Synchronous, with a Chance of Partition Tolerance

Yue Guo, Rafael Pass, and Elaine Shi

Thunder Research, Sunnyvale, USA
Cornell University, Ithaca, USA
{yue.guo,rafael,elaine}@thundercore.com

Abstract. Murphy, Murky, Mopey, Moody, and Morose decide to write a paper together over the Internet and submit it to the prestigious CRYPTO'19 conference that has the most amazing PC. They encounter a few problems. First, not everyone is online every day: some are lazy and go skiing on Mondays; others cannot use git correctly and they are completely unaware that they are losing messages. Second, a small subset of the co-authors may be secretly plotting to disrupt the project (e.g., because they are writing a competing paper in stealth).
Suppose that each day, sufficiently many honest co-authors are online (and use git correctly); moreover, suppose that messages checked into git on Monday can be correctly received by honest and online co-authors on Tuesday or any future day. Can the honest co-authors successfully finish the paper in a small number of days such that they make the CRYPTO deadline; and perhaps importantly, can all the honest co-authors, including even those who are lazy and those who sometimes use git incorrectly, agree on the final theorem?

1 Introduction

The "synchronous" model is one of the most commonly studied models in the past 30 years of distributed computing and cryptography literature. In the synchronous model, it is assumed that whenever an honest node sends a message, an honest recipient is guaranteed to have received it within a bounded delay Δ, and the protocol is aware of the maximum delay Δ.

We love the synchronous model because it allows us to achieve robustness properties that would otherwise be impossible. For example, assuming synchrony, we can achieve distributed consensus even when arbitrarily many nodes may be malicious [8]. In comparison, it is well-known that if message delays can be arbitrarily long [9], consensus is impossible in the presence of $\frac{1}{3}$ fraction of corrupt nodes. On the other hand, the synchrony assumption has been criticized for being too strong [3,19]: if an honest node ever experiences even a short outage (e.g., due to network jitter) during which it is not able to receive honest messages within Δ delay, this node is now considered as corrupt. From this point on, a consensus protocol proven secure under a synchronous model is not obliged to provide consistency and liveness to that node any more, even if the node may wake up shortly afterwards and wish to continue participating in the protocol. Similarly, as soon as P has even a short-term outage, a multi-party computation

(MPC) protocol proven secure under a synchronous model is not obliged to provide privacy for party P's inputs — for example, some protocols that aim to achieve fairness and guaranteed output would now have the remaining online parties reconstruct P's secret-shared input and thus P loses its privacy entirely.

We stress that this is not just a theoretical concern. Our work is in fact directly motivated by conversations with real-world blockchain engineers who were building and deploying a fast cryptocurrency and pointed out what seems to be a fatal flaw in a blockchain protocol [20] that was proven secure in the classical synchronous model: even when all nodes are benign and only a few crash in a specific timing pattern, transactions that were "confirmed" can be "undone" from the perspective of an honest node who just experienced short-term jitter possibly unknowingly (see the online full version [14] for a detailed description of this real-world example).

Not only so, in fact to the best of our knowledge, all known classical-style, *synchronous* consensus protocols [2, 15, 18] are *underspecified* and *unimplementable* in practice: if a node ever experiences even a short-term outage and receives messages out of sync, these protocols [2, 15, 18] provide no explicit instructions for such nodes to join back and continue to enjoy consistency and liveness!

Of course, one known solution to this problem is to simply adopt a partially synchronous [9] or asynchronous [6] model. In a partially synchronous or asynchronous model, a short-term outage would be treated in the same way as a long network delay, and a node that is transiently offline will not be penalized. For this reason, partially synchronous (or asynchronous) protocols are known to be arbitrarily *partition tolerant*; while synchronous protocols are *not*. Unfortunately, as mentioned, partially synchronous or asynchronous protocols can tolerate only 1/3 fraction of corruptions!

Can we achieve the best of both worlds, i.e., design distributed computing protocols that resist more than 1/3 *corruption and meanwhile achieve a practical notion of partition tolerance?*

1.1 Definitional Contribution: A "Weak Synchronous" Network

At a very high level, we show that synchrony and partition tolerance are not binary attributes, and that we can guarantee a notion called "best-possible partition tolerance" under a quantifiable measure of synchrony. To this end, we propose a new model called a χ-*weakly-synchronous* network.

A natural but overly restrictive notion. One natural way to quantify the degree of synchrony is to count the fraction of nodes that always respect the synchrony assumption. For example, we may want a distributed computing protocol to satisfy desired security properties (e.g., consistency, liveness, privacy), as long as more than χ fraction of nodes are not only honest but also always have good connectivity (i.e., bounded Δ delay) among themselves. This model, however, is overly restrictive especially in long-running distributed computing tasks such as a blockchain: after all, no node can guarantee 100% up-time [1], and over a few years duration, it could be that every node was at some point, offline.

χ-weak-synchrony. We thus consider a more general model that allows us to capture network churn. We now require only the following:

> [χ-weakly-synchronous assumption:] In every round, more than χ fraction nodes are not only honest but also *online*; however, the set of honest and online nodes in adjacent rounds need not be the same.

Throughout the paper we use the notation $\mathcal{O}_r$ to denote a set of at least $\lfloor \chi n \rfloor + 1$ honest nodes who are online in round r — henceforth $\mathcal{O}_r$ is also called the "honest and online set of round r". Note that the remaining set $[n] \backslash \mathcal{O}_r$ may contain a combination of honest or corrupt nodes and an honest node in $[n] \backslash \mathcal{O}_r$ is said to be offline in round r.

We assume that the network delivery respects the following property where multicast means "send to everyone":

> [network assumption:] when a node in $\mathcal{O}_r$ multicasts a message in round r, every node in $\mathcal{O}_t$ where $t \geq r + \Delta$ must have received the message in round t.

We allow the adversary to choose the honest and online set of each round (even after observing the messages that honest nodes want to send in the present round), and delay or erase honest messages, as long as the above χ-weak-synchrony and network delivery constraints are respected. For example, the adversary may choose to delay an honest but offline node's messages (even to online nodes) for as long as the node remains offline. The adversary can also selectively reveal an arbitrary subset of honest messages to an honest and offline node.

Therefore, our weak synchrony notion can be viewed as a generalization of the classical synchronous notion (henceforth also called strong synchrony). In a strongly synchronous network, it is required that the honest and online set of every round must contain all honest nodes.

We ask whether we can achieve secure distributed computing tasks under such a χ-weakly-synchronous network. With the exception of liveness (or guaranteed output) which we shall discuss shortly, we would like to guarantee all security properties, including consistency and privacy, for *all* honest nodes, regardless of whether or when they are online/offline. Defining liveness (or guaranteed output) in the χ-weakly-synchronous model, however, is more subtle. Clearly we cannot hope to guarantee liveness for an honest but offline node for as long as it remains offline. Therefore, we aim to achieve a "best-effort" notion of liveness: a protocol has T-liveness iff for any honest node that becomes online in some round $r \geq T$, it must have produced output by the end of round r.

The challenges. We are faced with a few apparent challenges when designing distributed protocols secure under χ-weak-synchrony. First, the online nodes may change rapidly in adjacent rounds. For example, if $\chi = 0.5$ and everyone is honest, the honest and online sets belonging to adjacent rounds can be almost disjoint. Second, we stress that *offline nodes may not be aware they are offline*, e.g., a DoS attack against a victim's egress router clearly will not announce itself in advance. Further, the adversary can selectively reveal a subset of messages to offline nodes such that they cannot detect they are offline from the protocol messages they receive either. Because of these facts, designing protocols in our

χ-weakly-synchronous model is significantly more challenging than the classical synchronous model (or even the above restrictive model where we may assume a sufficiently large set of honest and persistently online nodes).

1.2 Results: Consensus in a Weakly Synchronous Network

We consider the feasibility and infeasibility of achieving Byzantine Agreement (BA) in a χ-weakly-synchronous network. In a BA protocol, a designated sender has an input bit that it wants to convey to all other nodes. We would like to achieve the following guarantess for all but a negligible fraction of executions: 1) *consistency*, i.e., all honest nodes must output the same bit; 2) *validity*, i.e., if the designated sender is honest and online in the starting round (i.e., round 0) of the protocol, every honest node's output (if any) must agree with the sender's input bit; and 3) T-*liveness*, i.e., every node in $\mathcal{O}_r$ where $r \geq T$ must have produced an output by the end of round r. Note that if the designated sender is honest but offline initially, the protocol cannot make up for the time lost when the sender is offline — thus validity requires that the sender not only be honest but also online in the starting round.

As mentioned, we are primarily interested in protocols that tolerate more than 1/3 corruptions since otherwise one could adopt a partially synchronous or asynchronous model and achieve arbitrary partition tolerance. To avoid a well-known lower bound by Lamport et al. [17], throughout the paper we will assume the existence of a public-key infrastructure (PKI).

Impossibility when minority are honest and online. Unfortunately, we show that it is impossible to have a χ-weakly-synchronous consensus protocol for $\chi < 0.5 - 1/n$, i.e., if the honest and online set of each round contains only minority number of nodes (and this lower bound holds even assuming any reasonable setup assumption such as PKI, random oracle, common reference string (CRS), or the ability of honest nodes to erase secrets from memory). The intuition for the lower bound is simple: there can be two honest well-connected components that are partitioned from each other, i.e., the minority honest nodes inside each component can deliver messages to each other within a single round; however messages in between incur very long delay. In this case, by liveness of the consensus protocol, each honest well-connected component will reach agreement independently of each other. We formalize this intuition later in Section 4.

Best-possible partition tolerance. Due to the above impossibility, a consensus protocol that achieves consistency, validity, and liveness under 0.5-weak-synchrony is said to be *best-possible partition tolerant*.

A refinement of synchronous consensus. First, it is not hard to see that any best-possible partition tolerant Byzantine Agreement (BA) protocol (i.e., secure under 0.5-weak-synchrony) must also be secure under honest majority in the classical, strong synchronous model. On the other hand, the converse is not true. Interestingly, we examined several classical, honest-majority BA protocols [2, 15, 18, 20] and found none of them to satisfy best-possible partition tolerance. In this sense, our notion of best-possible partition tolerance can also be viewed as a refinement

of classical honest-majority BA, i.e., we can tease out a proper subset of honest-majority BA protocols that satisfy good-enough partition tolerance in practice — and we strongly recommend this robust subset for practical applications.

Round-efficient, best-possible partition tolerant BA. Of course, to show that our notion is useful, we must show existence of a best-possible partition tolerant BA that is efficient; and this turns out to be non-trivial.

Theorem 1 (Informal). *Assume the existence of a PKI and enhanced trapdoor permutations. Then, there exists an expected constant-round* BA *protocol secure under* 0.5*-weak-synchrony.*

Note that here, expected constant-round means that there is a random variable T whose expectation is constant, such that if an honest node becomes online in round $r \geq T$, it must have produced an output in round r.

We additionally show how to extend the above result and construct a best-possible partition tolerant BA protocol that is optimistically responsive [20]: specifically, under the following optimistic conditions, the honest and online nodes in $\mathcal{O}$ will produce an output in $O(\delta)$ amount of time where δ is the actual maximum network delay (rather than the a-priori upper bound Δ):

> **O** := "there exists a set $\mathcal{O}$ containing at least $3n/4$ honest and persistently online nodes, and moreover, the designated sender is not only honest but also online in the starting round"

Corollary 1 (Informal). *Assume the existence of a PKI and enhanced trapdoor permutations. Then, there exists an expected constant-round* BA *protocol secure under χ-weak-synchrony; moreover, if the optimistic conditions* **O** *specified above also holds, then the honest and online nodes in $\mathcal{O}$ would produce output in $O(\delta)$ time where δ is the actual maximum network delay.*

Classical, corrupt-majority BA *protocols inherently sacrifice partition tolerance.* As is well-known, in the classical, strongly synchronous model, we can achieve BA even when arbitrarily many nodes can be corrupt. We show, however, the set of corrupt-majority protocols are disjoint from the set of best-possible partition tolerant protocols. Not only so, we can show that the more corruptions one hopes to tolerate, the less partition tolerant the protocol becomes. Intuitively, the lower bound is simple because in a corrupt majority protocol, a minority honest well-connected component must independently reach agreement among themselves in a bounded amount of time; and obviously there can be two such components that are disconnected from each other and thus consistency among the two components is violated (with constant probability).

This simple observation, however, raises another philosophical point: if we adopted the classical synchronous model, it would be tempting to draw the conclusion that corrupt-majority BA is strictly more robust than honest-majority BA. However, we show that one must fundamentally sacrifice partition tolerance to trade for the ability to resist majority corruption and this tradeoff is, unfortunately, inherent.

1.3 Results: MPC in a Weakly Synchronous Network

We next consider the feasibility of realizing multi-party computation in a χ-weakly-synchronous network. Imagine that n parties would like to jointly evaluate the function $f(x_1, \ldots, x_n)$ over their respectively inputs $x_1, x_2, \ldots, x_n$ such that only the outcome is revealed and nothing else. Again, a couple of subtleties arise in formulating the definition. First, one cannot hope to incorporate the inputs of offline nodes if one would like online nodes to obtain outputs quickly. Thus, we require that at least $\lfloor \chi n \rfloor + 1$ number of honest nodes' inputs be included and moreover, every honest node who has always been online throughout the protocol should get their inputs incorporated. Concretely, we require that the ideal-world adversary submit a subset $\mathfrak{I} \subseteq [n]$ to the ideal functionality, such that $\mathfrak{I} \cap \mathsf{Honest} \geq \lfloor \chi n \rfloor + 1$ where Honest denotes the set of honest nodes, and moreover $\mathfrak{I}$ must include every honest node who has been online throughout the protocol. Henceforth, the subset $\mathfrak{I}$ is referred to as the "effective input set":

- for every $i \in \mathfrak{I}$ that is honest, the computation should use node i's true inputs;
- for every $i \in \mathfrak{I}$ that is corrupt, we allow the ideal-world adversary to replace the input to any value of its choice; and
- for every $i \notin \mathfrak{I}$, the computation simply uses a canonical input $\perp$ as its input.

Second, the notion of guaranteed output must be treated in the same manner as liveness for BA since we cannot hope that honest but offline nodes can obtain output for as long as they remain offline. We say that an execution of the multi-party protocol completes in T rounds, iff for any honest node in $\mathcal{O}_t$ where $t \geq T$, it must have produced an output by the end of round t.

Under the above definition, we prove the following theorem (informally stated):

Theorem 2 (Informal). *Assume the existence of a PKI, enhanced trapdoor permutations, and that the Learning with Errors (LWE) assumption holds. Then, there is an expected constant-round protocol that allows multiple parties to securely evaluate any function f under 0.5-weak-synchrony.*

We further extend our results in a non-trivial manner and and achieve optimistically responsive MPC in the online full version [14].

Additional related work. We provide comparison with additional related work in our online full version [14].

2 Technical Roadmap

The most technically non-trivial part of our result is how to realize Byzantine Agreement (BA) under 0.5-weak-synchrony. Existing synchronous, honest-majority protocols [15, 18] completely fail in our model. Since the honest and online set can change rapidly in every round, it could be that by the end of the protocol, very few or even no honest nodes have been persistently online, and everyone honest was offline at some point. In other words, it could be that from the view of every honest node, message delivery was asynchronous at some point

in the protocol. Indeed, interestingly many of our core techniques are in fact reminiscent of asynchronous consensus rather than synchronous approaches.

Although at a very high level, we follow a well-known recipe that constructs BA from a series of building blocks:

$$\text{Reliable Broadcast (RBC)} \Rightarrow \text{Verifiable Secret Sharing (VSS)}$$
$$\Rightarrow \text{Leader Election (LE)} \Rightarrow \text{Byzantine Agreement (BA)}$$

as it turns out, for all these building blocks, even how to define them was non-trivial: the definitional subtleties arise partly due to the new χ-weakly-synchronous model, and partly due to compositional issues.

2.1 Reliable Broadcast (RBC)

Definition. Reliable broadcast (RBC) allows a designated sender to convey a message to other nodes. The primitive can be viewed as a relaxed version of BA: assuming 0.5-weak-synchrony, RBC always guarantees the following for all but a negligible fraction of executions:

1. *Consistency:* if two honest nodes output x and x' respectively, it must be that $x = x'$. For technical reasons that will become clear later, we actually need a strengthening of the standard consistency notion, requiring that an efficient extractor can extract the value that honest nodes can possibly output, given honest nodes' transcript in the initial T rounds of the protocol.
2. *Validity:* if the sender is honest, then honest nodes' output must be equal to the honest sender's input;
3. *T-liveness (under an honest and initially online sender):* if the sender is not only honest but also online in the starting round, then every node in $\mathcal{O}_t$ where $t \geq T$ must have produced an output by the end of round t;
4. *Close termination:* if any honest node (even if offline) produces and output in round r, then anyone in $\mathcal{O}_t$ where $t \geq r + 2\Delta$ must have produced an output by the end of round t too.

Interestingly, note that the T-liveness property is reminiscent of classical synchronous definitions whereas the close termination property is reminiscent of asynchronous definitions.

Construction. At a very high level, our RBC construction combines techniques from classical synchronous "gradecast" [10,15] and asynchronous "reliable broadcast" [5, 6]. We defer the concrete construction to Section 5; the construction is constant round, i.e., achieves T-liveness where $T = O(1)$.

2.2 Verifiable Secret Sharing (VSS)

Definition. Verifiable secret sharing (VSS) allows a dealer to share a secret among all nodes and later reconstruct it. We propose a new notion of (a computationally secure) VSS that is composable and suitable for a 0.5-weakly-synchronous network. Somewhat imprecisely, we require the following properties:

– *Binding (formally referred to as Validity in Section 6.2).* Standard notions of VSS [6] require that the honest transcript of the sharing phase binds to the honestly reconstructed secret. For technical reasons needed later in the proof of the Leader Election (LE), we require a stronger notion: an efficient extractor $\mathcal{E}$, knowing honest nodes' public and secret keys, must be able to extract this secret from the honest transcript during the sharing phase, and the honestly reconstructed secret must agree with the extractor's output.
– *Secrecy and non-malleability.* If the dealer is honest, then the shared value must remain secret from the adversary before reconstruction starts. Not only so, we also need a non-malleablity: an adversary, after interacting in VSS instances each with an honest dealer, cannot act as a dealer in another VSS instance and share a secret that is related to the honest secrets.
– *Liveness.* For liveness, we require that if the dealer is honest and online in the initial round of the sharing phase, for $t \geq T$, everyone in $\mathcal{O}_t$ must have output "`sharing-succeeded`". Even when the dealer is corrupt or initially offline, if any honest node (even if offline) ever outputs "`sharing-succeeded`" in some round r, then everyone in $\mathcal{O}_t$ where $t \geq r + 2\Delta$ must have output "`sharing-succeeded`" by the end of round t. If some honest node has output "`sharing-succeeded`", then reconstruction must be successful and will terminate in T rounds for honest and online nodes.

Just like the RBC definition, our VSS definition also has both synchronous and asynchronous characteristics.

Construction. Informally our construction works as follows:

– Share. In the starting round of the sharing phase, the dealer secret splits its input s into n shares denoted $s_1, s_2, \ldots, s_n$ using a $(\lfloor n/2 \rfloor + 1)$-out-of-$n$ secret-sharing scheme. It then encrypts the share s_j to node i's public key pk_j using a public-key encryption scheme — let CT_j be the resulting ciphertext. Now, the node proves in zero-knowledge, non-interactively, that the ciphertexts $\mathsf{CT}_1, \ldots, \mathsf{CT}_n$ are correct encryptions of an internally consistent sharing of some secret — let π denote the resulting proof. Assuming PKI and honest majority, we can realize a Non-interactive Zero-Knowledge Proof (NIZK) system (without CRS) using a technique called multi-string honest majority NIZK proposed by Groth and Ostrovsky [13] (see online full version [14]). Finally, the dealer invokes an RBC instance (henceforth denoted RBC_0) to reliably broadcast the tuple $(sid, \{\mathsf{CT}_j\}_{j\in[n]}, \pi)$ to everyone — here sid denotes the current instance's unique identifier and this term is needed here and also included in the NIZK statement for compositional reasons.
Suppose that the RBC scheme employed satisfies T_{rbc}-liveness. Now in round T_{rbc} (assuming that the starting round is renamed to round 0), if a tuple has been output from the RBC_0 instance with a valid NIZK proof, then reliably broadcast the message "`ok`"; otherwise reliably broadcast the message $\perp$ — note that here n instances of RBC are spawned and each node i will act as the designated sender in the i-th instance. Finally, output "`sharing-succeeded`" iff not only RBC_0 has output a tuple with a valid NIZK proof but also at

least $\lfloor n/2 \rfloor + 1$ RBC instances have output "ok" — note that at this moment, the node (denoted i) can decrypt its own share s'_i from the corresponding ciphertext component contained in the output of RBC_0.
- Reconstruct: If the sharing phase has output "sharing-succeeded" and moreover the reconstruction phase has been invoked, then node i multicasts the decrypted share s'_i as well as a NIZK proof that the decryption was done correctly (in a way that is consistent with its public key). Finally, as soon as $\lfloor n/2 \rfloor + 1$ decryption shares with valid NIZK proofs are received, one can reconstruct the secret.

2.3 Leader Election (LE)

Definition. Leader Election (LE) is an inputless protocol that allow nodes to elect a leader denoted $L \in [n]$ among the n nodes. For the outcome of LE to be considered "good", we want that not only every honest node must agree on the leader, but also that this leader belongs to $\mathcal{O}_r$ for some a-priori known round r — jumping ahead, later in our BA protocol, everyone would attempt to propose a value during this round r and the proposal of the elected leader will be chosen.

Intuitively, we would like that the LE achieves a good outcome with $O(1)$ probability. Our actual definition turns out to be tricky due to compositional issues that arise due to multiple LE instances sharing the same PKI. We would like that even when multiple LE instances share the same PKI, roughly speaking, almost surely there is still *independent* constant probability that each individual instance's outcome is good. In formal definition (see Section 7), we will precisely specify which subset of honest coins that are freshly chosen in each LE instance allow us to capture this desired independence. Note that this independence property is desired because later in our BA protocol, we need to argue that after a bounded number of trials, an honest leader must be elected except with negligible probability.

Construction. Our LE protocol is in fact inspired by the *asynchronous* leader election protocol by Canetti and Rabin [6]. Since our LE construction is rather technical, we explain a high-level intuition here while deferring the full protocol to Section 7. The idea is for everyone i to choose n coins denoted $c_{i,1}, \ldots, c_{i,n} \in \mathbb{F}$, one for each person. All these coins will be committed to using a VSS protocol such that corrupt nodes cannot choose their coins after seeing honest coins. Each person j's *charisma* is the product of the coins chosen for him by at least $\lfloor n/2 \rfloor + 1$ others, i.e., $\prod_{i \in D_j} c_{i,j}$ where $D_j \subseteq [n]$ and $|D_j| \geq \lfloor n/2 \rfloor + 1$ — in this way, at least one in D_j is honest and has chosen a random coin. In our protocol, every person j will announce this set D_j itself through an RBC protocol. Ideally we would like nodes to agree on a set of candidates that contain many nodes in $\mathcal{O}_r$ for some r, and elect the candidate with the maximum charisma (lexicographically) from this set — unfortunately at this moment we do not have Byzantine Agreement yet. Thus we must accomplish this task without reaching agreement. Our idea is for each node to *independently calculate a sufficiently large set of candidates; and although honest nodes may not agree on this candidate set, honest nodes' candidate sets must all contain every node in* $\mathcal{O}_r$. We

stress that the challenge is that honest offline nodes' candidate sets must also satisfy this property even though they are receiving only an arbitrary subset of messages chosen by the adversary — note that these nodes basically have "asynchronous" networks. Perhaps more challengingly, it could be that every honest node may be offline in some round, and thus everyone's network may be asynchronous at some point.

Towards this end, we adapt Canetti and Rabin's leader election idea [6] to our weakly synchronous setting: specifically, everyone first reliably broadcasts a *tentative* candidate set S, but they keep maintaining and growing a local candidate set denoted $S^* \supseteq S$. They would keep adding nodes that they newly deem eligible to their local set S^*, until at some point, they decide that their local set S^* is sufficiently inclusive based on sufficiently many tentative candidate sets that have been reliably broadcast. At this moment, the node stops growing its local candidate set and outputs the candidate with maximum charisma from its current local set. We refer the reader to Section 7 for a detailed description.

2.4 Byzantine Agreement (BA)

The next question is how to construct BA given weakly synchronous LE. This step turns out to be non-trivial too. In particular, we stress that existing synchronous BA protocols [2, 15, 18] are broken under 0.5-weak-synchrony, not only because they lack a good leader election (or common coin) algorithm — in fact even if we replaced the leader election in existing schemes with an ideal version (e.g., our own leader election scheme in Section 7), the resulting BA schemes would still be broken under 0.5-weak-synchrony. All existing synchronous BA schemes make use of synchrony in a *strong* manner: they rely on the fact that if an honest node i sees some message m in round t, then i is surely able to propagate the message to *every* honest node by the end of round $t + \Delta$. This assumption is not true in our model since our model does not provide any message delivery guarantees for offline honest nodes. Instead, our protocol makes use of only weak synchrony and specifically the following observation (and variants of it): if $\lfloor n/2 \rfloor + 1$ number of nodes declare they have observed a message m by the end of round t, then at least one of them must be in $\mathcal{O}_t$ and if all of these nodes try to propagate the message m to others in round t, then everyone in $\mathcal{O}_{t^*}$ where $t^* \geq t + \Delta$ must have observed m by the end of round t^*.

At a very high level, our protocol proceeds in epochs. We make the following simplifying assumptions for the time being: 1) $\Delta = 1$, and 2) every node keeps echoing every message they have seen in every round (in our later technical sections we will remove the need for infinite echoing):

- *Propose:* For the first epoch, the designated sender's signature on a bit is considered a valid proposal. For all other epochs, at epoch start a leader election protocol is invoked to elect a leader. Recall that with constant probability, the leader election algorithm guarantees the following "good" event G: 1) the LE protocol guarantees that the elected leader is in $\mathcal{O}_r$ for some pre-determined round r; and 2) no two honest nodes output inconsistent leaders. Now imagine that in precisely round r of this epoch, everyone ten-

tatively proposes a random bit b — and if the node indeed gets elected as a leader the proposed bit will be recognized as a valid proposal[1].

- *Vote (formally called "Prepare" later):* Let T_{le} be the liveness parameter of the LE scheme. In round T_{le} of the epoch e, a node votes on the elected leader's proposal if in epoch $e-1$ majority nodes complained of not having received majority votes for either bit — in this case no honest node can have made a decision yet. Otherwise if the node has observed majority votes for some bit b' from the previous epoch $e-1$, it votes for b' — in this case some honest node might have made a decision on b' and thus we might need to carry on the decision. Henceforth the set of majority votes for b' from epoch $e-1$ is said to be an epoch-e pseudo-proposal for b'.
- *Commit:* In round $T_{\text{le}}+1$ of the epoch e, a node sends an epoch-e commit message for a bit b, iff it has observed majority epoch-e votes on b, and no epoch-e proposal or pseudo-proposal for $1-b$ has been seen.
- *Complain:* In round $T_{\text{le}}+2$ of the epoch e, send a complaint if neither bit gained majority votes in this epoch.

At any time, if $\lfloor n/2 \rfloor + 1$ number of commits from the same epoch and for the same bit b have been observed, output b and continue participating in the protocol (we describe a termination technique in the online full version [14]).

Remark 1. We point out that although our BA protocol might somewhat resemble the recent work by Abraham et al. [2], their protocol is in fact broken under 0.5-weak-synchrony (even if they adopted an ideal leader election protocol) for a couple of reasons. In their protocol, in essence a node makes a decision if the node itself has seen majority votes and no conflicting proposal. To ensure consistency under weak synchrony, our protocol makes a decision when majority votes have been collected and moreover, *majority nodes have declared that they have not seen a conflicting proposal (or pseudo-proposal).* Finally, we introduce a "complain" round, and technically this (and together with the whole package) allows us to achieve liveness under 0.5-weak-synchrony — in comparison, Abraham et al.'s protocol [2] appears to lack liveness under weak synchrony.

2.5 Multi-Party Computation

We now consider multi-party computation in a weakly synchronous network. Specifically, we will consider the task of secure function evaluation (SFE). Imagine that n nodes each has an input where node i's input is denoted x_i. The nodes would like to jointly compute a function $f(x_1, \ldots, x_n)$ over their respective inputs. The privacy requirement is that besides learning the outcome, each node must learn nothing else (possibly in a computational sense). Recall that earlier in our Byzantine Agreement (BA) protocols, there is no privacy requirement, and therefore our goal was to ensure that honest nodes who drop offline do not risk inconsistency with the rest of the network. With SFE, we would like to protect

[1] This is necessary because if a single proposer made a proposal *after* being elected, the adversary could make the proposer offline in that precise round.

not only the consistency but also the *input-privacy* of those who are benign but drop offline or have unstable network connection.

Of course, in a weakly synchronous environment, if we would like online nodes to obtain outputs in a bounded amount of time, we cannot wait forever for offline honest nodes to come online. Thus, in our definition, we require that 1) *at least* $n/2$ *honest nodes' inputs be included in the computation*; and 2) every honest node that remains online during the protocol must get their inputs incorporated. Note that the second requirement ensures that our notion is strictly stronger (i.e., more robust) than classical synchronous MPC under honest majority.

Construction. Our goal is to construct an expected constant-round SFE protocol secure under 0.5-weak-synchrony. The naïve approach of taking *any* existing MPC and replacing the "broadcast" with our weakly synchrounous BA (see earlier subsections of this section) may not solve the problem. Specifically, we need to additionally address the following challenges:

1. Classical synchronous MPC protocols are not required to provide secrecy for honest nodes who even temporarily drop offline. Once offline, an honest node's input may be reconstructed and exposed by honest nodes who still remain online.
2. Many standard MPC protocols [4, 11] require many pairs of nodes to have finished several rounds of pairwise interactions to make progress. Even if such protocols required only constant number of rounds in the classical synchronous model, they may suffer from bad round complexity in our model — recall that in a weakly synchronous network, nodes do not have persistent online presence; thus it can take (super-)linear number of rounds for sufficiently many pairs of nodes to have had an opportunity to rendezvous.

To tackle these challenges we rely on a Threshold Multi-Key Fully Homomorphic Encryption (TMFHE) scheme [3, 12]. In a TMFHE scheme [3],

1. Each node i can independently generate a public key denoted pk_i and register it with a PKI.
2. Now, each node i can encrypt its input x_i resulting in a ciphertext CT_i.
3. After collecting a set of ciphertexts $\{\mathsf{CT}_i\}_{i\in S}$ corresponding to the nodes $S \subseteq [n]$, any node can independently perform homomorphic evaluation (for the function f) on the ciphertext-set $\{\mathsf{CT}_i\}_{i\in S}$ and obtain an encryption (denoted $\widetilde{\mathsf{CT}}$) of $f(\{x_i\}_{i\in S})$.
4. Now, each node i can evaluate a partial decryption share of $\widetilde{\mathsf{CT}}$ such that if sufficiently many partial decryption shares are combined, one can reconstruct the plaintext evaluation outcome $f(\{x_i\}_{i\in S})$.

In our protocol, in round 0, every node i will compute an TMFHE ciphertext (denoted CT_i) that encrypts its own input and compute a NIZK proof (denoted π_i) attesting to well-formedness of the ciphertext. The pair (CT_i, π_i) will be broadcast by invoking an instance of our BA protocol described in Section 8. Let T_{ba} be the liveness parameter of BA. Now, every honest node in $\mathcal{O}_{T_{\mathrm{ba}}}$ will have

obtained outputs from all BA instances at the beginning of round T_{ba}. From the outputs of these BA instances, nodes in $\mathcal{O}_{T_{\text{ba}}}$ can determine the effective-input set $\mathfrak{I}$ — specifically if any BA instance that has produced a well-formed output with a valid NIZK proof, the corresponding sender will be included in the effective-input set. Observe that everyone in $\mathcal{O}_0$ will be included in $\mathfrak{I}$. Now, in round T_{ba}, any node who has produced outputs from all n BA instances will perform homomorphic evaluation independently over the collection of ciphertexts $\{\mathsf{CT}_i\}_{i\in\mathfrak{I}}$. They will then compute and multicast a partial decryption share and a NIZK proof vouching for the correctness of the partial decryption share. Now, everyone in $\mathcal{O}_t$ for $t \geq T_{\text{ba}}$ will have received sufficiently many decryption shares in round t to reconstruct the evaluation outcome.

Comparison with "lazy MPC". Interestingly, the recent work by Badrinarayanan et al. [3] propose a related notion called "lazy MPC"; and their goal is also to safeguard the inputs of those who are benign but drop out in the middle of the protocol. Their model, however, is overly restrictive:

1. first, Badrinarayanan et al. [3] require that a set of majority number of honest nodes to be online *forever*;
2. not only so, they also make the strong assumption that nodes who drop offline never come back (and thus we need not guarantee liveness for nodes who ever drop offline).

As mentioned, in long-running distributed computation environments (e.g., decentralized blockchains where a secure computation task may be repeated many times over the course of years), most likely no single node can guarantee 100% up-time (let alone majority). From a technical perspective, the existence of a majority "honest and persistent online" set also makes the problem significantly easier. For example, for BA, there is in fact a simple compiler that compiles any existing honest-majority, strongly synchronous BA to a setting in which the existence of majority "honest and persistent online" set is guaranteed: basically, simply run an honest-majority, strongly synchronous BA protocol denoted BA_0. If BA_0 outputs a value v, multicast a signed tuple ($\texttt{finalize}$, v). Output v iff $\lfloor n/2 \rfloor + 1$ number of ($\texttt{finalize}$, v) messages have been received with valid signatures from distinct nodes. In fact, this simple protocol also ensures liveness for drop-outs who come back online.

Under our definition of weak synchrony, realizing BA is highly non-trivial (see earlier subsections of this section). Once we realize BA, our approach for realizing MPC is reminiscent of Badrinarayanan et al. [3]. There is, in fact, a notable difference in a low-level subtlety: in Badrinarayanan et al. [3]'s lazy MPC model, they can afford to have sufficiently many pairs of nodes engage in several rounds of pairwise interaction, whereas in our model, it can take (super-)linear number of rounds for sufficiently many pairs of nodes to have had an opportunity to rendezvous. For this reason, we need to use a strengthened notion of Threshold Multi-Key Fully Homomorphic Encryption (TMFHE) in comparison with Badrinarayanan et al. [3]. More detailed discussion of these technicalities are inclued in the online full version [14].

3 Defining a Weakly Synchronous Execution Model

A protocol execution is formally modeled as a set of Interactive Turing Machines (ITMs). The execution proceeds in *rounds*, and is directed by a non-uniform probabilistic polynomial-time (p.p.t.) environment denoted $\mathcal{Z}(1^\kappa)$ parametrized by a security parameter $\kappa \in \mathbb{N}$. Henceforth we refer to ITMs participating in the protocol as *nodes* and we number the nodes from 1 to $n(\kappa)$ where n is chosen by $\mathcal{Z}$ and may be a polynomial function in κ.

3.1 Modeling Corruption and Network Communication

We assume that there is a non-uniform p.p.t. adversary $\mathcal{A}(1^\kappa)$ that may communicate with $\mathcal{Z}$ freely at any time during the execution. $\mathcal{A}$ controls a subset of nodes that are said to be *corrupt*. All corrupt nodes are fully within the control of $\mathcal{A}$: $\mathcal{A}$ observes a node's internal state the moment it becomes corrupt and henceforth all messages received by the corrupt node are forwarded to $\mathcal{A}$; further, $\mathcal{A}$ decides what messages corrupt nodes send in each round. In this paper, we assume that corruption is *static*, i.e., the adversary $\mathcal{A}$ decides which nodes to corrupt prior to the start of the protocol execution.

Nodes that are not corrupt are said to be *honest*, and honest nodes faithfully follow the prescribed protocol for as long as they remain honest. In each round, an honest node can either be *online* or *offline*.

Definition 1 (Honest and online nodes). *Throughout the paper, we shall use the notation $\mathcal{O}_r$ to denote the set of honest nodes that are online in round r. The set $\mathcal{O}_r$ is also called the "honest and online set" of round r. For $i \in \mathcal{O}_r$, we often say that i is honest and online in round r.*

We make the following assumption about network communication — note that our protocol is in the *multicast* model, i.e., every protocol message is sent to the set of all nodes:

Assumption 1 (Message delivery assumption) *We assume that if someone in $\mathcal{O}_r$ multicasts a message m in round r, then everyone in $\mathcal{O}_t$ where $t \geq r + \Delta$ will have received m at the beginning of round t.*

In other words, an honest and online node is guaranteed to be able to deliver messages to the honest and online set of nodes Δ or more rounds later. The adversary $\mathcal{A}$ may delay or erase honest messages arbitrarily as long as Assumption 1 is respected.

Remark 2 (Offline nodes' network communication). Note that the above message delivery assumption implies that messages sent by honest but offline nodes can be arbitrarily delayed or even completely erased by the adversary. Further, the adversary can control which subset of honest messages each offline node receives in every round; it can omit an arbitrary subset of messages or even all of them from the view of honest offline nodes for as long as they remain offline.

Remark 3. We stress that *a node is not aware whether it is online or offline.* This makes protocol design in this model more challenging since the adversary can carefully choose a subset of messages for an offline (honest) node to receive, such that the offline node's view can appear perfectly "normal" such that it is unable to infer that it is offline. Jumping ahead, a consensus protocol secure in our model should guarantee that should an offline node make a decision while it is offline, such decisions would nonetheless be safe and would not risk inconsistency with the rest of the network.

Our protocol needs to be aware of the parameters Δ and n. Throughout we shall assume that Δ and n are polynomial functions in κ. Formally, we can imagine that $\mathcal{Z}$ inputs Δ and n to all honest nodes at the start of the execution. Throughout the paper, we assume that $(\mathcal{A}, \mathcal{Z})$ respects the following constraints:

> $\mathcal{Z}$ always provides the parameters n and Δ to honest nodes at the start of the execution such that n is the total number of nodes spawned in the execution, and moreover, the adversary $\mathcal{A}$ respects Assumption 1.

Schedule within a round. More precisely, in each round r, the following happens:

1. First, each honest node receives inputs from $\mathcal{Z}$ and receives incoming messages from the network; note that at this moment, $\mathcal{A}$'s decision on which set of incoming messages an honest node receives will have bearings on whether this honest node can be included in $\mathcal{O}_r$;
2. Each honest node then performs polynomially bounded computation and decides what messages to send to other nodes — these messages are immediately revealed to $\mathcal{A}$. Further, after the computation each honest node may optionally send outputs to $\mathcal{Z}$.
3. At this moment, $\mathcal{A}$ decides which nodes will belong to $\mathcal{O}_r$ where r denotes the current round. Note that $\mathcal{A}$ can decide the honest and online set $\mathcal{O}_r$ of the present round after seeing what messages honest nodes intend to send in this round.
4. $\mathcal{A}$ now decides what messages each corrupt node will send to each honest node. Note also that $\mathcal{A}$ is *rushing* since it can see all the honest messages before deciding the corrupt nodes' messages.
5. Honest nodes send messages over the network to other nodes (which may be delayed or erased by $\mathcal{A}$ as long as Assumption 1 is satisfied).

Definition 2 (χ-weak-synchrony). *We say that $(\mathcal{A}, \mathcal{Z})$ respects χ-weak-synchrony (or that $\mathcal{A}$ respects χ-weak-synchrony), iff in every round r, $|\mathcal{O}_r| \geq \lfloor \chi \cdot n \rfloor + 1$.*

To aid understanding, we make a couple of remarks regarding this definition. First, observe that the set of honest and online nodes need not be the same in every round. This allows us to model churns in the network: nodes go offline and come online; and we wish to achieve consistency for *all* honest nodes, regardless of whether they are online or offline, as long as sufficiently many nodes are online in each round. Second, the requirement of χ-weak-synchrony also imposes a corruption budget. As an example, consider the special case when $\chi = 0.5$ and

n is an even integer: if $(\mathcal{A}, \mathcal{Z})$ respects 0.5-weak-synchrony, it means that the adversary controls at most $n/2-1$ nodes. It could be that the adversary in fact controls fewer, say, $n/3$ number of nodes. In this case, up to $n/2-1-n/3$ honest nodes may be offline in each round, and jumping ahead, in a consensus protocol we will require that consistency hold for these honest but offline nodes as well.

Finally, note also that our weakly-synchronous model is a generalization of the classical synchronous model: in the classical synchronous model, it is additionally required that for every r, $\mathcal{O}_r$ must be equal to the set of all nodes that remain honest till the end of round r (or later).

3.2 Modeling Setup Assumptions

In the plain model without any setup assumptions, Lamport et al. [17] showed that no consensus protocol could tolerate 1/3 or more corruptions; however for $< 1/3$ corruptions, one can construct protocols that tolerate arbitrary network partitions by adopting the partially synchronous model [7, 9, 16]. It is also known that assuming a public-key infrastructure (PKI) and computationally bounded adversaries, one can construct consensus protocols that tolerate arbitrarily many corruptions in the classical fully synchronous model. Thus the interesting open question is whether, assuming the existence of a PKI and computationally bounded adversaries, we can construct protocols that tolerate more than 1/3 corruptions and yet provide some quantifiable degree of partition tolerance. Therefore, throughout this paper we shall assume the existence of a PKI and computationally bounded adversaries. We assume that the adversary chooses which nodes to corrupt before the PKI is established.

3.3 Weakly Synchronous Byzantine Agreement

We now define Byzantine Agreement (BA) in a weakly synchronous network. The consistency definition is standard except that now we require consistency for honest nodes regardless of whether they are online or offline. For validity, if the sender is honest but offline initially, we cannot hope that the protocol will somehow make up for the time lost waiting for the sender to come online, such that honest and online nodes would output by the same deadline. Thus we require validity to hold only if the sender is not only honest but also online in the starting round. For liveness, we cannot hope that honest but offline nodes obtain outputs quickly without the risk of being inconsistent with the rest of the network. Thus, we require that as soon as an honest node is online at time T or greater (where T is also called the liveness parameter), it must produce an output if it has not done so already.

Syntax. A Byzantine Agreement (BA) protocol must satisfy the following syntax. Without loss of generality, we assume that node 1 is the designated sender. Before protocol start, the sender receives an input bit b from $\mathcal{Z}$; and all other nodes receive no input. The nodes then run a protocol, and during the protocol every node may output a bit.

Security. Let $T(\kappa, n, \Delta)$ be a polynomial function in the stated parameters. For $P \in \{\text{consistency, validity, } T\text{-liveness}\}$, a BA protocol is said to satisfy property P w.r.t. some non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that is allowed to spawn multiple possibly concurrent BA instances sharing the same PKI, iff there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability over the choice of protocol execution, the corresponding property as explained below is respected in all BA instances spawned — henceforth we rename the starting round of each BA instance to be round 0 and count rounds within the same instance accordingly afterwards:

- *Consistency.* If honest node i outputs b_i and honest node j outputs b_j, it must be that $b_i = b_j$.
- *Validity.* If the sender is in $\mathcal{O}_0$, any honest node's output must be equal to the sender's input.
- *T-liveness.* Any node in $\mathcal{O}_r$ for $r \geq T$ must have output a bit by the end of round r.

We say that a BA protocol satisfies property $P \in \{\text{consistency, validity, and } T\text{-liveness}\}$ under χ-weak-synchrony if it satisfies the property P w.r.t. any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects χ-weak-synchrony and is allowed to spawn multiple possibly concurrent BA instances sharing the same PKI. Henceforth, if a BA protocol satisfies consistency, validity, and T-liveness under χ-weak-synchrony, we also say that the protocol is a "χ-weakly-synchronous BA protocol".

Remark 4 (Worst-case vs expected notions of liveness). We note that T-liveness defines a worst-case notion of liveness. In the remainder of the paper, we sometimes use an expected round complexity notion. We say that our BA protocol is expected constant round, iff there is a random variable R whose expectation is constant such that everyone in $\mathcal{O}_r$ where $r \geq R$ should have produced an output by the end of round r.

Multi-valued agreement. The above definition can be extended to multi-valued agreement where nodes agree on a value from the domain $\{0,1\}^{\ell(\kappa)}$ rather than a single bit. Multi-valued agreement can be obtained by parallel composition of ℓ instances of BA. In this paper, we will refer to the multi-valued version as Byzantine Agreement (BA) too.

4 Lower Bounds

4.1 Impossibility of Weakly-Synchronous Consensus for $\chi \leq 0.5$

First, we show that for any $\chi \leq 0.5 - \frac{1}{n}$, it is impossible to achieve BA under χ-weak-synchrony. The intuition for this lower bound is simple: if a BA protocol allows a minority set of online nodes to reach agreement without hearing from the offline nodes, then two minority camps could independently reach agreement thus risking consistency. We formalize this intuition in the following theorem.

Theorem 3. *For any $\chi \leq 0.5 - \frac{1}{n}$, for any polynomial function T, no BA protocol Π can simultaneously achieve consistency, validity, and T-liveness under χ-weak-synchrony.*

Proof. Please refer to the online full version [14].

We point out that the above the lower bound holds even if $\mathcal{A}$ is restricted to scheduling the same honest and online set throughout, i.e., $\mathcal{O}_0 = \mathcal{O}_1 = \ldots$, has to decide the message delivery schedule in advance, and even when no node is corrupt. Moreover, the lower bound holds even for randomized protocols, allowing computational assumptions, and allowing additional setup assumptions (e.g., PKI, random oracle, or the erasure model).

Best-possible partition tolerance. In light of Theorem 3, a BA protocol secure under 0.5-weak-synchrony is also said to be best-possible partition tolerant.

4.2 Corrupt-Majority Protocols Sacrifice Partition Tolerance

It is well-known that there exist Byzantine Agreement protocols that tolerate arbitrarily many byzantine faults [8] under the classical synchronous model henceforth referred to as *strong* synchrony. If we adopted the classical strong synchrony model we might be misled to think that protocols that tolerate corrupt majority are strictly more robust than those that tolerate only corrupt minority. In this section, however, we show that corrupt-majority protocols (under strong synchrony) in fact sacrifice partition tolerance in exchange for tolerating corrupt majority, and this is inherent. As explained earlier, in real-world scenarios such as decentralized cryptocurrencies, partition tolerance seems to be a more important robustness property.

It is not too difficult to see that any corrupt-majority, strongly-synchronous protocol cannot be secure under 0.5-weak-synchrony. Specifically, with a corrupt-majority strongly-synchronous protocol, if the network partitions into multiple minority connected components, each component will end up reaching its own independent decision. We can generalize this intuition and prove an even stronger statement: any strongly-synchronous protocol that tolerates more than $\nu \geq 0.5$ fraction of corruptions cannot be secure under ν-weak-synchrony, i.e., such a protocol cannot guarantee consistency for all honest nodes (including offline ones) even if we make the strong assumption that at least ν fraction of honest nodes are online. In other words, *the more corruptions the protocol tolerates under strong synchrony, the less partition tolerant it becomes.* To state our theorem precisely, we introduce the following notation:

- We say that $(\mathcal{A}, \mathcal{Z})$ respects μ-strongly-synchronous iff at least $\lfloor \mu n \rfloor + 1$ nodes are honest and moreover all honest nodes are forever online. We say that a BA protocol satisfies property $P \in \{\text{consistency, validity, and } T\text{-liveness}\}$ under μ-strong-synchrony iff it satisfies property P w.r.t. any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects μ-strong-synchrony.
- Let $\mathcal{BA}\{\mu\}$ be the family that contains every protocol Π satisfying the following: $\exists$ a polynomial function $T(\cdot, \cdot, \cdot)$ s.t. Π that satisfy consistency, validity, and $T(\kappa, n, \Delta)$-liveness under μ-strong-synchrony.

– Let $\mathcal{BA}^+\{\chi\}$ be the family that contains every protocol Π satisfying the following: $\exists$ a polynomial function $T(\cdot,\cdot,\cdot)$ s.t. Π that satisfy consistency, validity, and T-liveness under χ-weak-synchrony.

Theorem 4. $\forall 0 < \mu < 0.5, \chi \leq 1 - \mu - 2/n,\ \ \mathcal{BA}\{\mu\} \cap \mathcal{BA}^+\{\chi\} = \emptyset$.

Proof. Please refer to the online full version [14].

5 Reliable Broadcast (RBC)

In our upper bound sections (Sections 5, 6.2, 7, 8, and the MPC upper bound in the online full version [14]) for convenience, we will make a slightly stronger assumption on the underlying network — but in fact this stronger assumption can be realized from Assumption 1 described earlier.

Assumption 2 (Strong message delivery assumption) *If $i \in \mathcal{O}_r$ and i has multicast or received a message* m *before the end of round r, then everyone in $\mathcal{O}_t$ where $t \geq r + \Delta$ will have received* m *at the beginning of round t.*

In the online full version [14], we describe how to realize Assumption 2 through a simple echo mechanism: roughly speaking, nodes echo and retry sending messages they have seen until they believe that the message has become part of the honest and online nodes' view.

5.1 Definition

We define a primitive called reliable broadcast (RBC) that allows a designated sender to broadcast a message, guaranteeing consistency regardless of whether the sender is honest or online, and additionally guaranteeing liveness when the sender is not only honest but also online in the starting round. We also require a "close termination" property: even when the designated sender is corrupt, we require that if some honest node outputs in round r, then everyone in $\mathcal{O}_t$ where $t \geq r + 2\Delta$ must have output by the end of round t too. The liveness notion is defined in a similar fashion as in Section 3.3: since under weak synchrony we cannot guarantee progress for offline nodes, we require that any honest node who comes back online in some time T or greater will have received output (assuming an honest and initially online sender). For technical reasons that will be useful later in the proof of our Leader Election (LE) protocol, we need a stronger version of the standard consistency property: not only must honest nodes' outputs agree, there must be an efficient extractor that outputs either a bit $b \in \{0,1\}$ or $\perp$ when given the PKI and the honest nodes' transcript in the initial T rounds as input. If any honest node indeed makes an output, the output must be consistent with the extractor's output b.

Syntax. An RBC protocol consists of the following algorithms/protocols:

– **PKI setup:** at the very beginning every node i registers a public key pk_i with the PKI;

– RBC **protocol:** all instances of RBC share the same PKI. In each RBC instance, a designated sender (whose identifier is pre-determined and publicly-known) receives a value x from the environment $\mathcal{Z}$ whereas all other nodes receive nothing. Whenever a node terminates, it outputs a value y. Henceforth we shall assume that an admissible $\mathcal{Z}$ must instruct all nodes to start protocol execution in the same round[2];
– **Extractor** $\mathcal{E}$: a polynomial-time deterministic extractor $\mathcal{E}$ that is needed only in our security definitions and proofs, not in the real-world protocol.

Security. Let $T(n, \Delta, \kappa)$ be a polynomial function in the stated parameters. For $P \in \{T\text{-consistency, validity, } T\text{-liveness, close termination}\}$, we say that an RBC protocol Π satisfies property P under χ-weak-synchrony iff for any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects χ-weak-synchrony and can spawn multiple instances of RBC sharing the same PKI, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$, except for $\mathsf{negl}(\kappa)$ fraction of the executions in the experiment $\mathsf{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \kappa)$, the following properties hold for every RBC instance:

– *T-consistency.* Let $y := \mathcal{E}(\{\mathsf{pk}_i\}_{i\in[n]}, \mathsf{Tr})$ where Tr denotes the transcript of all honest nodes in the initial T rounds of the RBC instance. Then, if any honest node ever outputs y', it must be that $y' = y$.
– *Validity.* If the sender is honest and its input is x, then if any honest node outputs x', it must be that $x' = x$.
– *T-liveness (under an honest and initially online sender).* If the sender is not only honest and but also online in the starting round of this RBC instance (henceforth the starting round is renamed to be round 0 for convenience), then every node that is honest and online in round $r \geq T$ will have produced an output by the end of round r.
– *Close termination.* If an honest node outputs in some round r, then every node that is honest and online in round $r' \geq r + 2\Delta$ will have output by the end of round r'.

Remark 5. Although in general, consistency and liveness can be parametrized by different delay functions, without loss of generality we may assume that two parameters are the same T (since we can always take the maximum of the two).

5.2 Construction

During the PKI setup phase (shared across all subsequent RBC instances), every node calls $(\mathsf{vk}, \mathsf{ssk}) \leftarrow \Sigma.\mathsf{K}(1^\kappa)$ and registers the vk with the PKI. The portion ssk is kept secret and henceforth the node will use ssk to sign protocol messages in all future RBC instances. Henceforth, although not explicitly noted, we assume that every message is by default tagged with the current session's identifier denoted *sid*. Every signature computation and verification will include the *sid*. We also assume that each message is tagged with the purported sender such that a recipient knows under which public key to verify the signature.

[2] Later in our VSS and LE protocols that invoke RBC, the fact that the RBC's environment $\mathcal{Z}$ is admissible is guaranteed by construction.

1. **Propose (round 0):** In round 0, the sender multicasts $(\texttt{propose}, x)$ where x is its input, attached with a signature on the tuple.
2. **ACK (round Δ):** At the beginning of round Δ, if a tuple $(\texttt{propose}, y)$ with a valid signature has been received from the sender, multicast $(\texttt{ack}, y)$ along with a signature on the tuple.
3. **Commit (round 2Δ):** At the beginning of round 2Δ, if the node has observed $\lfloor n/2 \rfloor + 1$ number of $(\texttt{ack}, y)$ messages for the same y and with valid signatures from distinct nodes, and moreover, it has not received any conflicting $(\texttt{propose}, y')$ message (with a valid signature from the sender) for $y' \neq y$, then multicast $(\texttt{commit}, y)$ along with a signature on the tuple.
4. **Finalize (any time):** At any time, if the node has received $\lfloor n/2 \rfloor + 1$ valid $(\texttt{commit}, y)$ messages for the same y and from distinct nodes, multicast $(\texttt{finalize}, y)$ along with a signature on the tuple. At any time, if a collection of $\lfloor n/2 \rfloor + 1$ $(\texttt{finalize}, y)$ messages with valid signatures from distinct nodes have been observed, output y.

We defer the constructor of the extractor $\mathcal{E}$ to the proofs since it is needed only in the security definitions and proofs and not in the real-world protocol.

Theorem 5. *Suppose that the signature scheme employed is secure, then the above* RBC *protocol satisfies* 2Δ*-consistency, validity,* 4Δ*-liveness, and close termination under* 0.5*-weak-synchrony.*

Proof. Please refer to the online full version [14].

6 Verifiable Secret Sharing (VSS)

6.1 Definitions

A Verifiable Secret Sharing (VSS) allows a dealer to share a secret among all nodes and later reconstruct the secret. Standard notions of VSS [6] require that the honest transcript of the sharing phase binds to the honestly reconstructed secret. For technical reasons needed later in the proof of the Leader Election (LE), we require a stronger notion, i.e., an efficient extractor $\mathcal{E}$, knowing honest nodes' public and secret keys, must be able to extract this secret from the honest transcript during the sharing phase (and later the honestly reconstructed secret must agree with the extractor's output). We need a composable notion of secrecy which we call non-malleability — note that composition was a non-issue in previous works that achieve security against unbounded adversaries [6]. Finally, for liveness, we require that if the dealer is honest and online in the initial round, for $t \geq T$, everyone in $\mathcal{O}_t$ must have output "`sharing succeeded`". Even when the dealer is corrupt or offline, if any honest node ever outputs "`sharing succeeded`" in some round r, then everyone in $\mathcal{O}_t$ where $t \geq r + 2\Delta$ must have output "`sharing succeeded`" by the end of round t. If some honest node has output "`sharing succeeded`", then reconstruction must be successful and will terminate in T rounds for honest and online nodes.

Syntax A Verifiable Secret Sharing (VSS) scheme for a finite field $\mathbb{F}$ consists of a setup algorithm K that is run once upfront and henceforth shared among all protocol instances where each protocol instance contains two sub-protocols called Share and $\mathsf{Reconstruct}$:

1. $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \mathsf{K}(1^\kappa)$: every node i calls this algorithm to generate a public and secret key pair denoted pk_i and sk_i; and pk_i is registered with the PKI.
2. Share: A designated node called the dealer receives an input $s \in \mathbb{F}$ from $\mathcal{Z}$ and all other nodes receive no input. Now all nodes execute the Share sub-protocol for the dealer to secret-share its input. We assume that for the same VSS instance, an admissible $\mathcal{Z}$ always instructs all honest nodes to start executing Share in the same round. Should execution of Share successfully terminate, a node would output a canonical output "`sharing succeeded`".
3. $\mathsf{Reconstruct}$: All nodes execute the $\mathsf{Reconstruct}$ sub-protocol to reconstruct a secret that is shared earlier in the Share sub-protocol. We assume that an admissible $\mathcal{Z}$ always instructs all honest nodes to start executing $\mathsf{Reconstruct}$ in the same round. Should execution of $\mathsf{Reconstruct}$ successfully terminate, a node would output a reconstructed secret $s' \in \mathbb{F}$.

Besides these real-world algorithms, a VSS scheme additionally has a polynomial-time extractor algorithm $\mathcal{E}$ that is needed later in the security definitions (including the definitions of validity and non-malleability). We shall explain the extractor $\mathcal{E}$ later when we define security.

***T*-Liveness** Consider a pair $(\mathcal{A}, \mathcal{Z})$ that may spawn multiple (concurrent or sequential) VSS instances all of which share the same n, PKI setup, and the same Δ. Let $T(n, \Delta, \kappa)$ be a polynomial function in n, Δ, κ. We say that a VSS protocol satisfies T-liveness under χ-weak-synchrony iff for any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects χ-weak-synchrony (and may spawn multiple instances sharing the same PKI), there exists $\mathsf{negl}(\cdot)$ such that for any $\kappa \in \mathbb{N}$, such that except with $\mathsf{negl}(\kappa)$ probability, the following holds for every VSS instance spawned:

1. *Termination of* Share *under honest and initially online dealer:* suppose that the Share sub-protocol is spawned in round r_0, and moreover the dealer is in $\mathcal{O}_{r_0}$, then any node in $\mathcal{O}_r$ for $r \geq r_0 + T$ must have output "`sharing succeeded`" by the end of round r;
2. *Close termination of* Share*:* if an honest node i has terminated the Share sub-protocol outputting "`sharing succeeded`" in round r, then for every $r' \geq r + 2\Delta$, every node in $\mathcal{O}_{r'}$ must have terminated the Share sub-protocol outputting "`sharing succeeded`" by the end of round r';
3. *Termination of* $\mathsf{Reconstruct}$*:* if by the end of some round r, some honest node has terminated the Share sub-protocol outputting "`sharing succeeded`", and moreover honest nodes have been instructed to start $\mathsf{Reconstruct}$, then, anyone in $\mathcal{O}_t$ for $t \geq r+T$ must have terminated the $\mathsf{Reconstruct}$ sub-protocol outputting some reconstructed value in $\mathbb{F}$ by the end of round t.

***T*-Validity** As before, we consider an $(\mathcal{A}, \mathcal{Z})$ pair that is allowed to spawn multiple (concurrent or sequential) VSS instances, all of which share the same n, PKI setup, and Δ. Let $T(n, \Delta, \kappa)$ be a polynomial function in its parameters. Henceforth let $\mathsf{Honest} \subseteq [n]$ denote the set of honest nodes. We say that a VSS protocol satisfies T-validity under χ-weak-synchrony, iff for every non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects χ-weak-synchrony (and may spawn multiple VSS instances sharing the same PKI where each instance has a unique sid), there exists a negligible function $\mathsf{negl}(\cdot)$ such that except with $\mathsf{negl}(\kappa)$ probability, the following holds for every VSS instance spawned: let $s' := \mathcal{E}(\{\mathsf{pk}_i\}_{i\in[n]}, \{\mathsf{sk}_i\}_{i\in\mathsf{Honest}}, \mathsf{Tr})$ where Tr denotes the transcript observed by all honest nodes in the initial T rounds of the Share sub-protocol; it must be that

(a) if an honest node ever outputs a reconstructed secret, the value must agree with s';
(b) if $\mathcal{E}$ outputs $\perp$, then no honest node ever outputs "`sharing succeeded`"[3];
(c) if the dealer is honest and online in the round in which the Share sub-protocol was invoked, and moreover it received the input s from $\mathcal{Z}$, then $s' = s$.

Non-Malleability Consider the following experiment $\mathsf{Expt}^{\mathcal{A}}(1^\kappa, s)$ involving an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$, as well as a challenge input $s \in \mathbb{F}$. We assume that throughout the experiment, *if an honest node outputs a string in any* VSS *instance, the adversary* $\mathcal{A}$ *is notified of the node's identifier, the identifier of the* VSS *instance, as well as the corresponding output.*

1. **Setup.** First, $\mathcal{A}$ chooses which set of nodes to corrupt. Henceforth the challenger $\mathcal{C}$ acts on behalf of all honest nodes and interact with $\mathcal{A}$. The honest nodes run the honest key generation algorithm such that each picks a public/secret-key pair. The public keys are given to $\mathcal{A}$. $\mathcal{A}$ now chooses corrupt nodes' public keys arbitrarily and sends them to $\mathcal{C}$.
2. **Queries.** The adversary $\mathcal{A}$ is now allowed to (adaptively) instruct $\mathcal{C}$ to spawn as many VSS instances as it wishes. The queries can be issued at any time, including before, during, or after the challenge phase (see the **Challenge** paragraph later).
   - Whenever $\mathcal{A}$ sends $\mathcal{C}$ a tuple $(sid, \mathsf{Share}, u, x)$ where $sid \in \{0,1\}^*$ and $u \in [n]$, $\mathcal{C}$ spawns instance sid with node u as the dealer. If u is honest, $\mathcal{A}$ must additionally specify the honest dealer u's input x in this instance (otherwise the field x is ignored). Now, $\mathcal{C}$ invokes the instance's Share sub-protocol (if this has not been done already);
   - Whenever $\mathcal{A}$ sends $\mathcal{C}$ a tuple $(sid, \mathsf{Reconstruct})$ where $sid \in \{0,1\}^*$, $\mathcal{C}$ does the following: if the instance sid has been spawned, then invoke the $\mathsf{Reconstruct}$ sub-protocol for that instance (if this has not been done).
   - Whenever $\mathcal{A}$ sends $\mathcal{C}$ a tuple $(sid, \mathsf{Extract})$ and instance sid has executed for at least T rounds, then $\mathcal{C}$ computes $\mathcal{E}(\{\mathsf{pk}_i\}_{i\in[n]}, \{\mathsf{sk}_i\}_{i\in\mathsf{Honest}}, \mathsf{Tr})$ where Tr is the transcript of honest nodes in the initial T rounds of the Share sub-protocol; $\mathcal{C}$ returns the result to $\mathcal{A}$.

[3] Note that (a) implies that if $\mathcal{E}$ outputs $\perp$, then no honest node will ever output a reconstructed secret.

3. **Challenge.** At any time, $\mathcal{A}$ may send the tuple $(\texttt{challenge}, sid, u)$ to $\mathcal{C}$ where u must be an honest node and the challenge sid must not be specified in any Extract or Reconstruct query throughout the experiment (in the past or future). $\mathcal{C}$ then spawns a challenge VSS instance identified by sid where u is the designated dealer and receives the input s; further $\mathcal{C}$ invokes the challenge instance's Share sub-protocol.
4. **Output.** Whenever the adversary $\mathcal{A}$ outputs a bit $b \in \{0, 1\}$, this bit is defined as the experiment's output.

We assume that an admissible $\mathcal{A}$ never attempts to create two VSS instances with the same sid, i.e., $\mathcal{A}$ chooses distinct session identifiers for all instances. Further, throughout the experiment, $\mathcal{A}$ is allowed to decide which honest nodes are online/offline in each round (after seeing the messages honest nodes want to send in that round). $\mathcal{A}$ also controls the message delivery schedule[4].

Definition 3 (Non-malleability for VSS). *We say that a* VSS *scheme satisfies non-malleability under χ-weak-synchrony iff for any non-uniform p.p.t. $\mathcal{A}$ that respects χ-weak-synchrony, there exists a negligible function* $\mathsf{negl}(\cdot)$ *such that for any* $s, s' \in \mathbb{F}$, $\left|\Pr[\mathsf{Expt}^{\mathcal{A}}(1^\kappa, s) = 1] - \Pr[\mathsf{Expt}^{\mathcal{A}}(1^\kappa, s') = 1]\right| \leq \mathsf{negl}(\kappa)$.

6.2 A 0.5-Weakly-Synchronous VSS Scheme

We show how to construct a 0.5-weakly synchronous VSS scheme. We will rely on the following cryptographic primitives:

1. let $\mathsf{NIZK} := (\mathsf{K}, \widetilde{\mathsf{K}}, \mathsf{P}, \mathsf{V})$ denote multi-CRS NIZK scheme that satisfies completeness, zero-knowledge, and simulation soundness (see the online full version [14]);
2. let $\mathsf{PKE} := (\mathsf{K}, \mathsf{Enc}, \mathsf{Dec})$ denote a perfectly correct public-key encryption scheme that preserves IND-CCA security; and
3. let RBC denote a reliable broadcast scheme that satisfies T_{rbc}-consistency, T_{rbc}-liveness, validity, and close termination under 0.5-weak-synchrony for some polynomial function T_{rbc}.

PKI setup (shared across all VSS instances): During the PKI setup phase, every node i performs the following:
- let $(\mathsf{epk}_i, \mathsf{esk}_i) \leftarrow \mathsf{PKE.K}(1^\kappa)$; $(\mathsf{vk}_i, \mathsf{ssk}_i) := \Sigma.\mathsf{K}(1^\kappa)$; $\mathsf{crs}_i \leftarrow \mathsf{NIZK.K}(1^\kappa)$; and let $(\mathsf{rpk}_i, \mathsf{rsk}_i) \leftarrow \mathsf{RBC.K}(1^\kappa)$;
- node i registers its public key $\mathsf{pk}_i := (\mathsf{epk}_i, \mathsf{crs}_i, \mathsf{vk}_i, \mathsf{rpk}_i)$ with the PKI; and it retains its secret key comprised of $\mathsf{sk}_i := (\mathsf{esk}_i, \mathsf{ssk}_i, \mathsf{rsk}_i)$.

Share (executed by the dealer): Let s be the input received from the environment, the dealer does the following:
- it splits s into n shares using a $(\lfloor n/2 \rfloor + 1)$-out-of-$n$ Shamir Secret Sharing scheme, where the i-th share is henceforth denoted s_i;

[4] Specifically, when honest nodes running inside $\mathcal{C}$ want to send messages, the messages are forwarded to $\mathcal{A}$, and $\mathcal{A}$ tells $\mathcal{C}$ when each honest node receives what message.

- for $i \in [n]$, it computes $\mathsf{CT}_i := \mathsf{PKE.Enc}_{\mathsf{epk}_i}(sid, s_i)$ where sid is the identifier of the current instance;
- it calls $\mathsf{NIZK.P}(\{\mathsf{crs}_i\}_{i\in[n]}, x, w)$ to compute a proof π where x and w are defined as below: $x := (sid, \{\mathsf{pk}_i, \mathsf{CT}_i\}_{i\in[n]})$ is the statement declaring that there is a witness $w := (s, \{s_i\}_{i\in[n]})$ such that for each $i \in [n]$, CT_i is a valid encryption[5] of (sid, s_i) under epk_i (which is part of pk_i); and moreover, the set of shares $\{s_i\}_{i\in[n]}$ is a valid sharing of the secret s.
- finally, the dealer relies on RBC to reliably broadcast the tuple $(sid, \{\mathsf{CT}_i\}_{i\in[n]}, \pi)$ — henceforth this RBC instance is denoted RBC_0.

Share (executed by everyone): Every node i does the following (where the starting round of Share is renamed round 0):

- **Any time:** whenever the RBC_0 instance outputs a tuple of the form $(sid, \{\mathsf{CT}_j\}_{j\in[n]}, \pi)$, call NIZK.V to verify the proof π w.r.t. the statement $(sid, \{\mathsf{pk}_i, \mathsf{CT}_i\}_{i\in[n]})$; and if the check succeeds, set $\mathsf{flag} := 1$ (we assume that flag was initially 0).
- **Round T_{rbc}:** if $\mathsf{flag} = 1$, reliably broadcast the message "ok"; else reliably broadcast the message "$\perp$";
- **Any time:** whenever more than $\lfloor n/2 \rfloor + 1$ RBC instances have output "ok" and RBC_0 has output a tuple; decrypt CT_i contained in the tuple output by RBC_0 using secret key esk_i; let $(\_, s_i)$ be the decrypted outcome; now record the share s_i and output "sharing-succeeded";

Reconstruct (executed by everyone): when the Reconstruct sub-protocol has been invoked, every node i waits till the instance's Share sub-protocol has output "sharing-succeeded" and then performs the following where the set $\mathbb{S}$ is initially empty:

- let s_i be the share recorded at the end of the Share sub-protocol;
- call $\mathsf{NIZK.P}(\{\mathsf{crs}_i\}_{i\in[n]}, x, w)$ to compute a proof (henceforth denoted π_i) for the following statement $x := (sid, i, s_i, \mathsf{CT}_i)$ declaring that there is random string that causes PKE.K to output the tuple $(\mathsf{epk}_i, \mathsf{esk}_i)$ where $\mathsf{epk}_i \in \mathsf{pk}_i$; and moreover, (sid, s_i) is a correct decryption of CT_i using esk_i — the witness w includes the randomness used in PKE.K, esk_i, and the randomness of PKE.Dec.
- multicast the tuple (sid, i, s_i, π_i);
- upon receiving a tuple (sid, j, s_j, π_j) such that π_j verifies w.r.t. the statement $(sid, j, s_j, \mathsf{CT}_j)$ where CT_j was the output of RBC_0 during the Share sub-protocol, add s_j to the set $\mathbb{S}$.
- whenever the set $\mathbb{S}$'s size is at least $\lfloor n/2 \rfloor + 1$, call the reconstruction algorithm of Shamir Secret Sharing to reconstruct a secret s, and if reconstruction is successful, output the result.

Since the extractor algorithm $\mathcal{E}$ is only needed in the proofs, we defer its presentation to the online full version [14].

Theorem 6. *Without loss of generality, assume that $T_{\mathrm{rbc}} \geq 3\Delta$ (if not, we can simply define $T_{\mathrm{rbc}} := 3\Delta$); and moreover assume that the* RBC *scheme employed satisfies T_{rbc}-liveness, validity, T_{rbc}-consistency, and close termination*

[5] For simplicity, we omit writing the randomness consumed by PKE.Enc which is also part of the witness.

under 0.5*-weak-synchrony; the* NIZK *scheme employed satisfies zero-knowledge and simulation soundness; and the* PKE *scheme satisfies IND-CCA security and is perfectly correct. Then, the above* VSS *protocol satisfies* $2T_{\text{rbc}}$*-liveness,* T_{rbc}*-validity, and non-malleability under* 0.5*-weak-synchrony.*

Proof. Please refer to the online full version [14].

7 Leader Election (LE)

7.1 Definition

A leader election (LE) protocol is an inputless protocol such that when a node terminates, it outputs an elected leader $L \in [n]$. For the outcome of LE to be considered good, we want that not only every honest node must agree on the leader, but also that this leader belongs to $\mathcal{O}_r$ for some a-priori known round r. We would like that the LE achieves a good outcome with $O(1)$ probability. Our actual definition below is somewhat tricky due to compositional issues that arise due to multiple LE instances sharing the same PKI. We would like that even when multiple LE instances share the same PKI, roughly speaking, almost surely there is still *independent* constant probability that each individual instance's outcome is good. In our formal definition below, we will precisely specify which subset of honest coins that are freshly chosen in each LE instance allow us to capture this desired independence. Note that this independence property is desired because later in our BA protocol, we need to argue that after super-logarithmically many trials, an honest leader must be elected except with negligible probability. We formalize the definitions below.

T-liveness. Consider an $(\mathcal{A}, \mathcal{Z})$ pair that is allowed to spawn multiple concurrent or sequential LE instances all of which share the same n, PKI setup, and Δ.

Let $T(n, \Delta, \kappa)$ be a polynomial function in its parameters. We say that an LE protocol denoted Π satisfies T-liveness under χ-weak-synchrony if for every non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects χ-weak-synchrony and may spawn multiple LE instances sharing the same PKI, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for every $\kappa \in \mathbb{N}$, except with $\mathsf{negl}(\kappa)$ probability, the following holds for every LE instance spawned (for the LE instance of interest, we rename its starting round to round 0):

every node in $\mathcal{O}_r$ for $r \geq T$ must have output by the end of round r.

(T^*, q)-quality. We consider an $(\mathcal{A}, \mathcal{Z})$ pair who can spawn $m(\kappa)$ LE instances possibly running concurrently. Henceforth let $\boldsymbol{\rho}^*_\ell$ denote the collection of the following randomness:

for each node honest and online in the starting round (i.e., round 0) of the ℓ-th instance: the first $d(\kappa, n)$ bits of randomness consumed by this node in this round,

where $d(\kappa, n)$ is an appropriate polynomial function that depends on the construction. Let $\boldsymbol{\rho}$ be all randomness consumed by the entire experiment (including

by $(\mathcal{A}, \mathcal{Z})$ and by honest nodes and the randomness of the PKI), and let $\boldsymbol{\rho} \backslash \boldsymbol{\rho}^*_\ell$ denote all other randomness besides $\boldsymbol{\rho}^*_\ell$.

We say that a leader election (LE) protocol satisfies (T^*, q)-quality under χ-weak-synchrony, iff for any polynomial function $m(\kappa)$, for any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects χ-weak-synchrony and spawns $m(\kappa)$ LE instances possibly executing concurrently, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, for every $1 \leq \ell \leq m(\kappa)$, except for a $\mathsf{negl}(\kappa)$ fraction of choices for $\boldsymbol{\rho} \backslash \boldsymbol{\rho}^*_\ell$, there exist at least q fraction of choices for $\boldsymbol{\rho}^*_\ell$, such that the experiment (determined by the joint randomness choice above) would guarantee the following good events for the ℓ-th instance:

1. *Consistency:* if an honest node outputs L and another honest node outputs L', it holds that $L = L'$; and
2. *Fairness:* let L be the leader output by an honest node, we have that $L \in \mathcal{O}_{T^*}$ (assuming that the start round of the ℓ-th instance is renamed round 0).

7.2 Construction

The construction is a bit involved and thus we refer the reader to Section 2.3 for an intuitive explanation of our protocol. Below we focus on a formal description.

Let VSS denote a verifiable secret sharing scheme for inputs over the finite field $\mathbb{F}$. (see Section 6.2) and let T_{vss} be its liveness parameter. We now show how to construct leader election from verifiable secret sharing. In our protocol below, there are n^2 instances of VSS. Henceforth we use $\mathsf{VSS}[i, j]$ to denote the j-th instance where node i is the designated dealer. Additionally, let RBC denote a reliable broadcast protocol (see Section 5) whose liveness parameter is denoted T_{rbc}. Let $\Sigma := (\mathsf{K}, \mathsf{Sign}, \mathsf{Ver})$ denote a digital signature scheme.

The following protocol is executed by every node, below we describe the actions taken by node $i \in [n]$ — for simplicity we implicitly assume that every message is tagged with its purported sender:

- **PKI setup** (shared across all LE instances): each node i calls $(\mathsf{rpk}_i, \mathsf{rsk}_i) \leftarrow \mathsf{RBC.K}(1^\kappa)$; $(\mathsf{vpk}_i, \mathsf{vsk}_i) \leftarrow \mathsf{VSS.K}(1^\kappa)$; and $(\mathsf{vk}_i, \mathsf{ssk}_i) \leftarrow \Sigma.\mathsf{K}(1^\kappa)$. Now its public key is $(\mathsf{rpk}_i, \mathsf{vpk}_i, \mathsf{vk}_i)$ and its secret key is $(\mathsf{rsk}_i, \mathsf{vsk}_i, \mathsf{ssk}_i)$.
  In the following, we describe the leader election (LE) protocol. We assume that all LE protocols share the same PKI. Moreover, whenever a node i uses ssk_i to sign messages, the message to be signed is always tagged with the session identifier *sid* of the current instance and signature verification also verifies the signature to the same *sid*.
- **Round 0**: Node i chooses n random coins $c_{i,1}, \ldots, c_{i,n} \in \mathbb{F}$. For instances $\mathsf{VSS}[i, 1]$, ..., $\mathsf{VSS}[i, n]$ where node i is the dealer, node i provides the inputs $c_{i,1}, \ldots, c_{i,n}$ respectively to each instance. Then, node i invokes the Share sub-protocol of all n^2 instances of VSS.
- **Any round**: At any time during the protocol, if in node i's view, all n VSS instances where node j is the dealer has terminated outputting "`sharing succeeded`", we say that node i now considers j as a *qualified dealer*.
- **Round** T_{vss}: If in round T_{vss}, at least $\lfloor n/2 \rfloor + 1$ qualified dealers have been identified so far: let D be the current set of all qualified dealers; reliably

broadcast the message (`qualified-set`, D) using RBC. Henceforth, we use $\mathsf{RBC}[j]$ to denote the RBC instance where j is the sender. If not enough qualified dealers have been identified, reliably broadcast the message $\perp$.

- **Any round**: In any round during the protocol, if $\mathsf{RBC}[j]$ has output (`qualified-set`, D_j) such that D_j is a subset of $[n]$ containing at least $\lfloor n/2 \rfloor + 1$ nodes, and moreover every node in D_j has become qualified w.r.t. node i's view so far, then node i considers j as a *candidate*, and node i records the tuple (j, D_j).
- **Round $T_{\text{vss}} + T_{\text{rbc}}$:** In round $T_{\text{vss}} + T_{\text{rbc}}$, do the following:
  - invoke the $\mathsf{Reconstruct}$ sub-protocol of all VSS instances;
  - if at least $\lfloor n/2 \rfloor + 1$ nodes are now considered candidates: let S be the set of all candidates so far; now multicast (`candidate-set`, S) along with a signature on the message.
- **Any round**: At any time, if a node i has observed a (`candidate-set`, S_j) message with a valid signature from the purported sender j where $S_j \subseteq [n]$ is at least $\lfloor n/2 \rfloor + 1$ in size, and moreover, every node in S_j is now considered a candidate by node i too, we say that node i becomes *happy* with j.
- **As soon as** node i becomes happy with at least $\lfloor n/2 \rfloor + 1$ nodes, let S_i^* be the current set of nodes that are considered candidates;
- **As soon as** the relevant VSS instances (needed in the following computation) have terminated the reconstruction phase outputting a reconstructed secret — henceforth let $c'_{u,v}$ be the secret reconstructed from instance $\mathsf{VSS}[u, v]$:
  - For every $u \in S_i^*$: let (u, D_u) be a previously recorded tuple when u first became a candidate; compute node u's *charisma* as $C_u := \prod_{v \in D_u} c'_{v,u}$.
  - Output the node $u^* \in S_i^*$ with maximum charisma (where ordering between elements in $\mathbb{F}$ is determined using lexicographical comparisons).

Theorem 7. *Suppose that the* VSS *scheme satisfies* T_{vss}*-liveness,* T_{vss}*-validity, and non-malleability under* 0.5*-weak-synchrony; the* RBC *scheme satisfies* T_{rbc}*-consistency,* T_{rbc}*-liveness, validity, and close termination under* 0.5*-weak-synchrony, and the signature scheme satisfies existential unforgeability under chosen-message attack. Then, the above* LE *scheme satisfies* $(2T_{\text{vss}}+T_{\text{rbc}})$*-liveness and* $(T_{\text{vss}}, 1/2)$*-quality under* 0.5*-weak-synchrony.*

Proof. Please refer to the online full version [14].

8 Byzantine Agreement

Let LE be a leader election scheme that satisfies T_{le}-liveness and $(T'_{\text{le}}, 1/2)$-quality under 0.5-weak-synchrony where $T_{\text{le}} > T'_{\text{le}}$.

PKI setup. Upfront, every node performs PKI setup as follows: every node calls $(\mathsf{LE.pk}, \mathsf{LE.sk}) \leftarrow \mathsf{LE.K}(1^\kappa)$; further, it calls $(\mathsf{vk}, \mathsf{ssk}) \leftarrow \Sigma.\mathsf{K}(1^\kappa)$. The tuple $(\mathsf{LE.pk}, \mathsf{vk})$ is the node's public key and registered with the PKI, and the tuple $(\mathsf{LE.sk}, \mathsf{ssk})$ is the node's secret key.

As before we assume that all messages, excluding the ones within the LE instance[6], are signed (using each node's ssk) and tagged with the purported

[6] Recall that the LE instance deals with its own message signing internally.

sender, and honest recipients verify the signature (using the purported sender's vk) upon receiving any message. To allow multiple BA instances to share the same PKI, we assume that a message is always tagged with the current instance's session identifier *sid* before it is signed and the verification algorithm checks the *sid* accordingly. Messages with invalid signatures are discarded immediately.

Protocol. In the following, an epoch-e commit evidence for $b \in \{0,1\}$ is a set of signatures from $\lfloor n/2 \rfloor + 1$ number of distinct nodes on the message $(\texttt{prepare}, e, b)$. Our protocol works as follows. For each epoch $e = 1, 2, \ldots$, do the following (henceforth the initial round of each epoch is renamed round 0 of this epoch):

- **Propose.** For the initial T_{le} rounds in each epoch, do the following:
  1. If the current epoch is $e = 1$, then in round 0 of epoch 1, the sender multicasts a signed tuple $(\texttt{propose}, b)$ where b is its input bit.
  2. Round 0 of every epoch: invoke an instance of the LE protocol.
  3. Round T'_{le} of every epoch: every node $i \in [n]$ flips a random coin $b_i \leftarrow_{\$} \{0,1\}$, and multicasts a signed tuple $(\texttt{propose}, b_i)$
- **Prepare (round $T_{\text{le}} + \Delta$ of each epoch).** If $e = 1$ and a node has heard an epoch-1 proposal for b from the sender, then it multicasts the signed tuple $(\texttt{prepare}, e, b)$. Else if $e > 1$, every node performs the following:
  1. if an epoch-e proposal of the form $(\texttt{propose}, e, b)$ has been heard from an eligible epoch-e proposer which is defined by the output of LE and moreover, either an epoch-$(e-1)$ commit evidence vouching for b or $\lfloor n/2 \rfloor + 1$ epoch-$(e-1)$ complaints from distinct nodes have been observed, multicast the signed tuple $(\texttt{prepare}, e, b)$.
     If LE has not produced an output in the range $[n]$ at the beginning of this round, act as if no valid proposal has been received.
  2. else multicast the signed tuple $(\texttt{prepare}, e, b)$ if the node has seen an epoch-$(e-1)$ commit evidence vouching for the bit b (if both bits satisfy this then send a prepare message for each bit).
- **Commit (round $T_{\text{le}} + 2\Delta$ of each epoch).** If by the beginning of the commit round of the current epoch e, a node
  1. has heard an epoch-e commit evidence for the bit b;
  2. has not observed a valid epoch-e proposal for $1 - b$ (from an eligible proposer); and
  3. has not observed any epoch-$(e-1)$ commit evidence for $1 - b$;

  then multicast the signed tuple $(\texttt{commit}, e, b)$.
- **Complain (round $T_{\text{le}} + 3\Delta$ of each epoch).** If no epoch-e commit evidence has been seen, multicast the signed tuple $(\texttt{complain}, e)$.
- End of this epoch and beginning of next epoch (round $T_{\text{le}} + 4\Delta$).

Finalization. At any time during the protocol, if a node has collected $\lfloor n/2 \rfloor + 1$ commit messages (from distinct nodes) for the same epoch and vouching for the same bit b, then output b if no bit has been output yet and continue participating in the protocol (we devise a termination technique in the online full version [14]).

Theorem 8. *Suppose that the* LE *scheme satisfies* T_{le}*-liveness and* $(T_{\text{le}}', 1/2)$*-quality under* 0.5*-weak-synchrony, the digital signature scheme employed is secure, and let* λ *be any super-logarithmic function in the security parameter* κ*. Then, the* BA *scheme above satisfies consistency, validity, and* $\lambda \cdot (T_{\text{le}} + 4\Delta)$*-liveness under* 0.5*-weak-synchrony.*

Proof. Please refer to the online full version [14].

References

1. Gmail and google drive are experiencing issues, and naturally people are complaining about it on twitter. `https://www.huffingtonpost.com/entry/gmail-issue_n_3099988`.
2. I. Abraham, S. Devadas, D. Dolev, K. Nayak, and L. Ren. Efficient synchronous byzantine consensus. In *Financial Crypto*, 2019.
3. S. Badrinarayanan, A. Jain, N. Manohar, and A. Sahai. Secure mpc: Laziness leads to god. Cryptology ePrint Archive, Report 2018/580, 2018.
4. M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10, 1988.
5. C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, pages 524–541, 2001.
6. R. Canetti and T. Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC*, pages 42–51, 1993.
7. M. Castro and B. Liskov. Practical byzantine fault tolerance. In *OSDI*, 1999.
8. D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *Siam Journal on Computing - SIAMCOMP*, 12(4):656–666, 1983.
9. C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 1988.
10. P. Feldman and S. Micali. An optimal probabilistic protocol for synchronous byzantine agreement. In *SIAM Journal of Computing*, 1997.
11. O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *ACM symposium on Theory of computing (STOC)*, 1987.
12. S. D. Gordon, F. Liu, and E. Shi. Constant-round MPC with fairness and guarantee of output delivery. In *CRYPTO*, pages 63–82, 2015.
13. J. Groth and R. Ostrovsky. Cryptography in the multi-string model. In *CRYPTO*, 2007.
14. Y. Guo, R. Pass, and E. Shi. Synchronous, with a chance of partition tolerance. Online full version of this paper, `https://eprint.iacr.org/2019/179.pdf`.
15. J. Katz and C.-Y. Koo. On expected constant-round protocols for byzantine agreement. *J. Comput. Syst. Sci.*, 75(2):91–112, Feb. 2009.
16. L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 1998.
17. L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 1982.
18. S. Micali and V. Vaikuntanathan. Optimal and player-replaceable consensus with an honest majority. MIT CSAIL Technical Report, 2017-004, 2017.
19. R. Pass and E. Shi. The sleepy model of consensus. In *Asiacrypt*, 2017.
20. R. Pass and E. Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Eurocrypt*, 2018.