that parties send messages in every round, while in the POW setting (honest) parties only communicate whenever they are able to produce a proof of work; otherwise, they remain silent. This also suggests that there may be honest parties that never diffuse a message[6] and thus it is feasible to drop communication costs to below n^2 (with a probabilistic guarantee; cf. Section 6.4).

7.5 Beyond Synchrony

The consensus protocols of [GKL15] in Figure 1 can be analyzed in the partial synchronous setting as well (refer to the the full version of [GKL14] as a starting point). Recall that the way the protocols operate in this setting is that a parameterisation of difficulty is hardcoded that provides a reasonable POW production rate over message passing time. The security of the protocols will then be at the theoretical maximum in terms of resiliency as long as the original estimate is close to being safe (network delay is low) and will degrade if the estimate is worse, dissipating entirely when the delay gets larger (for the full argument, see [PSS17], where it is shown how the blockchain protocol's consistency collapses when delay is arbitrarily large).

7.6 Property-based vs. simulation-based proofs

To our knowledge, there is no simulation-based treatment of consensus in the peer-to-peer setting, however it is easy to infer a functionality abstracting the problem. The only essential difference is that the actual number of parties involved in the execution are to be decided on the fly and will be unknown to the protocol participants.

8 Ledger Consensus

Ledger consensus (a.k.a. "Nakamoto consensus") is the problem where a set of servers (or nodes) operate continuously accepting inputs ("transactions") and incorporate them in a public data structure called the *ledger*. Clients are able to read the ledger and submit transactions to be added to it. The purpose of ledger consensus is to provide a unique view of the ledger to the clients. The properties that a ledger consensus protocol must satisfy are as follows:

- *Consistency* (or *Persistence*): This property mandates that if a client queries an honest node's ledger at time t_1 and receives the response $\mathscr{L}_1$, then a client querying an honest node's ledger at time $t_2 \geq t_1$ will receive a response $\mathscr{L}_2$ that satisfies $\mathscr{L}_1 \preceq \mathscr{L}_2$, where $\preceq$ denotes the standard prefix operation.
- *Liveness:* If a transaction tx is given as input to all honest nodes for a certain number of rounds denoted by u, and a client queries any honest node's ledger after that, then the node will respond with a value $\mathscr{L}$ that includes tx.

In the traditional distributed systems literature, such problem is often referred to as *state machine replication* [Sch90]. Consistency ensures that parties have the same view of the log of transactions, while Liveness ensures the quick incorporation of transactions.

Given a consensus protocol it is tempting to apply it in sequential composition in order to solve ledger consensus. The reduction indeed holds but some special care is needed. First, let us consider the case where no setup is available. The construction in the synchronous network model is as follows. First, suppose that we have at our disposal a consensus protocol that satisfies Agreement, (Strong) Validity, and Termination after u rounds. The protocol has all nodes collect transactions and

[6] Note the similarity with standard consensus in the eventual-delivery setting (Section 6.5), where not all honest parties' inputs may be accounted for.

then run the consensus protocol with the set of transactions as their input. When the protocol terminates after u rounds, the nodes assign an index to the output (call it the i-th entry to the ledger) and move on to the next consensus instance. It is easy to see that Consistency is satisfied because of Agreement, while Liveness is satisfied with parameter u because of Strong Validity and Termination. It is worth noting that "plain" Validity by itself is not enough, since a ledger protocol is supposed to run for any given set of transactions and as a result it is possible that no two honest nodes would ever agree on a set of inputs. In this case, Validity might just provide that honest parties' agree on an adversarial value, which might be the empty string. As a result the ledger would be empty and Liveness would be violated. However it is possible to deal with this problem without resorting to the full power of Strong Validity. For instance, it is sufficient to consider a variant of consensus where each party has an input set X_i and the joint output set S satisfies that $X_i \subseteq S$. We note that such a "union" consensus protocol can be implied by Interactive Consistency, as defined in [PSL80], and it has also recently been considered explicitly as a consensus variant [DG17]. Other intermediate notions of Validity such as a *predicate-based* notion [CKPS01] can be useful here as well.

Let us now comment how the reduction can be performed under different setup and network assumptions. First, if a setup assumption is used, observe that the above reduction will require the availability of the setup every u rounds. Given this might be impractical, one may consider how to emulate the sequence of setups using a single initial setup. This approach is non-black-box on the underlying protocol and may not be straightforward. For instance, when sequentially composing a POW-based consensus protocol that relies on a public setup, the security of the protocol may non-trivially rely on the unpredictability of the i-th setup. Techniques related to sequential composition of a basic building block protocol have appeared in a number of ledger protocols, including [KRDO17, BPS16, Mic16]. Regarding network aspects, we observe that the reduction can proceed in essentially the same way in the peer-to-peer setting as in the point-to-point setting. Finally, note that when simultaneous termination is not available in the underlying consensus protocol, special care is needed in applying composition (cf. [CCGZ16]).

Ledger consensus was brought forth as an objective of the Bitcoin blockchain protocol. For this reason, in the remaining of the paper, we only consider the problem in the peer-to-peer setting (in the point-to-point setting it is possible to adapt standard BFT methods to solve the problem, see e.g., [MXC+16] for an example of this approach. A pictorial overview of our protocol classification is presented in Figure 2.

8.1 Number of parties

As in the case of peer-to-peer consensus (Section 7.1), the actual number of parties n is not known in advance and may be assumed to fall within a range of operational parameters $n \in [n_{\min}, n_{\max}]$. This is also related to the concept of "sporadic participation" that was considered in [PS17a], where certain honest parties may "go to sleep" for arbitrary amounts of time.

In the POW setting, recall that each party has a fixed quota of queries that it can perform to a hash function per unit of time and thus the number of parties is directly proportional to the total computational or hashing power that is available. In this setting, first [GKL15] showed that ledger consensus can be achieved when the number of corrupted parties is strictly below $n/2$. This bound was also preserved in the partially synchronous setting, as shown by Pass *et al.* [PSS17]. With respect to lower bounds, it is easy to see that $n/2$ is tight.

The above results refer to a static setting where there are no large deviations in the number of parties throughout the execution. The setting where the population of parties running the protocol can dynamically (and quite drastically) change over time with the environment introducing new parties and deactivating parties that have participated was considered for ledger consensus for the first time in [GKL17]. Their main result is that ledger consensus can be achieved in the POW setting, assuming an honest majority appropriately restated by considering the number of parties as they change over time: Assuming n_i are the active parties at time unit i, it holds that the number of adversarial parties is bounded away from $n_i/2$.

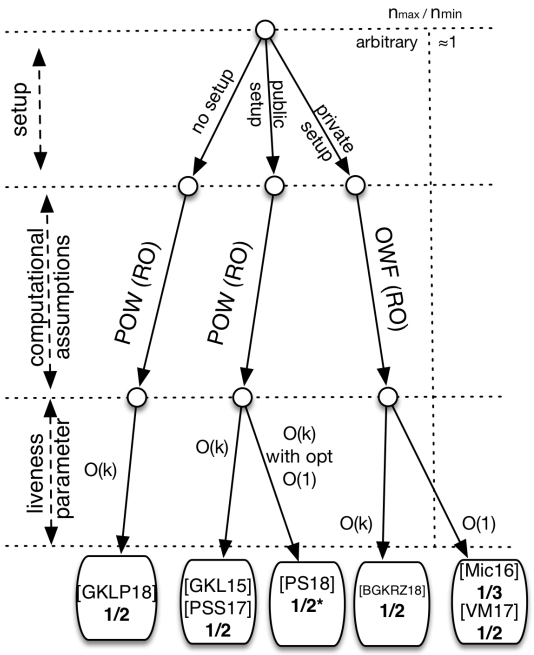

Figure 2: *The taxonomy of ledger consensus protocols (peer-to-peer setting).*

Assuming a private setup and a setting where the adversary gets t Byzantine corruptions and s asleep parties, in [PS17a] it is shown that ledger consensus can be achieved as long as t is strictly bounded by $a/2$ where $a = n - s$ is the number of "alert" parties, i.e., the number of asleep parties may be larger than $n/2$ and hence an arbitrary participation ratio can also be achieved in this setting without resorting to POWs. With respect to lower bounds, in the case of sleep corruptions the bound can be generalized to $a/2$; see [PS17a]. A dynamic setting of parties was also considered in [KRDO17, BPS16, DGKR18], providing a similar type of results assuming a PKI with honest "stake" majority. An important deficiency shared by these works is that new parties have to be chaperoned into the system by receiving advice consistent with the views of the honest parties. This was highlighted as the "bootstrapping from genesis" problem in [BGK+18], who also put forth a more refined model of dynamic participation, called *dynamic availability*. This model allows finer control from the environment's point of view in terms of disconnecting parties, or having parties lose access to resources such as the clock or the hash function computing procedure.

Finally, in terms of participation tolerance, we observe that an arbitrary $n_{\max}/n_{\min}$ can be achieved by protocols such as [PS17a, BGK+18] while Algorand, [Mic16], requires $n_{\max}/n_{\min}$ to be (approximately) 1 since the expected participation is a hardcoded value in the protocol.

8.2 Transaction processing time

Contrary to a consensus protocol, a ledger consensus protocol is a protocol that is supposed to be running over an arbitrary, potentially long, period of time. Thus, the relevant measure in this context is the amount of time that it takes for the system to insert a transaction in the log that is maintained by the participants. This relates to the parameter u introduced as part of the Liveness property, which determines the number of rounds needed in the execution model for a transaction to be included in the log. Observe that Liveness is only provided for transactions that are produced by honest participants or are otherwise unambiguously provided to the honest parties running the protocol.

In this setting we observe that [GKL15] achieves ledger consensus with processing time $O(\kappa)$ rounds of interaction, where κ is the security parameter. This result is replicated in the partially synchronous setting, where processing time takes $O(\kappa\Delta)$ rounds, and where Δ is the maximum delay

that is imposed on message transmission. The above results assume the adversarial bounds consistent with honest majority which are tight (cf. [PS17b]). Considering a weaker adversarial setting it is possible to improve Liveness; for instance, Algorand [Mic16] achieves expected-constant number of rounds while, Thunderella [PS17b], shows that the processing time can be dropped to $O(1)$ rounds worst-case, assuming an honest super-majority (i.e., adversarially controlled number of parties strictly below $n/4$) and the existence a certain specific party called the *accelerator* to be honest.

8.3 Trusted setup

Ledger consensus can be achieved in the public- or private-state setup setting. Protocols falling in the former category are [GKL15, PSS17, GKL17], whereas protocols consistent with the latter are [Mic16, PS17a, KRDO17, BPS16, GHM+17]. In the absence of a trusted setup, it has been shown that it is possible to "bootstrap" a ledger consensus protocol from "scratch," either directly [GKLP18] or via setting up a public-key directory using proofs of work [AD15b].

An important further consideration between public and private setup is that in the peer-to-peer setting, the former represents what typically is consistent with the so-called persmissionless setting, while the latter is consistent with the permissioned setting. This follows from the fact that anyone that has access to the peer-to-peer channel is free to participate in the protocol, if no setup or a public setup is assumed. On the other hand, in the private setup setting, a higher level of permissioning is implied: The parties that are eligible to run the protocol need to get authorized either by the setup functionaliy so that they receive the private information that is related to the protocol execution, or, alternatively, interact with the parties that are already part of the protocol execution so they can be inducted. Note that the point-to-point setting is—by definition—permissioned. via access to the RMT functionality.

8.4 Communication cost

Given that ledger consensus is an ongoing protocol that processes incoming transactions, defining communication costs requires some care. To our knowledge, no formal definitions of communication costs for ledger consensus have been proposed. A first approach to the problem is to consider a type of "communication overhead" on top of the transactions that are transmitted in the system. It follows that the minimum communication necessary for each bit of transaction transmitted is the diffusion of this bit. Given the above, the communication costs of ledger consensus protocols based on blockchain protocols can be seen to be constant in the sense that parties transmit, up to constant factors, more data.

8.5 Property-based vs. simulation-based proofs

The first simulation-based definition of ledger consensus was presented by Badertscher et al. [BMTZ17]. A refinement of this definition was presented in [BGK+18], where it was also shown how to adapt it in a setting where a private setup is available. In terms of composability, an (expected) disadvantage for POW-based protocols highlighted in the work of [BMTZ17] is that access to the random oracle should be specific to the current ledger protocol session.

8.6 Beyond synchrony

Initial work in ledger consensus protocols in the public setup [GKL15,GKL17] and the no setup setting [GKLP16, GKLP18] assumed a rushing adversary and synchronous operation. This can be extended to the partial synchrony setting as shown in [PSS17] as well as in the full version of [GKL14] with the same limitations explained in Section 7.5.

9 Acknowledgements

The second author was supported by H2020 Project Priviledge # 780477. The authors are grateful to Christian Cachin for helpful suggestions in an earlier version of this draft.

References

[AD15a] Ittai Abraham and Danny Dolev. Byzantine agreement with optimal early stopping, optimal resilience and polynomial complexity. In Rocco A. Servedio and Ronitt Rubinfeld, editors, *Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, STOC 2015, Portland, OR, USA, June 14-17, 2015*, pages 605–614. ACM, 2015.

[AD15b] Marcin Andrychowicz and Stefan Dziembowski. Pow-based distributed cryptography with no trusted setup. In *Advances in Cryptology - CRYPTO 2015 - 35th Annual Cryptology Conference, Santa Barbara, CA, USA, August 16-20, 2015, Proceedings, Part II*, pages 379–399, 2015.

[ADH08] Ittai Abraham, Danny Dolev, and Joseph Y. Halpern. An almost-surely terminating polynomial protocol forasynchronous byzantine agreement with optimal resilience. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Principles of Distributed Computing, PODC 2008, Toronto, Canada, August 18-21, 2008*, pages 405–414, 2008.

[AJK05] James Aspnes, Collin Jackson, and Arvind Krishnamurthy. Exposing computationally-challenged Byzantine impostors. Technical Report YALEU/DCS/TR-1332, Yale University Department of Computer Science, July 2005.

[BCG93] Michael Ben-Or, Ran Canetti, and Oded Goldreich. Asynchronous secure computation. In Kosaraju et al. [KJA93], pages 52–61.

[BDDS92] Amotz Bar-Noy, Danny Dolev, Cynthia Dwork, and H. Raymond Strong. Shifting gears: Changing algorithms on the fly to expedite byzantine agreement. *Inf. Comput.*, 97(2):205–233, 1992.

[BE03] Michael Ben-Or and Ran El-Yaniv. Resilient-optimal interactive consistency in constant time. *Distributed Computing*, 16(4):249–262, 2003.

[Bea96] Donald Beaver. Correlated pseudorandomness and the complexity of private computations. In Miller [Mil96], pages 479–488.

[BGJ+16] Nir Bitansky, Shafi Goldwasser, Abhishek Jain, Omer Paneth, Vinod Vaikuntanathan, and Brent Waters. Time-lock puzzles from randomized encodings. In Madhu Sudan, editor, *Proceedings of the 2016 ACM Conference on Innovations in Theoretical Computer Science, Cambridge, MA, USA, January 14-16, 2016*, pages 345–356. ACM, 2016.

[BGK+18] Christian Badertscher, Peter Gazi, Aggelos Kiayias, Alexander Russell, and Vassilis Zikas. Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability. *IACR Cryptology ePrint Archive*, 2018:378, 2018.

[BGP92a] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. *Bit Optimal Distributed Consensus*, pages 313–321. Springer US, Boston, MA, 1992.

[BGP92b] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. Optimal early stopping in distributed consensus (extended abstract). In Segall and Zaks [SZ92], pages 221–237.

[BGW88] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation (extended abstract). pages 1–10, 1988.

[BMTZ17] Christian Badertscher, Ueli Maurer, Daniel Tschudi, and Vassilis Zikas. Bitcoin as a transaction ledger: A composable treatment. In Katz and Shacham [KS17], pages 324–356.

[BO83] Michael Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols (extended abstract). In Robert L. Probert, Nancy A. Lynch, and Nicola Santoro, editors, *PODC*, pages 27–30. ACM, 1983.

[Bor96] Malte Borderding. Levels of authentication in distributed agreement. In *Distributed Algorithms, 10th International Workshop, WDAG '96, Bologna, Italy, October 9-11, 1996, Proceedings*, pages 40–55, 1996.

[BPS16] Iddo Bentov, Rafael Pass, and Elaine Shi. Snow white: Provably secure proofs of stake. *IACR Cryptology ePrint Archive*, 2016:919, 2016.

[BR93] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *CCS '93, Proceedings of the 1st ACM Conference on Computer and Communications Security, Fairfax, Virginia, USA, November 3-5, 1993.*, pages 62–73, 1993.

[Bra84] Gabriel Bracha. An asynchronou [(n-1)/3]-resilient consensus protocol. In Tiko Kameda, Jayadev Misra, Joseph G. Peters, and Nicola Santoro, editors, *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing, Vancouver, B. C., Canada, August 27-29, 1984*, pages 154–162. ACM, 1984.

[BSA+17] Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, and George Danezis. Consensus in the age of blockchains. *CoRR*, abs/1711.03936, 2017.

[Can96] Ran Canetti. *Studies in Secure Multiparty Computation and Applications*. PhD thesis, Weizmann Institute of Science, 1996.

[Can00a] Ran Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[Can00b] Ran Canetti. Universally composable security: A new paradigm for cryptographic protocols. *IACR Cryptology ePrint Archive*, 2000:67, 2000.

[CCD87] David Chaum, Claude Crépeau, and Ivan Damgård. Multiparty unconditionally secure protocols (abstract) (informal contribution). page 462, 1987.

[CCGZ16] Ran Cohen, Sandro Coretti, Juan A. Garay, and Vassilis Zikas. Probabilistic termination and composability of cryptographic protocols. In Matthew Robshaw and Jonathan Katz, editors, *Advances in Cryptology - CRYPTO 2016 - 36th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2016, Proceedings, Part III*, volume 9816 of *Lecture Notes in Computer Science*, pages 240–269. Springer, 2016.

[CD89] Benny Chor and Cynthia Dwork. Randomization in byzantine agreement. *Advances in Computing Research*, 5:443–497, 1989.

[CFF+05] Jeffrey Considine, Matthias Fitzi, Matthew Franklin, Leonid A. Levin, Ueli Maurer, and David Metcalf. Byzantine agreement given partial broadcast. *J. Cryptol.*, 18(3):191–217, July 2005.

[CFGN96] Ran Canetti, Uriel Feige, Oded Goldreich, and Moni Naor. Adaptively secure multi-party computation. In Miller [Mil96], pages 639–648.

[CGHZ16] Sandro Coretti, Juan A. Garay, Martin Hirt, and Vassilis Zikas. Constant-round asynchronous multiparty computation based on one-way functions. In Jung Hee Cheon and Tsuyoshi Takagi, editors, *Advances in Cryptology - ASIACRYPT 2016 - 22nd International Conference on the Theory and Application of Cryptology and Information Security, Hanoi, Vietnam, December 4-8, 2016, Proceedings, Part II*, volume 10032 of *Lecture Notes in Computer Science*, pages 998–1021, 2016.

[CGMA85] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults (extended abstract). In *26th Annual Symposium on Foundations of Computer Science, Portland, Oregon, USA, 21-23 October 1985*, pages 383–395. IEEE Computer Society, 1985.

[CGR11] Christian Cachin, Rachid Guerraoui, and Lus Rodrigues. *Introduction to Reliable and Secure Distributed Programming*. Springer Publishing Company, Incorporated, 2nd edition, 2011.

[Cha81] David Chaum. Untraceable electronic mail, return addresses, and digital pseudonyms. *Commun. ACM*, 24(2):84–88, 1981.

[CKPS01] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Advances in Cryptology - CRYPTO 2001, 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23, 2001, Proceedings*, pages 524–541, 2001.

[CKS05] Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. *J. Cryptology*, 18(3):219–246, 2005.

[CL02] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[CPS07] Ran Canetti, Rafael Pass, and Abhi Shelat. Cryptography from sunspots: How to use an imperfect reference string. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings* [DBL07], pages 249–259.

[CR93] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In Kosaraju et al. [KJA93], pages 42–51.

[CW89] Brian A. Coan and Jennifer L. Welch. Modular construction of nearly optimal byzantine agreement protocols. In Piotr Rudnicki, editor, *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing, Edmonton, Alberta, Canada, August 14-16, 1989*, pages 295–305. ACM, 1989.

[DBL07] *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings*. IEEE Computer Society, 2007.

[DG17] Florian Dold and Christian Grothoff. Byzantine set-union consensus using efficient set reconciliation. *EURASIP Journal on Information Security*, 2017(1):14, Jul 2017.

[DGKR18] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *Advances in Cryptology - EUROCRYPT 2018 - 37th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Tel Aviv, Israel, April 29 - May 3, 2018 Proceedings, Part II*, volume 10821 of *Lecture Notes in Computer Science*, pages 66–98. Springer, 2018.

[DLS88] Cynthia Dwork, Nancy A. Lynch, and Larry J. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, 1988.

[DM90] Cynthia Dwork and Yoram Moses. Knowledge and common knowledge in a byzantine environment: Crash failures. *Inf. Comput.*, 88(2):156–186, 1990.

[DN92] Cynthia Dwork and Moni Naor. Pricing via processing or combatting junk mail. In *CRYPTO*, pages 139–147, 1992.

[DPPU88] Cynthia Dwork, David Peleg, Nicholas Pippenger, and Eli Upfal. Fault tolerance in networks of bounded degree. *SIAM J. Comput.*, 17(5):975–988, 1988.

[DR85] Danny Dolev and Rüdiger Reischuk. Bounds on information exchange for byzantine agreement. *J. ACM*, 32(1):191–204, 1985.

[DRS90] Danny Dolev, Rüdiger Reischuk, and H. Raymond Strong. Early stopping in byzantine agreement. *J. ACM*, 37(4):720–741, 1990.

[DS83] Danny Dolev and H. Raymond Strong. Authenticated algorithms for Byzantine agreement. *SIAM J. Comput.*, 12(4):656–666, 1983.

[Fel88] Paul Feldman. *Optimal algorithms for Byzantine agreement.* PhD thesis, Massachusetts Institute of Technology, 1988.

[FG03] Matthias Fitzi and Juan A. Garay. Efficient player-optimal protocols for strong and differential consensus. In *PODC*, pages 211–220, 2003.

[Fit03] Matthias Fitzi. *Generalized communication and security models in Byzantine agreement.* PhD thesis, ETH Zurich, Zürich, Switzerland, 2003.

[FL82] Michael J. Fischer and Nancy A. Lynch. A lower bound for the time to assure interactive consistency. *Inf. Process. Lett.*, 14(4):183–186, 1982.

[FLM86] Michael J. Fischer, Nancy A. Lynch, and Michael Merritt. Easy impossibility proofs for distributed consensus problems. *Distributed Computing*, 1(1):26–39, 1986.

[FLP85] Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.

[FM97] Pesech Feldman and Silvio Micali. An Optimal Probabilistic Protocol for Synchronous Byzantine Agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

[GHM+17] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 51–68. ACM, 2017.

[GKKO07] Juan A. Garay, Jonathan Katz, Chiu-Yuen Koo, and Rafail Ostrovsky. Round complexity of authenticated broadcast with a dishonest majority. In *48th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2007), October 20-23, 2007, Providence, RI, USA, Proceedings* [DBL07], pages 658–668.

[GKKZ11] Juan A. Garay, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. Adaptively secure broadcast, revisited. In Cyril Gavoille and Pierre Fraigniaud, editors, *Proceedings of the 30th Annual ACM Symposium on Principles of Distributed Computing, PODC 2011, San Jose, CA, USA, June 6-8, 2011*, pages 179–186. ACM, 2011.

[GKL14] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. *IACR Cryptology ePrint Archive*, 2014:765, 2014.

[GKL15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015.

[GKL17] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol with chains of variable difficulty. In Katz and Shacham [KS17], pages 291–323.

[GKLP16] Juan A. Garay, Aggelos Kiayias, Nikos Leonardos, and Giorgos Panagiotakos. Bootstrapping the blockchain - directly. *IACR Cryptology ePrint Archive*, 2016:991, 2016.

[GKLP18] Juan A. Garay, Aggelos Kiayias, Nikos Leonardos, and Giorgos Panagiotakos. Bootstrapping the blockchain, with applications to consensus and fast PKI setup. In Michel Abdalla and Ricardo Dahab, editors, *Public-Key Cryptography - PKC 2018 - 21st IACR International Conference on Practice and Theory of Public-Key Cryptography, Rio de Janeiro, Brazil, March 25-29, 2018, Proceedings, Part II*, volume 10770 of *Lecture Notes in Computer Science*, pages 465–495. Springer, 2018.

[GKP17] Juan A. Garay, Aggelos Kiayias, and Giorgos Panagiotakos. Proofs of work for blockchain protocols. *IACR Cryptology ePrint Archive*, 2017:775, 2017.

[GM98] Juan A. Garay and Yoram Moses. Fully polynomial byzantine agreement for $n > 3t$ processors in $t + 1$ rounds. *SIAM J. Comput.*, 27(1):247–290, 1998.

[GMW86] Oded Goldreich, Silvio Micali, and Avi Wigderson. Proofs that yield nothing but their validity and a methodology of cryptographic protocol design (extended abstract). pages 174–187, 1986.

[Gol01] Oded Goldreich. *The Foundations of Cryptography - Volume 1, Basic Techniques*. Cambridge University Press, 2001.

[GP92] Juan A. Garay and Kenneth J. Perry. A continuum of failure models for distributed computing. In Segall and Zaks [SZ92], pages 153–165.

[HLP11] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In Phillip Rogaway, editor, *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, volume 6841 of *Lecture Notes in Computer Science*, pages 132–150. Springer, 2011.

[HZ10] Martin Hirt and Vassilis Zikas. Adaptively secure broadcast. In *Advances in Cryptology - EUROCRYPT 2010, 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30 - June 3, 2010. Proceedings*, pages 466–485, 2010.

[KJA93] S. Rao Kosaraju, David S. Johnson, and Alok Aggarwal, editors. *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing, May 16-18, 1993, San Diego, CA, USA*. ACM, 1993.

[KK09] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for Byzantine agreement. *Journal of Computer and System Sciences*, 75(2):91 – 112, 2009.

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Katz and Shacham [KS17], pages 357–388.

[KS05] Klaus Kursawe and Victor Shoup. Optimistic asynchronous atomic broadcast. In Luís Caires, Giuseppe F. Italiano, Luís Monteiro, Catuscia Palamidessi, and Moti Yung, editors, *Automata, Languages and Programming, 32nd International Colloquium, ICALP 2005, Lisbon, Portugal, July 11-15, 2005, Proceedings*, volume 3580 of *Lecture Notes in Computer Science*, pages 204–215. Springer, 2005.

[KS16] Valerie King and Jared Saia. Byzantine agreement in expected polynomial time. *J. ACM*, 63(2):13:1–13:21, 2016.

[KS17] Jonathan Katz and Hovav Shacham, editors. *Advances in Cryptology - CRYPTO 2017 - 37th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 20-24, 2017, Proceedings, Part I*, volume 10401 of *Lecture Notes in Computer Science*. Springer, 2017.

[LLR06] Yehuda Lindell, Anna Lysyanskaya, and Tal Rabin. On the composition of authenticated byzantine agreement. *J. ACM*, 53(6):881–917, 2006.

[LSP82] Leslie Lamport, Robert E. Shostak, and Marshall C. Pease. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, 1982.

[Lyn96] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[Mic16] Silvio Micali. ALGORAND: the efficient and democratic ledger. *CoRR*, abs/1607.01341, 2016.

[Mil96] Gary L. Miller, editor. *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*. ACM, 1996.

[ML14] Andrew Miller and Joseph J. LaViola. Anonymous Byzantine consensus from moderately-hard puzzles: A model for bitcoin. University of Central Florida. Tech Report, CS-TR-14-01, April 2014.

[MXC+16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. The honey badger of BFT protocols. In Edgar R. Weippl, Stefan Katzenbeisser, Christopher Kruegel, Andrew C. Myers, and Shai Halevi, editors, *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*, pages 31–42. ACM, 2016.

[Nak08a] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. http://bitcoin.org/bitcoin.pdf, 2008.

[Nak08b] Satoshi Nakamoto. "The proof-of-work chain is a solution to the Byzantine Generals' problem". The Cryptography Mailing List, https://www.mail-archive.com/cryptography@metzdowd.com/msg09997.html, November 2008.

[Nak09] Satoshi Nakamoto. Bitcoin open source implementation of p2p currency. http://p2pfoundation.ning.com/forum/topics/bitcoin-open-source, February 2009.

[Nei94] Gil Neiger. Distributed consensus revisited. *Inf. Process. Lett.*, 49(4):195–201, 1994.

[NY89] Moni Naor and Moti Yung. Universal one-way hash functions and their cryptographic applications. In David S. Johnson, editor, *Proceedings of the 21st Annual ACM Symposium on Theory of Computing, May 14-17, 1989, Seattle, Washigton, USA*, pages 33–43. ACM, 1989.

[Oku05a] Michael Okun. Agreement among unacquainted Byzantine generals. In Pierre Fraigniaud, editor, *DISC*, volume 3724 of *Lecture Notes in Computer Science*, pages 499–500. Springer, 2005.

[Oku05b] Michael Okun. Distributed computing among unacquainted processors in the presence of Byzantine failures. Ph.D. Thesis Hebrew University of Jerusalem, 2005.

[PS17a] Rafael Pass and Elaine Shi. The sleepy model of consensus. In Tsuyoshi Takagi and Thomas Peyrin, editors, *Advances in Cryptology - ASIACRYPT 2017 - 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part II*, volume 10625 of *Lecture Notes in Computer Science*, pages 380–409. Springer, 2017.

[PS17b] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. Cryptology ePrint Archive, Report 2017/913, 2017. `https://eprint.iacr.org/2017/913`.

[PSL80] Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *Journal of the ACM*, 27(2):228–234, 1980.

[PSS17] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part II*, pages 643–673, 2017.

[PW92] Birgit Pfitzmann and Michael Waidner. Unconditional byzantine agreement for any number of faulty processors. In *STACS*, volume 577, pages 339–350. Springer, 1992.

[Rab83] Michael O. Rabin. Randomized Byzantine Generals. In *FOCS*, pages 403–409. IEEE Computer Society, 1983.

[Sch90] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, December 1990.

[SJS+18] Nicholas Stifter, Aljosha Judmayer, Philipp Schindler, Alexei Zamyatin, and Edgar R. Weippl. Agreement with satoshi - on the formalization of nakamoto consensus. *IACR Cryptology ePrint Archive*, 2018:400, 2018.

[SZ92] Adrian Segall and Shmuel Zaks, editors. *Distributed Algorithms, 6th International Workshop, WDAG '92, Haifa, Israel, November 2-4, 1992, Proceedings*, volume 647 of *Lecture Notes in Computer Science*. Springer, 1992.

[TC84] Russell Turpin and Brian A. Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Information Processing Letters*, 18(2):73–76, 1984.

[Upf92] Eli Upfal. Tolerating linear number of faults in networks of bounded degree. In Norman C. Hutchinson, editor, *Proceedings of the Eleventh Annual ACM Symposium on Principles of Distributed Computing, Vancouver, British Columbia, Canada, August 10-12, 1992*, pages 83–89. ACM, 1992.

[Yao82] Andrew Chi-Chih Yao. Protocols for secure computations (extended abstract). pages 160–164, 1982.

A Ideal Functionalities

The ideal functionality that captures RMT is presented in Figure 3 assuming a synchronous operation; (cf. Section 3.2 where we discuss how the synchrony requirement can be relaxed). Given that not all parties may be required to send a message in each communication round, the functionality has to keep track of party activations and advance to the next "round" only when all parties have been given a chance to act (note that an activation does not necessarily imply performing any protocol tasks).

The ideal functionality capturing the diffuse operation is presented in Figure 4 assuming again synchronous network operation (likewise refer to Section 3.2 where we discuss how the synchrony requirement can be relaxed). A salient feature of protocols running in the $\mathcal{F}_{\text{Diffuse}}$ setting, is that the session id may just provide an abbreviation of the universe of parties $\mathcal{P} = \{P_1, \ldots, P_n\}$, of which only a subset may be activated.

The functionality capturing the formalisation of a hash function as a random oracle is shown in Figure 5.

Functionality $\mathcal{F}_{\text{RMT}}$

The functionality interacts with an adversary $\mathcal{S}$ and a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of parties. Initialize a Boolean flag $\mathsf{flag}(P_i)$ to false and a string $\mathsf{inbox}(P_i)$ to empty, for all $i = 1, \ldots, n$.

- Upon receiving $(\mathsf{Send}, \mathsf{sid}, P_i, P_j, m)$ from P_i, store $(\mathsf{Send}, \mathsf{sid}, P_i, P_j, m)$ and hand $(\mathsf{Send}, \mathsf{sid}, P_i, P_j, m, mid)$ to $\mathcal{S}$, where mid is a unique identifier.
- Upon receiving $(\mathsf{Activate}, \mathsf{sid}, P_i)$ from P_i, set $\mathsf{flag}(P_i)$ to true. If it holds that $\wedge_{i=1}^{n} \mathsf{flag}(P_i)$, then for all $i = 1, \ldots, n$, set $\mathsf{flag}(P_i)$ to false, and for any $(\mathsf{Send}, \mathsf{sid}, P_i, P_j, m, mid)$ that is recorded as unsent, mark it as sent, and copy $(\mathsf{Send}, \mathsf{sid}, P_i, P_j, m, mid)$ to $\mathsf{inbox}(P_j)$.
- Upon receiving $(\mathsf{Deliver}, \mathsf{sid}, mid)$ from $\mathcal{S}$, assuming $(\mathsf{Send}, \mathsf{sid}, P_i, P_j, m, mid)$ is recorded as unsent, mark it as sent, and copy $(\mathsf{Send}, \mathsf{sid}, P_i, P_j, m)$ to $\mathsf{inbox}(P_j)$.
- Upon receiving $(\mathsf{Fetch}, \mathsf{sid}, P_i)$ from P_i, return $\mathsf{inbox}(P_i)$ to P_i and set $\mathsf{inbox}(P_i)$ to empty.

Figure 3: *The reliable message transmission ideal functionality in the synchronous setting.*

Functionality $\mathcal{F}_{\text{Diffuse}}$

The functionality interacts with an adversary $\mathcal{S}$ and a set $\mathcal{U}$ of parties. Initialize a subset $A \subseteq \mathcal{U}$ to $\emptyset$, a Boolean flag $\mathsf{flag}(P_i)$ to false, and a string $\mathsf{inbox}(P_i)$ to empty, for all i such that $P_i \in \mathcal{U}$.

- Upon receiving $(\mathsf{Send}, \mathsf{sid}, P_i, m)$ from P_i, set $\mathsf{flag}(P_i)$ to true, store $(\mathsf{Send}, \mathsf{sid}, P_i, m)$ and hand $(\mathsf{Send}, \mathsf{sid}, P_i, m, mid)$ to $\mathcal{S}$, where mid is a unique identifier.
- Upon receiving $(\mathsf{Activate}, \mathsf{sid}, P_i)$ from P_i, set $A = A \cup \{P_i\}$ and $\mathsf{flag}(P_i)$ to true. If it holds that $\wedge_{i \in A} \mathsf{flag}(P_i)$, then for all $i = 1, \ldots, n$, set $\mathsf{flag}(P_i)$ to false, and for any P_j, $j \in A$, and any $(\mathsf{Send}, \mathsf{sid}, P_i, m, mid)$ that is recorded as unsent for P_j, mark it as sent for P_j, and copy $(\mathsf{Send}, \mathsf{sid}, P_i, P_j, m, mid)$ to $\mathsf{inbox}(P_j)$.
- Upon receiving $(\mathsf{Deliver}, \mathsf{sid}, mid, P_i', P_j)$ from $\mathcal{S}$ and $j \in A$, assuming $(\mathsf{Send}, \mathsf{sid}, P_i, m, mid)$ is recorded as unsent for P_j, mark it as sent for P_j, and copy $(\mathsf{Send}, \mathsf{sid}, P_i', P_j, m)$ to $\mathsf{inbox}(P_j)$.
- Upon receiving $(\mathsf{Fetch}, \mathsf{sid}, P_i)$ from P_i, return $\mathsf{inbox}(P_i)$ to P_i and set $\mathsf{inbox}(P_i)$ to empty.

Figure 4: *The peer-to-peer diffuse ideal functionality in the synchronous setting.*

Functionality $\mathcal{F}_{\text{RO}}$

The functionality interacts with an adversary $\mathcal{S}$ and a set $\mathcal{P} = \{P_1, \ldots, P_n\}$ of parties.

- Upon receiving $(\mathsf{Eval}, \mathsf{sid}, x)$ from P_i (resp. $\mathcal{S}$), return ρ to P_i (resp. $\mathcal{S}$) if $(x, \rho) \in T$. If no entry for x is in T, then choose $\rho \leftarrow \{0,1\}^{\kappa}$, add (x, ρ) in T and return ρ to P_i.

Figure 5: *The random oracle ideal functionality.*