Synchronous, with a Chance of Partition Tolerance

Yue Guo Rafael Pass Elaine Shi

Thunder Research and Cornell

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?

Contents

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 [12]. In comparison, it is well-known that if message delays can be arbitrarily long [13], 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 [4, 31]: 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 [33] 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 Section A 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 [3, 20, 27] are *underspecified* and *unimplementable* in practice: if a node ever experiences even a short-term outage and receives messages out of sync, these protocols [**?**, 20, 27] 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 [13] or asynchronous [7] 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 [2], 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. [23], 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 [3, 20, 27, 33] 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 1.1 (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 [33]: 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 1.2 (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 1.3 (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 Section B.

1.4 Additional Related Work

We compare with additional related work in this section. As mentioned, known classical-style, synchronous consensus protocols are underspecified and unimplementable in practice. To the best of our knowledge, the only known synchronous consensus (tolerating minority Byzantine fault) that has been deployed is Nakamoto's blockchain [15, 28–30], which was the first *longest-style* protocol. Subsequent works [10, 11, 21, 31, 34] have shown how to remove the Proof-of-Work assumption and yet emulate the stochastic process of Nakamoto consensus. Pass and Shi [30, 31] were the first to observe that in fact, this class of protocols can be proven secure in a so-called *sleepy* model that is more relaxed model than classical synchrony. Unlike classical synchrony which treats only those who have always been online since the very beginning as honest, the sleepy model allows an honest node to join late as well as have short-term outages and later join back. Pass and Shi [30, 31] show that known longest-chain-style consensus protocols can achieve consistency and liveness, as long as at any time, among the *online* nodes, the majority (possibly measured by compute-power) are honest — obviously such protocols also achieve security under strong synchrony assuming honest majority.

Although both the sleepy model and our work allow honest nodes to join late or have temporary outages, the two models are in fact *mutually exclusive*. The reason is that the sleepy model requires progress even if only 1% of the anticipated number of nodes show up — such protocols are henceforth said to be *availability-favoring*. Pass and Shi [30, 31] formally prove that such availability-favoring protocols cannot be partition tolerant. In this paper, we explore a *consistency-favoring* model instead where we favor consistency over availability under network partitions. Intuitively, a minority partition should prefer to get stuck rather than risking inconsistency with the majority.

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 [20, 27] 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:

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

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)

Reliable broadcast. 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 $x = x'$. In fact, 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.

Weakly synchronous RBC construction. At a very high level, our RBC construction combines techniques from classical synchronous "gradecast" [14,20] and asynchronous "reliable broadcast" [6, 7]. 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)

Verifiable secret sharing (VSS) allows a dealer to share a secret among all nodes and later be able to reconstruct the secret. 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 [7] 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 later 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.

Weakly synchronous VSS 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 [18] (see Section C.1). 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)

A leader election (LE) protocol is an inputless protocol that allow nodes to elect a leader denoted $L \in [n]$ among the n nodes[1]. 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.

Weakly synchronous LE construction. Our LE protocol is in fact inspired by the *asynchronous* leader election protocol by Canetti and Rabin [7]. 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 here 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 [7] 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

[1]Some recent works such as Dfinity [19] and Algorand [9] proposed to adopt a Verifiable Random Function [26] (VRF) to perform leader election in consensus protocols. This approach does not work in our context since the adversary is allowed to decide which nodes are offline in this round after examining what messages honest nodes want to send in the present round. If a VRF is employed, the adversary can examine who is the leader and make the leader offline. Such an adversary can hamper liveness indefinitely.

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 of this protocol.

2.4 Byzantine Agreement (BA)

The next question is how to construct BA given a 0.5-weakly-synchronous LE scheme. This step turns out to be non-trivial too. In particular, we stress that existing synchronous BA protocols [3, 20, 27] 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 works as follows. The 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 tentatively 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[2].

- *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.

[2]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.

- *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 point of time, if $\lfloor n/2 \rfloor + 1$ number of commits from the same epoch and for the same bit b has been observed, output the decision b and continue participating in the protocol (we shall describe a termination technique in Section E).

Remark 2.1. We point out that although our BA protocol might somewhat resemble the recent work by Abraham et al. [3], 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 [3] 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 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.

Weakly synchronous MPC 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 [5, 16] 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 [4, 17]. In a TMFHE scheme [4],

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 $\mathcal{O}_{T_{\text{ba}}}$ 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. [4] 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. [4] 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 (`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. [4]. There is, in fact, a notable difference in a low-level subtlety: in Badrinarayanan et al. [4]'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. [4]. We defer a more detailed discussion of these technicalities to Sections 9 and C.2.

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 3.1 (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 3.2 (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 3.3. 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 3.4 (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. [23] 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 [8, 13, 22]. 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 3.5 (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 and its proof.

Theorem 4.1. *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.*

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).

Proof. For the sake of contradiction, suppose that there exists some BA protocol Π that achieves consistency, validity, and T-liveness under χ-weak-synchrony for $\chi \leq 0.5 - \frac{1}{n}$. Consider two randomized executions of Π denoted EXEC_0 and EXEC_1 respectively, each with n nodes and with $\Delta = 1$. We assume that n is even and recall that node 1 is the designated sender. In both executions, all nodes are honest. We consider the following two sets of nodes $S_a = \{1, \ldots, n/2\}$ and $S_b = \{n/2+1, \ldots, n\}$. The adversary declares S_a to be the set of honest and online nodes in every round and declares all nodes in S_b to be offline throughout. In both executions, $\mathcal{A}$ always delivers all messages between nodes in S_a immediately at the beginning of the next round, and it does the same for S_b. However, no messages are delivered in between S_a and S_b. Clearly, such an adversary $\mathcal{A}$ respects χ-weak-synchrony. The only difference between the two executions is that in EXEC_b, the sender receives the input bit $b \in \{0,1\}$.

Now consider EXEC_b: due to validity and liveness, except with negligible probability, all nodes in S_a would output b by the end of round T. For nodes in S_b, their view in the execution is identically distributed as an execution in which all nodes in S_b form the honest and online set and all nodes in S_a are offline — in such an execution $\mathcal{A}$ respects χ-weak-synchrony too. Thus by liveness, except with negligible probability, nodes in S_b would output a bit too by the end of round T; by consistency, they must output the bit b. However, observe that the joint view of nodes in S_0 is identically distributed for both EXEC_0 and EXEC_1. Thus we have reached a contradiction. □

Best-possible partition tolerance. In light of Theorem 4.1, a BA protocol secure under 0.5-weak-synchrony is also said to be best-possible partition tolerant.

4.2 Classical Corrupt-Majority Protocols Sacrifice Partition Tolerance

It is well-known that there exist Byzantine Agreement protocols that tolerate arbitrarily many byzantine faults [12] 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.*

The remainder of the section not only formalizes the above intuitive reasoning. Henceforth we use the following notation:

- We say that $(\mathcal{A}, \mathcal{Z})$ respects μ-strong-synchrony 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 4.2. $\forall 0 < \mu < 0.5, \chi \leq 1 - \mu - 2/n,\ \ \mathcal{BA}\{\mu\} \cap \mathcal{BA}^+\{\chi\} = \emptyset$.

Proof. For the sake of contradiction, we assume there exists a protocol Π such that $\Pi \in \mathcal{BA}\{\mu\} \cap \mathcal{BA}^+\{\chi\}$ for some $0 < \mu < 0.5$ and some $\chi < 1 - \mu - 2/n$. Then there exists some polynomial

functions $T(\cdot,\cdot,\cdot)$ and $T^+(\cdot,\cdot,\cdot)$, such that Π satisfies T-liveness under μ-strong-synchrony and T^+-liveness under χ-weak-synchrony.

Consider two randomized executions (henceforth denoted EXEC_0 and EXEC_1) of Π each with n nodes and with $\Delta = 1$. All nodes are honest and recall that the designated sender is node 1. Messages between $S = \{1, \ldots, n - (\lfloor \mu n \rfloor + 1)\}$ and $S' = [n] \backslash S$ sent in or before the round $\max(T, T^+) + 1$ are delayed till the beginning of the round $\max(T, T^+) + 2$. All other messages are delivered immediately at the beginning of the next round (including messages between S and S' sent after the round $\max(T, T^+)+1$). It is not hard to see that $|S| \geq \lfloor \chi \cdot n \rfloor + 1$ and $|S'| \geq \lfloor \mu \cdot n \rfloor + 1$. The only difference between the two executions is that $\mathcal{Z}$ gives input bit 0 to the designated sender (i.e., node 1) in EXEC_0 and input bit 1 to the designated sender in EXEC_1.

Suppose that the adversary declares the set S (which includes the designated sender) to be the honest and online set throughout, then clearly $(\mathcal{A}, \mathcal{Z})$ respects χ-weak-synchrony in both executions. By liveness and validity under χ-weak-synchrony, except with negligible probability, all nodes in S have output b in EXEC_b for $b \in \{0, 1\}$ by the end of round $\max(T, T^+) + 1$; further, all nodes in S' have output b by the end of round $\max(T, T^+) + 2$ for each $b \in \{0, 1\}$ (since nodes in S' come back online in round $\max(T, T^+) + 2$).

On the other hand, in both EXEC_0 and EXEC_1, before round $\max(T, T^+) + 2$, the joint view of nodes in S' is identically distributed as one in which only nodes in S' are honest and can communicate with each other within a single round, and all other nodes (including the designated sender) are faulty and have crashed. Thus from the joint view of nodes in S' in both executions before round $\max(T, T^+) + 2$, the adversary respects μ-strong-synchrony. By liveness under μ-strong-synchrony, in both EXEC_0 and EXEC_1, a node in S' must output before the round $\max(T, T^+) + 1$; and moreover, its output in EXEC_0 must be identically distributed as its output in EXEC_1. We have thus reached a contradiction.

□

Remark 4.3. Note that the proof in fact implies that an even stronger version of the theorem holds where we broaden both the family $\mathcal{BA}\{\mu\}$ and the family $\mathcal{BA}^+\{\chi\}$:

- We may broaden $\mathcal{BA}\{\mu\}$ to include even protocols that are only required to satisfy consistency, validity, and polynomial liveness w.r.t. any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects μ-strong-synchrony, and moreover, all corrupt nodes can only exhibit *crash fault*.
- We may broaden $\mathcal{BA}^+\{\chi\}$ to include even protocols that are only required to satisfy consistency, validity, and polynomial liveness w.r.t. any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects χ-weak-synchrony, and moreover *makes no corruptions*.

5 Reliable Broadcast (RBC)

5.1 Additional Preliminary

In our upper bound sections (Sections 5, 6.2, 7, 8, and 9), 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 either i has multicast or has 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 Section D, 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.

5.2 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[3];
- **Extractor** $\mathcal{E}$: a polynomial-time deterministic extractor denoted $\mathcal{E}$ that is a construct used 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 property $P \in \{T\text{-consistency, validity, } T\text{-liveness, close termination}\}$, we say that an RBC protocol Π satisfies property P under χ-weak-synchrony iff for a non-uniform p.p.t. pair $(\mathcal{A}, \mathcal{Z})$ 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 instance of RBC:

- *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 $x' = x$.

[3]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.

- *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 5.1. 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).

5.3 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.

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 a construct needed only in the security definitions and proofs and not in the real-world protocol.

5.4 Proof

Henceforth although not explicitly noted, we by default assume that $(\mathcal{A}, \mathcal{Z})$ is non-uniform p.p.t. and respects 0.5-weak-synchrony. As before, a good execution is one in which no honest node ever sees a forged signature in any RBC instance. Assuming that the signature scheme is secure and recall that all messages are signed with the session's identifier, all but a negligible fraction of executions must be good. Henceforth we consider only good executions. In the proofs below we focus on what happens in a specific challenge instance spawned by $(\mathcal{A}, \mathcal{Z})$; however, in a good execution, all the statements and proofs apply to all instances spawned by $(\mathcal{A}, \mathcal{Z})$.

Henceforth we say that a message m is *in honest and online view* in round r in some execution, if for every $t \geq r$, every node in $\mathcal{O}_t$ must have observed m at the beginning of t in this execution.

5.4.1 Consistency

Lemma 5.2. *Consider a good execution: if some node in $\mathcal{O}_{2\Delta}$ multicasts $(\texttt{commit}, y)$ and another node in $\mathcal{O}_{2\Delta}$ multicasts $(\texttt{commit}, y')$, then $y = y'$.*

Proof. For the sake of contradiction, we assume that two nodes $i, i' \in \mathcal{O}_{2\Delta}$ multicast $(\texttt{commit}, y)$ and $(\texttt{commit}, y')$ respectively for $y \neq y'$. Then node i must have received $\lfloor n/2 \rfloor + 1$ valid $(\texttt{ack}, y)$ messages, and one of them must have been signed by some node $j \in \mathcal{O}_{\Delta}$ during round Δ. Now, node j must have observed a valid $(\texttt{propose}, y)$ message at the beginning of round Δ and thus by our strong message delivery assumption (Assumption 2), everyone in $\mathcal{O}_{2\Delta}$ will have observed the same at the beginning of round 2Δ. Thus no one in $\mathcal{O}_{2\Delta}$ will multicast $(\texttt{commit}, y')$ and this contradicts our assumption. □

Construction of the extractor $\mathcal{E}$. Consider the following extractor $\mathcal{E}$ that takes in all nodes' public keys and the honest nodes' transcript in the initial 2Δ rounds as input and outputs the following:

- If m number of honest nodes multicast $(\texttt{commit}, y)$ in round 2Δ for some y, and moreover $m + f \geq \lfloor n/2 \rfloor + 1$ where f denotes the number of corrupt nodes, then output y.
- Otherwise, output $\perp$.

Note that if at least $\lfloor n/2 \rfloor + 1 - f$ number of honest nodes multicast $(\texttt{commit}, y)$ in round 2Δ, at least one of these honest nodes must be in $\mathcal{O}_{2\Delta}$. Thus by Lemma 5.2, the output of the extractor must be *uniquely defined*.

Theorem 5.3 (Consistency)**.** *The* RBC *protocol in Section 5.3 satisfies 2Δ-consistency under 0.5-weak-synchrony.*

Proof. Consider a good execution: we prove that if an honest node (denoted j) ever outputs y, then the extractor $\mathcal{E}$ must output y too. Node j must have seen $\lfloor n/2 \rfloor + 1$ $(\texttt{finalize}, y)$ messages signed by distinct nodes. At least one of these `finalize` messages is from an honest node i. Node i must have seen $\lfloor n/2 \rfloor + 1$ number of $(\texttt{commit}, y)$ messages signed by distinct nodes. This means at least $\lfloor n/2 \rfloor + 1 - f$ number of honest nodes must have multicast $(\texttt{commit}, y)$ in round 2Δ. This means that the extractor $\mathcal{E}$'s outcome must be y too — note that as argued above, the extractor's outcome is uniquely defined. □

5.4.2 Validity

Theorem 5.4 (Validity)**.** *The* RBC *protocol in Section 5.3 satisfies validity under 0.5-weak-synchrony.*

Proof. If the sender is honest and its input is x, in a good execution without forged signatures in honest view, no honest node will multicast $(\texttt{ack}, x')$ for any $x' \neq x$; thus no honest node will ever observe $\lfloor n/2 \rfloor + 1$ `ack` messages for $x' \neq x$ and no honest node will multicast $(\texttt{commit}, x')$ for $x' \neq x$. Thus there cannot be $\lfloor n/2 \rfloor + 1$ $(\texttt{commit}, x')$ messages for $x' \neq x$ in honest view. Thus no honest node will multicast $(\texttt{finalize}, x')$ for $x' \neq x$ and there cannot be $\lfloor n/2 \rfloor + 1$ $(\texttt{finalize}, x')$ messages in honest view. □

5.4.3 Liveness

Theorem 5.5 (Liveness)**.** *The* RBC *protocol in Section 5.3 satisfies 4Δ-liveness under 0.5-weak-synchrony.*

Proof. If the sender is in $\mathcal{O}_0$ and proposes the value y in round 0, then a valid `propose` message for y will appear in honest and online view in round Δ, and there is no other valid `propose` message for $y' \neq y$ ever in honest view. Thus everyone in $\mathcal{O}_\Delta$ will multicast an `ack` message for y in round Δ, and all of these messages will be in honest and online view in round 2Δ. Thus everyone in $\mathcal{O}_{2\Delta}$ will multicast an `commit` message for y in round 2Δ and all of these messages will be in honest and online view in round 3Δ. Now everyone in $\mathcal{O}_{3\Delta}$ will multicast a `finalize` message for y in round r and all of these messages will be in honest and online view in round 4Δ. □

5.4.4 Close Termination

Theorem 5.6 (Close termination)**.** *The* RBC *protocol in Section 5.3 satisfies close termintion under* 0.5*-weak-synchrony.*

Proof. Consider a good execution. Let r be the first round in which there is some honest node who has observed $\lfloor n/2 \rfloor + 1$ number of valid $(\texttt{finalize}, x)$ messages from distinct nodes for some value x. One of these $(\texttt{finalize}, x)$ messages must have been sent by someone $i^* \in \mathcal{O}_r$ in round r or earlier. Now i^* must have observed a collection of $\lfloor n/2 \rfloor + 1$ number of $(\texttt{commit}, x)$ messages from distinct nodes at the beginning of round r. Thus by our strong message delivery assumption (Assumption 2), this collection of `commit` messages will appear in honest and online view in round $r + \Delta$. Thus everyone in $\mathcal{O}_{r+\Delta}$ will have multicast $(\texttt{finalize}, x)$ in round $r + \Delta$ if they have not done so earlier. By our strong message delivery assumption (Assumption 2), a collection of $\lfloor n/2 \rfloor + 1$ number of $(\texttt{finalize}, x)$ messages from distinct nodes will appear in honest and online view in round $r + 2\Delta$. □

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 [7] 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 achieves security against unbounded adversaries [7]. 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 initially 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.

We give a formal description below.

6.1.1 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 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. Reconstruct: All nodes execute the 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 Reconstruct in the same round. Should execution of 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.

6.1.2 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* 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 Reconstruct, then, anyone in $\mathcal{O}_t$ for $t \geq r + T$ must have terminated the Reconstruct sub-protocol outputting some reconstructed value in $\mathbb{F}$ by the end of round t.

6.1.3 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`"[4];

(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 $s' = s$.

6.1.4 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. **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 $\mathsf{Extract}$ or $\mathsf{Reconstruct}$ 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[5].

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

[5]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.

Definition 6.1 (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 Section C.1);
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;
- 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[6] 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 $(sid, \{\mathsf{CT}_j\}_{j\in[n]}, \pi)$, it calls $\mathsf{NIZK.V}$ 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, it sets $\mathsf{flag} := 1$ (we assume that flag was initially 0).
- **Round T_{rbc}:** if $\mathsf{flag} = 1$, reliably broadcast the message "$\mathtt{ok}$"; else reliably broadcast the message "$\perp$";

[6]For simplicity, we omit writing the randomness consumed by $\mathsf{PKE.Enc}$ which is also part of the witness.

- **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 Section 6.3.

6.3 Proofs

6.3.1 Liveness

Theorem 6.2. *Without loss of generality, assume that $T_{\text{rbc}} \geq 3\Delta$ (if not, we can simply define $T_{\text{rbc}} := 3\Delta$); and moreover assume that the* RBC *scheme employed satisfies T_{rbc}-liveness, validity, T_{rbc}-consistency, and close termination under 0.5-weak-synchrony; the* NIZK *scheme employed satisfies simulation soundness; and the* PKE *scheme is perfectly correct. Then, the above* VSS *scheme satisfies $2T_{\text{rbc}}$-liveness under 0.5-weak-synchrony.*

Proof. Henceforth we ignore the negligible fraction of bad views in which the following types of bad events take place: the T_{rbc}-liveness, validity, consistency, or the close termination properties of the RBC scheme are violated; or an honest node accepts a zero-knowledge proof for an untruthful statement.

Termination of Share under an honest and initially online dealer. Assume that the dealer is honest and online in the round in which Share is invoked (henceforth rename this round to be round 0). Then, by the T_{rbc}-liveness and validity of the RBC scheme, for every $r \geq T_{\text{rbc}}$, everyone in $\mathcal{O}_r$ will have output a tuple from the RBC_0 instance by the end of round r, and moreover the tuple must be what the dealer inputs into the RBC_0 instance. By completeness of the NIZK scheme, everyone in $\mathcal{O}_{T_{\text{rbc}}}$ will have passed the checks in the Share sub-protocol and will have reliably broadcast "ok" in round T_{rbc}. Now by the T_{rbc}-liveness and validity of the RBC scheme again, for any $r \geq 2T_{\text{rbc}}$, everyone in $\mathcal{O}_r$ will have output "sharing succeeded".

Close termination. We next prove the close termination property. The following claim is easy to see (due to the validity of RBC):

Claim 6.3. *If an honest node outputs "*`sharing succeeded`*" in round r, then the following must have happened by round r: some honest node i^* must have reliably broadcast "*`ok`*" and this node i^* must have output a tuple $(sid, \{\mathsf{CT}_j\}_{j\in[n]}, \pi)$ from the RBC_0 instance and the proof π in the tuple must have verified correctly by i^*.*

Now, suppose an honest node outputs "`sharing succeeded`" in round r. By Claim 6.3 and by consistency and close termination of the RBC scheme, in every round $r' \geq r + 2\Delta$, everyone in $\mathcal{O}_{r'}$ will have

1. seen at least $\lfloor n/2 \rfloor + 1$ RBC instances output "`ok`"; and
2. seen the RBC_0 instance output a tuple and the tuple must be the same as what i^* outputs from its RBC_0 (see Claim 6.3) and thus the tuple is well-formed.

Thus everyone in $\mathcal{O}_{r'}$ will have output "`sharing-succeeded`" by the end of round r'.

Termination of $\mathsf{Reconstruct}$. Finally, we prove termination of the $\mathsf{Reconstruct}$ protocol. Suppose that some honest node has output "`sharing-succeeded`" in some round r, and that the $\mathsf{Reconstruct}$ sub-protocol has also been invoked by the end of round r. By close termination of the Share phase, for every $r' \geq r + 2\Delta$, everyone in $\mathcal{O}_{r'}$ must have output "`sharing-succeeded`". Thus everyone $i \in \mathcal{O}_{r+2\Delta}$ must have multicast (sid, i, s_i, π_i) by the end of round $r + 2\Delta$; and for every $t \geq r + 3\Delta$, everyone in $\mathcal{O}_t$ must have received these messages multicast by everyone in $\mathcal{O}_{r+2\Delta}$ by round t. Due to the completeness of NIZK, the proof π_i in each such received message must verify.

So far we have shown that for every $t \geq r + 3\Delta$, everyone in $\mathcal{O}_t$ will have a large enough decrypted share set $\mathbb{S}$ to start the reconstruction process of the secret sharing scheme. We have yet to prove that reconstruction will be successful, i.e., all the values in $\mathbb{S}$ form an internally consistent sharing of some secret.

By consistency of RBC_0, all honest nodes' output tuple from RBC_0 must be the same, and let $(sid, \{\mathsf{CT}_j\}_{j\in[n]}, \pi)$ be this tuple. By Claim 6.3, some honest node i^* must have verified the proof π. Further, before an honest node adds a share to the set $\mathbb{S}$ it must have verified a proof showing that the share is consistent with what is encrypted inside one of the ciphertexts denoted CT_j contained in the tuple $(sid, \{\mathsf{CT}_j\}_{j\in[n]}, \pi)$. Due to simulation soundness of the NIZK scheme (which implies the standard notion of soundness) and perfect correctness of PKE, except with negligible probability the shares in $\mathbb{S}$ must be an internally consistent sharing of some secret. □

6.3.2 Definition of the Extractor Algorithm $\mathcal{E}$

The extractor algorithm $\mathcal{E}$ is defined as follows: given $\{\mathsf{pk}_i\}_{i\in[n]}$, $\{\mathsf{esk}_i\}_{i\in\mathsf{Honest}}$ and the initial T_{rbc} rounds of the honest nodes' transcript in the Share sub-protocol, perform the following:

- call the RBC scheme's $\mathcal{E}_{\mathrm{rbc}}$ to extract a tuple;
- if the tuple output by $\mathcal{E}_{\mathrm{rbc}}$ is of the form $(sid, \{\mathsf{CT}_j\}_{j\in[n]}, \pi)$ and the proof π verifies w.r.t. the statement $(sid, \{\mathsf{pk}_i, \mathsf{CT}_i\}_{i\in[n]})$, then
  1. for $j \in \mathsf{Honest}$, use esk_j to decrypt CT_j and obtain s'_j;
  2. now call the secret sharing scheme's reconstruction algorithm to recover a secret s' from the set of shares $\{s'_j\}_{j\in\mathsf{Honest}}$;

3. output the reconstructed secret (if reconstruction is successful);

- else output $\perp$ if either the tuple output by $\mathcal{E}_{\text{rbc}}$ is not of the anticipated form, or at least one the above checks fails, or the reconstruction step above is unsuccessful.

6.3.3 Validity

Theorem 6.4. *Assume that* RBC *satisfies* T_{rbc}*-consistency and* T_{rbc}*-validity,* NIZK *satisfies simulation soundness,* PKE *satisfies perfect correctness, then the* VSS *scheme satisfies* T_{rbc}*-validity under* 0.5*-weak-synchrony.*

Proof. Henceforth we ignore the negligible fraction of bad executions in which RBC's T_{rbc}-consistency, or its T_{rbc}-validity, or NIZK's simulation soundness is violated. The following statements hold for the remaining good executions. By T_{rbc}-consistency of the RBC scheme, every honest node's RBC_0 must output what the $\mathcal{E}_{\text{rbc}}$ extractor (inside the construction of $\mathcal{E}_{\text{vss}}$) extracted if any output is ever produced.

1. If the tuple output by $\mathcal{E}_{\text{rbc}}$ is not a well-formed tuple of the form $\left(sid, \{\mathsf{CT}_j\}_{j\in[n]}, \pi\right)$ or if π does not verify for the statement $\left(sid, \{\mathsf{CT}_j, \mathsf{pk}_j\}_{j\in[n]}\right)$, then $\mathcal{E}_{\text{vss}}$ will output $\perp$. In this case, every honest node will input $\perp$ to RBC and by validity of RBC, no honest node will collect enough "`ok`" outputs and thus will never output "`sharing succeeded`".

2. Else let $\mathcal{E}_{\text{rbc}}$'s output be $\left(sid, \{\mathsf{CT}_j\}_{j\in[n]}, \pi\right)$ where π verifies for the statement $\left(sid, \{\mathsf{CT}_j, \mathsf{pk}_j\}_{j\in[n]}\right)$.

   In this case, $\mathcal{E}_{\text{rbc}}$ will decrypt the ciphertexts $\{\mathsf{CT}_j\}_{j\in\mathsf{Honest}}$ and use the decrypted results to reconstruct s'. Further, honest nodes in the protocol will decrypt some set of at least $\lfloor n/2 \rfloor + 1$ ciphertexts and use the decrypted outcome to reconstruct the secret. By simulation soundness of the NIZK and perfect correctness of PKE, it must be that all decrypted shares form an internally consistent sharing of the secret and thus no matter which set of $\lfloor n/2 \rfloor + 1$ ciphertexts are decrypted for reconstruction, the reconstructed outcome will be the same.

It remains to prove that if the dealer is honest, the reconstructed outcome will be the dealer's input for the VSS scheme. This holds in a straightforward manner because in this case, the shares encrypted in the ciphertexts will form an internally consistent sharing of the dealer's input secret. □

6.3.4 Non-Malleability

We define the following hybrid experiment.

Experiment $\mathsf{Hyb}^{\mathcal{A}}(1^\kappa, s)$**.** Experiment $\mathsf{Hyb}^{\mathcal{A}}(1^\kappa, s)$ is almost identical as the real non-malleability experiment $\mathsf{Expt}^{\mathcal{A}}(1^\kappa, s)$, except the following modifications: each honest node i calls $(\widetilde{\mathsf{crs}}_i, \tau_i) \leftarrow \mathsf{NIZK}.\widetilde{\mathsf{K}}$. The challenger will use $\widetilde{\mathsf{crs}}_i$ as part of node i's public key and retain the trapdoor τ_i. Whenever an honest node needs to compute a proof, instead of calling the real NIZK.P, the challenger instead calls the simulated prover $\mathsf{NIZK}.\widetilde{\mathsf{P}}$ using the trapdoors $\{\tau_i\}_{i\in\mathsf{Honest}}$ and generates a proof without using any witness.

Claim 6.5. *Suppose that the* NIZK *scheme satisfies zero-knowledge. For any* s*, the adversary* $\mathcal{A}$*'s view in* $\mathsf{Hyb}^{\mathcal{A}}(1^\kappa, s)$ *is computationally indistinguishable from its view in* $\mathsf{Expt}^{\mathcal{A}}(1^\kappa, s)$*.*

Proof. Straightforward from the zero-knowledge property of the NIZK scheme. □

Lemma 6.6. *Assume that the* RBC *scheme satisfies validity, the* NIZK *scheme satisfies simulation soundness, and the* PKE *scheme satisfies IND-CCA security and perfect correctness, then for any s, s', $\mathcal{A}$'s view in experiment $\mathsf{Hyb}^{\mathcal{A}}(1^\kappa, s')$ is computationally indistinguishable from its view in experiment $\mathsf{Hyb}^{\mathcal{A}}(1^\kappa, s)$.*

Proof. We define $\widetilde{\mathsf{Hyb}}_i^{\mathcal{A}}(1^\kappa, s)$ to be a hybrid experiment where in the challenge phase, the challenger computes a secret sharing of both s let $(s_1, \ldots s_n)$ denote the resulting shares. Now, for each honest user j,

- if j is among the first i honest users (assume honest users are ordered by their respective identities), CT_j is computed by encrypting s_j;
- else CT_j is computed by encrypting 0.

Henceforth, let $h = \lfloor n/2 \rfloor + 1$ denote the number of honest users (without loss of generality, it suffices to consider an adversary that always corrupts a fixed number of nodes such that $h = \lfloor n/2 \rfloor + 1$). Clearly $\widetilde{\mathsf{Hyb}}_h^{\mathcal{A}}(1^\kappa, s) = \mathsf{Hyb}^{\mathcal{A}}(1^\kappa, s)$.

Claim 6.7. *Suppose that the assumptions in Lemma 6.6 hold. For any s, $\mathcal{A}$'s view in $\widetilde{\mathsf{Hyb}}_h^{\mathcal{A}}(1^\kappa, s)$ is computationally indistinguishable from its view in $\widetilde{\mathsf{Hyb}}_0^{\mathcal{A}}(1^\kappa, s)$.*

Proof. Due to the hybrid argument, it suffices to prove that for any s, $\mathcal{A}$'s view in $\widetilde{\mathsf{Hyb}}_i^{\mathcal{A}}(1^\kappa, s)$ is computationally indistinguishable from its view in $\widetilde{\mathsf{Hyb}}_{i-1}^{\mathcal{A}}(1^\kappa, s)$ for any $i \in [1, h]$.

We construct a reduction $\mathcal{B}$ who can break IND-CCA security if there an adversary $\mathcal{A}$ that can distinguish $\widetilde{\mathsf{Hyb}}_i^{\mathcal{A}}(1^\kappa, s)$ and $\widetilde{\mathsf{Hyb}}_{i-1}^{\mathcal{A}}(1^\kappa, s)$. $\mathcal{B}$ follows how Hyb would interact with $\mathcal{A}$ except the following modifications:

- $\mathcal{B}$ obtains a challenge public-key epk^* from the IND-CCA challenger. Now $\mathcal{B}$ will embed epk^* into the i-th honest node's public key; for every other honest node $j \neq i$, $\mathcal{B}$ generates its epk_j and esk_j by calling the honest $\mathsf{PKE.K}(1^\kappa)$;
- For the challenge instance, $\mathcal{B}$ creates secret shares for s and let $(s_1, \ldots s_n)$ denote the resulting shares. $\mathcal{B}$ submits the challenge plaintexts s_i and 0 to the IND-CCA challenger, and obtains a set of ciphertexts CT_i^*. Now, $\mathcal{B}$ computes $\mathsf{CT}_j^* := \mathsf{Enc}(\mathsf{epk}_j, s_j)$ for $j < i$ and $j \in \mathsf{Honest}$; it computes $\mathsf{CT}_j^* := \mathsf{Enc}(\mathsf{epk}_j, 0)$ for $j > i$ and $j \in \mathsf{Honest}$. $\mathcal{B}$ now uses the ciphertext CT_j^* in place of the CT_j's in the challenge phase for each $j \in \mathsf{Honest}$ (note that at this moment, all NIZKs from honest nodes are being simulated without using any witness);
- For the challenge instance, honest nodes do not attempt to decrypt its own share at the end of Share before outputting "`sharing-succeeded`";
- For any non-challenge instances, if an honest node observes $\lfloor n/2 \rfloor + 1$ RBC instances output "`ok`", it needs to call the decryption algorithm to decrypt its own share; since $\mathcal{B}$ cannot decrypt honest node i's ciphertexts itself, it calls the IND-CCA challenger to help with decryption instead;
- Similarly, whenever $\mathcal{A}$ makes an $\mathsf{Extract}$ query for a non-challenge instance, the extractor $\mathcal{E}$ (simulated by $\mathcal{B}$ now) must perform decryption if certain checks pass (see description of $\mathcal{E}$ earlier) — $\mathcal{B}$ also calls the IND-CCA challenger to decrypt ciphertexts pertaining to i;
- Finally, $\mathcal{B}$ makes the same guess as what $\mathcal{A}$ outputs.

Let $(sid^*, \{\mathsf{CT}_j^*\}_{j\in[n]}, \pi^*)$ be the dealer u's input into RBC_0 during the challenge instance. We now argue the following claim:

Fact 6.8. *If the* NIZK *scheme is simulation sound,* PKE *is perfectly correct, and* RBC *satisfies validity, then except with negligible probability in the above experiment, $\mathcal{B}$ will never ask the IND-CCA challenger to decrypt CT_i^*.*

Proof. By the definition of the honest algorithm and validity of RBC, and also by the $\mathcal{E}$ definition, the challenger will only ask the IND-CCA challenger to decrypt CT_i^* iff the challenger has seen a proof π that verifies for the statement $(sid, \{\mathsf{pk}_j, \mathsf{CT}_j\}_{j\in[n]})$. Since the NIZK is simulation sound, except with negligible probability the statement $(sid, \{\mathsf{pk}_j, \mathsf{CT}_j\}_{j\in[n]})$ must be true. Now by the perfect correctness of PKE, if the sid in the above tuple is not equal to sid^*, then $\mathsf{CT}_i \neq \mathsf{CT}_i^*$. □

It is not difficult to see that if the IND-CCA challenger encrypted s_i, then $\mathcal{A}$'s view in the above experiment is identically distributed as $\widetilde{\mathsf{Hyb}}_i^{\mathcal{A}}(1^\kappa, s)$; otherwise it is identically distributed as $\widetilde{\mathsf{Hyb}}_{i-1}^{\mathcal{A}}(1^\kappa, s)$. Thus $\mathcal{A}$'s advantage in distinguishing $\widetilde{\mathsf{Hyb}}_i^{\mathcal{A}}(1^\kappa, s)$ and $\widetilde{\mathsf{Hyb}}_{i-1}^{\mathcal{A}}(1^\kappa, s)$ would directly translate to $\mathcal{B}$'s advantage in distinguishing which challenge plaintext the IND-CCA encrypted (with only negligible loss due to possible failure of the RBC's validity or the simulation soundness of the NIZK). □

Finally, Lemma 6.6 follows by observing that for any s, s', $\widetilde{\mathsf{Hyb}}_0^{\mathcal{A}}(1^\kappa, s)$ is identically distributed as $\widetilde{\mathsf{Hyb}}_0^{\mathcal{A}}(1^\kappa, s')$ due to the property of Shamir Secret Sharing. □

Theorem 6.9 (Non-malleability)**.** *Suppose that the* RBC *scheme satisfies validity, the* NIZK *scheme satisfies zero-knowledge and simulation soundness, and the* PKE *scheme satisfies IND-CCA security and perfect correctness. Then, for any non-uniform p.p.t. $\mathcal{A}$ satisfying the requirements of Definition 6.1, for any s and s' $\mathcal{A}$'s views in $\mathsf{Expt}^{\mathcal{A}}(1^\kappa, s)$ and in $\mathsf{Expt}^{\mathcal{A}}(1^\kappa, s')$ are computationally indistinguishable.*

Proof. The proof follows directly due to Claim 6.5 and Lemma 6.6. □

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 $\vec{\rho}^*_\ell$ 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 $\vec{\rho}$ 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 $\vec{\rho} \backslash \vec{\rho}^*_\ell$ denote all other randomness besides $\vec{\rho}^*_\ell$.

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 $\vec{\rho} \backslash \vec{\rho}^*_\ell$, there exist at least q fraction of choices for $\vec{\rho}^*_\ell$, 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 $L \in \mathcal{O}_{T^*}$ where $\mathcal{O}_{T^*}$ denotes the set of honest nodes that are online in round T^* (assuming that the start round of the ℓ-th instance is renamed to be 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 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 point of 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 is a subset of $[n]$ of size at least $\lfloor n/2 \rfloor + 1$, 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).

7.3 Proof

7.3.1 Definitions and Notation

Consider a pair $(\mathcal{A}, \mathcal{Z})$ that spawns multiple possible concurrent instances of LE after a shared PKI setup. Henceforth we define $\vec{\rho}_\ell^*$ to be the following collection of coins consumed in by honest nodes

in the starting round (i.e., round 0) of the ℓ-th LE instance:

$$\{c_{i,j} : i \in \mathsf{Honest}, j \in [n]\}$$

Good execution. Henceforth, we define a good execution to be one in which 1) no VSS instance violated T_{vss}-liveness or T_{vss}-validity; 2) no RBC instance violated T_{rbc}-consistency, validity, T_{rbc}-liveness (under an honest and initially online sender), or close termination; and 3) there is no signature from an honest node i in honest view that i never signed.

The following fact is easy to see.

Fact 7.1. *For every polynomial m, for every non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects 0.5-weak-synchrony and spawns m possibly concurrent LE instances, there exists a negligible function $\mathsf{negl}(\cdot)$ such that conditioning on the randomness $\{\vec{\rho}^*_\ell\}_{\ell \in [m]}$ being fixed to any string of appropriate length, the probability of a good execution is $1 - \mathsf{negl}(\kappa)$.*

Without loss of generality we may assume that $T_{\text{rbc}} \geq 2\Delta$ and $T_{\text{vss}} \geq 2\Delta$ — if not we can just let $T_{\text{rbc}} := 2\Delta$ or let $T_{\text{vss}} := 2\Delta$.

Fact 7.2. *In a good execution, the following holds for every LE instance where the relevant instance's starting round is renamed to round 0 for convenience: every honest node in $\mathcal{O}_0$ is considered as a qualified dealer by every node in $\mathcal{O}_{T_{\text{vss}}}$; thus, every node in $\mathcal{O}_{T_{\text{vss}}}$ will reliably broadcast a $(\texttt{qualified-set}, D)$ message where D includes every node in $\mathcal{O}_0$.*

Proof. Follows directly from T_{vss}-liveness of the VSS scheme. □

Fact 7.3. *In a good execution, the following holds for every LE instance where the relevant instance's starting round is renamed to round 0: if a node in $\mathcal{O}_{T_{\text{vss}}}$ reliably broadcasts some $(\texttt{qualified-set}, D)$, then in round $T_{\text{vss}} + T_{\text{rbc}}$, every node in $\mathcal{O}_{T_{\text{vss}}+T_{\text{rbc}}}$ will consider j as a candidate. Combining Fact 7.2 we may also conclude that every node $j \in \mathcal{O}_{T_{\text{vss}}+T_{\text{rbc}}}$ will send a $(\texttt{candidate-set}, S_j)$ message in round $T_{\text{vss}} + T_{\text{rbc}}$ where $\mathcal{O}_{T_{\text{vss}}} \subseteq S_j$.*

Proof. By T_{rbc}-liveness (under an honest and initially online sender) and validity of the RBC scheme, if $i \in \mathcal{O}_{T_{\text{vss}}}$ reliably broadcasts $(\texttt{qualified-set}, D)$, every node in $\mathcal{O}_{T_{\text{vss}}+T_{\text{rbc}}}$ will have received the output $(\texttt{qualified-set}, D)$ from the corresponding RBC instance at the beginning of round $T_{\text{vss}} + T_{\text{rbc}}$. If suffices to prove that in this round, every node in $\mathcal{O}_{T_{\text{vss}}+T_{\text{rbc}}}$ will consider everyone in D a qualified dealer. This follows from the close termination property of the VSS scheme and the fact that $T_{\text{rbc}} \geq 2\Delta$. □

Fact 7.4. *In a good execution, the following holds for every LE instance where the relevant instance's starting round is renamed to round 0: every honest node i's local set S^*_i will contain every node in $\mathcal{O}_{T_{\text{vss}}}$.*

Proof. Every honest node i decides S^*_i only when it has become happy with at least $\lfloor n/2 \rfloor + 1$ nodes. Thus at least one of these nodes, denoted j, must be from $\mathcal{O}_{T_{\text{vss}}+T_{\text{rbc}}}$.

Recall that i becomes happy with j if it has received a $(\texttt{candidate-set}, S_j)$ message where $S_j \subseteq [n]$ is of size at least $\lfloor n/2 \rfloor + 1$, and moreover everyone in S_j is considered a candidate by i. Since there is no signature forgery in a good execution, the $(\texttt{candidate-set}, S_j)$ message i received from j must be the one that j sent in round $T_{\text{vss}} + T_{\text{rbc}}$. By Fact 7.3, $\mathcal{O}_{T_{\text{vss}}} \subseteq S_j$. By the definition of "happy with", S_j must be a subset of node i's local candidate set when i is happy with j. □

7.3.2 Liveness

Henceforth, if an honest node i considers j as a candidate due to having output a message (`qualified-set`, D_j) from $\mathsf{RBC}[j]$, we say that i considers j a candidate with the coin-set D_j.

Fact 7.5. *In a good execution, the following holds for every* LE *instance where the relevant instance's starting round is renamed to round 0: if in some round r, some honest node i considers j a candidate with the coin-set D_j, then the following holds:*

> *In any round $r' \geq \max(r, T_{\text{vss}} + T_{\text{rbc}}) + T_{\text{vss}}$, for every instance $\mathsf{VSS}[d, j]$ where $d \in [n]$: every node in $\mathcal{O}_{r'}$ will have reconstructed some secret $c'_{d,j} \in \mathbb{F}$.*

Proof. If j is considered a candidate by an honest node i in round r, it means that in i's view in round r, $\mathsf{RBC}[j]$ has output (`qualified-set`, D_j) where $D_j \subseteq [n]$ is at least $\lfloor n/2 \rfloor + 1$ in size, and moreover, i considers everyone in D_j qualified. This means that for everyone $d \in D_j$ and every $u \in [n]$, i has output "`sharing succeeded`" from the instance $\mathsf{VSS}[d, u]$. The remainder of the proof follows directly from the T_{vss}-liveness and T_{vss}-validity property of VSS. □

Theorem 7.6 (Liveness)**.** *Assume that the* VSS *scheme satisfies T_{vss}-liveness and T_{vss}-validity; the* RBC *scheme satisfies T_{rbc}-consistency, T_{rbc}-liveness, validity, and close termination; and the signature scheme satisfies existential unforgeability under chosen-message attack. Then, the leader election protocol defined above satisfies $(2T_{\text{vss}} + T_{\text{rbc}})$-liveness.*

Proof. Let $t^* := 2T_{\text{vss}} + T_{\text{rbc}}$ and recall that $T_{\text{vss}} \geq 2\Delta$. Henceforth consider only good executions and ignore the negligible fraction of bad executions. The following statements hold for all LE instances in a good execution where the relevant instance's starting round is renamed to round 0. Due to Fact 7.3, every node in $\mathcal{O}_t$ for $t \geq t^*$ will have received a (`candidate-set`, _) message from every node in $\mathcal{O}_{T_{\text{vss}}+T_{\text{rbc}}}$ by the end of round t. Further, due to the close termination property of the RBC scheme and liveness of the VSS scheme, if some node $i \in \mathcal{O}_{T_{\text{vss}}+T_{\text{rbc}}}$ sent (`candidate-set`, S_i), then every node in $\mathcal{O}_t$ will consider every node in S_i as a candidate by the end of round t. Thus, for every $i \in \mathcal{O}_t$, its local set S_i^* will have been determined by the end of round t. Finally, by Fact 7.5, every node in $\mathcal{O}_t$ will have reconstructed the relevant secrets by the end of round t such that its LE protocol will have produced an output by the end of round t. □

7.3.3 Quality

Let $\mathcal{E}_{\text{rbc}}$ denote the extractor for the RBC scheme; and let $\mathcal{E}_{\text{vss}}$ be the extractor for the VSS scheme.

Experiment $\mathsf{Expt}^{(\mathcal{A},\mathcal{Z}),m,\ell}(1^\kappa)$. We define an experiment $\mathsf{Expt}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ that is basically the real-world execution but augmented with new random variables defined as below. Imagine that $(\mathcal{A}, \mathcal{Z})$ spawns m possibly concurrent LE instances. When the ℓ-instance has executed for $T_{\text{vss}} + T_{\text{rbc}}$ rounds, we define the following random variables related to the execution:

1. $\{\widetilde{c}_{i,j}\}_{i\in[n],j\in[n]}$: If i is honest, then $\widetilde{c}_{i,j}$ (where $j \in [n]$) denotes the random coin chosen by honest node i in round 0; else $\widetilde{c}_{i,j}$ is obtained as

$$\widetilde{c}_{i,j} := \mathcal{E}_{\text{vss}}(\{\mathsf{VSS.pk}_d\}_{d\in[n]}, \{\mathsf{VSS.sk}_d\}_{d\in\mathsf{Honest}}, \mathsf{Tr}(\mathsf{VSS}[i,j]))$$

   where $\mathsf{Tr}(\mathsf{VSS}[i, j])$ denotes the transcript of honest nodes in the initial T_{vss} rounds of instance $\mathsf{VSS}[i, j]$.

2. $\{\widetilde{D}_i\}_{i\in[n]}$: If i is honest and reliably broadcasted a `qualified-set` message in round T_{vss}, let $\widetilde{D}_i$ be the qualified set included by i in this message; if i is honest but reliably broadcasted $\perp$ in round T_{vss}; simply let $\widetilde{D}_i := \perp$. If i is corrupt, we define $\widetilde{D}_i$ as below:
   - let D^* be the outcome of applying $\mathcal{E}_{\text{rbc}}$ to the transcript of the honest nodes in the initial T_{rbc} rounds of instance $\mathsf{RBC}[i]$;
   - let $\widetilde{D}_i$ be D^* iff D^* is a subset of $[n]$ at least $\lfloor n/2 \rfloor + 1$ in size; else let $\widetilde{D}_i := \perp$.
3. $\{\widetilde{C}_i\}_{i\in[n]}$: henceforth we assume that for any $x \in \mathbb{F}$, $(\perp \cdot x) = (x \cdot \perp) = \perp$; we define $\{\widetilde{C}_i\}_{i\in[n]}$ as below:
$$\widetilde{C}_i := \begin{cases} \prod_{j\in\widetilde{D}_i} \widetilde{c}_{j,i} & \text{if } \widetilde{D}_i \neq \perp \\ \perp & \text{o.w.} \end{cases}$$
4. $Q \in \{0,1\}$: $Q := 1$ iff the following good event happens: for some $i \in \mathcal{O}_{T_{\text{vss}}}$,
$$\widetilde{C}_i = \max_j \{\widetilde{C}_j : j \in [n] \text{ and } \widetilde{C}_j \neq \perp\} \tag{1}$$

Experiment $\widetilde{\mathsf{Expt}}^{(\mathcal{A},\mathcal{Z}),m,\ell}_{\vec{\rho}^*_\ell}(1^\kappa)$. We use the notation $\widetilde{\mathsf{Expt}}^{(\mathcal{A},\mathcal{Z}),m,\ell}_{\vec{\rho}^*_\ell}$ to denote the same experiment as $\mathsf{Expt}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ except that we now fix the random string $\vec{\rho}^*_\ell$; all other random bits consumed by the experiment are chosen at random as before.

Hybrid experiment $\mathsf{Hyb}^{(\mathcal{A},\mathcal{Z}),m,\ell}(1^\kappa)$. Consider a hybrid experiment $\mathsf{Hyb}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ that is almost the same as $\mathsf{Expt}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ except the following modification: in the ℓ-th LE instance, every honest node i uses a canonical input denoted 0 in all VSS instances where i serves as the designated dealer; however, the value $\widetilde{c}_{i,j}$ is still chosen at random.

Lemma 7.7. *Assume that the VSS scheme satisfies non-malleability. Then, for every fixed polynomial m, there is some negligible function $\mathsf{negl}(\cdot)$, such that for every $\ell \leq m$, for every fixed choice of $\vec{\rho}^*_\ell$, the fraction of $\vec{\rho}\backslash\vec{\rho}^*_\ell$ that causes the event Q to differ in $\mathsf{Hyb}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ and in $\widetilde{\mathsf{Expt}}^{(\mathcal{A},\mathcal{Z}),m,\ell}_{\vec{\rho}^*_\ell}$ is upper bounded by $\mathsf{negl}(\kappa)$.*

Proof. Suppose the claim is not true, i.e., there exists some polynomial m and p, some particular choice of $\ell \leq m$ and $\vec{\rho}^*_\ell$ such that the fraction of $\vec{\rho}\backslash\vec{\rho}^*_\ell$ that causes the event Q to differ for $\mathsf{Hyb}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ and $\widetilde{\mathsf{Expt}}^{(\mathcal{A},\mathcal{Z}),m,\ell}_{\vec{\rho}^*_\ell}$ is at least $1/p(\kappa)$.

We now construct a non-uniform p.p.t. reduction $\mathcal{B}$ that leverages $(\mathcal{A},\mathcal{Z})$ to break the non-malleability of the VSS scheme. We consider an M-multi-challenge variant of the non-malleability game where in the challenge phase, the non-malleability adversary $\mathcal{B}$ may specify M simultaneous challenge instances in the challenge phase, and the challenger (denoted $\mathcal{C}$) will either share the secret vector $\vec{s}$ or $\vec{s}'$ where each coordinate of the vector is shared in one of the challenge instances. $\mathcal{B}$ is not allowed to invoke $\mathsf{Reconstruct}$ or send $\mathsf{Extract}$ queries for any of the challenge instances. Through a standard hybrid argument, it is not difficult to see that given a VSS scheme, if no efficient adversary can distinguish which challenge secret is shared in the single-challenge non-malleability game, then no efficient adversary can distinguish which challenge vector is shared in the multi-challenge non-malleability game.

$\mathcal{B}$ will now leverage $(\mathcal{A},\mathcal{Z})$ to break non-malleability in the M-multi-challenge game. Now consider the following experiment: $\mathcal{B}$ simulates the behavior of all honest nodes in the LE instances when interacting with $(\mathcal{A},\mathcal{Z})$ (who may spawn multiple instances of LE). $\mathcal{B}$ simply follows the

honest LE algorithm when simulating honest nodes, except that when $\mathcal{B}$ serves as a man-in-the-middle for between $(\mathcal{A}, \mathcal{Z})$ and $\mathcal{C}$ for simulating the VSS instances. Specifically, $\mathcal{B}$ calls the VSS's non-malleability secrecy challenger (denoted $\mathcal{C}$) to compute honest nodes' public keys and messages pertaining to all VSS instances and forwards them to $(\mathcal{A}, \mathcal{Z})$; further, $\mathcal{B}$ forwards to $\mathcal{C}$ the part of all messages and public keys received that pertain to the VSS instances.

When the ℓ-th LE instances starts, let Γ denote all honest-dealer VSS instances contained in the i-th LE instance — this will be the challenge VSS instances in the non-malleability game. $\mathcal{B}$ submits two challenge vectors to $\mathcal{C}$, $\vec{0}$ and $\vec{\rho}^*_\ell$ where $(\ell, \vec{\rho}^*_\ell)$ denotes the bad choice that causes Lemma 7.7 to fail. $\mathcal{C}$ secret shares one of challenge vectors. For VSS instances contained in all other LE instances, $\mathcal{B}$ still simulates them as before by forwarding between $\mathcal{C}$ and $(\mathcal{A}, \mathcal{Z})$. Finally, the experiment stops when it has completed $(T_{\text{rbc}} + T_{\text{vss}})$ rounds of the ℓ-th LE instance. At this point, $\mathcal{B}$ may additionally make some Extract queries to the VSS challenger $\mathcal{C}$ (for corrupt-dealer VSS instances) to evaluate whether the good event Q has happened. $\mathcal{B}$ outputs 1 if Q is true and outputs 0 otherwise.

Fact 7.8. *When interacting with $\mathcal{C}$, $\mathcal{B}$ never makes a* Reconstruct *or* Extract *query on the challenge* VSS *instances.*

It is not difficult to see that if $\mathcal{C}$ chooses the challenge vector $\vec{0}$, the above experiment is identically distributed as in $\mathsf{Hyb}^{(\mathcal{A},\mathcal{Z}),m,\ell}$; otherwise it is identically distributed as $\widetilde{\mathsf{Expt}}^{(\mathcal{A},\mathcal{Z}),m,\ell}_{\vec{\rho}^*_\ell}$. The proof follows by observing that if the fraction of $\vec{\rho}\backslash\vec{\rho}^*_\ell$ that causes the event Q to differ for $\mathsf{Hyb}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ and $\widetilde{\mathsf{Expt}}^{(\mathcal{A},\mathcal{Z}),m,\ell}_{\vec{\rho}^*_\ell}$ is at least $1/p(\kappa)$, then the probability that $\mathcal{B}$ outputs 1 in $\mathsf{Hyb}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ and $\widetilde{\mathsf{Expt}}^{(\mathcal{A},\mathcal{Z}),m,\ell}_{\vec{\rho}^*_\ell}$ differs by at least $1/p$ too.

□

Lemma 7.9. *Suppose that the* VSS *scheme satisfies T_{vss}-liveness, T_{vss}-validity, and non-malleability under 0.5-weak-synchrony and that the* RBC *scheme satisfies T_{rbc}-consistency, T_{rbc}-liveness, and validity. For any polynomial function $m(\cdot)$, for any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects 0.5-weak-synchrony and spawns $m(\kappa)$* LE *instances, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all $\kappa \in \mathbb{N}$, for any $1 \leq \ell \leq m$, except for $\mathsf{negl}(\kappa)$ fraction of $\vec{\rho}\backslash\vec{\rho}^*_\ell$, the good event Q holds for at least $\frac{1}{2} + \frac{1}{9n}$ fraction of the choices for $\vec{\rho}^*_\ell$ in $\mathsf{Expt}^{(\mathcal{A},\mathcal{Z}),m,\ell}$.*

Proof. Follows directly from the following Claim 7.10 and Claim 7.11.

Claim 7.10. *Suppose that the* VSS *scheme satisfies non-malleability. Assume that $(\mathcal{A}, \mathcal{Z})$ is non-uniform p.p.t., respects 0.5-weak-synchrony, and spawns m possibly concurrent* LE *instances. Suppose the* VSS *scheme employed satisfies T_{vss}-validity and non-malleability under 0.5-weak-synchrony. Moreover, suppose that there is some $q(\kappa, n)$ and some negligible function $\nu(\cdot)$ such that for any $\ell \leq m$, for $1 - \nu(\kappa)$ fraction of $\vec{\rho}\backslash\vec{\rho}^*_\ell$, the good event Q holds for at least $q(\kappa, n)$ fraction of the $\vec{\rho}^*_\ell$ in $\mathsf{Hyb}^{(\mathcal{A},\mathcal{Z}),m,\ell}$. Then there is some negligible function $\nu'(\cdot)$ and $\nu''(\cdot)$ such that for any $\ell \leq m$, for $1 - \nu'(\kappa)$ fraction of $\vec{\rho}\backslash\vec{\rho}^*_\ell$, the good event Q holds for at least $q(\kappa, n) - \nu''(\kappa)$ fraction of the $\vec{\rho}^*_\ell$ in $\mathsf{Expt}^{(\mathcal{A},\mathcal{Z}),m,\ell}$.*

Proof. The proof follows from Lemma 7.7 as follows. Let k_1 be the length of $\vec{\rho}^*_\ell$ and let k_2 be the length of $\vec{\rho}\backslash\vec{\rho}^*_\ell$. If the above is true, then the number of $(\vec{\rho}^*_\ell, \vec{\rho}\backslash\vec{\rho}^*_\ell)$ combinations that cause Q to differ for $\mathsf{Hyb}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ and $\mathsf{Expt}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ is upper bounded by $2^{k_1+k_2} \cdot \mathsf{negl}(\kappa)$.

On the other hand, assume that the claim we are proving does not hold, i.e., there is some polynomial function $p(\cdot)$, and $p'(\cdot)$ such that for $1/p(\kappa)$ fraction of $\vec{\rho}\backslash\vec{\rho}^*_i$, the good event Q holds for only less than $q(\kappa, n) - 1/p'(\kappa)$ fraction of the $\vec{\rho}^*_i$ in $\mathsf{Expt}^{(\mathcal{A},\mathcal{Z}),m,\ell}$. In this case, the number of

$(\vec{\rho}^*_\ell,\ \vec{\rho}\backslash\vec{\rho}^*_\ell)$ combinations that cause Q to differ for $\mathsf{Hyb}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ and $\mathsf{Expt}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ is lower bounded by

$$(1/p - \mathsf{negl}) \cdot 2^{k_2} \cdot (1/p') \cdot 2^{k_1} = 2^{k_1+k_2} \cdot (1/\mathsf{poly}(\kappa))$$

Thus we have reached a contradiction. □

Claim 7.11. *Suppose that the* VSS *scheme satisfies T_{vss}-liveness, T_{vss}-validity, and non-malleability under 0.5-weak-synchrony. For every polynomial m and every non-uniform p.p.t. $(\mathcal{A},\mathcal{Z})$ that respects 0.5-weak-synchrony and spawns m* LE *instances sharing the same PKI, there is a negligible function $\mathsf{negl}(\cdot)$ such that for every $\ell \le m$, for all but $\mathsf{negl}(\kappa)$ fraction of $\vec{\rho}\backslash\vec{\rho}^*_\ell$, the good event Q holds for $\frac{1}{2} + \frac{1}{8n}$ fraction of $\vec{\rho}^*_\ell$ in $\mathsf{Hyb}^{(\mathcal{A},\mathcal{Z}),m,\ell}$.*

Proof. In Hyb, note that the honest coins $\vec{\rho}^*_\ell$, i.e., the set of values $\{\widetilde{c}_{i,j}\}_{i\in\mathsf{Honest},j\in[n]}$ in the ℓ-th instance are independent of $(\mathcal{A},\mathcal{Z})$'s view in the experiment, independent of whether the execution is "good" defined in Section 7.3.1, and independent of the choice of $\{\widetilde{c}_{i,j}\}_{i\in\mathsf{Crupt},j\in[n]} \cup \{\widetilde{D}_i\}_{i\in[n]}$.

Define the following event denoted X: for every $i \in \mathcal{O}_{T_{\text{vss}}}$, $\widetilde{D}_i$ is a subset of $[n]$ at least $\lfloor n/2 \rfloor + 1$ in size; moreover, for every $j \in \widetilde{D}_i$ that is corrupt, it holds that $\widetilde{c}_{j,u} \neq \perp$ for every $u \in [n]$. Clearly, the event X is independent of $\vec{\rho}^*_\ell$ in Hyb.

Now, for every choice of every $\vec{\rho}\backslash\vec{\rho}^*_\ell$ such that X is true and moreover the execution is good by the definition in Section 7.3.1, it is not hard to see that the good event Q holds for $0.5 + \frac{1}{3n}$ fraction of the $\vec{\rho}^*_\ell$ choices. To conclude the proof, we show that X is true for all but a negligible fraction of $\vec{\rho}\backslash\vec{\rho}^*_\ell$. Since the execution is good except with negligible probability over the choice of $\vec{\rho}\backslash\vec{\rho}^*_\ell$, it suffices to show that X is true for all good executions of Hyb. Observe that in a good execution of Hyb, Fact 7.2 also (note that earlier the proof of Fact 7.2 was for the original experiment $\mathsf{Expt}^{(\mathcal{A},\mathcal{Z}),m,\ell}$ but the same proof holds for Hyb too). Thus in a good execution of Hyb, in the ℓ-th LE instance, every node in $\mathcal{O}_{T_{\text{vss}}}$ will reliably broadcast a (`qualified-set`, D) message where D includes every node in $\mathcal{O}_0$. Further, in a good execution, if D is multicast by an honest node, for every corrupt node $j \in D$, for every $u \in [n]$, $\widetilde{c}_{j,u} \neq \perp$ by the validity and liveness of VSS. □

□

Theorem 7.12 (Quality)**.** *Suppose that the* VSS *scheme satisfies T_{vss}-liveness, T_{vss}-validity, and non-malleability under 0.5-weak-synchrony; that the* RBC *scheme satisfies T_{rbc}-consistency, T_{rbc}-liveness, validity, and close termination; and that the signature scheme employed satisfies existential unforgeability under chosen-message attack; then, the above* LE *protocol satisfies $(T_{\text{vss}}, 1/2)$-quality under 0.5-weak-synchrony.*

Proof. We start by proving the following claim.

Claim 7.13. *In a good execution, if the event Q is true, then the following holds: let $i^* \in \mathcal{O}_{T_{\text{vss}}}$ be the honest node for which Equation (1) holds; then, every honest node's output must be i^*.*

Proof. Observe that in any good execution, Fact 7.4 holds. Further, in a good execution, if an honest node successfully evaluated a node i's charisma in the execution, then the charisma value must agree with $\widetilde{C}_i$. Now, if the good event Q also holds in this execution, and let $i^* \in \mathcal{O}_{T_{\text{vss}}}$ be the honest node for which Equation (1) holds, then every honest node's output must be i^*. □

By Fact 7.1, we know that for every choice of $\vec{\rho}^*_\ell$, all but a negligible fraction of the choices of $\vec{\rho}\backslash\vec{\rho}^*_\ell$ lead to a good execution. Thus the total number of $(\vec{\rho}^*_\ell,\ \vec{\rho}\backslash\vec{\rho}^*_\ell)$ combinations that lead to a bad execution must be at most $\mathsf{negl}(\kappa) \cdot 2^{k_1+k_2}$ where k_1 denotes the length of $\vec{\rho}^*_\ell$ and k_2 denotes the length of $\vec{\rho}\backslash\vec{\rho}^*_\ell$. Now suppose for the sake of contradiction that the theorem is not true, i.e., there

exists some $1/\mathsf{poly}(\kappa)$ fraction of $\vec{\rho}\backslash\vec{\rho}^*_\ell$ such that less than $1/2$ of the choices of $\vec{\rho}^*_\ell$ lead to the good event Q. Due to Lemma 7.9 and the above claim, it must be that for each of $1/\mathsf{poly}(\kappa) - \mathsf{negl}(\kappa)$ fraction of the choices of $\vec{\rho}\backslash\vec{\rho}^*_\ell$, there exist $1/\mathsf{poly}'(\kappa)$ fraction of $\vec{\rho}^*_\ell$ that cause the execution to be bad. In this case, we know that the number of $(\vec{\rho}^*_\ell, \vec{\rho}\backslash\vec{\rho}^*_\ell)$ combinations that lead to a bad execution must be at least $2^{k_1+k_2} \cdot (1/\mathsf{poly}''(\kappa))$. Thus we have reached a contradiction. □

8 Byzantine Agreement

8.1 Protocol Description

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[7], are signed (using each node's ssk) and tagged with the purported 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.

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

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.

Remark 8.1. In the above scheme, a node would continue participating in the protocol forever after producing an output. We devise a termination technique in Section E.

8.2 Proof

Henceforth we ignore the negligible fraction of bad executions in which there exists a forged signature in honest view. We refer the remaining set of (all but negligible fraction of) executions as good executions. Throughout the proof, we shall assume by default that $(\mathcal{A}, \mathcal{Z})$ is non-uniform p.p.t. and respects 0.5-weak-synchrony.

8.2.1 Consistency

Lemma 8.2 (Consistency within the same epoch)**.** *In a good execution, if for some e, an honest node multicasts $(\texttt{commit}, e, b)$, then there cannot be an epoch-e commit evidence for $1-b$ ever in honest view.*

Proof. Henceforth for convenience, an epoch-$(e-1)$ commit evidence for $1-b$ is said to be an epoch-e pseudo-proposal for $1-b$.

Suppose for the sake of contradiction that the above is not true, i.e., some honest node i multicasts $(\texttt{commit}, e, b)$ and there is at some point an epoch-e commit evidence for $1-b$ too in honest view. At the beginning of the commit round of epoch e, node i must have seen an epoch-e commit evidence for b, and must not have observed an epoch-e valid proposal or pseudo-proposal for $1-b$. On the other hand, since there is an epoch-e commit evidence for $1-b$ in honest view, some honest node that is online in the prepare round of epoch e must have sent an epoch-e prepare message for $1-b$ — this node must have observed a valid epoch-e proposal or pseudo-proposal for $1-b$ at the beginning of the prepare round of epoch e. Due to our strong message delivery assumption (Assumption 2), node i must have observed either a valid epoch-e proposal or pseudo-proposal for $1-b$ at the beginning of the commit round of epoch e. Thus we have reached a contradiction. □

Lemma 8.3 (Consistency across epochs)**.** *In a good execution, if there is some $e \in \mathbb{N}$ and $b \in \{0,1\}$ such that an honest node ever sees a set of $\lfloor n/2 \rfloor + 1$ $(\texttt{commit}, e, b)$ messages with valid signatures from distinct nodes, then for $e' \geq e$, there cannot be an epoch-e' commit evidence for $1-b$ ever in honest view.*

Proof. By Lemma 8.2, there cannot be an epoch-e commit evidence for $1-b$ in honest view.

Now, some honest node that is online in the commit round of epoch e must have sent an epoch-e commit message for the bit b — this node must have observed an epoch-e commit evidence for b at the beginning of the commit round of epoch e. Thus every honest node that is online in the complain round of epoch e must have observed the same commit evidence by our strong message delivery assumption (Assumption 2), and thus will not send an epoch-e complain message. Consider the prepare round of epoch $e+1$: every honest node that is online in this round 1) must have observed an epoch-e commit evidence for b; 2) must not have observed $\lfloor n/2 \rfloor + 1$ number of epoch-e complain messages from distinct nodes; and 3) must not have observed any epoch-e commit evidence for $1-b$ by Lemma 8.2. Thus every honest node that is online in the prepare round of epoch $e+1$ will send an epoch-$(e+1)$ prepare message for b, and will not send an epoch-$(e+1)$ prepare message for $1-b$. Thus there will not be any epoch-$(e+1)$ commit evidence for $1-b$ in honest view.

Now, every honest node that is online in the commit round of epoch $e+1$ must have seen $\lfloor n/2 \rfloor + 1$ number of epoch-$(e+1)$ commit messages for b. By induction, we can thus prove the statement for all $e' \geq e$. □

Theorem 8.4 (Consistency)**.** *In a good execution, if an honest node outputs b and another honest node outputs b', it must be that $b = b'$.*

Proof. If an honest node outputs b due to having received $\lfloor n/2 \rfloor + 1$ number of valid epoch-e commit messages for b from distinct nodes, we say that the node *finalizes to b due to epoch e*.

It is not hard to see that if there is no epoch-e commit evidence for b in honest view, then no honest node will finalize to b due to epoch e. Now, let e^* be the smallest epoch such that some honest node finalizes to some bit due to epoch e^*, and let b^* denote this finalized bit. By Lemma 8.3 it holds that for any $e \geq e^*$, there cannot be an epoch-e commit evidence for $1-b$ in honest view. Thus no honest node will finalize to $1-b$ due to epoch e. □

8.2.2 Validity

Theorem 8.5 (Validity)**.** *In a good execution in which the sender is honest and online in the initial round, all honest nodes must output the sender's input bit.*

Proof. If the sender (i.e., node 1) is honest and online in the starting round, every honest node that is online in the prepare round of the first epoch will multicast a prepare message for the sender's input bit b. Thus every honest node that is online in the commit round of the first epoch will multicast a commit message for the sender's input b. Thus, every honest node that is online in the complain round of the first epoch will have observed a collection of $\lfloor n/2 \rfloor + 1$ number of $(\texttt{commit}, e = 1, b)$ messages from distinct nodes and will have output the bit b. The remainder of the proof follows from Theorem 8.3. □

8.2.3 Liveness

Henceforth we say that a message m is *in honest and online view* in round r, if for every $t \geq r$, every node in $\mathcal{O}_t$ must have observed m at the beginning of t.

Fact 8.6. *At the beginning of every epoch $e > 1$, either there is an epoch-$(e-1)$ commit evidence in honest and online view, or there are $\lfloor n/2 \rfloor + 1$ epoch-$(e-1)$ complaints from distinct nodes in honest and online view.*

Proof. Let r be the complain round of epoch $e-1$ and let t be the starting round of epoch e. At the beginning of round r,

- either some node in $\mathcal{O}_r$ has seen an epoch-$(e-1)$ commit evidence for some $b \in \{0,1\}$ in which case the commit evidence will appear in honest and online view in the starting round of epoch e by our strong message delivery assumption (Assumption 2);
- or no node in $\mathcal{O}_r$ has seen an epoch-$(e-1)$ commit evidence in which case all nodes in $\mathcal{O}_r$ will multicast an epoch-e complaint in round r. In this case, $\lfloor n/2 \rfloor + 1$ number of epoch-$(e-1)$ complaints from distinct nodes will appear in honest and online view in the starting round of epoch e.

□

Theorem 8.7 (Liveness)**.** *Let T be a random variable representing the first round such that every node that is honest and online in round $t \geq T$ will have produced an output by the end of round t. For any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects 0.5-weak-synchrony, we have that there is some suitable constant C such that for every $\lambda \in \mathbb{N}$, except for $\exp(-C\lambda)$ fraction of the executions, $T \leq \lambda \cdot (T_{\text{le}} + 4\Delta)$.*

Proof. Consider an execution during which no honest node observes a forged signature from an honest node. Suppose that one of the following is true for some epoch $e > 1$, we say that epoch e is a lucky epoch: let r denote the starting round of epoch e,

- either $\lfloor n/2 \rfloor + 1$ epoch-$(e-1)$ complaints from distinct nodes are in honest and online view in round r; or
- there is some bit $b \in \{0,1\}$ such that 1) an epoch-$(e-1)$ commit evidence for b is in honest and online view in round r, 2) the proposer of epoch e, denoted i^*, is honest and online in the round (within epoch e) in which it makes the proposal; and 3) i^* proposes the same bit b too.

If an epoch $e > 1$ is lucky, everyone in $\mathcal{O}_e^{\text{prepare}}$ will multicast $(\texttt{prepare}, e, b)$, and will not multicast $(\texttt{prepare}, e, 1-b)$ where $\mathcal{O}_e^{\text{prepare}}$ denotes the honest and online set in the prepare round of epoch e (and the meaning of similar notations below can be likewise inferred). Thus we have that 1) an epoch-e commit evidence for b will appear in honest and online view at the beginning of epoch $e+1$; and 2) there cannot be an epoch-e commit evidence for $1-b$ ever in honest view. Note, however, that nodes in $\mathcal{O}_e^{\text{commit}}$ may not multicast $(\texttt{commit}, e, b)$ since it might have observed an epoch-$(e-1)$ commit evidence for $1-b$ at the beginning of the commit round of epoch e.

Now, suppose that epoch $e > 1$ is lucky and let b be the bit proposed in epoch e by the honest epoch-e proposer. Suppose that additionally, the proposer of epoch $e+1$ is honest and online in the round in which it makes the epoch-$(e+1)$ proposal, and moreover it happens to propose the bit b in epoch $e+1$ too. In this case, it is not difficult to see that epoch $e+1$ is lucky too, and moreover, since there is no epoch-e commit evidence for $1-b$ in honest view, it must be that everyone in $\mathcal{O}_{e+1}^{\text{commit}}$ will multicast $(\texttt{commit}, e+1, b)$ in the commit round of epoch $e+1$. Thus for every honest node i that is online in round t where t is at least Δ rounds after the commit round of epoch $e+1$, i must have produced an output at the beginning of round t.

Finally, to prove the theorem, it suffices to observe the following:

Claim 8.8. *Suppose that the* LE *scheme satisfies* $(T_{\text{le}}', 1/2)$*-quality. Then the probability that there exists two consecutive lucky epochs in* $\lambda(T_{\text{le}} + 4\Delta)$ *rounds is at least* $1 - \exp(-C\lambda) - \nu(\kappa)$ *for some appropriate constant* C *and appropriate negligible function* $\nu(\cdot)$*.*

Proof. First, imagine that the LE scheme satisfied a stronger property where the negligible function $\mathsf{negl}(\cdot)$ in the quality definition is 0 instead. In other words, for every ℓ, in the ℓ-th LE instance, conditioned on all randomness consumed by the experiment upto the starting round of the ℓ-th LE instance (inclusive) but not including $\vec{\rho}_\ell^*$, there are at least $1/2$ choices of ρ_ℓ^* that would cause the good event Q to happen for the ℓ-th instance. In this case, clearly conditioned on all randomness consumed by the experiment upto the starting round of the ℓ-th LE instance (inclusive) but not including $\vec{\rho}_\ell^*$, there are at least $1/4$ choices of (ρ_ℓ^*, b^*) that would cause the ℓ-th epoch to be lucky where b^* is the random bit to propose chosen by the resulting honest proposer of epoch ℓ. Thus after $\lambda(T_{\text{le}} + 4\Delta)$ rounds the probability that there do not exist two consecutive lucky epochs is upper bounded by $\exp(-C\lambda)$.

In reality, our LE scheme satisfies only a weaker property. There is a negligible probability the randomness chosen by the experiment $\vec{\rho}$ will be bad in the following sense: there exists some ℓ such that for the particular the choice of $\vec{\rho} \backslash \vec{\rho}_\ell^*$, less than $1/2$ of the choices for $\vec{\rho}^*$ would lead to the good event Q in the ℓ-th LE instance. The claim then follows by union bound. □

□

9 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 ensure that honest nodes who drop offline not only do not risk inconsistency from the rest of the network, but also do not risk losing the privacy of their respective inputs.

Of course, in a weakly synchronous environment, if we would like online nodes to finish the protocol 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 $\lfloor n/2 \rfloor + 1$ number of honest inputs be included in the computation; and
2. for every honest node that remains online during the protocol, their inputs must be included (our formal definitions later are a slight generalization: we require that all honest nodes who remain online till a specific round during the protocol must be able to get their inputs incorporated).

Note that the second requirement makes sure that our security requirement is strictly stronger than SFE protocols secure under classical synchrony, i.e., any SFE protocol secure under our definition must be secure under classical synchrony assuming honest majority.

9.1 Definition

We consider secure function evaluation (SFE) protocols under χ-weak-synchrony, assuming the existence of a PKI. Our definition is a UC-style, simulation-based notion assuming a weakly synchronous network model. In the following, we assume that the real-world adversary $\mathcal{A}$ (or the ideal-world adversary $\mathcal{S}$) can communicate at any time and arbitrarily with the environment $\mathcal{Z}$.

Real-world experiment $\mathsf{Real}^{\mathcal{A},\mathcal{Z},\Pi}(1^\kappa)$**.**

- **Corrupt.** Let $\mathsf{Crupt} \leftarrow \mathcal{A}(1^\kappa)$ where $\mathsf{Crupt} \subseteq [n]$. Henceforth let $\mathsf{Honest} := [n] \backslash \mathsf{Crupt}$.
- **PKI Setup.** For every $i \in \mathsf{Honest}$, let $(\mathsf{pk}_i, \mathsf{sk}_i)$ be a public- and secret-key pair output by the honest key generation algorithm. Let $\{\mathsf{pk}_i\}_{i \in \mathsf{Crupt}} \leftarrow \mathcal{A}(\{\mathsf{pk}_i\}_{i \in \mathsf{Honest}})$.
- **Input.** $\mathcal{Z}$ chooses honest nodes' inputs $\{x_i\}_{i \in \mathsf{Honest}}$.
- **Execute.** Honest nodes interact with $\mathcal{A}$ who controls the corrupt nodes. $\mathcal{A}$ can decide the message delivery schedule and the honest and online set $\mathcal{O}_r$ for each round r (possibly subject to certain constraints, e.g., χ-weak-synchrony).
- **Output.** When an honest node produces an output, the output is sent to $\mathcal{Z}$.

Ideal-world experiment $\mathsf{Ideal}^{\mathcal{S},\mathcal{Z},T}(1^\kappa)$**.**

- **Corrupt.** Let $\mathsf{Crupt} \leftarrow \mathcal{S}(1^\kappa, z)$ where $\mathsf{Crupt} \subseteq [n]$. Henceforth let $\mathsf{Honest} := [n] \backslash \mathsf{Crupt}$.
- **Input.** $\mathcal{Z}$ chooses honest nodes' inputs $\{x_i\}_{i \in \mathsf{Honest}}$.
- **Honest and online sets.** In each round r, $\mathcal{S}$ must output a set $\mathcal{O}_r \subseteq \mathsf{Honest}$ such that $|\mathcal{O}_r| \geq \lfloor \chi n \rfloor + 1$;
- **Effective inputs.** In some round $r^* \leq T$ during the ideal execution, $\mathcal{S}$ additionally outputs $(\{\widetilde{x}_i\}_{i \in \mathsf{Crupt}}, \mathfrak{I})$ where it is required that $\mathfrak{I} \subseteq [n]$, $|\mathfrak{I} \cap \mathsf{Honest}| \geq \lfloor \chi n \rfloor + 1$, and moreover $\mathfrak{I}$ includes any honest node that has always remained online so far. Intuitively, this means that among the honest nodes, the subset $(\mathfrak{I} \cap \mathsf{Honest})$'s inputs will be taken into account. Further, in this step, the ideal-world adversary $\mathcal{S}$ has an opportunity to replace corrupt nodes' inputs too.

  Now, define the ideal-world outcome $\{y_i\}_{i \in [n]} := f(\{x_i^*\}_{i \in [n]})$ where x_i^* is defined as:

$$x_i^* := \begin{cases} x_i & \text{if } i \in \mathfrak{I} \cap \mathsf{Honest} \\ \bot & \text{if } i \in \mathsf{Honest} \backslash \mathfrak{I} \\ \widetilde{x}_i & \text{o.w.} \end{cases}$$

- **Output.** For each round $r \geq r^*$, $\mathcal{S}$ must also output an honest subset $L_r \subseteq \mathsf{Honest}$ who are supposed to receive an output in this round. $\mathcal{S}$ must respect the following constraints:

  1. each honest node is included in no more than one set L_r;
  2. if for some $r \geq T$, some honest node $i \in \mathcal{O}_r$ but i has not been included in any L_t for $t < r$, then i must be included in L_r.

  Now honest node $i \in L_r$ will output y_i to $\mathcal{Z}$ in round r.

Definition 9.1 (Weakly synchronous SFE)**.** Let $T(\kappa, n, \Delta)$ be a polynomial function in the stated parameters. We say that a protocol Π securely evaluates the function f in T time under χ-weak-synchrony iff for every real-world non-uniform p.p.t.$\mathcal{A}$ that respects χ-weak-synchrony, there exists an ideal-world non-uniform p.p.t. adversary $\mathcal{S}$, such that for every non-uniform p.p.t. environment $\mathcal{Z}$, its views in the experiments $\mathsf{Real}^{\mathcal{A},\mathcal{Z},\Pi}(1^\kappa)$ and $\mathsf{Ideal}^{\mathcal{S},\mathcal{Z},T}(1^\kappa)$ are computationally indistinguishable.

Note that in the above, we use a single definition to capture the *correctness*, *privacy*, and *liveness* requirements of the SFE task.

9.2 Construction

We show how to construct a 0.5-weakly-synchronous SFE protocol that allows n nodes to securely evaluate any polynomial function $f(x_1, \ldots, x_n)$ over their respective inputs. As mentioned, to circumvent the well-known 1/3 lower bound in the plain, authenticated-channels model, our protocol assumes the existence of a PKI.

For simplicity, we describe our protocol assuming that all nodes want to receive the same output, i.e., $y_1 = y_2 = \ldots = y_n$ in the definition of the ideal experiment $\mathsf{Ideal}^{\mathcal{S},\mathcal{Z},T}(1^\kappa)$ earlier. Later in Remark 9.3, we argue that it is not difficult to extend our protocol to the case when different nodes want to receive different outputs.

We will leverage the following building blocks:

1. a 0.5-weakly-synchronous Byzantine Agreement (BA) protocol whose liveness parameter is denoted T_{ba} (see Sections 3.3 and 8);
2. a $(\lfloor n/2 \rfloor + 1)$-out-of-$n$ Threshold Multi-Key Fully Homomorphic Encryption (TMFHE) scheme [4, 17] denoted $\mathsf{TMFHE} := (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{PartDec}, \mathsf{FinDec})$ — see Section C.2 for a formal definition of TMFHE; and
3. an honest-majority multi-CRS non-interactive zero-knowledge proof (NIZK) system denoted $\mathsf{NIZK} := (\mathsf{K}, \mathsf{P}, \mathsf{V})$ — see Section C.1 for a formal definition;

We have explained the intuition behind our construction earlier in Section 2.5. We now give a formal descryption of our protocol.

- **PKI setup:** every node $i \in [n]$ calls BA's key generation algorithm and obtains $(\mathsf{BA.pk}_i, \mathsf{BA.sk}_i)$, calls $\mathsf{NIZK.K}$ and obtains crs_i. calls $\mathsf{TMFHE.Gen}$ and obtains $(\mathsf{TMFHE.pk}_i, \mathsf{TMFHE.sk}_i)$. The node now registers the tuple $(\mathsf{BA.pk}_i, \mathsf{crs}_i, \mathsf{TMFHE.pk}_i)$ with the PKI.
- **Round** 0**:** Each node $i \in [n]$ does the following (else do nothing):
  1. call $\mathsf{CT}_i \leftarrow \mathsf{TMFHE.Enc}(\{\mathsf{TMFHE.pk}_j\}_{j\in[n]}, x_i)$ where x_i denotes its input — henceforth let ρ_i be the random coins consumed by $\mathsf{TMFHE.Enc}$;
  2. call $\pi_i \leftarrow \mathsf{NIZK.P}(\{\mathsf{crs}_j\}_{j\in[n]}, \mathsf{stmt}, w)$ using the statement $\mathsf{stmt} := (\{\mathsf{TMFHE.pk}_j\}_{j\in[n]}, \mathsf{CT}_i, i)$ and the witness $w := (x_i, \rho_i)$, and (stmt, w) is considered to be in the language L iff CT_i is a correct encryption corresponding to node i of the plaintext x_i under the public keys $\{\mathsf{TMFHE.pk}_j\}_{j\in[n]}$ and using randomness ρ_i;
  3. call BA to broadcast the tuple (CT_i, π_i) — henceforth the BA instance in this step where node i is the designated sender is denoted $\mathsf{BA}[i]$;
- **Any round:** if all $\{\mathsf{BA}[j]\}_{j\in[n]}$ instances have produced an output, a node i does the following:

1. for each $j \in [n]$, let (CT_j, π_j) be the output from $\mathsf{BA}[j]$: if $\mathsf{NIZK.V}(\{\mathsf{crs}_j\}_{j\in[n]}, (\{\mathsf{TMFHE.pk}_j\}_{j\in[n]}, \mathsf{CT}_j, j), \pi_j)$ outputs 1, add j to the "effective-input" set $\mathfrak{I}$ (which was initially empty);
2. note that at this moment, the variables (CT_j, π_j) for $j \in [n]$ and $\mathfrak{I}$ must be well-defined;

- **Round T_{ba}:** if all $\{\mathsf{BA}[j]\}_{j\in[n]}$ instances have produced an output (and thus the relevant variables are now well-defined), a node i does the following (else it does nothing):

  1. let $\widetilde{\mathsf{CT}} \leftarrow \mathsf{TMFHE.Eval}(f^{\mathfrak{I}}, \{\mathsf{CT}_j\}_{j\in\mathfrak{I}})$ where $f^{\mathfrak{I}}$ the restricted version of f where for every $i \notin \mathfrak{I}$ its input is set to a default value $\perp$;
  2. let $p_i \leftarrow \mathsf{TMFHE.PartDec}(\mathsf{TMFHE.sk}_i, \widetilde{\mathsf{CT}})$;
  3. let $\pi'_i \leftarrow \mathsf{NIZK.P}((\{\mathsf{crs}_j\}_{j\in[n]}, \mathsf{stmt}, w)$ using the statement $\mathsf{stmt} := (p_i,\ f,\ \mathfrak{I},\ \{\mathsf{CT}_j\}_{j\in[n]}, \{\mathsf{TMFHE.pk}_j\}_{j\in[n]}, i)$ and the witness $w := (\widetilde{\mathsf{CT}}, \psi_i, \mathsf{TMFHE.sk}_i)$, and (stmt, w) is considered to be in the language iff $\widetilde{\mathsf{CT}}$ is a correct outcome of evaluating $\mathsf{TMFHE.Eval}(f^{\mathfrak{I}}, \{\mathsf{CT}_j\}_{j\in\mathfrak{I}})$, and moreover, p_i is a correct outcome of evaluating $\mathsf{TMFHE.PartDec}(\mathsf{TMFHE.sk}_i, \widetilde{\mathsf{CT}})$ where $(\mathsf{TMFHE.pk}_i, \mathsf{TMFHE.sk}_i)$ must match the output of $\mathsf{TMFHE.Gen}$ when consuming randomness ψ_i.
  4. multicast the message (p_i, π'_i) (tagged with the purported sender i).

- **Every round $r > T_{\mathrm{ba}}$:** if all $\{\mathsf{BA}[j]\}_{j\in[n]}$ instances have produced an output (and thus relevant variables are now well-defined), a node i does the following:

  1. for every tuple (p_j, π'_j) received from some purported sender $j \in [n]$, add j to the set S (which was initially empty) iff the following verification passes:

$$\mathsf{NIZK.V}(\{\mathsf{crs}_k\}_{k\in[n]}, (p_j, f, \mathfrak{I}, \{\mathsf{CT}_k\}_{k\in[n]}, \{\mathsf{TMFHE.pk}_k\}_{k\in[n]}, j), \pi'_j) = 1$$

  2. whenever the set S's size is at least $\lfloor n/2 \rfloor + 1$, call $\mu \leftarrow \mathsf{TMFHE.FinDec}(\{p_j\}_{j\in S})$ and output μ.

Theorem 9.2 (Weakly synchronous multi-party computation)**.** *Suppose that the* TMFHE *scheme satisfies simulation security, the* BA *scheme satisfies consistency, validity, and T_{ba}-liveness under* 0.5*-weak-synchrony, and the multi-CRS* NIZK *scheme satisfies zero-knowledge and simulation sound extractability. Then, the above SFE protocol securely evaluates the function f in $T_{\mathrm{ba}} + \Delta$ time under* 0.5*-weak-synchrony.*

We will present the full proofs of the above theorem in Section 9.3.

Remark 9.3 (On different outputs for different nodes)**.** It is not hard to generalize the above protocol to allow different nodes to receive different outputs. Specifically, we can have every node include the public-key of a public-key encryption (PKE) scheme in the PKI — henceforth node i's encryption public-key is denoted epk_i and the corresponding secret decryption key is denoted esk_i. We now use the above protocol for all nodes to compute the common output $\{\mathsf{PKE.Enc}_{\mathsf{pk}_i}(y_i)\}_{i\in[n]}$ where y_i is node i's output after evaluating f over the effective-input set. Then, each node simply decrypts its portion of the ciphertext to obtain its output.

9.3 Proofs

We construct the following sequence of hybrid experiments.

Hyb_0. Hyb_0 is identical to the real-world experiment Real except that now, the adversary $\mathcal{A}$ interacts with a challenger $\mathcal{C}$ which internally emulates the execution of all honest nodes.

Hyb_1. Hyb_1 is almost identical to Hyb_0, except the following modifications:

- Instead of calling the real-world $\mathsf{NIZK.K}$ algorithm, $\mathcal{C}$ instead calls the simulated CRS generation algorithm $\widetilde{\mathsf{K}}$, such that for for each $i \in \mathsf{Honest}$, $\mathcal{C}$ obtains $(\widetilde{\mathsf{crs}}_i, \tau_i, \mathsf{ek}_i)$. $\mathcal{C}$ uses $\widetilde{\mathsf{crs}}_i$ as node i's NIZK CRS, and keeps the simulation trapdoor τ_i and extraction key ek_i to itself.
- Whenever $\mathcal{C}$ needs to compute a NIZK on behalf of an honest node i, it instead calls the simulated prover $\widetilde{\mathsf{P}}$ algorithm supplying the trapdoor τ_i to compute a simulated proof without using the witness.

Claim 9.4. *Assume that* NIZK *satisfies computational zero-knowledge, then* $\mathcal{Z}$*'s view in* Hyb_0 *is computationally indistinguishable from its view in* Hyb_1*.*

Proof. The proof is standard and can be achieved through a sequence of hybrid experiments, where we first replace the real-world setup with a simulated setup, and then replace the NIZK proofs one by one with simulated proofs; in every step, we show that adjacent hybrids are indistinguishable by a straightforward reduction to the NIZK's computational zero-knowledge property. □

Hyb_2. Hyb_2 is almost identical to Hyb_1, except that now, if any of the following bad events happen, the challenger $\mathcal{C}$ simply aborts:

1. *Failure of* BA*'s consistency, validity, or* T_{ba}*-liveness:* the BA scheme's consistency, validity, or T_{ba}-liveness is violated;
2. *Failure of* NIZK*'s simulation sound extractability:* whenever $\mathcal{A}$ supplies a NIZK proof in any message on behalf of a corrupt node, $\mathcal{C}$ calls the NIZK's extraction algorithm using the honest nodes extraction keys $\{\mathsf{ek}_j\}_{j \in \mathsf{Honest}}$ to extract a witness; however, the witness is not valid for the statement being proven.

Henceforth, whenever a message of the form (CT_j, π_j) for $j \in \mathsf{Crupt}$ is received from $\mathcal{A}$ where π_j verifies, let $(\widetilde{x}_j, \widetilde{\rho}_j)$ denote the extracted plaintext and randomness.

Claim 9.5. *Suppose that* BA *satisfies consistency, validity, and* T_{ba}*-liveness, and that* NIZK *satisfies simulation sound extractability. Then,* Hyb_2 *does not abort except with negligible probability; and thus* $\mathcal{Z}$*'s views in* Hyb_1 *and in* Hyb_2 *are computationally indistinguishable.*

Proof. If $\mathcal{Z}$ can distinguish whether it is in Hyb_1 or Hyb_2 with non-negligible probability, we can easily construct a reduction that leverages $(\mathcal{A}, \mathcal{Z})$ to break either the consistency, validity, or T_{ba}-liveness of BA, or the simulation sound extractability of NIZK. Specifically, for applying simulation sound extractability, note that all NIZK statements in the protocol is tagged with the identity of the prover and thus no statement can be reused. □

Hyb_3. Let $\mathsf{Sim}_1, \mathsf{Sim}_2$ denote the simulator algorithms for the TMFHE scheme. Hyb_3 is almost identical to Hyb_2, except the following modifications:

- In round 0, instead of calling $\mathsf{TMFHE.Enc}$ to compute TMFHE ciphertexts on behalf of honest nodes, $\mathcal{C}$ instead calls $\mathsf{TMFHE.Sim}_1$ to compute the corresponding ciphertexts without using the actual plaintext.

- In round T_{ba}, instead of calling $\mathsf{TMFHE.PartDec}$ to compute decryption shares on behalf of honest nodes, $\mathcal{C}$ instead calls $\mathsf{TMFHE.Sim}_2(\mathsf{state}, \mu, \widetilde{\mathsf{CT}}, \mathsf{Honest}, \{\mathsf{TMFHE}.\widetilde{\rho}_j\}_{j\in\mathsf{Crupt}})$, where
  - state denotes the internal state output by $\mathsf{TMFHE.Sim}_1$ earlier;
  - for $j \in \mathsf{Crupt}$, $\widetilde{\rho}_j$ is the randomness extracted from the corresponding NIZK proof that $\mathcal{A}$ submitted together with the corrupt node's ciphertext if $j \in \mathfrak{I}$; else $\widetilde{\rho}_j := \vec{0}$.
  - $\mu := f^{\mathfrak{I}}(\{x_j^*\}_{j\in\mathfrak{I}})$ where $x_i^* := x_i$ if $i \in \mathsf{Honest} \cap \mathfrak{I}$ and $x_i^* := \widetilde{x}_i$ (i.e., the plaintext extracted from a corresponding NIZK proof supplied by $\mathcal{A}$) if $i \in \mathsf{Crupt} \cap \mathfrak{I}$.

  Note that in Hyb_3, all honest nodes must obtain the same $\widetilde{\mathsf{CT}}$ after calling $\mathsf{TMFHE.Eval}$ in any instance. Note also that although $\mathcal{C}$ obtains $|\mathsf{Honest}|$ number of simulated partial decryptions shares after calling $\mathsf{TMFHE.Sim}_2$, it may only need to send a subset of these shares to $\mathcal{A}$.

Lemma 9.6. *Suppose that* TMFHE *satisfies simulation security. Then* $\mathcal{Z}$*'s views in* Hyb_2 *and* Hyb_3 *are computationally indistinguishable.*

Proof. Suppose for the sake of contradiction that $\mathcal{Z}$ can distinguish its views in Hyb_2 and Hyb_3 with non-negligible probability, we will construct a reduction $\mathcal{R}$ that breaks the simulation security of TMFHE with non-negligible probability.

Imagine that on one hand, $\mathcal{R}$ is interacting with either $\mathsf{Real}^{\mathcal{R}}(1^\kappa, d, n)$ experiment or the $\mathsf{Ideal}^{\mathcal{R},(\mathsf{Sim}_1,\mathsf{Sim}_2)}(1^\kappa, d, n)$ experiment of the TMFHE scheme. On the other hand, $\mathcal{R}$ is also interacting with $(\mathcal{A}, \mathcal{Z})$.

- First, $\mathcal{R}$ specifies the same set of corrupt nodes as $\mathcal{A}$ outputs.
- $\mathcal{R}$ receives from the TMFHE experiment a set of public keys $\{\mathsf{TMFHE.pk}_i\}_{i\in\mathsf{Honest}}$ which it will embed into the interactions with $\mathcal{A}$. For $i \in \mathsf{Honest}$, $\mathcal{R}$ calls BA's key generation algorithm to generate a public- and secret-key pair denoted $(\mathsf{BA.pk}_i, \mathsf{BA.sk}_i)$ for node i; moreover, it calls $(\widetilde{\mathsf{crs}}_i, \tau_i, \mathsf{ek}_i) \leftarrow \mathsf{NIZK}.\widetilde{\mathsf{K}}$. It hands $\{\mathsf{BA.pk}_i, \widetilde{\mathsf{crs}}_i, \mathsf{TMFHE.pk}_i\}_{i\in\mathsf{Honest}}$ to $\mathcal{A}$, and waits for $\mathcal{A}$ to specify $\{\mathsf{BA.pk}_i, \widetilde{\mathsf{crs}}_i, \mathsf{TMFHE.pk}_i\}_{i\in\mathsf{Crupt}}$. $\mathcal{R}$ passes the terms $\{\mathsf{TMFHE.pk}_i\}_{i\in\mathsf{Crupt}}$ to the TMFHE experiment it is interacting with.
- At any time, if $\mathcal{R}$ receives a NIZK proof from $\mathcal{A}$ on behalf of corrupt nodes, it calls the NIZK scheme's extractor and use the extraction keys $\{\mathsf{ek}_j\}_{j\in\mathsf{Honest}}$ to extract a witness.
- At any time, if the bad events defined above (that violate either the security of the BA scheme or the simulation sound extractability of the NIZK scheme) happen in any instance, $\mathcal{R}$ simply aborts.
- In round 0, when $\mathcal{R}$ needs to generate TMFHE ciphertexts on behalf of honest nodes, it calls the TMFHE experiment supplying the challenge plaintexts $\{x_i\}_{i\in\mathsf{Honest}}$ and uses the ciphertexts returned by the TMFHE experiment to compute its messages to $\mathcal{A}$. Note that $\mathcal{R}$ can compute the simulated NIZK proofs without using the witness (i.e., the plaintext and the randomness used in the encryption algorithm).
- Assuming that $\mathcal{R}$ does not abort, then in round T_{ba}, some honest nodes must have produced output from all $\{\mathsf{BA}[j]\}_{j\in[n]}$ instances. For each $j \in \mathsf{Crupt}$,
  1. if some honest node's $\mathsf{BA}[j]$ has output (CT_j, π_j) and π_j passes the NIZK verification, then $\mathcal{R}$ must have extracted a witness $(\widetilde{x}_j, \widetilde{\rho}_j)$ such that CT_j is a valid TMFHE ciphertext for $\widetilde{x}_j$ using randomness $\widetilde{\rho}_j$. In this case, $\mathcal{R}$ submits to the TMFHE experiment the tuple $(\widetilde{x}_j, \widetilde{\rho}_j)$ for generating node j's ciphertext;

2. else $\mathcal{R}$ submits to the TMFHE experiment a canonical tuple $(\vec{0}, \vec{0})$ for generating node j's ciphertext — note that these ciphertexts will be excluded from the evaluation anyway.

- In round T_{ba}, when $\mathcal{R}$ needs to compute partial decryption shares on behalf of honest nodes, $\mathcal{R}$ forwards the corresponding query $(\mathfrak{I}, f^{\mathfrak{I}}, \mathsf{Honest})$ to the TMFHE experiment and uses the partial decryption shares returned by the TMFHE experiment. Here again $\mathcal{R}$ can compute the simulated NIZK proofs without using a valid witness.

- Besides the above modifications, $\mathcal{R}$ otherwise follows how Hyb_2 would interact with $(\mathcal{A}, \mathcal{Z})$.

The proof follows by making the following observation: if $\mathcal{R}$ is interacting with the $\mathsf{Real}^{\mathcal{R}}(1^\kappa, d, n)$ experiment of TMFHE, then $\mathcal{Z}$'s view is identically distributed as in Hyb_2; else if $\mathcal{R}$ is interacting with the $\mathsf{Ideal}^{\mathcal{R},(\mathsf{Sim}_1,\mathsf{Sim}_2)}(1^\kappa, d, n)$ experiment of TMFHE, then $\mathcal{Z}$'s view in the above experiment is identically distributed as in Hyb_3. Thus if Hyb_2 can be distinguished from Hyb_3 by an efficient $\mathcal{Z}$ with non-negligible probability, $\mathcal{R}$ can leverage $(\mathcal{A}, \mathcal{Z})$ to have non-negligible probability of distinguishing whether it is interacting with the $\mathsf{Real}^{\mathcal{R}}(1^\kappa, d, n)$ experiment or the $\mathsf{Ideal}^{\mathcal{R},(\mathsf{Sim}_1,\mathsf{Sim}_2)}(1^\kappa, d, n)$ experiment of TMFHE. □

Construction of $\mathcal{S}$ and $\mathsf{Ideal}^{\mathcal{S},\mathcal{Z},T}$. We now describe an ideal experiment involving an ideal-world adversary $\mathcal{S}$.

- **Corrupt:** $\mathcal{S}$ outputs the same set of corrupt nodes as $\mathcal{A}$.

- **Round 0:** $\mathcal{S}$ interacts with $\mathcal{A}$ in the same way as how the challenger $\mathcal{C}$ in Hyb_3 interacts with $\mathcal{A}$. Note that in round 0, $\mathcal{S}$ can now interact with $\mathcal{A}$ using Sim_1 and the NIZK's simulated prover without knowing honest nodes' inputs.

- **Honest and online sets:** In every round r, $\mathcal{S}$ outputs the same set of honest and online set $\mathcal{O}_r$ as $\mathcal{A}$.

- **Round T_{ba} and effective inputs:** In round T_{ba}, if $\mathcal{S}$ has not aborted, it must have computed the effective-input set $\mathfrak{I}$ and must have extracted the inputs for the set $\mathfrak{I} \cap \mathsf{Crupt}$. At this point, $\mathcal{S}$ outputs 1) the effective-input set $\mathfrak{I}$; and 2) the replaced inputs for $\mathfrak{I} \cap \mathsf{Crupt}$, and for $\mathsf{Crupt} \backslash \mathfrak{I}$ it outputs the input $\vec{0}$. Now $\mathcal{S}$ obtains the ideal-world outcome μ. From this point on $\mathcal{S}$ can compute the partial decryption shares on behalf of honest nodes knowing μ by calling Sim_2 and the NIZK's simulated prover; thus $\mathcal{S}$ can continue to interact with $\mathcal{A}$ in the same way as the challenger $\mathcal{C}$ in Hyb_3 would interact with $\mathcal{A}$.

- **Output:** Just like the challenger $\mathcal{C}$ in Hyb_3, here $\mathcal{S}$ is also emulating the honest nodes in its head. If in some round $r \geq T_{\text{ba}}$, an honest node produces an output, $\mathcal{S}$ would include the node in the L_r set it outputs.

Clearly, the above description of $\mathsf{Ideal}^{\mathcal{S},\mathcal{Z},T}$ is identical to Hyb_3 since it is just a rewriting of Hyb_3. To complete the proof that our protocol securely evaluates the function f under 0.5-weak-synchrony in $T := T_{\text{ba}} + \Delta$ time, it now suffices to prove that in the above $\mathsf{Ideal}^{\mathcal{S},\mathcal{Z},T}$ experiment, assuming $\mathcal{S}$ does not abort, then $\mathcal{S}$ respects the constraints specified by the $\mathsf{Ideal}^{\mathcal{S},\mathcal{Z},T}$ experiment for $T := T_{\text{ba}} + \Delta$ (see Section 9.1). More specifically, the following must hold:

1. in each round r, the $\mathcal{O}_r \subseteq \mathsf{Honest}$ set $\mathcal{S}$ outputs is at least $\lfloor n/2 \rfloor + 1$ in size;
2. $\mathcal{S}$ outputs a set $\mathfrak{I}$ such that $\mathcal{O}_0 \subseteq \mathfrak{I}$; and

3. $\mathcal{S}$ respects the constraints (w.r.t. $T := T_{\text{ba}} + \Delta$) on the $\{L_r\}_{r \geq T_{\text{ba}}}$ sets it outputs.

The first fact holds trivially since $(\mathcal{A}, \mathcal{Z})$ respects 0.5-weak-synchrony. The second fact holds since by validity and T_{ba}-liveness of BA, everyone in $\mathcal{O}_0$ must be included in $\mathfrak{I}$ due to the honest protocol definition. The third fact holds since by the validity and T_{ba}-liveness of BA, for everyone in $\mathcal{O}_{T_{\text{ba}}}$, all of their $\{\mathsf{BA}[j]\}_{j \in [n]}$ instances must have produced an output at the beginning of round T_{ba}, and they must send a partial decryption share and the corresponding NIZK proof in round T_{ba}. Thus everyone in $\mathcal{O}_t$ for $t \geq T_{\text{ba}} + \Delta$ must have produced an output from all $\{\mathsf{BA}[j]\}_{j \in [n]}$ instances and moreover received $\lfloor n/2 \rfloor + 1$ partial decryption shares whose NIZK proofs verify.

Acknowledgments

We gratefully acknowledge the brilliant engineering team at ThunderCore — the insightful observations they made while implementing (a practical instantiation of) the Thunderella protocol partly inspired this work. We also thank T-H. Hubert Chan, Kai-Min Chung, Leo Fan, Andrew Miller, Antigoni Polychroniadou, Ling Ren, Kartik Nayak, and Muthu Venkitasubramaniam for helpful technical discussions.

References

[1] `https://www.lamborghini.com/`.

[2] 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`.

[3] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Efficient synchronous byzantine consensus. In *Financial Crypto*, 2019.

[4] Saikrishna Badrinarayanan, Aayush Jain, Nathan Manohar, and Amit Sahai. Secure mpc: Laziness leads to god. Cryptology ePrint Archive, Report 2018/580, 2018.

[5] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *STOC*, pages 1–10, 1988.

[6] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *CRYPTO*, pages 524–541, 2001.

[7] Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *STOC*, pages 42–51, 1993.

[8] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance. In *OSDI*, 1999.

[9] Jing Chen and Silvio Micali. Algorand: The efficient and democratic ledger. https://arxiv.org/abs/1607.01341, 2016.

[10] Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proofs of stake. In *Financial Crypto*, 2019.

[11] Bernardo Machado David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In *Eurocrypt*, 2018.

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

[13] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 1988.

[14] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. In *SIAM Journal of Computing*, 1997.

[15] Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In *Eurocrypt*, 2015.

[16] O. Goldreich, S. Micali, and A. Wigderson. How to play any mental game. In *ACM symposium on Theory of computing (STOC)*, 1987.

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

[18] Jens Groth and Rafail Ostrovsky. Cryptography in the multi-string model. In *CRYPTO*, 2007.

[19] Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview seriesconsensus system. `https://dfinity.org/tech`.

[20] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. *J. Comput. Syst. Sci.*, 75(2):91–112, February 2009.

[21] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In *Crypto*, 2017.

[22] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

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

[24] Chen-Da Liu-Zhang, Julian Loss, Ueli Maurer, Tal Moran, and Daniel Tschudi. Robust mpc: Asynchronous responsiveness yet synchronous security. Unpublished manuscript.

[25] Julian Loss and Tal Moran. Combining asynchronous and synchronous byzantine agreement: The best of both worlds. Cryptology ePrint Archive, Report 2018/235, 2018.

[26] Silvio Micali, Salil Vadhan, and Michael Rabin. Verifiable random functions. In *FOCS*, 1999.

[27] Silvio Micali and Vinod Vaikuntanathan. Optimal and player-replaceable consensus with an honest majority. MIT CSAIL Technical Report, 2017-004, 2017.

[28] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.

[29] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Eurocrypt*, 2017.

[30] Rafael Pass and Elaine Shi. Rethinking large-scale consensus (invited paper). In *CSF*, 2017.

[31] Rafael Pass and Elaine Shi. The sleepy model of consensus. In *Asiacrypt*, 2017.

[32] Rafael Pass and Elaine Shi. The thunder protocol. White-paper for ThunderCore, `https://docs.thundercore.com/thunder-whitepaper.pdf`, 2018.

[33] Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In *Eurocrypt*, 2018.

[34] Elaine Shi. Analysis of deterministic longest-chain protocols. In *CSF*, 2019.

A Optimistically Responsive Byzantine Agreement

A.1 Background on Optimistically Responsive Synchronous Consensus

Optimistically responsive synchronous consensus was a notion raised in a recent work called Thunderella [33] and later extended by Loss and Moran [25]. The goal is to have *responsive* confirmation (i.e., as fast as the actual network delay) when a set of *optimistic* conditions denoted **O** holds; while the basic security properties including consistency and liveness are guaranteed other a set of weaker, *worst-case* conditions denoted **W**. In Thunderella [33], nodes are separated into two logical roles, the *leader* and a *committee* of *voters*, and informally speaking, the optimistic and worst-case conditions are defined as follows:

- **O** := "the leader and $\frac{3n}{4}$ fraction of the committee are honest"; and
- **W** := "majority of the committee are honest".

For our informal description here, we will use "state-machine-replication" version of Thunderella which agrees on a linearly-ordered log through repeated consensus over time. In Thunderella, consensus is achieved through an asynchronous "fast path" and a synchronous "slow chain" combination — one can abstractly think of the slowchain as any standard synchronous state machine replication protocol. A leader proposes blocks on the fast path and the committee vote on the proposed blocks. Once votes from $\frac{3}{4}$ fraction of the committee have been collected, one can immediately confirm the block on the fast path without waiting for the underlying slowchain. Henceforth a collection of at least $\frac{3n}{4}$ votes from distinct committee members on a block is called a *notarization*. It is not difficult to see that as long as the majority of the committee remain honest (i.e., conditions **W**) and honest voters do not double-vote at each sequence number, then only a unique block can gain notarization at each sequence number.

Note that as long as the leader and at least $\frac{3n}{4}$ of the committee are honest (i.e., conditions **O**), then blocks are confirmed on the fast path responsively (i.e., at the actual network speed) without having to wait for any a-priori set synchronization delay. If there is no progress on the fast path for a certain amount of time, a backup procedure is invoked to "fall back" to the slowchain. During this fallback, everyone will leverage the slowchain to reach agreement on what blocks have been confirmed on fast path, such that any block that has already been confirmed by an honest node on the fast path will not be rolled back. This is accomplished by having everyone post notarizations of blocks they have seen on the fast path to the slowchain (a practical optimization is to periodically post digests of the fast-path log to the slowchain to form "checkpoints", such that during a fallback, nodes only need to post notarizations since the last checkpoint [32, 33]). After the fallback completes, transactions can be posted to the slowchain to be confirmed; and importantly, at this point nodes can use the slowchain to decide how to reconfigure rebootstrap the fast path.

Subsequently, the work by Loss and Moran [25] built upon Thunderella's idea and showed how to construct an optimistically responsive, *leaderless*, single-shot consensus protocol.

Thunderella is not best-possible partition tolerant. Thunderella was proven secure under the classical synchronous model [33]. However, Thunderella is not best-possible partition tolerant (i.e., secure under 0.5-weak-synchrony) even if the underlying synchronous slowchain is best-possible partition tolerant.

We will describe a scenario in which all everyone is honest but a few honest nodes crash with an unfortunate timing pattern. In this scenario, an honest committee member (e.g., Coinbase) can possibly experience a confirmed block being rolled back, leading to financial loss. Suppose that initially, the leader is honest and online. Moreover, exactly $\lceil 3n/4 \rceil$ committee members are honest and online and the remaining committee members are offline. Blocks thus get confirmed on the fast path. At some point the leader proposes the block B. At this moment, the leader crashes. Moreover, one committee member, say, Coinbase, experiences a network-layer attack and goes offline — recall that an adversary can selectively deliver messages for offline nodes. Now, the adversary chooses to deliver $\lceil 3n/4 \rceil$ committee members' votes to Coinbase (including the vote from Coinbase itself), but it prevents everyone else from seeing Coinbase's vote. Thus no-one except Coinbase has seen a notarization for the block B. Since Coinbase is unaware that its network is experiencing a partial outage, it informs its customer, Andy, to ship his Lamborghini [1] to a buyer, and indeed Andy does.

Recall that since the leader has crashed, the fast path will shut down and this will trigger a fallback to the slowchain. When the fallback completes, everyone temporarily has to post transactions to the slowchain to confirm until they figure out a way to rebootstrap the fast-path. Of course, Coinbase would try to post the notarized block B to the slowchain too, but for as long as it remains offline, the adversary can delay this message. When Coinbase eventually comes back online, the fallback has already completed and the notarized block B, which Coinbase and Andy believed to be confirmed, is now effectively undone.

It is not hard to see that a similar attack applies to Loss and Moran [25] too.

A.2 Optimistically Responsive, Weakly Synchronous Consensus

We now show how to construct an optimistically responsive protocol that is secure under 0.5-weak-synchrony. To keep the notations simple, we will describe it for Byzantine Agreement (i.e., single-shot consensus) rather than state-machine replication (i.e., repeated consensus).

Under classical strong synchrony, a node who has seen a notarization (on some value) believes that he is capable of propagating the knowledge to all other nodes very quickly (assuming himself to be honest since we care about achieving security only for honest nodes), and thus it is safe to consider the value confirmed. Under 0.5-weak-synchrony, an honest but offline node may not be able to propagate this knowledge to others any time soon. It is, however, safe to confirm a value if "many" honest nodes have seen its notarization too, such that in any round at least one of these honest nodes will be online and capable of propagating the notarization to others (and posting it to the slow path in the event of a fallback).

Concretely, we require an additional amplification voting-round on the fast path. Nodes do not immediately output a value upon seeing a notarization (i.e., a collection of $\frac{3n}{4}$ votes); instead, they vote on the notarization again. Once a node has seen a "notarization on notarization", it is safe to output the notarized value on the fast path. Note that having seen a "notarization on notarization" in round r implies that in every round in or after r, some node in $\mathcal{O}_r$ must have seen the notarization (and thus it voted on the "notarization on notarization"). This node will then be able to ensure that this knowledge is propagated to others in the network.

Last but not the least, we need to make sure that the underlying (slow) synchronous consensus protocol is secure under 0.5-weak-synchrony too — thus for the slow path we adopt the weakly

Fast path:

- Sender multicasts its input bit b attached with a signature.
- Upon hearing a bit b attached with a valid signature from the sender for the first time, sign the tuple $(\texttt{ack}, b)$ and multicast the tuple and signature.
- A notarization for $b \in \{0, 1\}$ is a collection of at least $\frac{3n}{4}$ signatures on the tuple $(\texttt{ack}, b)$ from distinct nodes. Upon collecting a notarization for b, sign the tuple $(\texttt{commit}, b)$ and multicast the tuple and signature.
- If at least $\frac{3n}{4}$ commit messages with valid signatures from distinct nodes vouching for the same bit b have been observed by the beginning of round 3Δ, output b immediately but continue participating in the protocol.

Slow path:

- At the beginning of round 3Δ: for every $i \in [n]$, fork an instance of BA denoted $\mathsf{BA}[i]$ where the designated sender is i. For the instance $\mathsf{BA}[i]$, the i-th node's input is defined as follows:
  - if a notarization for b has been observed, then the input is b and its notarization;
  - else if a sender's signature on the bit b has been observed, the input is b and the sender's signature (include sender signatures for both bits if both have been seen);
  - else the input is $\emptyset$.
- When all n BA instances have produced output: examine the n outputs from the BA instances and output a bit as follows (if no bit has been output so far):
  - if among the n outputs, a notarization for some bit b exists, then output b;
  - else if a valid sender signature for the bit 0 exists among the n outputs, output 0;
  - else output 1.

Figure 1: An optimistically responsive, 0.5-weakly-synchronous Byzantine Agreement protocol called BA^*.

synchronous Byzantine Agreement protocol described in Section 8.

In Figure 1, we formalize the above intuition and describe the resulting protocol BA^*.

Theorem A.1 (Optimistically responsive, 0.5-weakly synchronous consensus)**.** *Let $T(\cdot, \cdot, \cdot)$ be some polynomial function. Suppose that the BA protocol employed by BA^* satisfies consistency, validity, and T-liveness under 0.5-weak-synchrony. Then, BA^* satisfies consistency, validity, and $(T + 3\Delta)$-liveness under 0.5-weak-synchrony. Furthermore, if there is a set $\mathcal{O}$ containing at least $3n/4$ honest nodes who are persistently online, and moreover the designated sender belongs to $\mathcal{O}_0$, then, except with negligible probability, everyone in $\mathcal{O}$ will output in 3δ rounds where δ is the actual maximum network delay.*

We prove the above theorem in Section A.3.

A.3 Proofs

As before, a good execution is one in which no honest node ever sees a forged signature. Assuming that the signature scheme is secure, all but a negligible fraction of executions must be good.

Henceforth given an execution, we say that some message m is in honest view if an honest ever observes it during this execution.

A.3.1 Consistency

Lemma A.2 (Uniqueness of notarization)**.** *In a good execution, it cannot be that both bits have notarization in honest view. Further, if a bit b has at least $\frac{3n}{4}$ commit messages in honest view, this bit must have a notarization in honest view and thus $1-b$ cannot have notarization or $\frac{3n}{4}$ commit messages in honest view.*

Proof. Consider a good view. Each honest node will sign an $\mathtt{ack}$ message for at most one bit; each corrupt node can sign an $\mathtt{ack}$ message for both bits. Let $f < n/2$ be the number of corrupt nodes. The total number of $\mathtt{ack}$s for either 0 or 1 in honest view can be at most $n - f + 2f = n + f < 3n/2$. However, both bits have notarization in honest view, then the total number of $\mathtt{ack}$s must be at least $\frac{3n}{4} \times 2 = \frac{3n}{2}$. Note that an honest node will only send a commit message for b if it has seen a notarization for b. Therefore the remainder of the lemma follows in a straightforward manner. □

Fix some execution, if any honest node i outputs a bit b by tallying the n outputs of the BA instances (in this execution), we say that i outputs b on the slow path. If any honest node i outputs a bit b due to having observed $\frac{3n}{4}$ commit messages for b by the beginning of round 3Δ, we say that i outputs b on the fast path. By Lemma A.2, it holds that in any good execution, if an honest node i outputs b on the fast path and an honest node j outputs b' on the fast path, it must be that $b = b'$. Therefore, to prove consistency, it suffices to prove the following lemma.

Lemma A.3 (Fast-path slow-path agreement)**.** *Consider some good execution: if an honest node i outputs b on the fast path and an honest node j outputs b' on the slow path, it must be that $b = b'$.*

Proof. If in a good execution, some honest node outputs b on the fast path, it must be that by round 3Δ, the node sees a collection of at least $\frac{3n}{4}$ commit messages from distinct nodes vouching for b. Therefore, at least $\frac{3n}{4} - f$ honest nodes (henceforth denoted the set S) have signed $(\mathtt{commit}, b)$ by round 3Δ — by honest protocol definition, nodes in S will only sign it if they have observed a notarization on b.

Now, recall that $(\mathcal{A}, \mathcal{Z})$ respects 0.5-weak-synchrony, it must be that $S \cap \mathcal{O}_{3\Delta} \neq \emptyset$. Let $i \in S \cap \mathcal{O}_{3\Delta}$, i must input b and its notarization into the i-th BA instance. By validity of the BA scheme under a 0.5-weak-synchrony, any honest node's output in the i-th BA instance must be b and its notarization. Therefore, if any node outputs a bit b' on the slow path, it must be that $b' = b$. □

A.3.2 Validity

Consider some good execution where the sender is honest and online in the starting round, and its input is b. Clearly no honest node will send $(\mathtt{ack}, 1-b)$, and thus $1-b$ cannot gain notarization in honest view. Thus no honest node will send $(\mathtt{commit}, 1-b)$ either. Thus $1-b$ cannot gain $f+1$ commit messages in honest view. Thus if any honest node outputs a bit on the fast-path it must be b.

If the sender is in $\mathcal{O}_0$, everyone in $\mathcal{O}_{3\Delta}$ must have seen the sender's signature on b. Recall that $1-b$ also cannot gain notarization in honest view. By validity of the BA, at least one of the BA instances will output either the sender's signature on b or a notarization for b. Thus if any honest node outputs a bit on the slow-path, it must be b.

A.3.3 Liveness and Optimistic Responsiveness

If at least $3n/4$ honest nodes are persistently online and the sender belongs to $\mathcal{O}_0$, optimistic responsiveness of the protocol is straightforward.

We now prove (worst-case) liveness. Consider some good execution. By T-liveness of the BA employed in BA^*, any node honest and online in round t where $t \geq T + 3\Delta$ must have produced output in all n instances of BA. Therefore, any node in $\mathcal{O}_t$ for $t \geq T + 3\Delta$ must have output a bit in BA^* by the end of round t.

B Optimistically Responsive Multi-Party Computation

B.1 Definition

Defining optimistically responsive MPC is somewhat non-trivial. Let $\chi, \alpha \in (0, 1]$. Suppose that we would like to have a χ-weakly-synchronous MPC protocol that allows honest nodes to obtain outputs responsively under the following optimistic conditions:

$\mathbf{O}$ = "at least $\alpha \cdot n$ nodes are not only honest but also persistently online".

Due to a lower bound for optimistically responsive BA proven by Pass and Shi [33], this would only be possible iff $\alpha \geq 1 - \chi/2$. Additionally, note that if $\chi \geq \frac{2}{3}$, we can simply adopt asynchronous MPC. On the other hand, if $\chi < \frac{1}{2}$, it would be impossible to achieve fairness and guaranteed output. Thus we are mainly interested in the parameter range $\frac{1}{2} \leq \chi < \frac{2}{3}$.

Another interesting issue arises in the definition: if nodes must obtain output responsively under the optimistic conditions $\mathbf{O}$, the computation can wait for only $\lceil \alpha \cdot n \rceil$ number of nodes' inputs — henceforth we refer to this set the "effective input set". Now, since the adversary can always make the corrupt nodes' messages deliver faster than honest nodes, inevitably it can include f corrupt inputs in the effective input set where f denotes the number of corrupt nodes. Thus the smaller α (i.e., we want to obtain outputs responsively under a less stringent condition), the smaller fraction of honest inputs may be included in the effective input set, and this tradeoff is inevitable.

Formal definition. Like in Section 9, we consider the task of secure function evaluation (SFE). In our formal definition, we may use a standard simulation-based definition to capture the correctness, privacy, and worst-case liveness and guaranteed output delivery requirements. For cleanness, we do not directly capture the optimistic responsiveness in the simulation-based definition, since it is easy to define optimistic responsiveness as a separate property of the SFE protocol.

Therefore, our formal definition here is almost the same as the definitions in Section 9, except that now, in the ideal-world experiment, we require that the effective input set $\mathfrak{I} \subseteq [n]$ specified by the adversary must satisfy one of the following conditions:

1. either $|\mathfrak{I}| \geq \alpha \cdot n$ or
2. $|\mathfrak{I} \cap \mathsf{Honest}| \geq \lfloor \chi \cdot n \rfloor + 1$.

In particular, the former condition on $\mathfrak{I}$ corresponds to the case in which the effective input set is determined on the fast path where as the latter condition corresponds to the case in which the effective input set is determined in the slow, synchronous path.

Since our new ideal experiment is parametrized by the parameter $\alpha \geq 1 - \chi/2$, we now denote it as $\mathsf{Ideal}^{\mathcal{S},\mathcal{Z},T,\alpha}(1^\kappa)$.

Examples. We elaborate on two special-case parameterizations:

1. $\chi = 0.5$, $\alpha = 1$: in this case, our result can be viewed as an improvement over classical honest-majority MPC: not only do we provide an optimistically responsive path when everyone is honest and persistently online, we also achieve best-possible partition tolerance in the worst-case.

2. $\chi = 0.5$, $\alpha = 3/4$: in comparison with the best-possible partition tolerant MPC protocol earlier in Section 9, here we allow online nodes to obtain outputs optimistically responsively; however, we have to sacrifice the fraction of honest inputs in the effective input set (and as argued above, this tradeoff is inherent).

In an unpublished manuscript [24], Liu-Zhang et al. present a somewhat incomparable formulation of optimistically responsive MPC in the universal composition framework, assuming a classical, *fully synchronous* network. They then show how to construct a protocol under their formulation assuming trusted setup (and not just PKI). We will be able to provide more detailed comparisons with their work when it is published.

B.2 Building Block: Optimistically Responsive Set Agreement (SA)

B.2.1 Definition

Syntax. Each node i receives an input x_i from an appropriate finite domain. Sometime during the protocol, a node may output a set of the form $X := \{(i, x_i')\}_{i \in S, S \subseteq [n]}$ and moreover, all the indices i must be different.

Security. Let $T(\kappa, n, \Delta)$ be a polynomial function in the stated parameters, let $\frac{1}{2} \leq \chi < \frac{2}{3}$, and let $1 - \chi/2 < \alpha \leq 1$. We say that an SA protocol satisfies property $P \in \{\text{consistency, } \alpha\text{-validity, } T\text{-liveness}\}$ under χ-weak-synchrony, iff for any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that respects χ-weak-synchrony, there exists a negligible function $\mathsf{negl}(\cdot)$ such that the corresponding condition P stated below holds for $1 - \mathsf{negl}(\kappa)$ fraction of the executions:

1. *Consistency*: if an honest node outputs the set X and another honest node outputs the set X', it must be that $X = X'$.

2. *α-validity*: if an honest node outputs the set X, then X must satisfy at least one of the conditions below:

   (a) either $|X| \geq \alpha n$ and $X \cap \{(i, x_i)\}_{i \in \mathsf{Honest}} \geq n - f$ where f denotes the number of corrupt nodes; or

   (b) $X \cap \{(i, x_i)\}_{i \in \mathsf{Honest}} \geq \lfloor \chi n \rfloor + 1$.

3. *T-liveness*: every node in $\mathcal{O}_r$ where $r \geq T$ must have produced an output by the end of round r (as before, we assume that the starting round of the SA is renamed round 0).

Besides the above security properties, we would like our SA protocol to be *optimistically responsive*, i.e., as long as there exists a set $\mathcal{O}$ containing at least αn nodes who are not only honest but also persistently online, then except with negligible probability, everyone in $\mathcal{O}$ should produce an output in $T_{\text{opt}}(\kappa, n, \alpha, \delta)$ number of rounds where δ is the *actual* maximum network delay and T_{opt} is an appropriate polynomial function.

B.2.2 Construction

We rely on the following building blocks:

- let ARBC denote an asynchronous reliable broadcast protocol that achieves consistency and validity as long as at least $\lfloor \chi n \rfloor + 1$ nodes are honest, and achieves T_{arbc}-honest-sender-liveness and T_{arbc}-close-termination as long as at least αn nodes are honest (see Section C.3);
- let ABA denote an asynchronous Byzantine Agreement protocol that satisfies consistency, validity, and T_{aba}-liveness as long as more than $2n/3$ nodes are honest (see Section C.3);
- let SBA denote a χ-weakly-synchronous Byzantine Agreement protocol as defined in Section 3.3 and let T_{slow} be its liveness parameter.

We describe our optimistically responsive set agreement protocol in Figure 2. Our protocol is inspired by techniques from a few earlier works [6, 25] but we combine them in a new way.

Fix some execution, if any honest node i outputs a set Y by tallying the n outputs of the SBA instances (in this execution), we say that i outputs Y on the slow path. If any honest node i outputs a set Y due to having observed at least αn commit messages for Y by the end of round $2\Delta(T_{\text{arbc}} + T_{\text{aba}} + 1)$, we say that i outputs Y on the fast path.

Theorem B.1 (Optimistically responsive set agreement)**.** *Suppose that $\frac{1}{2} \leq \chi < \frac{2}{3}$ and $1 - \chi/2 \leq \alpha \leq 1$. Suppose that the* ARBC *protocol employed satisfies consistency and validity as long as at least $\lfloor \chi n \rfloor + 1$ nodes are honest and satisfies T_{arbc}-honest-sender-liveness and T_{arbc}-close-termination as long as at least αn nodes are honest, the* ABA *protocol employed satisfies consistency, validity, and T_{aba}-liveness as long as more than $2n/3$ nodes are honest, the* SBA *protocol employed satisfies consistency, validity, and T_{slow}-liveness under χ-weak-synchrony, and the digital signature scheme employed is secure. Then, the above* SA *protocol satisfies consistency, α-validity, and $(2\Delta(T_{\text{arbc}} + T_{\text{aba}} + 1) + T_{\text{slow}} + 1)$-liveness.*

Further, if there exists a set $\mathcal{O}$ of at least αn honest nodes who are persistently online, then except with negligible probability, everyone in $\mathcal{O}$ will output on the fast path.

Note that the above theorem implies that the protocol is responsive under the optimistic condition that at least αn nodes are honest and persistently online since the fast-path protocol is asynchronous in nature. We now prove the above theorem.

B.2.3 Consistency Proof

We first prove consistency. As before, a good execution is one in which no honest node ever sees a forged signature. Assuming that the signature scheme is secure, all but a negligible fraction of executions must be good. Henceforth given an execution, we say that some message m is in honest view if an honest ever observes it during this execution.

Lemma B.2 (Uniqueness of notarization)**.** *In a good execution, it cannot be that both bits have notarization in honest view. Further, if a bit b has at least αn commit messages in honest view, this bit must have a notarization in honest view and thus $1 - b$ cannot have notarization or αn commit messages in honest view.*

Proof. The proof is almost identical to Lemma A.2 except that now we have more generalized parameters. □

By Lemma B.2, it holds that in any good execution, if an honest node i outputs Y on the fast path and an honest node j outputs Y' on the fast path, it must be that $Y = Y'$. Therefore, to prove consistency, it suffices to prove the following lemma.

The protocol invokes n instances of asynchronous reliable broadcast denoted $\{\mathsf{ARBC}[i]\}_{i\in[n]}$, n instances of asynchronous Byzantine Agreement denoted $\{\mathsf{ABA}[i]\}_{i\in[n]}$, and n instances of χ-weakly-synchronous Byzantine Agreement $\{\mathsf{SBA}[i]\}_{i\in[n]}$. In $\mathsf{ARBC}[i]$ and $\mathsf{SBA}[i]$ where $i \in [n]$, node i is the designated sender. We assume that the asynchronous instances $\{\mathsf{ARBC}[i]\}_{i\in[n]}$ and $\{\mathsf{ABA}[i]\}_{i\in[n]}$ are invoked upfront at protocol start.

PKI setup:

- During the PKI registration phase, each node i run ARBC and ABA's setup algorithms (if any) and let $(\mathsf{ARBC.pk}_i, \mathsf{ARBC.sk}_i)$ and $(\mathsf{ABA.pk}_i, \mathsf{ABA.sk}_i)$ be the output key pairs. Further, generate a digital signature key pair denoted $(\mathsf{vk}_i, \mathsf{ssk}_i)$. Now, register the tuple $(\mathsf{ARBC.pk}_i, \mathsf{ABA.pk}_i, \mathsf{vk}_i)$ with the PKI.

Fast path:

- At protocol start, node i inputs x_i into $\mathsf{ARBC}[i]$ where x_i denotes its input value.
- As soon as $\mathsf{ARBC}[j]$ outputs a value x'_j, input 1 to $\mathsf{ABA}[j]$ if no value has been input into $\mathsf{ABA}[j]$ yet.
- As soon as at least αn instances of $\mathsf{ABA}[j]$ has output the bit 1, input 0 to the remaining instances of $\mathsf{ABA}[j]$.
- Once all instances $\{\mathsf{ABA}[j]\}_{j\in[n]}$ have completed, let $\mathfrak{I}_{\text{fast}} \subseteq [n]$ be the set containing every j such that $\mathsf{ABA}[j]$ has output the bit 1.
- As soon as all instances $\{\mathsf{ABA}[j]\}_{j\in[n]}$ have completed and thus $\mathfrak{I}_{\text{fast}}$ is well-defined and moreover the subset of instances $\{\mathsf{ARBC}[j]\}_{j\in\mathfrak{I}_{\text{fast}}}$ have output a value: let x'_j denote the output from $\mathsf{ARBC}[j]$. Now, if $|\mathfrak{I}_{\text{fast}}| \geq \alpha n$, node i sets $X_{\text{fast}} := \{(j, x'_j)\}_{j\in\mathfrak{I}_{\text{fast}}}$ and multicasts the message $(\mathtt{ack}, X_{\text{fast}})$ signed with its secret key ssk_i.
- Henceforth, a collection of at least αn number of $(\mathtt{ack}, Y_{\text{fast}})$ messages with valid signatures from distinct nodes and vouching for the same set Y_{fast} is said to be a *notarization* for Y_{fast}.

  As soon as a notarization for some Y_{fast} has been observed, a node i multicasts the message $(\mathtt{commit}, Y_{\text{fast}})$ signed with its secret key ssk_i.
- Once at least αn number of $(\mathtt{commit}, Y_{\text{fast}})$ messages have been received with valid signatures from distinct nodes and vouching for the same set Y_{fast}, and moreover the current round is no greater than $2\Delta(T_{\text{arbc}} + T_{\text{aba}} + 1)$, then output Y_{fast} and continue participating in the protocol.

Slow path:

- Round $2\Delta(T_{\text{arbc}} + T_{\text{aba}} + 1) + 1$: invoke the slow, χ-weakly-synchronous instances $\{\mathsf{SBA}[i]\}_{i\in[n]}$; moreover each node i's input in $\mathsf{SBA}[i]$ is defined as follows:
  1. if a notarization on some Y_{fast} has been observed, then the input is Y_{fast} and its notarization;
  2. else the input is x_i.
- Once all n instances $\{\mathsf{SBA}[i]\}_{i\in[n]}$ have produced an output, examine the outputs from all these instances and output a value as follows (if no value has been output yet):
  1. if among the n outputs, some set Y_{fast} and a valid notarization exists, output Y_{fast}.
  2. else the n outputs must be of the form $x_1, x_2, \ldots, x_n$: now, output the set $\{(j, x_j)\}_{j\in[n]}$.

Figure 2: Optimistically responsive set agreement SA.

Lemma B.3 (Fast-path slow-path agreement)**.** *Consider some good execution: if an honest node i outputs set Y on the fast path and an honest node j outputs Y' on the slow path, it must be that $Y = Y'$.*

Proof. If in a good execution, some honest node outputs Y on the fast path, it must be that by the end round $2\Delta(T_{\text{arbc}} + T_{\text{aba}} + 1)$, the node sees a collection of at least αn commit messages from distinct nodes vouching for b. Therefore, at least $\alpha n - f$ honest nodes (henceforth denoted the set S) have signed $(\texttt{commit}, Y)$ by round $2\Delta(T_{\text{arbc}} + T_{\text{aba}} + 1)$ — by honest protocol definition, nodes in S will only sign it if they have observed a notarization on Y.

Now, recall that $(\mathcal{A}, \mathcal{Z})$ respects 0.5-weak-synchrony, it must be that $S \cap \mathcal{O}_{2\Delta(T_{\text{arbc}}+T_{\text{aba}}+1)+1} \neq \emptyset$. Let $i \in S \cap \mathcal{O}_{2\Delta(T_{\text{arbc}}+T_{\text{aba}}+1)+1}$, i must input Y and its a notarization into $\mathsf{SBA}[i]$. By validity of the SBA scheme under a 0.5-weak-synchrony, any honest node's output in the $\mathsf{SBA}[i]$ instance must be Y and its notarization. Therefore, if any node outputs a bit Y' on the slow path, it must be that $Y' = Y$. □

B.2.4 Validity Proof

There are only two ways an honest node would output a set Y in the protocol:

1. the node has observed a notarization for Y;
2. the node has not observed a notarization for Y — in this case for $j \in [n]$, each $\mathsf{SBA}[j]$'s output must be of the form x_j, and the output must be $Y := \{(j, x_j)\}_{j \in [n]}$.

First, validity for the latter case is easy to see: due to the validty of SBA, in a good execution, an honest node's output set Y must satisfy $\{(j, x_j)\}_{j \in \mathcal{O}_{2\Delta(T_{\text{arbc}}+T_{\text{aba}}+1)+1}} \subseteq Y$.

It suffices to prove that in the former case, it must be that $|Y| \geq \alpha n$ and moreover $Y \cap \{(j, x_j)\}_{j \in \mathsf{Honest}} \geq n - f$ (if the execution is good). In the former case, there is a notarization for Y in honest view. This means that at least $\alpha n - f > 0$ honest nodes must have signed $(\texttt{ack}, Y)$ where f denotes the number of corrupt nodes. By honest protocol definition, an honest node would sign $(\texttt{ack}, Y)$ only if $|Y| \geq \alpha n$ and if for each $(j, y_j) \in Y$, y_j is the outcome of $\mathsf{ARBC}[j]$. The proof thus follows from the validity of ARBC since more than χn nodes must be honest.

B.2.5 Liveness Proof

The $(2\Delta(T_{\text{arbc}} + T_{\text{aba}} + 1) + T_{\text{slow}} + 1)$-livenes proof follows in a straightforward fashion from liveness of the T_{slow}-liveness of SBA.

B.2.6 Optimistic Responsiveness Proof

Suppose there is set $\mathcal{O}$ containing at least αn honest nodes who are persistently online. Note that since $\alpha \geq 1 - \chi/2$ and $\frac{1}{2} \leq \chi < \frac{2}{3}$, it holds that $\alpha > 2/3$. In this case, all the security properties of ABA will be respected.

Since at least αn nodes are honest, due to T_{arbc}-honest-sender-liveness of ARBC, by the end of round $T_{\text{arbc}}\Delta$, everyone in $\mathcal{O}$ will have produced an output in the instances $\{\mathsf{ARBC}[j]\}_{j \in \mathcal{O}}$. Thus by the end of round $T_{\text{arbc}}\Delta$, everyone in $\mathcal{O}$ must have input 1 to each instance in $\{\mathsf{ABA}[j]\}_{j \in \mathcal{O}}$. By T_{aba}-liveness and validity of ABA, by the end of round $(T_{\text{arbc}} + T_{\text{aba}})\Delta$, everyone in $\mathcal{O}$ must have produced an output of 1 for each instance in $\{\mathsf{ABA}[j]\}_{j \in \mathcal{O}}$. By the end of the same round $(T_{\text{arbc}} + T_{\text{aba}})\Delta$, everyone in $\mathcal{O}$ will have input a bit to every instance in $\{\mathsf{ABA}[j]\}_{j \notin \mathcal{O}}$ too. By T_{aba}-liveness of ABA, by the end of $(T_{\text{arbc}} + 2T_{\text{aba}})\Delta$, everyone in $\mathcal{O}$ will have produced an output in all instances $\{\mathsf{ABA}[j]\}_{j \in [n]}$. Further, if any $\mathsf{ABA}[j]$ where $j \in [n]$ has output 1 for someone in

$\mathcal{O}$, then it must be that someone $\widetilde{i} \in \mathcal{O}$ has input 1 to $\mathsf{ABA}[j]$ and $\widetilde{i}$'s $\mathsf{ARBC}[j]$ instance must have produced an output when this happened. Due to the T_{arbc}-close-termination of ARBC, the end of round $2\Delta(T_{\text{arbc}} + T_{\text{aba}})$, everyone $i \in \mathcal{O}$ must have produced an output from instance $\mathsf{ARBC}[j]$ if $\mathsf{ABA}[j]$'s output for node i is 1.

Since at least αn nodes are honest, we can apply the consistency condition of ARBC. Therefore for every $j \in [n]$, any two honest nodes' outputs in $\mathsf{ARBC}[j]$ must be the same. Thus, by the end of round $2\Delta(T_{\text{arbc}} + T_{\text{aba}})$, everyone in $\mathcal{O}$ will have multicast a signed tuple $(\texttt{ack}, Y)$ for the same set $Y \subseteq [n]$ such that $|Y| \geq \alpha n$. Thus by the end of round $2\Delta(T_{\text{arbc}} + T_{\text{aba}}) + \Delta$, everyone in $\mathcal{O}$ will have received the signed tuple $(\texttt{ack}, Y)$ from everyone in $\mathcal{O}$. Thus everyone in $\mathcal{O}$ will multicast a signed tuple $(\texttt{commit}, Y)$ in round $2\Delta(T_{\text{arbc}} + T_{\text{aba}}) + \Delta$ and thus by the end of round $2\Delta(T_{\text{arbc}} + T_{\text{aba}} + 1)$, everyone in $\mathcal{O}$ will have received the signed tuple $(\texttt{commit}, Y)$ from everyone in $\mathcal{O}$, and will output Y on the fast path.

B.3 Construction: Optimistically Responsive MPC

Recall that in the MPC protocol of Section 9, each node i relies on a Byzantine Agreement instance (denoted $\mathsf{BA}[i]$) to broadcast its homomorphically encrypted input and a NIZK proof that the encryption is computed correctly. In round T_{ba}, if a node has produced outputs from all $\{\mathsf{BA}[j]\}_{j\in[n]}$ instances, it would then perform homomorphic evaluation, compute a partial decryption share, and multicast the partial decryption share and a NIZK proof that the homomorphic evaluation and partial decryption is done correctly. Clearly, all nodes in $\mathcal{O}_{T_{\text{ba}}}$ would have produced outputs from all $\{\mathsf{BA}[j]\}_{j\in[n]}$ instances in round T_{ba}, and their decryption shares would be sufficient for the reconstruction. Note that in this protocol, all $\{\mathsf{BA}[j]\}_{j\in[n]}$ instances jointly realize the task of set agreement.

Our new optimistically responsive MPC protocol is almost the same as the protocol in Section 9 except that now, we rely on an optimistically responsive set agreement protocol SA to reach agreement on the effective input set. For completeness, we describe the new protocol below, highlighting the difference from Section 9 in blue. Below, suppose that $\frac{1}{2} \leq \chi < \frac{2}{3}$, and $1 - \chi/2 \leq \alpha \leq 1$. Moreover, suppose that SA is an optimistically responsive set agreement protocol that achieves consistency, α-validity, and T_{sa}-liveness under χ-weak-synchrony, and achieves responsiveness as long as at least αn nodes are not only honest but also online.

- **PKI setup:** every node $i \in [n]$ calls SA's key generation algorithm and obtains $(\mathsf{SA.pk}_i, \mathsf{SA.sk}_i)$, calls $\mathsf{NIZK.K}$ and obtains crs_i. calls $\mathsf{TMFHE.Gen}$ and obtains $(\mathsf{TMFHE.pk}_i, \mathsf{TMFHE.sk}_i)$. The node now registers the tuple $(\mathsf{BA.pk}_i, \mathsf{crs}_i, \mathsf{TMFHE.pk}_i)$ with the PKI.

- **Round** 0**:** Each node $i \in [n]$ does the following (else do nothing):

  1. call $\mathsf{CT}_i \leftarrow \mathsf{TMFHE.Enc}(\{\mathsf{TMFHE.pk}_j\}_{j\in[n]}, x_i)$ where x_i denotes its input — henceforth let ρ_i be the random coins consumed by $\mathsf{TMFHE.Enc}$;
  2. call $\pi_i \leftarrow \mathsf{NIZK.P}(\{\mathsf{crs}_j\}_{j\in[n]}, \mathsf{stmt}, w)$ using the statement $\mathsf{stmt} := (\{\mathsf{TMFHE.pk}_j\}_{j\in[n]}, \mathsf{CT}_i, i)$ and the witness $w := (x_i, \rho_i)$, and (stmt, w) is considered to be in the language L iff CT_i is a correct encryption corresponding to node i of the plaintext x_i under the public keys $\{\mathsf{TMFHE.pk}_j\}_{j\in[n]}$ and using randomness ρ_i;
  3. invoke SA and input the tuple (CT_i, π_i) to SA.

- **As soon as** SA has produced an output of the form $X := \{(j, \mathsf{CT}_j, \pi_j)\}_{j\in\mathfrak{I}'}$ for some $\mathfrak{I}' \subseteq [n]$ (and all the j's must be distinct), a node i does the following:

1. for each $j \in \mathfrak{I}'$, let $(j, \mathsf{CT}_j, \pi_j)$ be the corresponding tuple included in X: if $\mathsf{NIZK.V}(\{\mathsf{crs}_j\}_{j\in[n]}, (\{\mathsf{TMFHE.pk}_j\}_{j\in[n]}, \mathsf{CT}_j, j), \pi_j)$ outputs 1, add j to the "effective-input" set $\mathfrak{I}$ (which was initially empty);
2. let $\widetilde{\mathsf{CT}} \leftarrow \mathsf{TMFHE.Eval}(f^{\mathfrak{I}}, \{\mathsf{CT}_j\}_{j\in\mathfrak{I}})$ where $f^{\mathfrak{I}}$ the restricted version of f where for every $i \notin \mathfrak{I}$ its input is set to a default value $\perp$;
3. let $p_i \leftarrow \mathsf{TMFHE.PartDec}(\mathsf{TMFHE.sk}_i, \widetilde{\mathsf{CT}})$;
4. let $\pi'_i \leftarrow \mathsf{NIZK.P}((\{\mathsf{crs}_j\}_{j\in[n]}, \mathsf{stmt}, w)$ using the statement $\mathsf{stmt} := (p_i,\ f,\ \mathfrak{I},\ \{\mathsf{CT}_j\}_{j\in[n]}, \{\mathsf{TMFHE.pk}_j\}_{j\in[n]}, i)$ and the witness $w := (\widetilde{\mathsf{CT}}, \psi_i, \mathsf{TMFHE.sk}_i)$, and (stmt, w) is considered to be in the language iff $\widetilde{\mathsf{CT}}$ is a correct outcome of evaluating $\mathsf{TMFHE.Eval}(f^{\mathfrak{I}}, \{\mathsf{CT}_j\}_{j\in\mathfrak{I}})$, and moreover, p_i is a correct outcome of evaluating $\mathsf{TMFHE.PartDec}(\mathsf{TMFHE.sk}_i, \widetilde{\mathsf{CT}})$ where $(\mathsf{TMFHE.pk}_i, \mathsf{TMFHE.sk}_i)$ must match the output of $\mathsf{TMFHE.Gen}$ when consuming randomness ψ_i.
5. multicast the message (p_i, π'_i) tagged with the purported sender i;
6. Wait to receive sufficiently many decryption shares for reconstruction as follows:
   - for every tuple (p_j, π'_j) received from some purported sender $j \in [n]$, add j to the set S (which was initially empty) iff the following verification passes:
   $$\mathsf{NIZK.V}(\{\mathsf{crs}_k\}_{k\in[n]}, (p_j, f, \mathfrak{I}, \{\mathsf{CT}_k\}_{k\in[n]}, \{\mathsf{TMFHE.pk}_k\}_{k\in[n]}, j), \pi'_j) = 1$$
   - whenever the set S's size is at least $\lfloor n/2 \rfloor + 1$, call $\mu \leftarrow \mathsf{TMFHE.FinDec}(\{p_j\}_{j\in S})$ and output μ.

Theorem B.4 (Optimistically responsive and weakly synchronous multi-party computation)**.** *Suppose that $\frac{1}{2} \le \chi < \frac{2}{3}$, and $1 - \chi/2 \le \alpha \le 1$. Suppose that the* TMFHE *scheme satisfies simulation security, the* SA *scheme satisfies consistency, α-validity, and T_{sa}-liveness under χ-weak-synchrony, and the multi-CRS* NIZK *scheme satisfies zero-knowledge and simulation sound extractability. Then, the above SFE protocol securely evaluates the function f in $T_{\mathrm{sa}} + \Delta$ (worst-case) time under 0.5-weak-synchrony. Furthermore, if there exists a set $\mathcal{O}$ containing at least αn nodes who are not only honest, but also persistently online, then except with negligible probability everyone in $\mathcal{O}$ will obtain output in time that depends only on δ but not Δ where δ denotes the actual maximum network delay.*

Note that in our actual construction above, we may adopt an ARBC and an ABA scheme that are expected constant (asynchronous) round (see Section C.3). In this case, under optimistic conditions, an output will be produced in honest and online view in $O(\delta)$ time; and under worst-case conditions (i.e., 0.5-weak-synchrony), an output will be produced in honest and online view in $O(\Delta)$ time.

Proof. Responsiveness under the stated optimistic conditions follows directly from the consistency and responsiveness of SA under the same set of conditions, and the soundness of NIZK. The simulation proof can be done with straightforward modifications to the proof of Theorem 9.2 in Section 9.3 — basically, in the new proof, the ideal-world adversary $\mathcal{S}$ would call the simulated partial decryption algorithm and would output the effective input set $\mathfrak{I}$ and replaced inputs for $\mathfrak{I} \cap \mathsf{Crupt}$ not in round T_{ba}, but as soon as some honest node has produced an output from SA. □

C Additional Preliminaries

Throughout the paper, a function $\mathsf{negl}(\cdot)$ is said to be negligible if for every polynomial $p(\cdot)$, there exists some κ_0 such that $\mathsf{negl}(\kappa) \le 1/p(\kappa)$ for every $\kappa \ge \kappa_0$.

C.1 Honest-Majority Multi-CRS NIZK

An honest-majority multi-CRS non-interactive zero-knowledge proof (NIZK) system for the language L consists of the following algorithms where R_L is the witness relation for the language L:

- $\mathsf{crs} \leftarrow \mathsf{K}(1^\kappa)$: each node $i \in [n]$ runs K to generate a common reference string (CRS) which is published in the PKI;
- $\pi \leftarrow \mathsf{P}(\{\mathsf{crs}_i\}_{i\in[n]}, x, w)$: given a statement x and a witness $w \in R_L(x)$, and the set of all n CRSes denoted $\{\mathsf{crs}_i\}_{i\in[n]}$, compute a proof denoted π;
- $\{0,1\} \leftarrow \mathsf{V}(\{\mathsf{crs}_i\}_{i\in[n]}, x, \pi)$: given a statement x, the set of all CRSes $\{\mathsf{crs}_i\}_{i\in[n]}$ and a proof π, the verifier algorithm V outputs 0 or 1 denoting either reject or accept.
- $(\widetilde{\mathsf{crs}}, \tau) \leftarrow \widetilde{\mathsf{K}}(1^\kappa)$: a simulated CRS generation algorithm that generates a simulated $\widetilde{\mathsf{crs}}$ and a trapdoor τ;
- $\pi \leftarrow \widetilde{\mathsf{P}}(x, \{\widetilde{\mathsf{crs}}_i\}_{i\in[n]}, \{\tau_i\}_{i\in G})$ where $G \subseteq [n]$ and $|G| \geq \lfloor n/2 \rfloor + 1$: a simulated prover algorithm that produces a proof for the statement x without any witness, and the simulated prover needs to have access to at least $\lfloor n/2 \rfloor + 1$ number of trapdoors.

We care about achieving security against an adversary that corrupts only minority of the common reference strings. Thus we formally define minority-constrained below.

Definition C.1 (Minority-constrained adversary $\mathcal{A}$)**.** We say that an adversary $\mathcal{A}$ who has oracle access to an honest key generation oracle K is *minority-constrained*, iff with probability 1, it outputs a set $\{\mathsf{crs}_i\}_{i\in[n]}$ such that at least $\lfloor n/2 \rfloor + 1$ of these entries are outputs of the K oracle during its interactions with $\mathcal{A}$.

We would like to have the following security properties.

Completeness. Completeness requires that for any non-uniform p.p.t. minority-constrained $\mathcal{A}$, the following holds:

$$\Pr\left[(\{\mathsf{crs}_i\}_{i\in[n]}, x, w) \leftarrow \mathcal{A}^{\mathsf{K}}(1^\kappa),\ \pi \leftarrow \mathsf{P}(\{\mathsf{crs}_i\}_{i\in[n]}, x, w),\ \mathsf{V}(\{\mathsf{crs}_i\}_{i\in[n]}, x, \pi) = 0 \text{ and } w \in R_L(x)\right] \approx 0$$

Note that since $\mathcal{A}$ is minority-constrained, the set $\{\mathsf{crs}_i\}_{i\in[n]}$ it selects must contain at least $\lfloor n/2 \rfloor + 1$ number of CRSes output by the honest K oracle.

Zero-knowledge. An honest-majority multi-CRS NIZK system satisfies zero knowledge iff the following properties hold. First, we require that simulated reference strings are indistinguishable from real ones, i.e., for every non-uniform p.p.t. $\mathcal{A}$,

$$\Pr\left[\mathsf{crs} \leftarrow \mathsf{K}(1^\kappa) : \mathcal{A}(1^\kappa, \mathsf{crs}) = 1\right] \approx \Pr\left[(\widetilde{\mathsf{crs}}, \_) \leftarrow \widetilde{\mathsf{K}}(1^\kappa) : \mathcal{A}(1^\kappa, \widetilde{\mathsf{crs}}) = 1\right]$$

Moreover, we require that if the adversary $\mathcal{A}$ is minority-constrained, it cannot distinguish interactions with a real prover using real witnesses to prove statements and a simulated prover who proves statements without using witnesses, even when $\mathcal{A}$ obtains the trapdoors of the simulated CRSes. More formally, for any non-uniform p.p.t. minority-constrained adversary $\mathcal{A}$, we have that

$$\begin{aligned}&\Pr\left[(\{\mathsf{crs}_i\}_{i\in[n]}, x, w) \leftarrow \mathcal{A}^{\widetilde{\mathsf{K}}(1^\kappa,\cdot)}(1^\kappa), \pi \leftarrow \mathsf{P}(\{\mathsf{crs}_i\}_{i\in[n]}, x, w) : \mathcal{A}(\pi) = 1 \text{ and } w \in R_L(x)\right] \\ \approx &\Pr\left[(\{\mathsf{crs}_i\}_{i\in[n]}, x, w) \leftarrow \mathcal{A}^{\widetilde{\mathsf{K}}(1^\kappa,\cdot)}(1^\kappa), \pi \leftarrow \widetilde{\mathsf{P}}(\{\mathsf{crs}_i\}_{i\in[n]}, \vec{\tau}, x) : \mathcal{A}(\pi) = 1 \text{ and } w \in R_L(x)\right]\end{aligned}$$

where $\vec{\tau}$ is the following vector: for every CRS in the set $\{\mathsf{crs}_i\}_{i\in[n]}$ that is output by $\widetilde{\mathsf{K}}$, the vector $\vec{\tau}$ includes its corresponding trapdoor. Note that there are at least $\lfloor n/2 \rfloor + 1$ entries in $\vec{\tau}$ since $\mathcal{A}$ is minority-constrained.

Simulation soundness. Intuitively, simulation soundness requires that even though an $\mathcal{A}$ may adaptively interacts with a simulated prover and obtain simulated proofs of false statements, if $\mathcal{A}$ ever produces a fresh proof for some purposed statement x, then x must be true except with negligible probability. More formally, an honest-majority multi-CRS NIZK system satisfies simulation soundness iff for any non-uniform p.p.t. minority-constrained adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that the following holds:

$$\Pr\left[\left(\{\mathsf{crs}_i\}_{i\in[n]}, x, \pi\right) \leftarrow \mathcal{A}^{\mathcal{C}(1^\kappa,\cdot)} : \begin{array}{c} (x,\pi) \text{ not output from } \mathcal{C}(1^\kappa,\cdot), \text{ and} \\ x \notin L \text{ but } \mathsf{V}(\{\mathsf{crs}_i\}_{i\in[n]}, x, \pi) = 1 \end{array} \right] \leq \mathsf{negl}(\kappa)$$

where $\mathcal{C}(1^\kappa,\cdot)$ is the following oracle:

- Upon receiving input $\mathtt{gen}$ from $\mathcal{A}$, it runs $(\mathsf{crs}, \tau) \leftarrow \widetilde{\mathsf{K}}(1^\kappa)$; it then records τ and returns only crs to $\mathcal{A}$. Note that $\mathcal{A}$ can make such $\mathtt{gen}$ queries multiple times to attempt to generate CRSes for multiple nodes; and it can choose a subset of these to include in the output $\{\mathsf{crs}_i\}_{i\in[n]}$.
- Then, at some point, $\mathcal{A}$ outputs $\{\mathsf{crs}_i\}_{i\in[n]}$ — this set of CRSes must be consistent with the CRSes in $\mathcal{A}$'s final output.
- At this moment, $\mathcal{A}$ is allowed to send $(\mathtt{prove}, x)$ to the oracle multiple times; and for each such invocation, the oracle would call $\pi \leftarrow \widetilde{\mathsf{P}}(\{\mathsf{crs}_i\}_{i\in[n]}, \vec{\tau}, x)$ and return the resulting π to $\mathcal{A}$, where $\vec{\tau}$ is the following vector: for every CRS in the set $\{\mathsf{crs}_i\}_{i\in[n]}$ that is output by $\widetilde{\mathsf{K}}$, the vector $\vec{\tau}$ includes its corresponding trapdoor. Note that $\vec{\tau}$ must contain at least $\lfloor n/2 \rfloor + 1$ such trapdoors since $\mathcal{A}$ is minority-constrained.

Simulation sound extractability. Simulation sound extractability is a further strengthening of simulation soundness and directly implies simulation soundness. Intuitively, simulation sound extractability requires that even though an $\mathcal{A}$ may adaptively interacts with a simulated prover and obtain simulated proofs of false statements, if $\mathcal{A}$ ever produces a fresh proof for some purposed statement x, then except with negligible probability, some p.p.t. extractor must be able to extract a valid witness from the proof, using an extraction key that is produced during a simulated setup procedure.

More formally, an honest-majority multi-CRS NIZK system satisfies simulation sound extractability iff there exist p.p.t. algorithms $\widetilde{\mathsf{K}}_0$ and $\mathcal{E}$ such that the following is satisfied:

- $\widetilde{\mathsf{K}}_0$ outputs a triple denoted $(\widetilde{\mathsf{crs}}, \tau, \mathsf{ek})$ where the first two terms have an output distribution identical to that of $\widetilde{\mathsf{K}}$; and
- for any non-uniform p.p.t. minority-constrained adversary $\mathcal{A}$, there exists a negligible function $\mathsf{negl}(\cdot)$, such that the following holds:

$$\Pr\left[\begin{array}{l} \left(\{\mathsf{crs}_i\}_{i\in[n]}, x, \pi\right) \leftarrow \mathcal{A}^{\mathcal{C}(1^\kappa,\cdot)} : \\ w \leftarrow \mathcal{E}(\{\mathsf{crs}_i\}_{i\in[n]}, \vec{\mathsf{ek}}, x, \pi) \end{array} \begin{array}{c} (x,\pi) \text{ not output from } \mathcal{C}(1^\kappa,\cdot), \text{ and} \\ (x,w) \notin R_L \text{ but } \mathsf{V}(\{\mathsf{crs}_i\}_{i\in[n]}, x, \pi) = 1 \end{array} \right] \leq \mathsf{negl}(\kappa)$$

  where $\mathcal{C}(1^\kappa,\cdot)$ is the following oracle:

1. Upon receiving input $\texttt{gen}$ from $\mathcal{A}$, it runs $(\mathsf{crs}, \tau, \mathsf{ek}) \leftarrow \widetilde{\mathsf{K}}_0(1^\kappa)$; it then records τ and returns crs and ek to $\mathcal{A}$.
2. Then, at some point, $\mathcal{A}$ outputs $\{\mathsf{crs}_i\}_{i\in[n]}$ — this set of CRSes must be consistent with the CRSes in $\mathcal{A}$'s final output.

   At this moment, define the following notation:
   - $\vec{\tau}$ is the following vector: for every CRS in the set $\{\mathsf{crs}_i\}_{i\in[n]}$ that is output by $\widetilde{\mathsf{K}}$, the vector $\vec{\tau}$ includes its corresponding trapdoor. Note that $\vec{\tau}$ must contain at least $\lfloor n/2 \rfloor + 1$ such trapdoors since $\mathcal{A}$ is minority-constrained.
   - Similarly, the notation $\vec{\mathsf{ek}}$ denotes the following vector: for every CRS in the set $\{\mathsf{crs}_i\}_{i\in[n]}$ that is output by $\widetilde{\mathsf{K}}_0$, the vector $\vec{\mathsf{ek}}$ includes its corresponding extraction key ek included in the triple.
3. At this moment, $\mathcal{A}$ is allowed to send $(\texttt{prove}, x)$ to the oracle multiple times; and for each such invocation, the oracle would call $\pi \leftarrow \widetilde{\mathsf{P}}(\{\mathsf{crs}_i\}_{i\in[n]}, \vec{\tau}, x)$ and return the resulting π to $\mathcal{A}$.

Groth and Ostrovksy [18] showed how to construct a multi-CRS NIZK from standard cryptographic assumptions, resulting in the following theorem.

Theorem C.2 (Multi-CRS NIZK [18])**.** *Assume the existence of enhanced trapdoor permutations. Then, there exists a multi-CRS NIZK system that satisfies completeness, zero-knowledge, and simulation sound extractability.*

C.2 Threshold Multi-Key Fully Homomorphic Encryption

Threshold Multi-key Fully Homomorphic Encryption (TMFHE) was first constructed by Gordon, Liu, and Shi [17] for t-out-of-n threshold access structures assuming the existence of a common reference string (CRS)[8]. Subsequently, Badrinarayanan et al. [4] showed how to remove the CRS and further generalized the scheme to any monotone access structure. Our formulation below is a slight strengthening of Badrinarayanan et al.'s formulation (see also Remark C.3) but we only define it for t-out-of-n access structures (and not general monotone access structure).

A t-out-of-n Threshold Multi-Key Fully Homomorphic Encryption (TMFHE) scheme [4, 17] consists of the following p.p.t. algorithms.

$\{\mathsf{pk}_i, \mathsf{sk}_i\} \leftarrow \mathsf{Gen}(1^\kappa, d, n)$: each node i independently calls Gen to generate a public-/secret-key pair denoted $(\mathsf{pk}_i, \mathsf{sk}_i)$. Gen takes as input the security parameter κ, the maximum depth d of the circuit, and the total number of nodes n;

$\mathsf{CT}_i \leftarrow \mathsf{Enc}(\mathsf{pk}_1, \ldots, \mathsf{pk}_n, i, \mathsf{m})$: takes the nodes' public keys, an index $i \in [n]$, a message $\mathsf{m} \in \{0,1\}^\kappa$, and outputs a ciphertext denoted CT_i on behalf of node i;

$\widetilde{\mathsf{CT}} \leftarrow \mathsf{Eval}(C, \{\mathsf{CT}_j\}_{j\in L})$: a *deterministic* algorithm that takes in a subset of indices $L \subseteq [n]$ and a boolean circuit $C : (\{0,1\}^\kappa)^{|L|} \to \{0,1\}$, of depth at most d and of size polynomially bounded in κ, and ciphertexts $\{\mathsf{CT}_j\}_{j\in L}$, and outputs an evaluated ciphertext denoted $\widetilde{\mathsf{CT}}$;

$p_i \leftarrow \mathsf{PartDec}(\mathsf{sk}_i, \widetilde{\mathsf{CT}})$: takes in a secret key sk_i and an evaluated ciphertext $\widetilde{\mathsf{CT}}$, and outputs a partial decryption share denoted p_i;

[8] Although some subsequent works cite Gordon et al [17] as n-out-of-n Multi-Key FHE, the citations are incorrect. Their work in fact constructed t-out-of-n Threshold Multi-Key FHE and applied it to constant-round multi-party computation with guaranteed output.

$\mu \leftarrow \mathsf{FinDec}(\{p_i\}_{i\in S})$: takes partial decryption shares $\{p_i\}_{i\in S}$ for some subset $S \subseteq [n]$ of size at least t, output a decrypted message μ.

Correctness and compactness. We require that there is a polynomial $\mathsf{poly}(\cdot)$ such that for any $L \subseteq [n]$, for any set of messages $\{\mathsf{m}_j\}_{j\in L} \in (\{0,1\}^\kappa)^{|L|}$, any boolean circuit $C : (\{0,1\}^\kappa)^{|L|} \to \{0,1\}$, and for any $S \subseteq [n]$ where $|S| \geq t$, the following holds with probability 1:

> for $j \in [n]$, let $(\mathsf{pk}_j, \mathsf{sk}_j) \leftarrow \mathsf{Gen}(1^\kappa, n)$; for $i \in L$, let $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\{\mathsf{pk}_j\}_{j\in[n]}, i, \mathsf{m}_i)$; let $\widetilde{\mathsf{CT}} \leftarrow \mathsf{Eval}(C, \{\mathsf{CT}_i\}_{i\in L})$; for $j \in S$, let $p_j \leftarrow \mathsf{PartDec}(\mathsf{sk}_j, \widetilde{\mathsf{CT}})$; it must be that 1) $\mathsf{FinDec}(\{p_j\}_{j\in S}) = C(\{\mathsf{m}_i\}_{i\in L})$; and 2) $\widetilde{\mathsf{CT}} < \mathsf{poly}(\kappa, d, n)$, i.e., the evaluated ciphertext $\widetilde{\mathsf{CT}}$'s size depends only on κ, d, and n but does not depend on the circuit's size $|C|$.

Simulation security. Formally, simulation security requires the following. There exist p.p.t. algorithms $\mathsf{Sim} := (\mathsf{Sim}_1, \mathsf{Sim}_2)$ such that for any non-uniform p.p.t. adversary $\mathcal{A}$, we have that the following experiments $\mathsf{Real}^{\mathcal{A}}(1^\kappa, d, n)$ and $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}(1^\kappa, d, n)$ are computationally indistinguishable:

- $\mathsf{Real}^{\mathcal{A}}(1^\kappa, d, n)$:
  1. $\mathsf{Crupt} \leftarrow \mathcal{A}(1^\kappa, d, n)$ where $\mathsf{Crupt} \subseteq [n]$ and $|\mathsf{Crupt}| < t$. Henceforth let $\mathsf{Honest} := [n] \backslash \mathsf{Crupt}$;
  2. For $i \in \mathsf{Honest}$, run $\mathsf{pk}_i \leftarrow \mathsf{Gen}(1^\kappa, d, n)$. For each $j \in \mathsf{Crupt}$, the adversary $\mathcal{A}(\{\mathsf{pk}_i\}_{i\in\mathsf{Honest}})$, who receives the honest nodes' public keys as input, outputs $\{\mathsf{pk}_j\}_{j\in\mathsf{Crupt}}$. Henceforth let $\vec{\mathsf{pk}} := \{\mathsf{pk}_j\}_{j\in[n]}$.
  3. $(\{\mathsf{m}_i\}_{i\in\mathsf{Honest}}) \leftarrow \mathcal{A}$ where each message $\mathsf{m}_i \in \{0,1\}^\kappa$. Now for $i \in \mathsf{Honest}$, let $\mathsf{CT}_i \leftarrow \mathsf{Enc}(\vec{\mathsf{pk}}, i, \mathsf{m}_i)$.
  4. For every $j \in \mathsf{Crupt}$, the adversary $\mathcal{A}(\{\mathsf{CT}_i\}_{i\in\mathsf{Honest}})$, who receives the challenge ciphertexts computed in the previous step as input, outputs a pair $(\mathsf{m}_j, \rho_j^{\mathrm{enc}})$ where $\mathsf{m}_j \in \{0,1\}^\kappa$ denotes a message and ρ_j^{enc} is the random coins to be consumed by the encryption algorithm, and let $\mathsf{CT}_j \leftarrow \mathsf{Enc}(\vec{\mathsf{pk}}, j, \mathsf{m}_j)$ consuming the random coins ρ_j^{enc};
  5. The adversary $\mathcal{A}$ specifies polynomially many queries (indexed by k) of the form (L_k, C_k, S_k) where $L_k \subseteq [n]$ is a subset of indices, $C_k : (\{0,1\}^\kappa)^{|L_k|}$ denotes a circuit to be evaluated, and $S_k \subseteq \mathsf{Honest}$ denotes the subset of honest nodes' partial decryptions requested by $\mathcal{A}$. For each query indexed by k, let $\widetilde{\mathsf{CT}}_k \leftarrow \mathsf{Eval}(C_k, \{\mathsf{CT}_j\}_{j\in L_k})$, and for $j \in S_k$, let $p_{k,j} \leftarrow \mathsf{PartDec}(\mathsf{sk}_j, \widetilde{\mathsf{CT}}_k)$, and $\mathcal{A}$ receives the set $\{p_j\}_{j\in S_k}$;
  6. Output the adversary $\mathcal{A}$'s output.
- $\mathsf{Ideal}^{\mathcal{A},\mathsf{Sim}}(1^\kappa, d, n)$:
  1. $\mathsf{Crupt} \leftarrow \mathcal{A}(1^\kappa, d, n)$ where $\mathsf{Crupt} \subseteq [n]$ and $|\mathsf{Crupt}| < t$. Henceforth let $\mathsf{Honest} := [n] \backslash \mathsf{Crupt}$;
  2. For $i \in \mathsf{Honest}$, run $\mathsf{pk}_i \leftarrow \mathsf{Gen}(1^\kappa, d, n)$. For each $j \in \mathsf{Crupt}$, the adversary $\mathcal{A}(\{\mathsf{pk}_i\}_{i\in\mathsf{Honest}})$, who receives the honest nodes' public keys as input, outputs $\{\mathsf{pk}_j\}_{j\in\mathsf{Crupt}}$. Henceforth let $\vec{\mathsf{pk}} := \{\mathsf{pk}_j\}_{j\in[n]}$.
  3. $(\{\mathsf{m}_i\}_{i\in\mathsf{Honest}}) \leftarrow \mathcal{A}$ where each message $\mathsf{m}_i \in \{0,1\}^\kappa$. Now let $(\{\mathsf{CT}_i\}_{i\in\mathsf{Honest}}, \mathsf{state}) \leftarrow \mathsf{Sim}_1(\vec{\mathsf{pk}}, \mathsf{Crupt})$;
  4. For every $j \in \mathsf{Crupt}$, the adversary $\mathcal{A}(\{\mathsf{CT}_i\}_{i\in\mathsf{Honest}})$, who receives the challenge ciphertexts computed in the previous step as input, outputs a pair $(\mathsf{m}_j, \rho_j^{\mathrm{enc}})$ where $\mathsf{m}_j \in \{0,1\}^\kappa$ denotes a message and ρ_j^{enc} is the random coins to be consumed by the encryption algorithm, and let $\mathsf{CT}_j \leftarrow \mathsf{Enc}(\vec{\mathsf{pk}}, j, \mathsf{m}_j)$ consuming the random coins ρ_j^{enc};

5. The adversary $\mathcal{A}$ specifies polynomially many queries (indexed by k) of the form (L_k, C_k, S_k) where $L_k \subseteq [n]$ is a subset of indices, $C_k : (\{0,1\}^\kappa)^{|L_k|}$ denotes a circuit to be evaluated, and $S_k \subseteq \mathsf{Honest}$ denotes the subset of honest nodes' partial decryptions requested by $\mathcal{A}$. For each query indexed by k, let $\widetilde{\mathsf{CT}}_k \leftarrow \mathsf{Eval}(C_k, \{\mathsf{CT}_j\}_{j \in L_k})$, let $\{p_{k,j}\}_{j \in S_k} \leftarrow \mathsf{Sim}_2(\mathsf{state}, \mu_k, \widetilde{\mathsf{CT}}_k, S_k, \{\rho_j^{\mathrm{enc}}\}_{j \in \mathsf{Crupt}})$ where

$$\mu_k := \begin{cases} C(\{\mathsf{m}_j\}_{j \in L_k}) & \text{if } |S_k \cup \mathsf{Crupt}| \geq t \\ \bot & \text{o.w.} \end{cases}$$

   The adversary $\mathcal{A}$ receives the set $\{p_{k,j}\}_{j \in S_k}$;
6. Output the adversary $\mathcal{A}$'s output.

Remark C.3 (Difference from Badrinarayanan et al.'s formulation [4])**.** Our definition is a slight variant and slight strengthening of Badrinarayanan et al. [4]. In Badrinarayanan et al. [4]'s definition, there is a separate $\mathsf{DistSetup}$ phase and a KeyGen phase. In their security game, the adversary must explain to the experiment the randomness supplied to KeyGen for generating corrupt nodes' keys (but need not do so for $\mathsf{DistSetup}$). It is not hard to see that Badrinarayanan et al. [4]'s construction and proofs still hold even if the adversary does not explain to the experiment the randomness that is used to seed KeyGen to generate corrupt nodes' keys. In particular, the adversary already supplies to the experiment all randomness used to generate corrupt nodes' ciphertexts. Therefore, when Sim_2 simulates partial decryption shares, instead of using the corrupt secret keys (output by KeyGen) to decrypt terms from the corrupt nodes' ciphertext, the experiment can obtain the same terms from the adversary's explanation of the corrupt ciphertext and supply it to Sim_2.

With this observation, we may merge their KeyGen algorithm into the $\mathsf{DistSetup}$ stage (which we call Gen in our definition). In the new security experiment, the adversary need not explain the randomness used to seed Gen to generate corrupt nodes' keys. For our Sim_2 to simulate the partial decryption shares on behalf of honest nodes, Sim_2 is supplied with the randomness used to generate corrupt ciphertexts rather than the randomness used to generate corrupt keys.

Theorem C.4 (TMFHE [4])**.** *Suppose that the standard Learning With Errors (LWE) assumption holds. There exists a Threshold Multi-Key FHE (*TMFHE*) scheme that satisfies correctness, compactness, and simulation security as defined above.*

C.3 Asynchronous Consensus

An asynchronous protocol is one in which a node can only take action (including perform computation and send messages) when one of the following two events takes place:

1. at protocol start when an input is received from $\mathcal{Z}$ — in this paper we consider protocols in which $\mathcal{Z}$ provides input only once to each node;
2. upon receiving some message from the network.

From a programmer's perspective, an asynchronous protocol's implementation should only have two types of callback functions corresponding to the above two events respectively. In particular, the implementation cannot have timeouts or query any local or global clock.

Asynchronous network model. Our asynchronous building blocks achieve security even when the adversary $\mathcal{A}$ can arbitrarily delay honest nodes' messages and deliver them to different nodes at different times of its choice. However, $\mathcal{A}$ cannot erase honest nodes' messages and must eventually deliver them to the intended recipients.

The round complexity of an asynchronous protocol is accounted for using a standard notion called *asynchronous round* [7]. We refer the reader to Canetti and Rabin [7] for a formal definition of this notion.

C.3.1 Asynchronous Reliable Broadcast (ARBC)

Syntax. A designated sender $i \in [n]$ receives an input x from the enviornment, every other node receives no input. During the protocol, a node may output a value y.

Security. We say that an asynchronous reliable broadcast (ARBC) protocol satisfies consistency, validity, or T-liveness assuming that at least ρn nodes are honest, iff for any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that corrupts at most $n - \rho n$ nodes and spawns multiple ARBC instances sharing the same PKI, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all but $\mathsf{negl}(\kappa)$ fraction of the executions, the corresponding property defined below holds:

- *Consistency.* If an honest node outputs x and another honest node outputs y, it must be that $x = y$.
- *Validity.* If the designated sender is honest, then if any honest node outputs x, x must be the sender's input.
- *T-honest-sender-liveness.* If the designated sender is honest and rename the asynchronous round in which the designated sender inputs some value x to the protocol to be round 0, then all honest nodes will have output a value in asynchronous round T.
- *T-close-termination.* Moreover, if any honest node outputs a value in asynchronous round r, then all honest nodes will have output a value in asynchronous round $r + T$.

Theorem C.5. *Assume the existence of a PKI and digital signatures. Then, for any $0 < \chi \leq 1$, there exists an ARBC scheme that satisfies consistency and validity as long as $\lfloor \chi n \rfloor + 1$ nodes are honest, and satisfies 2-honest-sender-liveness and 2-close-termination as long as at least $(1 - \chi/2)n$ nodes are honest.*

Proof. Consider the following protocol parametrized by $0 < \chi \leq 1$.

- During PKI setup, everyone generates public and secret key pair for a digital signature scheme and regiesters the public key with the PKI.
- During the entire protocol, always echo (i.e., multicast) every fresh message that is observed for the first time.
- When the designated sender receives an input value x from $\mathcal{Z}$, it multicasts $(\texttt{propose}, x)$ with a signature on the tuple.
- Upon receiving a tuple of the form $(\texttt{propose}, x')$ with a valid signature from the designated sender, multicast $(\texttt{ack}, x')$ with a signature on the tuple.
- Upon observing at least $(1 - \chi/2)n$ signatures on the same tuple $(\texttt{propose}, x')$, output x'.

The consistency proof holds due to a very similar argument as the proofs of Lemma A.2 and Lemma B.2. For validity, note that if any honest node outputs x, the node must have observed $(1-\chi/2)n$ nodes' signatures on $(\texttt{ack}, x)$, and thus an honest node must have signed $(\texttt{ack}, x)$ and this honest node must have seen the designated sender's signature on $(\texttt{propose}, x)$. Honest-sender-liveness is obvious, and close termination is also obvious due to the fact that honest nodes echo every fresh message they see. □

C.3.2 Asynchronous Byzantine Agreement (ABA)

Observe that the broadcast version of Byzantine Agreement is impossible in the asynchronous setting since an honest sender with a really long network delay is indistinguishable from a corrupt sender that has crashed. Thus nodes cannot tell whether they should wait longer for the sender's message or simply give up and agree on a value without the sender. We thus define the agreement version of Byzantine Agreement like in the standard consensus literature.

Syntax. Every node receives an input bit $b \in \{0,1\}$ from the environment $\mathcal{Z}$. Every node eventually outputs a bit b'.

Security. We say that an asynchronous Byzantine Agreement (ABA) protocol satisfies consistency, validity, or T-liveness assuming that at least ρn nodes are honest, iff for any non-uniform p.p.t. $(\mathcal{A}, \mathcal{Z})$ that corrupts at most $n - \rho n$ nodes and spawns multiple ABA instances sharing the same PKI, there exists a negligible function $\mathsf{negl}(\cdot)$ such that for all but $\mathsf{negl}(\kappa)$ fraction of the executions, the corresponding property defined below holds:

- *Consistency.* If an honest node outputs x and another honest node outputs y, it must be that $x = y$.
- *Validity.* As long as at least ρn honest nodes obtain the same input bit b, then any honest node's output must be b.
- *T-liveness.* Rename the first round in which at least ρn number of honest nodes have obtained input from $\mathcal{Z}$ to be round 0, then all honest nodes will have output a value in asynchronous round T.

Theorem C.6 (Asynchronous Byzantine Agreement [7])**.** *Assume the existence of PKI and digital signatures. Then, there exists an* ABA *protocol that satisfies satisfies consistency, validity, and T-liveness for any $T(\cdot)$ that is super-logarithmic in κ, as long as at least $\lfloor 2n/3 \rfloor + 1$ nodes are honest. Further, honest nodes obtain outputs in the* ABA *protocol in expected constant number of asynchronous rounds.*

D Echo Mechanism

As mentioned earlier, a brute-force mechanism to realize Assumption 2 from Assumption 1 is to have nodes echo (i.e., multicast) all messages they have seen in every round. This approach, however, would be very expensive.

We now describe a more efficient mechanism to realize Assumption 2 from Assumption 1. Effectively, online nodes only need to retry a few times whereas offline nodes might need to keep retrying until shortly after they come online — however, since nodes are unaware whether they are online or offline, the actual protocol needs to look for a different indicator as to when to stop

retrying. The idea is to stop retrying when more than $n/2$ nodes have echoed the message. More concretely, nodes rely on the following echo mechanism to realize Assumption 2:

- An echo from a node i for a message m is of the format $(\texttt{echo}, \mathsf{m})$ tagged with node i's signature[9].
- Upon observing a fresh message m (including messages contained in an echo, messages input from $\mathcal{Z}$, messages received over the network, or messages the node tried to send itself), multicast an echo for m in every round until more than $n/2$ valid echos for m have been heard from distinct nodes. Note that nodes need not recursively send echos for an echo.

Theorem D.1. *Consider a good execution: the above echo mechanism satisfies Assumption 2.*

Proof. Suppose that some honest node stops echoing m in round r, we prove that everyone in $\mathcal{O}_{r'}$ where $r' \geq r + \Delta$ will have seen m.

Let $r'' \leq r$ be the first round in which some honest node hears more than $n/2$ echos for m. Some node $i^* \in \mathcal{O}_{r''-1}$ must have sent an echo for m; and since r'' is the first round in which some honest node hears more than $n/2$ echos for m, it must be that i^* sent an echo for m in round $r''-1$ too. Thus, by Assumption 1, every node honest and online in round $r' \geq r'' + \Delta$ will have seen m. □

E A Termination Technique for Byzantine Agreement

In our BA protocol in Section 8.1, honest nodes participate forever even after having output a bit. It is, however, quite easy to devise a termination technique such that terminated nodes need not send additional messages including even the echo messages in Section D that is necessary for realizing the strong message delivery assumption (Assumption 2). Henceforth a set of $\lfloor n/2 \rfloor + 1$ commit messages for b signed by distinct nodes pertaining to the same epoch is called a *finalization evidence* for b. Moreover, we assume that every message is signed together with the session identifier and tagged with the message's purported sender. Upon receiving a message with an invalid signature, the message is immediately discarded.

Basically, whenever a node sees a finalization evidence C for the bit b in its view, it not only outputs b but also multicasts the tuple $(\texttt{finalize}, b, C)$ every round, until the node has heard a valid `finalize` message (with a valid finalization evidence) from $\lfloor n/2 \rfloor + 1$ number of nodes — at this point it continues to multicast this tuple for $\Delta + 1$ additional rounds and then it terminates.

It is not difficult to see that if the original BA protocol in Section 8 satisfies consistency and validity w.r.t. $(\mathcal{A}, \mathcal{Z})$ then the modified BA protocol above satisfies consistency and validity w.r.t. $(\mathcal{A}, \mathcal{Z})$ too. The theoreom below states the termination/liveness property of the above modified BA scheme.

Theorem E.1 (Termination)**.** *Suppose that the original* BA *protocol in Section 8 satisfies T-liveness, then in the modified protocol, except with negligible probability, everyone in $\mathcal{O}_t$ where $t \geq T + 3\Delta$ will not only have output but also terminated by the end of round $t + \Delta$.*

Proof. Earlier we have shown that everyone in $\mathcal{O}_t$ where $t \geq T$ must have received $\lfloor n/2 \rfloor + 1$ commit messages pertaining to the same epoch and for the same bit by the end of round t. We first prove that in the modified scheme, some honest node must have terminated by the end of round

[9] We assume that all protocol instances share the same echo-PKI. For composition, we assume that the message signed is tagged with the session identifier *sid* and verification is aware of the *sid* too.

$T + 2\Delta$. Suppose that no honest node has terminated even after round $T + 2\Delta$ — we show that this is impossible since if so, then every node in $\mathcal{O}_T$ must send a finalize message for b in round T and thus every node in $\mathcal{O}_{T+\Delta}$ will have received $\lfloor n/2 \rfloor + 1$ number of `finalize` messages from distinct nodes in round $T + \Delta$ and will have terminated by the end of round $T + 2\Delta$.

Due to the same proof as in Section D, except with negligible probability the following statements also hold. Suppose that r is the first round in which some honest node hears valid `finalize` messages from $\lfloor n/2 \rfloor + 1$ number of nodes. Then, everyone in $\mathcal{O}_t$ where $t \geq r + \Delta$ will have seen at least one valid `finalize` message containing a valid finalization evidence at the beginning of round t. Thus everyone in $\mathcal{O}_{r+\Delta}$ will multicast a `finalize` message in round $r + \Delta$. Therefore everyone in $\mathcal{O}_{t'}$ where $t' \geq r + 2\Delta$ will have seen $\lfloor n/2 \rfloor + 1$ number of `finalize` messages from distinct nodes and will have terminated by the end of round $t' + \Delta$. □

F Supplemental Figure

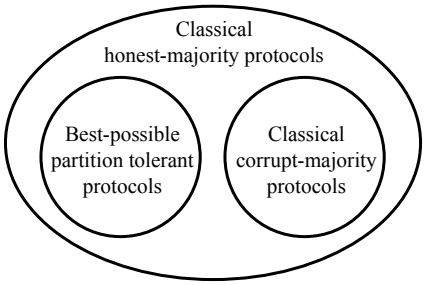

Figure 3: The set of best-possible partition tolerant BA protocols is a strict subset of classical honest-majority BA protocols. No BA protocol that tolerates corrupt majority satisfies best-possible partition tolerance.