Optimal Best-of-Both-Worlds Consensus

Fatima Elsheimy[1], Simon Holmgaard Kamp[2], Julian Loss[2], and Jesper Buus Nielsen[3]

[1] Yale University
[2] Ruhr University Bochum
[3] Aarhus University

Abstract. Consensus protocols face a fundamental trade-off: synchrony enables higher fault tolerance, whereas asynchrony provides responsiveness (latency proportional to actual network delays) and security under arbitrary delays. In particular, synchronous consensus achieves optimal resilience up to $t < n/2$ corruptions but relies on worst-case delay bounds that affect both latency and security, whereas asynchronous consensus tolerates only $t < n/3$ corruptions.

Two approaches have sought orthogonal compromises between the two. *Optimistically responsive protocols* [Pass and Shi, EC'18] achieve resilience beyond the feasibility limits of asynchronous security, and provide responsiveness under optimistic conditions. *Network-agnostic protocols* [Blum–Katz–Loss, TCC'19] are secure in both synchronous and asynchronous networks but are non-responsive due to relying on conservative worst-case waiting and, to date, do not achieve constant time in asynchrony.

We reconcile these approaches by constructing the first Byzantine agreement (BA) and validated Byzantine agreement (VBA) protocols that achieve network-agnostic security and optimistic responsiveness with optimal resilience tradeoffs. Concretely, for thresholds $t_s > t_r > t_a$ satisfying $n > 2t_s + t_a$ and $n > t_s + 2t_r$, our protocols satisfy:

- Both protocols are synchronously secure for up to t_s corruptions and asynchronously secure for up to t_a corruptions.
- Our VBA protocol is responsive when the number of corruptions is at most t_r, while our BA protocol is responsive if additionally at least $t_s + 1$ honest parties begin with the same input.
- Both protocols run in expected constant time with quadratic communication complexity, regardless of network conditions.

We prove matching impossibility results showing that the resilience tradeoffs are optimal, and our protocols additionally achieve optimal expected time and communication complexity.

Table of Contents

1 Introduction 2
2 Technical Overview 3
2.1 State of the Art 3
2.2 Reconciling Network-Agnostic and Optimistic Responsiveness: Shifting the Switch from Protocol-Level to Step-Level 4
2.3 Related Work 7
3 Preliminaries 10
3.1 Model 10
3.2 Problem definitions 13
4 Binary Provable Byzantine Agreement 15
4.1 Input Validation Gadget 16
4.2 Binary Provable Validated Byzantine Agreement 20
4.3 Justified Proxcensus 23
5 Optimality of our Thresholds 30
6 Conclusion and Future Directions 36

1 Introduction

Consensus a.k.a. Byzantine agreement (BA) [LSP82] is a fundamental problem in distributed computing and cryptography. In a consensus protocol, n parties attempt to agree on a common output value despite the presence of up to $t < n/2$ malicious (Byzantine) parties that may deviate arbitrarily from the protocol. Decades of research have delineated the conditions under which consensus can be solved securely. Broadly, these results fall into two models: (1) the *synchronous model:* the network guarantees that messages are delivered within a known upper bound Δ, and all parties have a common notion of time (e.g., global clocks). In this setting, many fundamental problems—including consensus and multi-party computation (MPC)—are solvable tolerating up to $t < n/2$ corruptions, and (2) the *asynchronous model*: the network provides no timing guarantees, ensuring only eventual message delivery. In this model, classic results show that consensus and MPC are solvable if and only if $t < n/3$. A key advantage of asynchronous protocols is *responsiveness*, meaning their running time only depends on the actual speed of the network, since there are no fixed round delays. By comparison, the added resilience of synchronous protocols (tolerating $n/2$ faults instead of $n/3$) comes at a significant cost: their running time is tied to the worst-case network delay Δ; specifically, the protocol proceeds in rounds and must wait Δ in each round to ensure all honest messages arrive. At the same time, if a message is ever delayed by more than Δ time, all guarantees are lost.

This was partially mitigated by Pass and Shi [PS18] when they introduced the concept of synchronously secure protocols with optimistic responsiveness (OR),

i.e., protocols that have higher corruption resilience than possible in asynchrony, and additionally responsive under some optimistic condition. However, protocols in this line of work are purely synchronously secure and have no guarantees under asynchrony. On the other hand, the line of work on network-agnostic (NA) protocols initiated by Blum et al. [BKL21] takes an orthogonal approach by preserving correctness under arbitrary network delays, while achieving higher resilience under synchrony. But, using current techniques, this comes with the cost of not only giving up on the responsiveness of asynchronous protocols, but also on constant asynchronous time complexity.

Our Contribution: Truly Best-Of-Both-Worlds Consensus. So far, approaches that compromise between the benefits and drawbacks of synchrony and asynchrony have followed completely orthogonal paths concerning responsiveness and NA security. A natural question is whether it is possible to achieve higher corruption tolerance than what is possible during asynchrony whenever the network is synchronous, while, at the same time, retaining both asynchronous security and responsiveness when fewer parties are corrupted.

We completely characterize the feasibility of best-of-both-worlds BA and validated BA protocols. For n parties and f Byzantine corruptions and any $t_s < n/2$, we give BA and VBA protocols achieving the following:

- Synchronous security/liveness when $f \leq t_s$.
- Asynchronous security/liveness when $f \leq t_a = n - 2t_s - 1$.
- Responsiveness when $f \leq t_r = \lceil (n - t_s)/2 \rceil - 1$ for validated BA, while BA additionally requires $t_s + 1$ honest parties to have the same input.
- Our protocols are *efficient*: we achieve, for the first time, constant-expected running time as well as $O(n^2)$ communication under any network condition.

We also provide new lower bounds for VBA and BA, showing that these conditions are minimal.

2 Technical Overview

2.1 State of the Art

Two distinct lines of research have sought to overcome the gap between the synchronous and asynchronous models:

Optimistic Responsiveness. Pass and Shi [PS18] introduced optimistically responsive consensus protocols, which are synchronously secure for up to $t_s < n/2$ corruptions and responsive when the actual number of faults f for a given execution stays below some threshold $t_r < t_s$. Additionally, they proved that no protocol with synchronous security for $f < n/2$ can be optimistically responsive if $f \geq n/4$. However, the drawback is that these protocols do not provide any guarantees if the network behaves asynchronously. For example, if messages are delayed arbitrarily (violating the synchrony assumption), an adversary can exploit timing assumptions to break agreement (honest parties output the same value) or validity (honest outputs are the input of an honest party). Synchronous

protocols typically rely on *timeouts* to collect quorum certificates of size $n - t_s$, which are guaranteed to be collected within Δ time in synchronous conditions. Suppose that all honest parties start with the same input value, but some honest messages are delayed beyond the timeout. Then, even in the presence of a single corrupted party, honest parties may fail to collect a quorum of size $n - t_s$ by time $c \cdot \Delta$ for some constant c and consequently, may think that not all honest parties started with the same value. As a result, they may proceed to agree on a value different from the honest parties' initial input, thereby violating validity.

Network-Agnostic Consensus. Blum, Katz, and Loss [BKL19] introduced the *network-agnostic (NA) model*, which unifies synchronous and asynchronous guarantees in a single protocol. A network-agnostic protocol is secure regardless of whether the network is synchronous or asynchronous: it tolerates up to t_s corruptions in a fully synchronous execution, and remains secure (and guarantees eventual termination) with up to t_a corruptions in an asynchronous network. More specifically, their BA protocol Π achieves security whenever the system satisfies $0 < t_a < \frac{n}{3} \leq t_s < \frac{n}{2}$ and $2t_s + t_a < n$. Furthermore, they establish that the bound $n > t_a + 2t_s$ is both *necessary* and *sufficient* for solving BA in this setting. Since the initial work of BKL19, the network-agnostic paradigm has seen widespread adoption. A series of follow-up results extended this paradigm, including general MPC [BZL20], distributed key generation [BCLZL23,ACC22], state-machine replication [BKL21,ABKL22], and other consensus variants [DE24,CGWW24,GLZW22]. However, all existing NA protocols essentially follow the same design blueprint as BKL19: they run a standard synchronous protocol Π_s for some predetermined running time (the worst-case time needed to run the protocol in case the network is synchronous), and if it has not terminated by then, parties assume the network is asynchronous and switch over to an asynchronous protocol Π_a (secure for t_a faults). This mode-switching approach ensures safety in both worlds but comes at the cost of efficiency. Concretely, a party can only be confident in switching to the asynchronous mode Π_a after waiting out the worst-case synchronous latency T. Waiting for T rounds is necessary to ensure that if Π_s failed to finish, then the network must indeed be asynchronous (and moreover that the number of corruptions $f \leq t_a$). As a result, network-agnostic protocols that follow this blueprint are inherently slow in the worst case: they cannot achieve expected constant rounds or any optimistic responsiveness, because even under good conditions every party must conservatively wait for T time. These efficiency limitations make it difficult to motivate NA protocols as best-of-both-worlds solutions and have kept them from being regarded as practical.

2.2 Reconciling Network-Agnostic and Optimistic Responsiveness: Shifting the Switch from Protocol-Level to Step-Level

Our key idea is to – in contrast to the network-agnostic tradition – move the logical switch between the synchronous and asynchronous execution paths from the *protocol* level to the level of individual protocol *steps*, and to invert the

order in which these paths are attempted. Whereas Blum et al. [BKL19] execute an entire synchronous protocol before falling back to an asynchronous one, we alternate between two paths at every step of the protocol. Concretely, each protocol step begins by optimistically attempting to execute logic which is safe up to t_s corruptions without relying on synchrony, but which may deadlock unless some optimistic condition is satisfied. We refer to this as a *fast step*. In a fast step, all parties send a message that allows the protocol to make progress if sufficiently many parties send the right type of messages. If the optimistic conditions are not satisfied, parties fall back to what we call a *slow step*; a step that is guaranteed to be executed under maximum corruption.

We first describe this mechanism in the context of VBA, which serves as the core building block of our BA construction. We then explain how the same fast/slow structure is extended to the input justification phase required for BA.

Fast Path: VBA. Our VBA protocol follows a standard framework, alternating between invoking proxcensus [FLZL21] (a variant of graded agreement) and a common coin. While the latter is trivially responsive given threshold signatures (since $n - t_s > t_s$ parties are honest [CKS00]), our core insight lies in providing a responsive way to make progress in an adaptation of an asynchronous *justified* proxcensus (see Definition 7) from [Kam25]. Here, *justified* means that inputs, outputs, and protocol messages are accompanied by proofs certifying their validity with respect to some predicate. Their protocol uses justified voting messages (Definitions 2, 3) and advances to the next round upon observing either two distinct justified votes or a quorum of $n - t_a$ unanimous votes. The safety of the protocol only relies on the fact that any two such quorums intersect on at least one honest party as $n - 2t_a > t_a$ in the fully asynchronous setting with $n > 3t_a$.

In our setting, we change this condition by saying that for any $t_a < t_r < t_s$ satisfying $n > 2t_s + t_a$ and $n > t_s + 2t_r$, we may safely use the same logic for quorums of size $n - t_r$ as long as up to t_s parties are corrupted. We call such $n - t_r$ unanimous votes a *fast quorum*. Since all that is required for this to be safe is that two such quorums intersect on an honest party, this is guaranteed since $n - 2t_r$ is more than the assumed maximal number of corruptions t_s. In particular, this means that whenever fewer than t_r parties are corrupted, parties can *responsively* collect messages from $n - t_r$ parties and move on to the next step based on either having received a fast quorum of $n - t_r$ unanimous votes, or two different justified votes.

Slow Path: VBA. The main challenge of designing a fallback mechanism for the synchronous non-responsive case (i.e., in the opposite direction of BKL19) is that it is impossible for a party to tell that the fast path of VBA has failed in their local view. This complicates matters compared to BKL19, in which an asynchronous fallback mechanism is employed when the clock indicates that the synchronous path has taken too long for the synchronous setting to be plausible

within the constraints of the model.[4] While this allows these prior works building on the BKL19 framework to directly infer that the network must be asynchronous (and thus only t_a parties are corrupted), it is also the reason that the design pattern incurs an inherent super-constant asynchronous time complexity.

Instead, what we rely on is that a party can locally reason in the following manner: "If the network is synchronous, I know that all honest parties must have sent their message for the *fast* path by time τ. Thus I would have received all of their messages by time $\tau + \Delta$. I can wait for this time (by which I will have received all honest messages for the fast path) to occur before I send anything towards the completion of the current step on the *slow* path."

This allows using quorums in a novel fashion: To complete a step on the slow path we will only require *slow* quorums of size $n - t_s$ to guarantee liveness. Such a quorum now has different properties based on whether the network is synchronous or asynchronous: If the network is asynchronous, then there are at most t_a corruptions, and its intersection with any other quorum includes an honest party (as all quorums have size at least $n - t_s$ and $n - 2t_s > t_a$). In this case, the protocol logic of Kamp [Kam25] goes through using the usual intersection argument. If on the other hand the network is synchronous and there are t_s corruptions, then we cannot conclude anything from the size of the quorum itself, except that it includes a single honest party (as $n - t_s > t_s$).

Thus, if we can ensure that all honest votes (fast and slow) are unanimous whenever a single honest party sends a slow vote, then any quorum of $n - t_s$ justified votes certifies a unique value, ensuring the safety of the slow path. To enforce this unanimity condition, we require honest parties to send a vote toward completion of a slow step only if they reach time $\tau + \Delta$ without observing any vote that disagrees with their initial fast vote. Note here that since the protocol can move on responsively based on seeing just two justified votes for different values, getting to time $\tau + \Delta$ without completing the step implies that either all honest votes are unanimous or the network is asynchronous.

Keeping honest parties synchronized. The core idea presented above relies on the fact that honest parties are able to tell that at some point they would have received all honest messages from a certain protocol step, *if* the network is synchronous. This is easy to achieve at the beginning of the execution, under the standard synchronous assumption that honest parties have synchronized clocks and start the protocol at some pre-agreed time. However, for all other steps of the protocol it would be cumbersome to rely on clocks for synchronization, since the good-case scenario is that parties move ahead much faster than round-by-round. By borrowing a common technique from partially synchronous and optimistically responsive protocols, we sidestep the issue of synchronization for all but the initial synchronization point by using justifiers for all messages. Since the justification predicates are designed to guarantee that any justified message satisfies the validity and consistency properties we would require of an honest

[4] Some later works (e.g., [DE24]) unconditionally run both paths sequentially and rely on the first synchronous phase to make honest inputs to the next phase consistent under synchrony.

message; an honest party can simply treat the first justified *fast* vote it sees as its own *fast* vote, immediately forward it and ignore a later input that would have allowed sending a vote for the same step. This has the effect that a party sending its own vote at time τ knows that if the network is synchronous, all honest parties will receive it by time $\tau + \Delta$, and thus it will have received all honest votes by time $\tau + 2\Delta$.

Extending the design pattern to BA. To obtain BA from VBA, we follow Cachin et al. [CKPS01]. In VBA, validity is defined by the existence of a justification showing that a value is admissible with respect to a predefined predicate, rather than by direct consistency with honest inputs. To recover BA in [CKPS01], this predicate is satisfied for any bit justified by $t + 1$ signatures from parties claiming that it was their input. In asynchrony, where $n > 3t$, this justification arrives within a round trip. However, in our setting, since t_s parties may be corrupted, we would need signatures from $t_s + 1$ parties to ensure validity. While this is easy to achieve whenever $t_s + 1$ honest parties have the same input, additional care is required to correctly justify VBA inputs otherwise. As a result, responsive progress for our BA depends not only on the responsiveness conditions of the underlying VBA instance, but also on $t_s + 1$ honest parties having the same input. In Theorem 8 we show that this is an inherent limitation for responsive BA, and in Section 4.1 we provide a slow path to complete the picture.

2.3 Related Work

Network-Agnostic Consensus. Blum, Katz, and Loss [BKL19] initiated the study of the *network-agnostic (NA)* model. They introduced a binary Byzantine agreement protocol that is secure under up to $t_a < n/3$ corruptions in the asynchronous setting and up to $t_s < n/2$ in the synchronous setting, provided the condition $n > t_a + 2t_s$ holds. Their protocol incurs $O(n^4\kappa)$ communication, and a running time of $O(n\Delta)$ in the synchronous setting, *even if* $t << t_a$, and then, in the asynchronous setting, an additional expected $O(\delta)$ to run the fallback phase.

Since the work of [BKL19], there has been increasing interest in designing round-efficient NA consensus protocols. All known approaches broadly follow the BKL blueprint of waiting for the synchronous BA bound before attempting the asynchronous fallback. Notably, Deligios et al. [DHLZ21] study round efficient synchronous BA with asynchronous fallback. However, they only consider round efficiency of synchronous executions. In order to terminate using the asynchronous path, it still needs to run λ (where λ is a statistical security parameter) rounds of the synchronous protocol before it can make progress.

In a recent work Deligios and Erbes [DE24] design a compiler that turns any pair of synchronous and asynchronous BA protocols into a network-agnostic BA. In order for their compiler to make black box use of the BA protocols, it has to run both the synchronous and the asynchronous BA in sequence. In spite of this, they still (up to a small additive constant) preserve the concrete expected

round complexity of the synchronous protocol by employing a gadget that makes the asynchronous protocol terminate deterministically whenever output of the synchronous BA is unique. Since the compiler allows any synchronous protocol to be used, it can be instantiated with Ghinea et al. [GGLZ22], which achieves an error probability $(cr)^r$ in r rounds and thus terminates in $O(\lambda/\log\lambda)$ rounds with $2^{-\lambda}$ error improving on Deligios et al. [DHLZ21] in worst case synchronous executions and thus also in asynchronous executions. While the protocol of Ghinea et al. never terminates before $O(\lambda/\log\lambda)$ rounds, one can transform it so that it terminates in expected constant time by using the compiler of Goldreich and Petrank to run $O(\log(\lambda/\log\lambda)$ iterations that each terminate with constant probability, before running the original protocol. Thus, by applying the compiler of Goldreich and Petrank [GP90] to Ghinea et al. [GGLZ22] and then using the resulting protocol in the compiler of Deligios and Erbes, one obtains the state of the art network-agnostic BA in terms of expected time complexity in the synchronous case (i.e., $O(\Delta)$ time) *and* worst case time complexity in synchrony and expected asynchronous complexity (at $O(\frac{\lambda}{\log\lambda}\Delta)$ and $O(\frac{\lambda}{\log\lambda}\Delta+\delta)$ respectively). The communication complexity is dominated by the Ghinea et al. BA at $O(n^4(\kappa + r\log r))$ for r rounds, which for $r = O(\lambda/\log\lambda)$ is $O(n^4\kappa)$ but can be reduced by a factor n using threshold signatures. Alternatively, Deligios and Erbes can be instantiated with the communication efficient version mentioned by the authors, which uses threshold signatures and the synchronous protocol by Momose and Ren [MR21] to achieve $O(n^2\kappa)$ communication.

When comparing to the state of the art in the network-agnostic setting, we provide the first BA which is simultaneously expected constant time and quadratic communication. These properties have only previously been achieved in isolation and, in the case of running time, only in synchrony.

(Optimistic) Responsiveness. Responsiveness was coined by Pass and Shi [PS17] as the property that the running time of a protocol only depends on the actual network delay δ rather than on an upper bound on the network delay Δ. This notion of the actual network delay δ matches the *period* of an execution defined by Canetti and Rabin [CR93] as the maximal time that a message has been in transit between two honest parties. Unfortunately, Pass and Shi demonstrate that it is impossible to guarantee responsiveness if $n \leq 3t$. On the positive side, in their follow up work Thunderella [PS18] Pass and Shi introduce *optimistic responsiveness* which circumvents the impossibility result by only achieving responsiveness when certain optimistic conditions are met, but without relying on these optimistic conditions for safety and liveness in general. In terms of broadcast and state machine replication, they show that it is solvable for $t_s < n-1$ corruptions with synchronous security and responsiveness whenever $\lceil (n-t_s)/2 \rceil - 1$ parties (in addition to a designated sender or leader) are honest. They also show that this is tight in terms of the tradeoff between number of corruptions needed for synchronous security and the number tolerated on the responsive path. Concretely, that such protocols cannot be responsive in the face of $(n-t_s)/2$ corruptions.

Asharov et al. [ACCI25] study the security of a approach to achieve optimistic responsiveness seen in practice, in which parties move on to the next round of

Protocol	Setting	Running Time	Communication	Responsive
[BKL19][BKL19]	Sync	$O(n\Delta)$	$O(n^4\kappa)$	✗
	Async	$O(n\Delta + \delta)$		
[DHL21][DHLZ21]	Sync	$O(\Delta)$	$O(n^3\kappa)$	✗
	Async	$O(\lambda\Delta + \delta)$	$O(\lambda n^3\kappa)$	
[DE24](A)[DE24]	Sync	$O(\Delta)$	$O(n^3\kappa)$	✗
	Async	$O(\frac{\lambda}{\log\lambda}\Delta + \delta)$		
[DE24](B)[DE24]	Sync	$O(n\Delta)$	$O(n^2\kappa)$	✗
	Async	$O(n\Delta) + O(\delta)$		
This work	Sync	$O(\Delta)$	$O(n^2\kappa)$	✗
	Resp	$O(\delta)$		✓
	Async			

Table 1. Comparison of protocols with $n > 2t_s + t_a$: running time, communication complexity, and optimistic responsiveness across network models. "Setting" denotes if synchrony and $f \leq t_s$ (Sync), $f \leq t_a$ (Async) or, synchrony, $t_s + 1$ honest parties with the same bit and $f < (n - t_s)/2$ (Resp) is assumed. [DE24](A) is the early termination instantiation of the Deligios and Erbes compiler described above. [DE24](B) is the communication optimized instantiation described in their paper. We report the communication of each protocol assuming threshold signatures where applicable.

a protocol as soon as they see messages from the previous round. This yields responsiveness under the condition that no parties are corrupted. They show that this approach is sound for a class of MPC protocols that are based on multiplication triples or have perfect security. Several works on SMR protocols follow the outline of Pass and Shi [AMN+20,SARN20,MCK20,ANS21] and focus on low concrete latency as a function of δ and Δ, but rely on a designated leader being honest to achieve responsiveness. While this is an inherent limitation in broadcast, it may seem unsatisfactory to make this compromise in protocols such as SMR and consensus that are not inherently tied to a designate leader. In particular, it leaves the optimistic condition vulnerable to be broken at the whim of an adversary with a small number of adaptive corruptions. From a more practical perspective, this corresponds to a denial of service attack against the single point of failure.

The only protocols that achieve synchronous security with optimistic responsiveness under low (but non-zero) corruption, regardless of who is corrupted, are by Liu-Zhang et al. [LZLM+20] and Loss and Moran [LM18]. Liu-Zhang et al. [LZLM+20] show that the bounds by Pass and Shi are also tight for MPC (with the additional restriction that for full security $n > 2t_s$ is needed), without relying on a designated leader for their positive result. Loss and Moran [LM18] provide the only feasibility result for BA. [5] They present a compiler that transforms any pair of synchronous and asynchronous BA protocols into a synchronously secure one that is optimistically responsive. Concretely, while they

[5] The construction of Liu-Zhang et al. [LZLM+20] does rely on an optimistically responsive BA, but it is not synchronously secure. Specifically, it does not have validity against t_s corruptions.

distinguish between the number of corruptions that are tolerated to ensure consistency, responsiveness, and validity. If we restrict the scope to protocols that are safe in synchrony under t_s corruptions, then their compiler gives solutions for $n > 2t_s + t_r$ where additionally t_r is capped at $n/4$. In comparison, our protocols do not have the $t_r < n/4$ limitation, allowing full responsiveness when less than $n - 2t_s$ parties are corrupted regardless of the input configuration, and pushing the threshold to $n > t_s + 2t_r$ when responsiveness is characterized by the input configuration in addition to the number of corruptions. (Although in fairness, it appears unproblematic to adapt their compiler to achieve the same guarantees. Albeit without any asynchronous security.) Loss and Moran also show that the $n > 2t_s + t_r$ condition is necessary *when the responsiveness condition exclusively considers the number of corruptions*. This is a special case of Theorem 8, in which we show that synchronously secure BA with up to t_s corruptions can be responsive *if and only if* less than $(n - t_s)/2$ parties are corrupted and more than t_s honest parties have the same input.

There is no explicit construction of optimistically responsive validated BA in the literature. But it is straightforward to implement VBA from any of the many aforementioned SMR protocols. (The construction is trivial: let all parties send inputs and validity proofs through the SMR.) Among these, the original Thunderella framework by Pass and Shi [PS18] and in the particular the dishonest majority solution is noteworthy because it yields the only dishonest majority VBA which is resilient against a dishonest majority in synchrony – albeit with reliance on a designated party staying honest. Similarly, the subprotocol used in the input stage of the MPC protocol by Liu-Zhang et al. [LZLM+20] can also be used to instantiate VBA. In contrast to the works mentioned, our contribution is to combine synchronous security with optimistic responsiveness with asynchronous security with optimal tradeoffs.

3 Preliminaries

3.1 Model

Network and Adaptive Corruption. We consider a set of n parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, where each pair of parties shares an authenticated communication channel and message delivery is controlled by the adversary $\mathcal{A}$. We assume that $\mathcal{A}$ is computationally-bounded and adaptive, meaning that it may corrupt parties at any point during a protocol execution. Upon corrupting a party P, the adversary learns the state of P and may have it deviate arbitrarily from the protocol. In addition, we allow the adversary to delete or replace any undelivered messages previously sent by a newly corrupted P_i while it was still honest. We denote by t the number of parties corrupted in an execution and specify bounds on t below.

Executions. We define an execution of a protocol Π as a tuple specifying the security parameter κ, a set of n parties $\mathcal{P}$, and the adversary $\mathcal{A}$, in addition to

the inputs (including random tapes) of $\mathcal{A}$ and all parties in $\mathcal{P}$.

$$(\Pi, \kappa, \mathcal{A}, x_1, \ldots, x_n, r_1, \ldots, r_n, r_{\mathcal{A}})$$

This tuple determines all events that occur in the view of all parties during the execution (including the order in which they occur).

Clocks and Time Complexity. Time is modeled via a global clock. An execution is synchronous if all messages between two honest parties are delivered within a known bound Δ and all parties can query the clock reliably.

In asynchronous executions: there are no guarantees on when messages are delivered, queries to the clock return arbitrary values decided by the adversary, and termination guarantees are only stated in terms of the maximal duration in which a message between two honest parties has been en route. In line with Canetti and Rabin [CR93], for any finite execution, we use the maximal such duration (which we call δ) to measure asynchronous time complexity. In synchronous executions we measure time complexity in Δ.

Responsiveness. In line with Pass and Shi [PS17,PS18], we say that a protocol is *responsive* if its running time is a function of the actual network delay (i.e., the period δ used to measure asynchronous time complexity). As we consider efficient protocols this means that there is some polynomial p, such that $\delta \cdot p(\kappa)$ upper bounds the running time of the protocol with overwhelming probability.

Admissible executions. We use admissible execution to denote an execution which respects the assumptions made in a security definition. That is, for the protocols we present and any choice of parameters $t_a + 2t_s < n$, we say that an execution is admissible if the adversary corrupts up to f parties, where $f \leq t_s$ if the execution is synchronous and $f \leq t_a$ when it is asynchronous. When our security definitions state that a property holds for a protocol, this means that it holds for all but a negligible fraction of admissible executions.

Threshold Signatures and PKI. We assume the existence of a public key infrastructure (PKI) supporting a digital signature scheme with efficient signing and verification algorithms, denoted Sign and Verify, respectively. Each party P_i holds a public key vk_i known to all other parties, and a corresponding private key sk_i known only to P_i. A party P_i can generate a signature on message m as $\mathsf{sig}(m) := \mathsf{Sign}(\mathsf{sk}_i, m)$, which can be verified using $\mathsf{Verify}(\mathsf{vk}_i, \mathsf{sig}(m), m)$. We say a signature $\mathsf{sig}(m)$ is *valid* if $\mathsf{Verify}(\mathsf{vk}_i, \mathsf{sig}(m), m) = 1$. Crucially, we follow the Dolev-Yao model [DY81] and idealize signatures schemes as ideal objects producing κ-bit sized signatures with perfect unforgeability.

Additionally, we will also make use of threshold signatures with reconstruction thresholds $t_s + 1$, $t_a + 1$, $n - t_s$, and $n - t_r$, and define it as follows:

Definition 1 (t-Threshold Signatures). *A t-out-of-n threshold signature scheme with unique signatures is a tuple of efficient algorithms* (Setup_t, Sig_t, Ver_t, $\mathsf{ShareVerify}_t$, $\mathsf{Combine}_t$) *defined as follows.*

- $\mathsf{Setup}_t(1^\kappa, n, t)$: *On input the security parameter* κ*, the number of parties* n*, and threshold* t*, it outputs a verification key* vk *and secret signing shares* $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$*;*
- $\mathsf{Sig}_t(\mathsf{sk}_i, m)$: *On input a message* m *and a secret signing share* sk_i*, it outputs a partial signature* σ_i*;*
- $\mathsf{ShareVerify}_t(\mathsf{vk}_i, m, \sigma_i)$: *On input the verification key* vk_i*, a message* m*, and a partial signature* σ_i*, it outputs* 1 *if* σ_i *is valid, and* 0 *otherwise;*
- $\mathsf{Combine}_t(\{\sigma_{i_1}, \ldots, \sigma_{i_t}\})$: *On input a set of* t *valid partial signatures, it outputs either a unique signature* σ *or* $\perp$*;*
- $\mathsf{Ver}_t(\mathsf{vk}, m, \sigma)$: *On input the verification key* vk*, a message* m*, and a signature* σ*, it outputs* 1 *if* σ *is valid, and* 0 *otherwise.*

When a party receives a signature in our protocol descriptions, these are implicitly verified and ignored upon failure.

Common Coin. We assume an ideal protocol for common coin Π_{CC}. If all honest parties invoke Π_{CC} with an identifier id, the functionality gives a unique coin c_{id} to all honest parties. The value of this coin is unpredictable until the first honest party invokes Π_{CC} with id. This can be implemented asynchronously using threshold signatures [CKS00], assuming an honest majority (implied by $2t_s + t_a < n$ and $t_s + 2t_r < n$). Threshold-signature-based constructions typically incur $O(n^2\kappa)$ communication and have a single round of interaction.

Justified messages, inputs and outputs. Inputs, outputs, and messages sent in our protocols are often accompanied by a proof *justifying* them with respect to a predicate. This is used to force the adversary to behave consistently with honest behavior. As an example, consider the previously discussed reduction from BA to VBA, where parties send a $t+1$ threshold signature to demonstrate that an honest party received a bit as input [CKPS01]. We follow a similar notion of justifiers as defined in [Kam25] although in our case, a justification predicate does not depend on the state of the party evaluating it, but comes with an explicit proof to align with the definitions by Cachin et al. [CKPS01].

Definition 2 (Justification predicate). *We call an efficiently computable predicate* $J : \{0,1\}^* \times \{0,1\}^* \to \{\top, \perp\}$ *a justification predicate. Whenever* $J(m, \pi) = \top$ *for a message* m *and a proof* π*, we say that* (m, π) *constitutes a valid justification for* m*.*

By convention, for protocols where the security definition establishes that output values satisfy a justifier J_{OUT}, we will write $\Pi.J_{\text{OUT}}$ to disambiguate the output justification predicate of protocol Π from those of other protocols.

Our security definitions sometimes claim that for a justification predicate, J, some property, P, holds for *all* J*-justified values*. By this we mean that P holds not only for all honest values, but even for values which the adversary sends as long as they satisfy J. More specifically, we require that the adversary cannot cook up a value v and proof π for which $J(v, \pi) = \top$ unless v satisfies P.

Definition 3 (J-Justified Values). *Let J be a justification predicate and let protocol Π be executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$ under attack by $\mathcal{A}$ in an arbitrary admissible execution $\mathcal{E}$. When we quantify over* all J-justified values *with respect to $\mathcal{E}$, this refers to the set of all values v for which some honest party at some point during the execution $\mathcal{E}$ has v and a corresponding proof π such that $J(v, \pi) = \top$ in its view. If this set consists of a single element, we say that this is the* only J-justified value *with respect to $\mathcal{E}$.*

The definition above defines J-justified values with respect to an execution. In the context of our security definitions, such an execution is always implicitly fixed, as properties stated in the security definitions are stated with respect to an execution and claimed to hold for all but a negligible fraction of admissible executions. Therefore, we will usually just write J-justified values whenever the corresponding execution is clear from the context.

We can define statements involving different justifiers to obtain more fine-grained requirements and statements on input/output values of protocols. For example, we may say that if P holds for all J-justified values, then P' holds for all J'-justified values.

3.2 Problem definitions

We include definitions of the studied primitives.

Byzantine Agreement. Byzantine agreement [LSP82] allows a set of parties to agree on a single bit that corresponds to the input of an honest party.

Definition 4 (Byzantine Agreement). *Let Π be a protocol executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, where each party P_i holds an input $x_i \in \{0, 1\}$ and outputs a value $y_i \in \{0, 1\}$. We say that Π is a secure BA protocol if the following properties hold:*

Liveness: *If every honest party P_i has input x_i, then eventually every honest party P_j outputs y_i.*
Agreement: *If honest parties P_i and P_j have output y_i and y_j then $y_i = y_j$.*
Validity: *If an honest party P_i has output y_i, then some honest party P_j has input $x_j = y_i$.*

Provable Validated Byzantine Agreement. Cachin et al. [CKPS01] introduced Validated BA (VBA) as a relaxation of BA in which the standard validity requirement is replaced with the weaker condition that the outputs of honest parties satisfy an external validity predicate $J_{\text{IN}} : \{0, 1\} \times \{0, 1\}^* \mapsto \{0, 1\}$. Concretely, this is implemented by having each party provide a witness that its input satisfies the predicate and also obtain such a witness for its output. We defer the formal description to Definition 9 in Section 5 as we instead a stronger (in the sense that it implies VBA) primitive: Provable VBA. Our definition of PVBA adds an output justifier J_{OUT}, where we require that outputs satisfying J_{OUT} are

consistent in addition to satisfying J_{IN}. This definition morally matches PVBA as described by Dai et al. [DZJR23], which ensures that each honest party receives a transferable certificate alongside the output such that the pair satisfy a predicate R. However, their security definition requires that if a corrupted party outputs a value and proof satisfying R, then an honest party must have already output a value and proof satisfying R. This does ensure that certified outputs inherit the safety properties of honest outputs. However, it is not implementable in a standard setting since, on one hand an honest party who receives output before all other parties must have a provable output, but on the other hand a corrupted party who follows the protocol in a way that is indistinguishable from this honest party is not allowed to obtain a provable output. Instead, we follow the approach of [Kam25] and impose agreement and (external) validity properties on outputs that satisfy the output predicate J_{OUT}, although we adapt the justifiers to come with an explicit proof string, rather than be a function of each party's state.

Definition 5 (Provable Validated Byzantine Agreement). *Let $\Pi^{J_{\text{IN}}}$ be a protocol parameterized by justification predicate J_{IN}, and executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, where each party P_i receives an input $x_i \in \{0,1\}$ and proof $\pi_{x_i} \in \{0,1\}^*$, and outputs a value $y_i \in \{0,1\}$ and proof $\pi_{y_i} \in \{0,1\}^*$. Additionally, $\Pi^{J_{\text{IN}}}$ defines an output justification predicate J_{OUT}. We say that $\Pi^{J_{\text{IN}}}$ is a secure PVBA protocol if the following properties hold:*

Liveness: *If every honest party P_i has justified input x_i and π_{x_i} (where $J_{\text{IN}}(x_i, \pi_{x_i}) = \top$), then eventually every honest party P_j outputs y_j with proof π_{y_j}.*
Justified Agreement: *For all J_{OUT}-justified values y and y': $y = y'$.*
Justified Validity: *For all outputs y that are J_{OUT}-justified, y is J_{IN}-justified.*

Provable BA. Similarly, BA can be extended with a provability notion, which comes for free in our implementation.

Definition 6 (Provable Byzantine Agreement). *Let Π be a protocol executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, where each party P_i receives an input $x_i \in \{0,1\}$, and outputs a value $y_i \in \{0,1\}$ and proof $\pi_{y_i} \in \{0,1\}^*$. Additionally, $\Pi^{J_{\text{IN}}}$ defines an output justification predicate J_{OUT}. We say that Π is a secure PBA protocol if the following properties hold:*

Liveness: *If every honest party P_i has justified input x_i, then eventually every honest party P_j outputs y_j with proof π_{y_j}.*
Justified Agreement: *For all J_{OUT}-justified values y and y': $y = y'$.*
Justified Validity: *For all outputs y that are J_{OUT}-justified, the input of some honest party P_i is $y_i = y$.*

Justified Proxcensus. Similarly, we adapt the definition of justified proxcensus from [Kam25] to be compatible with our justifier definition. This is a version of proxcensus [FLZL21] with external validity and provable output.

Definition 7 (Justified Proxcensus). *Let $\Pi^{J_{\text{IN}}}$ be a protocol for n parties, parameterized by a justification predicate J_{IN}, where each party P_i receives an input $x_i \in \{0,1\}$ and proof $\pi_{x_i} \in \{0,1\}^*$, and outputs a value $y_i \in \{0,\ldots,G{-}1\}$ alongside a proof $\pi_{y_i} \in \{0,1\}^*$. Additionally, the protocol is associated with an output justification predicate J_{OUT}. We say that Π is a secure* JProx-G *protocol if the following holds:*

Liveness: *If every honest party P_i has justified input x_i, π_{x_i} (where $J_{\text{IN}}(x_i, \pi_{x_i}) = \top$), then eventually every honest party P_j will have justified output y_j and π_{y_j} where $J_{\text{OUT}}(y_j, \pi_{y_j}) = \top$.*

Justified Graded Agreement: *For all J_{OUT}-justified values y and y': $|y{-}y'| \leq 1$.*

Justified Validity: *If b is the only J_{IN}-justified bit, then $b \cdot (G{-}1)$ is the only J_{OUT}-justified output value.*

4 Binary Provable Byzantine Agreement

The classical goal of Byzantine Agreement (Definition 4) consists of two properties: (1) agreement, requiring that all honest parties output the same value, and (2) validity, requiring that if all honest parties begin with the same input value v, then v is the decided output. Provable Byzantine agreement (PBA) is a slightly stronger notion in which parties output, in addition to the decided value, a proof justifying the output. Accordingly, the agreement and validity properties are strengthened to justified agreement, requiring that all honest parties agree on some justified value, and justified validity, requiring that if all honest parties begin with the same input, then this value is the only justified output. As mentioned earlier, we follow Cachin et al., [CKPS01], and reduce the PBA problem to provable validated Byzantine agreement (PVBA). In contrast to BA, provable validated Byzantine agreement relaxes the justified validity condition. Instead of requiring the output to be directly consistent with the parties' initial inputs, PVBA requires that if the output value is justified with respect to some *output justification predicate*, then it is also valid with respect to some *input justification predicate*. To obtain full provable Byzantine agreement, we will: (1) define the input justifier so that it guarantees standard validity—that is, only the value that all honest parties initially held can be justified—and (2) design a protocol that runs before PVBA, during which parties obtain justified inputs. In particular, if all parties start with the same value, then by the end of these rounds they should collectively obtain a proof that justifies this value, w.r.t. PVBA's input justifier. We call this protocol Π_{IVG} the *Input Validation Gadget (IVG)*.

Obtaining justified inputs for PVBA in an optimistically responsive manner is subtle. In the favorable case where at least $t_s + 1$ honest parties start with the same input value v, every honest party can obtain a justification proof in the form of $t_s + 1$ signatures on v: since at most t_s parties are corrupted in the synchronous case, and at most $t_a < t_s + 1$ parties are corrupted in the asynchronous case, any such $(t_s + 1)$-proof necessarily contains an honest party

and is therefore sound. However, if honest parties do not have $t_s + 1$ matching inputs, then waiting for a $(t_s + 1)$-proof can cause a deadlock indefinitely. To break this deadlock, we let each party multicast its input and optimistically wait for $t_s + 1$ identical inputs. In an asynchronous execution, such a $t_s + 1$-proof is guaranteed to eventually appear and is always sound, since $n > 2t_s + t_a$ and at most t_a parties are malicious. Furthermore, if at least $t_s + 1$ honest parties share the same input, then a $(t_s + 1)$-proof will be collected within δ network delay. On the other hand, if honest parties begin with different inputs, such a proof may fail to materialize. Consequently, if a party waits for Δ and still does not observe a $(t_s + 1)$-proof, it can safely conclude that either:

1. honest parties started with different input values, in which case both bits are valid, or
2. the network is asynchronous, in which case a weaker justification proof—signed by only $t_a + 1$ parties—suffices to demonstrate validity.

Downgrading the threshold. Since a justification proof for an input value needs to be validated by other parties, the informal notion of "waiting long enough" must be replaced with a verifiable proof. To this end, we require parties to first broadcast an *intention to downgrade* the threshold from t_s to t_a when either observing that the network is behaving asynchronously or that no bit b that is held by $t_s + 1$ honest parties exists. A downgraded proof of only $t_a + 1$ signed input value is considered valid only if it also includes $t_s + 1$ signed downgrading intentions. This ensures that at least one honest party waited Δ and verified that downgrading does not break validity. Consequently, we are left to define the input justifier for justified Byzantine agreement such that it validates a value if it is accompanied with $(t_s + 1)$-proof for value v or $(t_s + 1)$-proof for downgrading intention along with $(t_a + 1)$-proof for value v.

In the next section, we present and prove the construction of Π_{IVG} and then use the black-box properties of Π_{IVG} and Π_{PVBA} to obtain a binary Byzantine agreement protocol.

4.1 Input Validation Gadget

The protocol Π_{IVG} corresponds to the input-validation step in [CKPS01], where $t + 1$ threshold signatures are used to justify values as being the input of at least one honest party. In their case, having $n > 3t$ guarantees that every party receives signed inputs from at least $n - t$ parties of which at least $\lceil (n - t)/2 \rceil > t$ must be signatures on the same bit meaning that every honest party eventually receives $t + 1$ signatures on some value. In our setting, however, this step is more involved than sending and collecting shares of a $t + 1$ threshold signatures as we cannot know whether the number of corruptions is bounded by t_s or t_a. We first formally define the security of Π_{IVG} as follows:

Definition 8 (Input Validation Gadget). *Let Π be a protocol executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, where each party P_i holds an input $x_i \in \{0, 1\}$*

and may output a value $y_i \in \{0,1\}$ and a proof $\pi_{y_i} \in \{0,1\}^*$. Additionally, Π defines an output justifier J_{OUT}. We say that Π is a secure IVG if the following properties hold:

Liveness: *If every honest party P_i has input x_i, then eventually every honest party P_j outputs y_j, and proof π_{y_i} (where $J_{\text{OUT}}(y_j, \pi_{y_i}) = \top$).*

Justified Validity: *For all outputs y that are J_{OUT}-justified, the input of some honest party P_i is $y_i = y$.*

To construct the input justifier in Fig. 1, we let each party (at the same time) multicast its input bit and then optimistically attempt to collect signature shares toward forming a justification proof on the same bit from at least $t_s + 1$ parties. If at least $t_s + 1$ honest parties hold the same input and the network is synchronous, then this proof is obtained within Δ of the last honest party sending its input. If an honest party receives such messages, it outputs this bit and the threshold signature as a justification proof. Note that since we assume synchronized start of the BA protocol, Δ is enough time to wait.

If a party waits Δ without receiving $t_s + 1$ identical inputs and has not yet terminated, it sends a message declaring this fact and then attempts to collect both (1) $t_s + 1$ such declarations and (2) $t_a + 1$ signed inputs on some bit b. These two components serve distinct purposes: (1) The $t_a + 1$ signatures on b demonstrate that, even in the asynchronous case, b was the input of at least one honest party. (2) The $t_s + 1$ declarations guarantee that at least one honest party has explicitly confirmed that either the network is asynchronous or, if synchronous, then both input bits are valid due to the lack of honest pre-agreement. Consequently, if an honest party collects such messages, it outputs bit b along with both threshold signatures as its justification proof. A final concern is that if most honest parties terminate early according to the optimistic scenario: too few honest parties may remain to satisfy the second scenario. To prevent this, we require every party to forward its justified output upon termination. Consequently, if a party who has not terminated receives a justified output from another party, it adopts this output, forwards it to other parties and terminates.

To ensure overall communication remains $O(n^2\kappa)$, the output is justified by a threshold signature which combines shares from $t_s + 1$ parties directly on the bit, or alternatively only $t_a + 1$ directly on the bit and $t_s + 1$ shares from parties who claim that either there was not honest pre-agreement or that the network is asynchronous.

Theorem 1. *Let $n > t_s + 2t_r$, and $n > 2t_s + t_a$. The protocol Π_{IVG} given in Fig. 1 is a secure IVG as defined in Definition 8, and has $O(n^2\kappa)$ communication complexity and $O(\max(\delta, \Delta))$ running time. Furthermore, whenever at least t_s+1 honest parties start with the same input it has running time at most $O(\delta)$.*

Proof. For every property, we distinguish between synchronous and asynchronous networks.

Justified validity. *Synchronous network*: any justified output bit b must be accompanied by either (1) a $(t_s + 1)$ threshold signature on bit b, or (2) a

Input Validation Gadget Π_{IVG}

- On input b_i, P_i:
  1. records current time τ_i
  2. computes signature shares of threshold signature schemes with thresholds t_a+1 and t_s+1, i.e., $\sigma_{t_a} = \mathsf{Sig}_{t_a}(\mathsf{sk}_i, (b_i, \textsc{Fast})$ and $\sigma_{t_s} = \mathsf{Sig}_{t_s}(\mathsf{sk}_i, (b_i, \textsc{Fast}))$
  3. multicasts $(b_i, \textsc{Fast}, \sigma_{t_s,i}, \sigma_{t_a,i})$
- Upon receiving $(b, \textsc{Fast}, \sigma_{t_s,j}, \sigma_{t_a,j})$ valid signature shares for some bit b from t_s+1 distinct parties, party P_j:
  1. combines the shares of the t_s+1 threshold signature scheme, i.e., $\pi_{y_i} = \mathsf{Combine}_{t_s}(\{\sigma_{t_s,i_1}, \ldots, \sigma_{t_s,i_{t_s+1}}\})$
  2. sets $y_i = b$
  3. multicasts $(y_i, \textsc{SyncQuorum}, \pi_{y_i})$
  4. outputs y_i and π_{y_i}
- At time $\tau_i + \Delta$ if P_i did not yet give output:
  1. It computes a signature share via the threshold signature scheme with threshold t_s+1, i.e., $\sigma_{t_s,i} = \mathsf{Sig}_{t_s}(\mathsf{sk}_i, (\textsc{AsyncOrBothValid}))$
  2. It multicasts $(\textsc{AsyncOrBothValid}, \sigma_{t_s,i})$.
- Upon receiving $(\textsc{AsyncOrBothValid}, \sigma_{t_s,j})$ from t_s+1 distinct parties and $(b, \textsc{Fast}, \sigma_{t_s,j}, \sigma_{t_a,j})$ for some bit b from t_a+1 distinct parties, party P_i:
  1. combines the shares of the t_s+1 threshold signature scheme on $(\textsc{AsyncOrBothValid})$., i.e., $\mathsf{AnyCert} = \mathsf{Combine}_{t_s}(\{\sigma_{t_s,i_1}, \ldots, \sigma_{t_s,i_{t_s+1}}\})$
  2. combines the shares of the t_a+1 threshold signature scheme on $(b, \textsc{Fast})$, i.e., $\mathsf{bCert} = \mathsf{Combine}_{t_a}(\mathsf{vk}, \{\sigma_{t_a,i_1}, \ldots, \sigma_{t_a,i_{t_a+1}}\})$
  3. sets $y_i = b$ and $\pi_{y_i} = \mathsf{bCert}, \mathsf{AnyCert}$. .
  4. multicasts $(y_i, \textsc{AsyncQuorum}, \pi_{y_i})$
  5. outputs $y_i = b$ and π_{y_i}
- Upon receiving a justified output $(y_j, \textsc{SyncQuorum}, \pi_{y_j})$ or $(y_j, \textsc{AsyncQuorum}, \pi_{y_j})$ from P_j, such that $J_{\text{OUT}}(y_j, \pi_{y_j}) = \top$:
  1. sets $y_i = y_j$ and $\pi_{y_i} = \pi_{y_j}$
  2. multicasts $(y_i, \textsc{SyncQuorum}, \pi_{y_i})$ or $(y_i, \textsc{AsyncQuorum}, \pi_{y_i})$ accordingly.

Remark 1. We define $J_{\text{OUT}}(y, \pi_y) = \top$ if $y \in \{0,1\}$ and π_y consists of either (1) a t_s+1 valid threshold signature on $(y, \textsc{Fast})$ or (2) a t_s+1 threshold signature on $(\textsc{AsyncOrBothValid})$ and a t_a+1 threshold signature on $(y, \textsc{Fast})$

Fig. 1. An input validation gadget reducing BA to VBA.

$(t_a + 1)$ threshold signature on bit b and a $(t_s + 1)$ threshold signature on

AsyncOrBothValid. In case (1), the threshold signature implies that at least one honest party signed on b, as there are at most t_s corruptions. In case (2), the party outputting the justification proof must have waited at least Δ time without receiving a proof of the form in (1). This means that no $(t_s + 1)$ signature on b was observed during that period. The absence of such a proof after Δ implies that not all honest parties had the same input, as we assumed perfectly synchronized start. Consequently, the $(t_a + 1)$ threshold signature on b must be some honest party's input as it is a binary protocol.
Asynchronous network: in case (1), the $(t_s + 1)$-threshold signature implies that at least one honest party signed on b as there are at most t_a corruptions in asynchrony and $t_s > t_a$. In case (2) the $(t_a + 1)$ threshold signature on b must include at least one honest party, so b must be the input of at least one honest party. In both cases, b must have been the input of at least one honest party.
Liveness. A party outputs and terminates once it collects either a (1) a $(t_s + 1)$ threshold signature on bit b, or (2) a $(t_a + 1)$ threshold signature on bit b and a $(t_s + 1)$ threshold signature on AsyncOrBothValid. We prove that a party always collects either in both networks. *Synchronous network:* either parties collect $(t_s + 1)$ threshold if started with same input or a $(t_s + 1)$ downgrading threshold within Δ. As $n - t_s > 2t_a$, a $t_a + 1$ certificate on some value b will be formed.
Asynchronous network: observe that eventually $n - t_a > 2t_s$ inputs are sent and the majority bit among them is signed by at least $t_s + 1$ honest parties.
Communication Complexity. In every step, each party sends a message that either contains its own signature or a threshold signature to every other party.
Running Time. The protocol runs for at most 2Δ in a synchronous network, and δ if $t_s + 1$ honest parties have the same input. Note, this is guaranteed in asynchrony, where $t_a < n - 2t_s$, as each honest party receives $t_s + 1$ same input message within δ time.

Given Π_{IVG} and Π_{PVBA} (Definition 5) as two building blocks the protocol shown in Fig. 2 implements binary provable Byzantine agreement as defined in Definition 6.

Binary Provable Byzantine Agreement $\Pi_{\mathbf{PBA}}$

- On input x_i, P_i runs Π_{IVG} with input x_i.
- On output (b_i, π_{b_i}) (for which $\Pi_{\mathsf{IVG}}.J_{\text{OUT}}(b_i, \pi_{b_i}) = \top$) from Π_{IVG}, run $\Pi_{\text{PVBA}}^{\Pi_{\mathsf{IVG}}.J_{\text{OUT}}}$) with input (b_i, π_{b_i}) (i.e., an instance of PVBA where inputs are justified when they are justified outputs of Π_{IVG}).
- On output y_i, π_{y_i} from Π_{PVBA}, output y_i, π_{y_i}.

Fig. 2.

Lemma 1. *Assume Π_{PVBA} is a secure PVBA protocol as per Definition 9 and Π_{IVG} is a secure IVG protocol as per Definition 8. Then, the protocol Π_{PBA} given*

in Fig. 2 is a secure binary provable Byzantine agreement protocol as defined in Definition 6.

Proof. Follows directly from the security definitions shown to be satisfied by Π_{PVBA} and Π_{IVG}. Note that the output of Π_{IVG} is used as an input to Π_{PVBA}.
Justified Validity follows from justified validity property of Π_{IVG} and justified validity property of Π_{PVBA}.
Justified Agreement follows from justified agreement of Π_{PVBA}.

And finally, we conclude this section by stating the theorem that captures our main result, and delegate the proofs to Section 4.3.

Theorem 2. *Assume $n > t_s + 2t_r$, $n > 2t_s + t_a$, and $t_s > t_r > t_a$. The protocol Π_{PBA} given in Fig. 2 is a secure binary provable Byzantine agreement protocol as defined in Definition 6. In particular, it is synchronously secure whenever at most t_s parties are corrupted, asynchronously secure whenever at most t_a parties are corrupted, and responsive provided that (i) at least $t_s + 1$ honest parties begin with the same input and (ii) at most t_r parties are corrupted. In expectation, it has communication complexity $O(n^2\kappa)$ and a running time of $O(\max(\Delta, \delta))$ which is additionally bounded by $O(\delta)$ whenever at most t_r parties are corrupted and $t_s + 1$ honest parties have the same input.*

4.2 Binary Provable Validated Byzantine Agreement

We present our binary provable validated Byzantine agreement protocol, which roughly follows the approach of Feldman and Micali [FM97] using graded agreement (proxcensus) and a common coin to reach consensus in an expected constant number of rounds. At a high level, the protocol proceeds in iterations of proxcensus protocol ($\Pi_{\mathsf{JProx\text{-}}G}$). After each iteration, parties wait for the output $g^r \in \{0, G-1\}$: if the output is the highest grade (biased toward bit 1) or the lowest grade (biased toward bit 0), they can safely terminate with the corresponding value. This is because once a value is assigned the extreme grade, it becomes the only value that can be justified with proof in subsequent iterations; hence no two honest parties can terminate with conflicting justified outputs. If the graded output lies in between, parties update their input for the next iteration using a common coin. The coin takes the same value for all honest parties and thus, justified validity of $\Pi_{\mathsf{JProx\text{-}}G}$ ensures that the following iteration outputs either the highest or lowest grade. Once a party outputs the highest or lowest grade along with a proof, it then forwards them to all parties so that they can also terminate with this value.

In order to avoid explicit use of session identifiers, we will exploit the fact that each invocation of justified proxcensus uses a unique justifier and (abusing notation slightly) use the justifier to distinguish between different sessions.

We start by showing, that if only one proposal b is justified in some iteration, then the output that may become justified in that iteration is b, and there are no justified proposals for subsequent iterations. This is useful for both liveness and

justified agreement. In addition, it directly implies justified validity, as it shows that the protocol stops in iteration $r = 0$ with b as the only justified output if b is the only justified input.

Lemma 2. *Let $n > t_s+2t_r$ and $n > 2t_s+t_a$. Assume $\Pi_{\mathsf{JProx\text{-}9}}$ is a secure* JProx-9 *protocol as per Definition 7. Fix an iteration r. Suppose there is some $b \in \{0,1\}$ such that for all J_r-justified values b': $b' = b$. Then, there are no $J_{r'}$-justified inputs for iterations $r' > r$ and the only value which may be J_{OUT}-justified using a $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}$-justified value as a proof is b.*

Proof. Assume b is the only J_r-justified bit. By *Justified Validity* of JProx-9, if b is the only J_r-justified bit, then $b\cdot(G-1)$ is the only $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}$-justified output value. Hence, the only bit for which J_{OUT} may be satisfied by a $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}$-justified value is b. Since $b \cdot (G-1) \in \{0, 8\}$ is the only $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}$-justified value, no bit can become J_{r+1}-justified for the next iteration, as the J_{r+1} can only be satisfied for grades in the range 1 to 7. And by extension, the same holds for subsequent iterations.

Lemma 3. *Let $n > t_s + 2t_r$, and $n > 2t_s + t_a$. Assume $\Pi_{\mathsf{JProx\text{-}9}}$ is a secure* JProx-9 *protocol as per Definition 7. Then, protocol Π_{PVBA} presented in Fig. 3 terminates in expected* 3 *iterations.*

Proof. The proof follows from (1) Lemma 2 ensuring that if there is only one justified proposal bit in some iteration r, then r is the last iteration, and (2) each iteration establishing that there is only one justified input bit in the following iteration with independent probability at least 1/2 (giving expected at most 2 iterations before Lemma 2 can be applied).

From (1), we only get that no further messages are justified after an iteration r has a unique J_r-justified proposal. We still need argue that parties output and terminate based on this. There are two cases: either all honest parties input and participate in the protocol, then all honest parties obtain an output (from liveness of $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}$). Otherwise, the only reason an honest party would stop participating in $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}$ would be if it had a J_{OUT} justified value. But in that case it would multicast this value before terminating and all honest parties would receive it and terminate.

I only remains to show (2): for any r where both bits are J_r-justified, at most one bit is J_{r+1}-justified with probability at least 1/2. To see that this is true, observe that an honest party only invokes the coin flip protocol Π_{CC} for iteration r upon receiving an output from $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}$. This means that the output of the coin c_r is unpredictable to the adversary until at least one value g_i^r is $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}$-justified. If $g_i^r < 6$, then any other $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}$-justified $g_j^r \neq g_i^r$ would have to be at most 6 (due to justified graded agreement of JProx-9). If $c_r = 0$, then 0 is the only J_r-justified value (as $J_r(1,\pi)$ can only be satisfied if $c_r = 1$ or $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}(7,\pi) = \top$). Since g_i^r is output while c_r is unpredictable to the adversary, this happens with probability 1/2. We argue symmetrically for the where $g_i^r > 3$. In that case any other $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}$-justified $g_j^r \neq g_i^r$ would have to be at least 3 (due to justified graded agreement

of JProx-9). If $c_r = 1$, then 1 is the only J_r-justified value (as $J_r(0, \pi)$ can only be satisfied if $c_r = 0$ or $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}(1, \pi) = \top$). Since g_i^r is output while c_r is unpredictable to the adversary, this happens with probability 1/2.

So, the expected number of iterations before there is a unique J_r justified bit is 2. And that point the protocol terminates for all parties at the latest in iteration J_r.

Provable Validated Byzantine Agreement $\Pi_{\mathbf{PVBA}}{}^{J_{\text{IN}}}$

- Upon receiving input x_i, π_{x_i} where $J_{\text{IN}}(x_i, \pi_{x_i}) = \top$ (locally or from any other party), party P_i initiates an iteration counter $r = 0$ and defines $x_i^0 = x_i$ and $J_0 = J_{\text{IN}}$. P_i then executes the following steps.
  1. P_i runs $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}$ with input $x_i^r, \pi_{x_i^r}$. Let g_i^r and $\pi_{g_i^r}$ be the respective output.
  2. On output from $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}$, P_i invokes Π_{CC} with identifier r.
  3. On output c_r from Π_{CC}, P_i waits until it also has a justified output $g_i^r, \pi_{g_i^r}$ from $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}$ and then acts according to the following rules:
     - If $g_i^r = 0$ (or $g_i^r = 8$): let $y_i = 0$ (or $y_i = 1$) and $\pi_{y_i} = \pi_{g_i^r}$, multicast and output y_i, π_{y_i}.
     - Else if $g_i^r = 1$ (or $g_i^r = 7$): let $x_i^{r+1} = 0$ (or $x_i^{r+1} = 1$) and $\pi_{x_i^{r+1}} = \pi_{g_i^r}$.
     - Otherwise, let $x_i^{r+1} = c_r$ and $\pi_{x_i^{r+1}} = \pi_{g_i^r}$.
- If P_i did not yet output, P_i defines J_{r+1} be the predicate which is true iff either of the following holds:[a]
  - $y = 1$ and $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}(g_i^r = 7, \pi_y)$
  - $y = 0$ and $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}(g_i^r = 1, \pi_y)$
  - $y = c_{r+1}$ and $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}(g_i^r \in [2, 6], \pi_y)$

  It then sets $r = r + 1$, and goes back to the first step of running proxcensus.
- Upon receiving an output y_j and π_{y_j}, such that $\Pi_{\text{PVBA}}{}^{J_{\text{IN}}}.J_{\text{OUT}}(y_j, \pi_{y_j}) = \top$, P_i lets $y_i = y_j$, $\pi_{y_i} = \pi_{y_j}$, multicasts and outputs y_i, π_{y_i}.

Remark 2. We define $\Pi_{\text{PVBA}}{}^{J_{\text{IN}}}.J_{\text{OUT}}(y, \pi_y) = \top$ if either (1) $y = 1$ and $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}(8, \pi_{y_i})$, or $y = 0$ and $\Pi_{\mathsf{JProx\text{-}9}}{}^{J_r}.J_{\text{OUT}}(0, \pi_{y_i})$

[a] Note that it is not satisfied by $g_i^r \in \{0, 8\}$.

Fig. 3.

We prove that the PVBA protocol is secure according to Definition 5, under the assumption that $\Pi_{\mathsf{JProx\text{-}9}}$ is secure according to Definition 7. Furthermore,

its communication and running time asymptotically matches that of $\Pi_{\mathsf{JProx\text{-}9}}$ in expectation. Looking ahead, we will instantiate this theorem with our $\Pi_{\mathsf{JProx\text{-}9}}$ protocol in Corollary 1 to get our main result for validated BA.

Theorem 3. *Assume $\Pi_{\mathsf{JProx\text{-}9}}$ is a secure* JProx-9 *protocol as per Definition 7. The protocol Π_{PVBA} presented in Fig. 3 is a secure provable validated Byzantine agreement protocol according to Definition 5. It has communication complexity* $\max(\text{Comm}(\Pi_{\mathsf{JProx\text{-}9}}), O(n^2\kappa))$*, and running time* $3 \cdot (\text{RunTime}(\Pi_{\mathsf{JProx\text{-}9}}) + \Delta)$ *in expectation.*

Proof.
- **Justified Validity.** If there are two J_{IN}-justified bits, then justified validity is a vacuous statement. So, we only have to show that if there is a unique J_{IN}-justified bit b, then the only J_{OUT}-justified bit is b. But this follows directly from Lemma 2, as the inputs to the first iteration have to be $J_0 = J_{\text{IN}}$-justified. Since only b is J_0-justified, the only bit that can become J_{OUT} justified in iteration 0 is b, and there are no J_r-justified values for $r > 0$.
- **Justified Agreement.** If a bit b is ${\Pi_{\text{PVBA}}}^{J_{\text{IN}}}.J_{\text{OUT}}$-justified, then b is justified by grade $g = 8 \cdot b$ being a ${\Pi_{\mathsf{JProx\text{-}9}}}^{J_r}.J_{\text{OUT}}$-justified output for some round r. Assume that r is the first iteration in which a value $g \in \{0, 8\}$ ${\Pi_{\mathsf{JProx\text{-}9}}}^{J_r}.J_{\text{OUT}}$-justified. We let $b = g = 0$. (The other case ($b = 1$ and $g = 8$) is symmetric.) This means that any other ${\Pi_{\mathsf{JProx\text{-}9}}}^{J_r}.J_{\text{OUT}}$-justified value $g' \neq g = 0$ would be $g' = 1$ as it must satisfy $|g - g'| \leq 1$ by justified graded agreement of $\Pi_{\mathsf{JProx\text{-}9}}$. As J_{r+1} can only be justified by a ${\Pi_{\mathsf{JProx\text{-}9}}}^{J_r}.J_{\text{OUT}}$-justified values in the range 1 to 7 and $g' = 1$ only justifies $b = 0$, there is at most one J_{r+1}-justified value: b. Similarly, for the symmetric case, $b = 1$ and $g = 8$, any J_{r+1}-justified bit would have to be justified by $g' \neq g$ which when restricted by $|g - g'| \leq 1$ leaves $g' = 7$, which can only justify 1 as proposal in iteration $r + 1$. Since b is the only J_{r+1}-justified value, we can now invoke Lemma 2 to say that b is the only value that may become J_{OUT}-justified based on ${\Pi_{\mathsf{JProx\text{-}9}}}^{J_{r+1}}.J_{\text{OUT}}$, and that there are no J_{r+2}-justified proposals. As r was assumed to be the first iteration which gave rise to a J_{OUT}-justified value. The bit b is the only J_{OUT}-justified value.
- **Liveness.** See Lemma 3.
- **Communication and Time Complexity.** From Lemma 3, Π_{PVBA} terminates in three iterations in expectation. In each iteration, Π_{PVBA} invokes $\Pi_{\mathsf{JProx\text{-}9}}$ followed by a common coin protocol, Π_{CC}. From Section 3, Π_{CC} incurs $O(n^2\kappa)$ communication and has a single round of interaction. Each iteration requires a constant number of sequential messages in addition to the running time of $\Pi_{\mathsf{JProx\text{-}9}}$.

4.3 Justified Proxcensus

We abstract an idea from [Kam25] which defines justified proxcensus (c.f. Definition 7) and show how to convert the underlying logic to be sound in the network-agnostic setting with optimal corruption resilience.

The goal is to start from having bit inputs which are justified, and then in each iteration k allow parties to output an integer value in the range from 0 to 2^k

which is justified and satisfies that for all other justified values for the iteration they must be on an adjacent value. That is, after one iteration, parties obtain a value between 0 and 2, with the guarantee that 0 and 2 cannot simultaneously be justified. Similarly, after one more iteration, the outputs obtained comply with justified graded agreement with maximum grade 4, etc. Additionally, if only one input bit b satisfies the predicate J_{IN}, then at each step that input is doubled so that the only J_{OUT}-justified output is $b \cdot 2^k$ to maintain justified validity.

The protocol logic follows the structure of [Kam25] and [FLZL21] by constructing $\Pi_{\mathsf{JProx\text{-}}2s-1}$ from $\Pi_{\mathsf{JProx\text{-}}s}$. In the first round, each party begins by running an instance of the Proxcensus protocol with smaller grade, $\Pi_{\mathsf{JProx\text{-}}s}$, on its local input (x_i, π_{x_i}) that satisfies the justification predicate J_{IN}. Once the party obtains the output z_i and justification proof π_i, it records the current time and generates threshold signature shares for both the responsive threshold $(n - t_r)$ and the synchronous threshold $(n - t_s)$ on the output. It then broadcasts a proposal message containing the output, its proofs, and the two signature shares.

Consequently, if the party receives two justified proposals for two adjacent values z and $z-1$, it means both values are valid and it outputs $y_i = 2z - 1$ and π_{y_i} to be the combination of these two justification proofs, broadcasts this pair, and terminates with that output. Alternatively, If a party collects proposals for the same value z from at least $n - t_r$ distinct parties, it can combine the responsive signature shares into a responsive quorum certificate, FastQuorum. It then outputs $y_i = 2z$ and $\pi_{y_i} =$ FastQuorum, broadcasts this pair, and terminates. This case corresponds to the "optimistic responsive" path. Finally, if neither of these conditions is met quickly, the protocol eventually falls back to a slow path. After waiting for a timeout (two network delays), each party checks whether it has collected proposals for some value z from at least $n - t_s$ distinct parties. If so, it creates and multicasts a synchronous signature share for (SlowProposal, z). Once $t_s + 1$ such synchronous shares are gathered, they can be combined into a quorum certificate. The party then outputs $y_i = 2z$ and $\pi_{y_i} =$ SlowQuorum, broadcasts this pair, and terminates. By following this recursive process, each layer of the protocol either doubles the justified grade (in the responsive or synchronous case) or threads together adjacent outputs (in the mixed case). Over multiple recursive layers, this ensures that validity and agreement are preserved, while guaranteeing termination: in the optimistic case the protocol finishes quickly, but if the network is slow or adversarial, it still falls back to the synchronous guarantees.

The proof proceeds by induction, assuming a secure implementation for $\mathsf{JProx\text{-}}s$ when implementing $\mathsf{JProx\text{-}}2s - 1$. We use the trivial protocol of considering your input x your output y and $J_{\text{OUT}} = J_{\text{IN}}$ as a base case defining $\mathsf{JProx\text{-}}2$. This together with the following induction proof gives us instances of $\mathsf{JProx\text{-}}2^k + 1$ for all $k \in \mathbb{N}$.

The output justifier J_{OUT} is defined via three separate justifiers: J_{FastPair}, $J_{\text{FastQuorum}}$, and J_{Slow}. Towards proving justified graded agreement, we show that there is at most one value justified by each of these predicates individually.

We start by showing this for values justified by the predicate J_{FASTPAIR}, i.e., by a pair of outputs from the inner protocol. Since we inductively assume that the inner protocol satisfies justified graded agreement, any justifying pair is uniquely defined, which makes any resulting value unique.

Lemma 4. *For any two J_{FASTPAIR}-justified values y and y', $y = y'$.*

Proof. If a value y is J_{FASTPAIR}-justified (and honest party has y and π in it view s.t. $J_{\text{FASTPAIR}}(y, \pi) = \top$) then $y = 2z-1$ and the values z and $z-1$ $\Pi_{\mathsf{JProx}\text{-}s}{}^{J_{\text{IN}}}.J_{\text{OUT}}$-justified. Since we assume $\Pi_{\mathsf{JProx}\text{-}s}$ satisfies justified graded agreement, no values apart from z and $z-s$ can be $\Pi_{\mathsf{JProx}\text{-}s}{}^{J_{\text{IN}}}.J_{\text{OUT}}$-justified, thus no value apart from y can be J_{FASTPAIR}-justified in $\Pi_{\mathsf{JProx}\text{-}2s-s}$.

Validated Proxcensus $\Pi_{\mathsf{JProx\text{-}}2s-1}{}^{J_{\text{IN}}}$

We implement $\Pi_{\mathsf{JProx\text{-}}2s-1}$ from $\Pi_{\mathsf{JProx\text{-}}s}$ for $s \geq 2$. Letting $\Pi_{\mathsf{JProx\text{-}}2}{}^{J_{\text{IN}}}$ be the protocol which lets $(y_i, \pi_{y_i}) = (x_i, \pi_{x_i})$ and $J_{\text{OUT}} = J_{\text{IN}}$.

- Upon receiving input x_i, π_{x_i} where $J_{\text{IN}}(x_i, \pi_{x_i}) = \top$ (locally or from any other party), P_i runs $\Pi_{\mathsf{JProx\text{-}}s}{}^{J_{\text{IN}}}$ with input x_i, π_i and waits for it to output z_i, π_{z_i}. Then P_i: records the current time τ_i, computes a signature share for a threshold signature scheme with threshold $n - t_r$, i.e., $\sigma_{n-t_r} = \mathsf{Sig}_{n-t_r}(\mathsf{sk}_i, (\text{FastProposal}, z_i))$ multicasts $(\text{FastProposal}, z_i, \pi_{z_i}, \sigma_{n-t_r})$.
- Upon receiving $(\text{FastProposal}, z, \pi_z, \sigma_{n-t_r})$ and $(\text{FastProposal}, z-1, \pi_{z-1}, \sigma_{n-t_r})$ where $\Pi_{\mathsf{JProx\text{-}}s}.J_{\text{OUT}}(z, \pi_z) = \Pi_{\mathsf{JProx\text{-}}s}.J_{\text{OUT}}(z-1, \pi_{z-1}) = \top$, P_i: sets $y_i = 2z - 1$, and $\pi_{y_i} = \pi_z, \pi_{z-1}$, multicasts $(\text{FastPair}, y_i, \pi_{y_i})$ and terminates with output y_i, π_{y_i}.
- Upon receiving $(\text{FastProposal}, z, \pi_z, \sigma_{n-t_r})$ from $n - t_r$ distinct parties, P_i: combines the shares of the $n - t_r$ threshold signature scheme, i.e., $\pi_{y_i} = \mathsf{Combine}_{n-t_r}(\{\sigma_{n-t_r,i_1}, \ldots, \sigma_{n-t_r,i_{n-t_r}}\})$, sets $y_i = 2z$, multicasts $(\text{FastQuorum}, y_i, \pi_{y_i})$ and terminates with output y_i and π_{y_i}.
- After time $\tau_i + 2\Delta$ and having collected $(\text{FastProposal}, z, \pi_z, \sigma_{n-t_r})$ from at least $n - t_s$ distinct parties, P_i: computes a signature share of threshold signature schemes with thresholds $t_s + 1$, i.e., $\sigma_{t_s+1} = \mathsf{Sig}_{t_s+1}(\mathsf{sk}_i, (\text{SlowProposal}, z))$, multicasts $(\text{SlowProposal}, z, \sigma_{t_s+1})$.
- Upon receiving $t_s + 1$ $(\text{SlowProposal}, z, \sigma_{t_s+1})$ messages from t_s+1 distinct parties, P_i: combines the shares of the t_s+1 threshold signature scheme, i.e., $\pi_{y_i} = \mathsf{Combine}_{t_s+1}(\{\sigma_{t_s,i_1}, \ldots, \sigma_{t_s,i_{t_s+1}}\})$ sets $y_i = 2z$ multicasts $(\text{SlowQuorum}, y_i, \pi_{y_i})$ and terminates with output y_i and π_{y_i}
- Upon receiving output y_j and π_{y_j} such that $J_{\text{OUT}}(y_j, \pi_{y_j}) = \top$, P_i lets $y_i = y_j$, $\pi_{y_i} = \pi_{y_j}$, multicasts and outputs y_i, π_{y_i}.

Remark 3. The justifications predicates are defined as follows:

- $J_{\text{FastPair}}(y, \pi_y) = \top$ if $y = 2z - 1$ and $\pi_y = \pi_z, \pi_{z-1}$ such that $\Pi_{\mathsf{JProx\text{-}}s}{}^{J_{\text{IN}}}.J_{\text{OUT}}(z, \pi_z) = \top$ and $\Pi_{\mathsf{JProx\text{-}}s}{}^{J_{\text{IN}}}.J_{\text{OUT}}(z-1, \pi_{z-1}) = \top$
- $J_{\text{FastQuorum}}(y, \pi_y) = \top$ if $y_i = 2z$ and π_{y_i} is valid $n - t_r$ threshold signature on $(\text{FastProposal}, z, \pi_z)$
- $J_{\text{slow}}(y, \pi_y) = \top$ if $y_i = 2z$ and π_{y_i} is a valid $t_s + 1$ threshold signature on $(\text{SlowProposal}, z, \pi_z)$
- $J_{\text{OUT}}(y, \pi_y) = \top$ if (1) $J_{\text{FastPair}}(y, \pi_y) = \top$, (2) $J_{\text{FastQuorum}}(y, \pi_y) = \top$, or (3) $J_{\text{slow}}(y, \pi_y) = \top$.

Fig. 4. A recursive description of the justified proxcensus protocol.

Next, we show that the same holds for outputs that are justified by the $J_{\text{FastQuorum}}$ predicate, i.e., the other way to give output using the fast path. This is because it is justified by a $n - t_r$ threshold signature, leaving at least one honest contribution in the $n - 2t_r > t_s$ sized intersection of two such quorums.

Lemma 5. *Assume $n > t_s + 2t_r$. For any two values y and y' that are $J_{\text{FastQuorum}}$-justified: $y = y'$.*

Proof. Any two $n - t_r$ threshold signatures justifying y and y' have shares from $n - 2t_r > t_s$ parties in common. Regardless of the network condition, at most t_s parties are corrupted, so $y = y'$ as no honest party sends more than one FastProposal on different values.

To show justified graded agreement, we still need to consider the J_{slow}-justified values which may be produced through the slow path. The proof for such a value is a $(t_s + 1)$-threshold signature which only guarantees that a single honest party sent a SlowProposal. Now, the consistency argument takes two different forms based on whether the network is asynchronous (with only t_a corruptions) or synchronous. In the asynchronous case, we rely on the single honest party who sent SlowProposal having seen $n - t_s$ FastProposal for the same value, and that this makes it unique when there is less than $t_a < n - 2t_s$ corruptions. In the synchronous case, we will instead show that honest parties only send a SlowProposal for a value z if all honest FastProposal messages are also for z.

Lemma 6. *Assume $n > t_s + 2t_r$ and $n > 2t_s + t_a$. For any two values y and y', each of them either $J_{\text{FastQuorum}}$-justified or J_{slow}-justified: $y = y'$.*

Proof. The case where both y and y' are $J_{\text{FastQuorum}}$-justified was covered in Lemma 5. So, WLOG let y be J_{slow}-justified by a $t_s + 1$ threshold signature on some value $z = y/2$, which implies that at least one honest party sent a SlowProposal for z (i.e., sent the message $(\text{SlowProposal}, z, \sigma_{t_s+1})$). We consider the asynchronous and synchronous setting separately:

Asynchronous setting: In asynchrony, we use that the honest party who signed z saw $n - t_s$ FastProposal messages for z before doing so. If this were to happen for another value $z' \neq z$, then it would require that $n - 2t_s > t_a$ parties equivocate, which cannot happen in asynchrony with only t_a corruptions. So, no $y' \neq y$ can be J_{slow}-justified. Similarly, a $y' \neq y$ being $J_{\text{FastQuorum}}$-justified would require $n - t_r - t_s > n - 2t_s > t_a$ parties equivocating, and is also ruled out by the lower corruption threshold assumed for asynchronous security.

Synchronous setting: If the network is synchronous, we rely on the fact that a single honest party sending a SlowProposal for z, implies that every honest party sent a FastProposal for z. To see why this is true: consider the first honest party P_i that sends a FastProposal message and starts the timeout at time τ_i. Because the network is synchronous, all honest parties receive the justified message from P_i no later than $\tau_i + \Delta$. When this happens, they send their own FastProposal which reaches all honest parties no later than $\tau_i + 2\Delta$. Since P was

the first honest party to start waiting, no honest party finishes waiting before $\tau_i + 2\Delta$, at which point all honest parties have seen the FastProposal from all honest parties. If two different FastProposal have been sent by honest parties, then they would be received by all honest parties, and they would already have terminated the protocol before time $\tau_i + 2\Delta$. Thus, any honest party sending a SlowProposal message does so on the single value z that was included in all honest FastProposal messages. This immediately gives that there cannot be a J_{SLOW}-justified or $J_{\text{FASTQUORUM}}$-justified $y' \neq y$ as that would require at least one honest party to send a FastProposal for $y'/2$ as both $n - t_s$ and $n - t_r$ are greater than maximum number of corruptions t_s.

From Lemma 4 and Lemma 6, we conclude that values justified using any of the three predicates (J_{FASTPAIR}, $J_{\text{FASTQUORUM}}$, and J_{SLOW}) satisfy graded agreement.

Lemma 7. *For any J_{OUT}-justified values y and y': $|y - y'| \leq 1$.*

Proof. We know from Lemma 4 and Lemma 6 that there is at most one value that can be J_{FASTPAIR}-justified and at most one value that can be $J_{\text{FASTQUORUM}}$-justified or J_{SLOW}-justified. So, we only need to consider the case where there is one of each and show that their difference is one (they are by definition odd and even integers, respectively). Let y be J_{FASTPAIR}-justified and y' be $J_{\text{FASTQUORUM}}$-justified or $J_{\text{FASTQUORUM}}$-justified. We show that $|y - y'| = 1$. By Justified Graded Agreement of ${\Pi_{\mathsf{JProx\text{-}}s}}^{J_{\text{IN}}}$, we know that the two values z and $z - 1$ justifying $y = 2z - 1$ are the only possible justified outputs of ${\Pi_{\mathsf{JProx\text{-}}s}}^{J_{\text{IN}}}$. Since y' must be two times some justified output from ${\Pi_{\mathsf{JProx\text{-}}s}}^{J_{\text{IN}}}$, we have $y' \in \{2z, 2(z-1)\}$, which means that $y = 2z - 1 = y' \pm 1$. □

We show that the protocol runs for a number of sequential steps which is logarithmic in number of possible outputs G. Each step is of length at most $\max(\delta, \Delta)$ or *if* the number of corruptions is below t_r: only δ.

Lemma 8 (Liveness and Optimistic Responsiveness). *If an honest party has input to $\Pi_{\mathsf{JProx\text{-}}G}$ at time τ, then all honest parties receive a justified output by time $\tau + O(\max(\Delta, \delta) \log G)$. If at most t_r parties are corrupted, then all honest parties receive a justified output by time $\tau + O(\delta \log G)$.*

Proof. We consider the running time of a single iteration and note that the protocol runs for $O(\log G)$ iterations. When the first party sends its FastProposal message at time τ, all honest parties will within one network delay (by time $\tau + \delta$) send a FastProposal message using either their own justified input, or on received over the network. If two honest parties propose different values, then all honest parties can terminate with a J_{FASTPAIR} justified output by time $\tau + 2\delta$. This is because the inner protocol satisfies justified graded agreement, so these two values (which are justified since they are proposed by honest parties) would be some z and $z - 1$, allowing to output the J_{FASTPAIR}-justified value $2z - 1$.

So, we only have to account for the case where all honest send the same value z: Since every honest party sends (FastProposal, $z, \dots$): if there are at most t_r corrupted parties, then every honest party receives $n - t_r$ valid signature shares

for z and can output a $J_{\text{FASTQUORUM}}$-justified value by time $\tau + 2\delta$. This, together with the case where honest parties do send different values above, covers the responsive case for $f \leq t_s$.

If instead there are up to t_s corruptions, then all honest parties receive at least $n - t_s$ valid $(\text{FASTPROPOSAL}, z, \dots)$ messages. Now, it may happen that at least one of them also receives a valid $(\text{FASTPROPOSAL}, z', \dots)$ with $z' \neq z$, allowing any such party to output a J_{FASTPAIR}-justified value. In that case it also forwards its output to all parties allowing them to output. Alternatively, no honest party receives a $(\text{FASTPROPOSAL}, z', \dots)$ message with $z' \neq z$. In that case they all trigger the rule to send a partial $t_s + 1$ threshold signature on $(\text{SLOWPROPOSAL}, z, \cdots)$ by time $\tau + \delta + \max(2\Delta, \delta)$. Consequently, every honest party receives $t_s + 1$ signatures shares on $(\text{SLOWPROPOSAL}, \cdots)$ and terminates.

Theorem 4. *Assume $n > t_s + 2t_a$, $n > t_s + 2t_r$. Then, $\Pi_{\mathsf{JProx}\text{-}G}$ is a secure JProx-G protocol as per Definition 7. Furthermore, it has $O(n^2 \cdot \kappa \cdot G \cdot \log G)$ communication complexity, running time bounded by $O(\max(\Delta, \delta) \log G)$, and additionally bounded by $O(\delta \log G)$ whenever the corruption is less than t_r.*

Proof. – **Justified Graded Agreement.** See Lemma 7.

- **Justified Validity.** Let $G = 2s-1$ and assume for induction that ${\Pi_{\mathsf{JProx}\text{-}s}}^{J_{\text{IN}}}$ satisfies justified validity. Then, if b if the only J_{IN}-justified value, $b \cdot (s-1)$ is the only ${\Pi_{\mathsf{JProx}\text{-}s}}^{J_{\text{IN}}}.J_{\text{OUT}}$-justified value. Now, since any ${\Pi_{\mathsf{JProx}\text{-}2s-1}}^{J_{\text{IN}}}.J_{\text{OUT}}$-justified value would be produced by doubling a single justified output or adding two justified outputs, and $b \cdot (s-1)$ is uniquely ${\Pi_{\mathsf{JProx}\text{-}s}}^{J_{\text{IN}}}.J_{\text{OUT}}$-justified, $2b \cdot (s-1) = b \cdot (2s-1) - 1 = b \cdot (G-1)$ is the only ${\Pi_{\mathsf{JProx}\text{-}2s-1}}^{J_{\text{IN}}}.J_{\text{OUT}}$-justified value.
- **Liveness.** See Lemma 8.
- **Communication complexity.** In every step, each party sends a message that either contains its own signature or up to two justifiers. Each justifier is either a threshold signature or a justifier from the previous round. As there are $\log G$ rounds and potentially expands by a factor 2 per round, the maximum message size is bounded $2^{\log G}\kappa = G\kappa$, thus the communication is bounded by $O(n^2 G \log(G)\kappa)$. (This multiplicative dependence on G can be eliminated using a trick to keep justifiers of constant size described in [Kam25]. But as we only use a small constant G, we sidestep this complication.)
- **Running Time.** See Lemma 8.

This together with Theorem 3 gives us the claimed PVBA result.

Corollary 1. *The protocol Π_{PVBA} in Fig. 3 when instantiated with the protocol $\Pi_{\mathsf{JProx}\text{-}9}$ in Fig. 4 is a secure PVBA protocol according to Definition 5. In expectation, it has communication complexity $O(n^2\kappa)$ and a running time of $O(\max(\Delta, \delta))$ which is additionally bounded by $O(\delta)$ whenever at most t_r parties are corrupted.*

Proof. Follows from instantiating Theorem 3 with Theorem 4.

Finally, we prove the main theorem from Section 4.

Theorem 2. *Assume $n > t_s + 2t_r$, $n > 2t_s + t_a$, and $t_s > t_r > t_a$. The protocol Π_{PBA} given in Fig. 2 is a secure binary provable Byzantine agreement protocol as defined in Definition 6. In particular, it is synchronously secure whenever at most t_s parties are corrupted, asynchronously secure whenever at most t_a parties are corrupted, and responsive provided that (i) at least $t_s + 1$ honest parties begin with the same input and (ii) at most t_r parties are corrupted. In expectation, it has communication complexity $O(n^2\kappa)$ and a running time of $O(\max(\Delta, \delta))$ which is additionally bounded by $O(\delta)$ whenever at most t_r parties are corrupted and $t_s + 1$ honest parties have the same input.*

Proof. **Security:** from Corollary 1, Π_{PVBA} is a secure PVBA protocol as per Definition 5. From Theorem 1, Π_{IVG} is a secure IVG protocol as per Definition 8. Thus, the statement follows from Lemma 1.
Communication Complexity and Running Time: Π_{PBA} runs protocol Π_{IVG} followed by Π_{PVBA}. Adding up the complexities stated in Theorem 1 and Corollary 1, we get that Π_{PBA} has has expected communication complexity $O(n^2\kappa)$ and expected running time of $O(\max(\Delta, \delta))$. Additionally, when the optimistic conditions are met for both Π_{PVBA} (that at most t_r parties are corrupted) and Π_{IVG} (that at least $t_s + 1$ honest parties have the same input), then we get that the expected running time is bounded by $O(\delta)$.

5 Optimality of our Thresholds

In this section we show that the conditions under which we achieve synchronous security, asynchronous security and responsiveness are optimal for both BA and VBA respectively. The bounds hold even for a static adversary.

We first show that VBA requires $n > 3t_a$ for asynchrony, $n > 2t_s + t_a$ for network-agnostic security, and that responsiveness requires that there are fewer than $(n - t_s)/2$ corruptions. This matches the bounds for state machine replication and reliable broadcast; reflecting that in executions where an honest party may decide asynchronously, it must use a quorum which intersects any other such quorum on an honest party, in order to maintain consistency. Finally, we show that for BA, responsiveness can only be achieved when all the previously mentioned conditions hold and additionally $t_s + 1$ honest parties have the same input.

Impossibility results for external validity. Unlike standard Byzantine agreement, VBA [CKPS01] does not require that the decided value is derived from the inputs of honest parties. Rather, validity is defined with respect to an external predicate J_{IN}, requiring only that the decided value satisfies the predicate. Each honest party is assumed to hold an input value that is valid with respect to J_{IN} (together with a corresponding validity proof), while additional values may become valid during the execution once suitable validating information is obtained. That is, parties may decide on a value input by a corrupted party as long as honest parties can validate it.

Definition 9 (Validated Byzantine Agreement). *Let $\Pi^{J_{\text{IN}}}$ be a protocol parameterized by a justification predicate J_{IN}, and executed among parties $\mathcal{P} = \{\mathsf{P}_1, \ldots, \mathsf{P}_n\}$, where each party P_i has an input $x_i \in \{0,1\}$ and proof $\pi_{x_i} \in \{0,1\}^*$, and may output a value $y_i \in \{0,1\}$ and proof $\pi_{y_i} \in \{0,1\}^*$. We say that $\Pi^{J_{\text{IN}}}$ is a secure validated BA protocol if the following properties hold:*

Liveness: *If every honest party P_i has an input x_i, π_{x_i} (where $J_{\text{IN}}(x_i, \pi_{x_i}) = \top$), then eventually every honest party P_j will have output y_j with proof π_{y_j}.*
Agreement: *If honest parties P_i and P_j have output y_i and y_j then $y_i = y_j$.*
External Validity: *If an honest party P_i outputs (y_i, π_i), then $J_{\text{IN}}(y_i, \pi_i) = \top$.*

This means that the typical technique used to prove infeasibility of consensus in certain regimes cannot directly be applied to consensus with external validity. For instance, the extensive characterization of validity conditions by Civit et al. [CGG+23] only considers validity conditions which can be expressed as a function of the honest inputs and explicitly leaves external validity as an open question. The same holds for the study of validity conditions with network-agnostic security by Constantinescu et al. [CDGW25].

This is not an artificial issue. In fact, it is easy to see that if the external predicate is trivial in the sense that satisfying assignments can be computed for all inputs, then VBA can be implemented by locally computing a validity proof for a fixed output value, and thus the number of corruptions is irrelevant. The VBA definition by Cachin et al. [CKPS01] has an *integrity* property designed to rule out such a trivial solution by saying that *if* all parties are honest, then the output should be the input of an honest party. However, this property is typically not included in more recent definitions of the primitive[AMS19,DZJR23,LLTW20]. So, to make our lower bounds apply most generally, including to more recent versions that have been proven useful without relying on integrity, we will instead show that this wider interpretation of VBA is infeasible beyond certain resilience thresholds for a large class of natural predicates. Namely, take any predicate for which there are two inputs for which it is not guaranteed that honest parties obtain a validity proof. Then the standard indistinguishability argument can be applied. We stress that this is not a contrived example but rather the usual case, as validity predicates typically involve verifying cryptographic certificates attesting to some fact external to the protocol. Similarly, while we present the lower bounds for a binary definition of VBA, they also apply to the multi-valued flavor whenever the predicate yields two values that are not trivially satisfied.

In the following proofs for VBA, our arguments typically construct two executions that must lead to different outputs, given that validity proofs for arbitrary values are hard to compute, and then introduce hybrid executions that remain indistinguishable to some honest party, yielding a contradiction with agreement.

Impossibility of Asynchronous VBA for $n \leq 3t_a$. To show that $n = 3t_a$ is impossible in asynchrony, consider an execution in which only t_a parties are corrupted and remain silent while the remaining honest parties are given the same input and a corresponding validity proof with respect to the external predicate. Eventually these honest parties have to terminate as the corruption bound

is respected. Since they only have a validity proof for the input they received, they must output this value. Since these sets may have an intersection of only t_a parties, it is possible to corrupt the entire intersection and simulate views that are identically distributed to the two original executions to break agreement.

Theorem 5. *Assume a fully asynchronous network with $n \leq 3t_a$. There is no protocol that satisfies Validated Byzantine Agreement (Definition 5) against t_a Byzantine corruptions for general predicates J_{IN}.*

Proof. Assume protocol $\Pi^{J_{\text{IN}}}$ is secure for $n = 3t_a$ and let values 0 and 1 have corresponding validity proofs π_0 and π_1. Choose two sets of parties S_0 and S_1 of size $n - t_a$ with intersection $C = S_0 \cap S_1$ of size $n - 2t_a = t_a$.

Consider two executions E_0 and E_1. In execution E_b, all parties in S_b are honest and begin with input b, π_b, and no honest party ever obtains a validity proof for $1 - b$. The remaining parties are crashed, meaning that they are statically Byzantine corrupted and do not send any messages. By liveness, all honest parties in S_b must decide a value, since the protocol is resilient to t_a faults. By external validity, they all decide b as this is the only proof of validity they have seen, they have to output a proof of validity for their output, and we assumed it was computationally hard to compute validity proofs for J_{IN}.

Now construct a third execution E^* in which exactly the parties in C are corrupted, and the remaining parties in S_b receive the same inputs as in E_b. The adversary delays all messages between honest parties in $S_0 \setminus C$ and $S_1 \setminus C$, and corrupts the parties in C. It lets the parties in C equivocate, so as to simulate E_0 toward $S_0 \setminus C$ and E_1 toward $S_1 \setminus C$. Thus, honest parties in $S_0 \setminus C$ decide 0 and honest parties in $S_1 \setminus C$ decide 1, violating agreement.

Impossibility of Network-Agnostic VBA for $n \leq 2t_s + t_a$. To show that network-agnostic VBA cannot be solved for $n = 2t_s + t_a$, we use two synchronous executions. Similarly to before, only one value can be validated in each of these executions for two respective sets S_0 and S_1 of $n - t_s$ honest parties. We then use an asynchronous execution in which for each of the two original (synchronous) executions a party who was honest in one of them has an identically distributed view. To this end, we will use the fact that t_a parties in $S_0 \cap S_1$ equivocate and messages from the remaining t_s now honest parties are being delayed.

Theorem 6. *If $n \leq 2t_s + t_a$ and $t_a < t_s$, no protocol satisfies Validated Byzantine Agreement (Definition 5) against t_a Byzantine corruptions in asynchrony and t_s Byzantine corruptions in synchrony for general predicates J_{IN}.*

Proof. Assume protocol $\Pi^{J_{\text{IN}}}$ is secure for $n = 2t_s + t_a$ and let 0 and 1 have corresponding validity proofs π_0 and π_1. Let S_0 and S_1 be sets of size $n - t_s$ with $|S_0 \cap S_1| = t_a$ (i.e., pick each set as the complement of two disjoint sets of size t_s). Consider two executions E_0 and E_1 in which t_s parties are corrupted and the network is synchronous. By the network being synchronous, we mean that the adversary follows some deterministic message scheduling algorithm which does not violate the Δ bound.

In execution E_b, all parties in S_b are honest and begin with input b, π_b. All parties $\mathcal{P} \setminus S_b$ are corrupted and remain silent. Additionally, the validity proof for the bit $1-b$ never appears in the view of an honest party. By Liveness, all honest parties in S_b eventually output some y, π. By External Validity, $J_{\text{IN}}(y, \pi) = \top$, which means that $y = b$ as they have not received and cannot compute a witness for $1 - b$.

Now construct an *asynchronous* execution E^* in which exactly the parties in $C = S_0 \cap S_1$ are corrupted. This is admissible since $|C| = t_a$. Assign initial inputs in E^* such that parties in $S_0 \setminus C$ start with $0, \pi_0$ and parties in $S_1 \setminus C$ start with $1, \pi_1$. Because the network is asynchronous, the adversary can delay messages arbitrarily between S_b, and because all parties in C are corrupted, they can equivocate and act consistent with E_b towards parties in S_b. We let the adversary deliver all messages between parties in S_b as described by the scheduling algorithm used in execution E_b, and all parties in C act consistently with their input and simulated state in E_b towards the parties in S_b. Thus the adversary makes every honest party in $S_b \setminus C$ have a view which is identically distributed to its view in E_b (until the party gives output), and it must still output b. This violates agreement as there are honest parties in both sets S_0 and S_1.

Impossibility of VBA Responsiveness for $f \geq (n - t_s)/2$. Using a similar argument to Pass and Shi [PS18], we show that VBA cannot be solved with synchronous security for t_s corruptions and responsiveness with t_r corruptions unless $n > t_s + 2t_r$. Note that while the VBA we have presented matches this tightly, it is only secure when additionally $n > 2t_s$. So, in relation to our protocol: the following proof only shows that it is optimally resilient in the regime where we also insist on security in full asynchrony, since this requires $n > 2t_s + t_a$.

While the responsiveness guarantees we prove for our protocols hold except with negligible probability (whenever the optimistic conditions are met), our impossibility results will rule out that a protocol can achieve responsiveness (for all adversaries) with any polynomial probability given "unsound" conditions. This is to rule out that a weaker form of responsiveness could be achieved with any meaningful probability given, too many corruptions or too few honest parties in pre-agreement.

Theorem 7. *A protocol implementing VBA (Definition 9) with synchronous security against t_s corruptions and responsiveness (with polynomial probability) against t_r corruptions must satisfy $n > t_s + 2t_r$.*

Proof. Assume for contradiction that a protocol $\Pi^{J_{\text{IN}}}$ solves VBA with synchronous security against t_s corruptions with non-negligible probability and responsiveness against t_r corruptions where $n = t_s + 2t_r$. Partition the parties into three sets S_0, S_1, and S, such that $|S_0| = |S_1| = t_r$ and $|S| = t_s$. We consider three separate executions:

For both bits $b \in \{0, 1\}$ we consider an execution E_b in which the parties in S_b and S are honest and get input b (each with a proof that b satisfies the validity

predicate), and the parties in S_{1-b} are corrupted and stay silent. As $\Pi^{J_{\text{IN}}}$ is assumed to be responsive for all adversaries with some non-negligibe probability p when only t_r parties are corrupted, we may assume that we restricted to an adversary that delivers all messages within an arbitrarily small δ. Now, let f be the polynomial such that $\delta \cdot f(\kappa)$ upper bounds the running time of Π with polynomial probability p under these conditions, and pick $\delta < \Delta/f(\kappa)$. This allows us to assume that all parties in S_b obtain output by some time τ_b which with probability p is earlier than Δ after the execution starts. Since the parties in S_b only have a proof that b satisfies J_{IN} in execution E_b, their output is b.

Finally, we consider an execution E^* in which the parties in S_b have identical views as in execution E_b: To this end the parties in S are corrupted and simulate the honest behavior from E_b towards the parties in S_b while all messages between S_0 and S_1 are delayed until time $\max(\tau_0, \tau_1) < \Delta$. Since the executions E^* and E_b are indistinguishable in the view of the parties in S_b, their outputs are identically distributed. Thus, for both bits $b \in \{0, 1\}$, honest parties in S_b output b in execution E^* which breaks the agreement property with (non-negligible) probability at least p^2.

Implications for BA, PVBA and PBA. It is easy to see that VBA can be reduced to (binary) BA by letting parties take their VBA input v, π where $J_{\text{IN}}(v, \pi) = \top$, use v as input to a BA instance, and then multicast π. Since the output of the BA is guaranteed to be the input of an honest party, it will satisfy J_{IN} and the corresponding witness π will arrive at all honest parties, allowing them to output in accordance with the VBA definition. Similarly, BA and VBA can trivially be reduced to PBA and PVBA, as these only satisfy the additional condition that the output comes with a proof. Thus the bounds established for VBA in Theorems 5, 6, and 7 apply to BA, PBA, and PVBA.

Impossibility of BA Responsiveness with $\leq t_s$ Honest Support. We finally show that the conditions under which the BA protocol presented in Section 4 is responsive are optimal. Namely, that a BA protocol which is synchronously secure when $f \leq t_s$, can be optimistically responsive only when $f < (n - t_s)/2$ *and* $t_s + 1$ honest parties have the same input. While the $f < (n - t_s)/2$ is needed for consistency and holds because BA implies VBA, the reason that at least $t_s + 1$ need to have the same input is intuitively that there is no way to reliably detect negative events (such as the absence of honest pre-agreement which we used to enable an "anything goes" clause on the slow path in Π_{IVG}) without waiting for an assumed upper bound on the network delay. Thus an honest party can only know that a bit is a valid output if either it was its own input, or if it hears sufficiently many parties that one of them must be honest say that it was their input.

Theorem 8. *Let h_b be the number of honest parties with input b. A BA protocol (Definition 4) with synchronous security against t_s corruptions cannot be responsive (with any polynomial probability) when $h_b \leq t_s$ for both bits $b \in \{0, 1\}$.*

Proof. Assume for contradiction that a protocol Π solves BA and is responsive with at least some polynomial probability p when the maximum number of

honest parties with the same input b is $h_b \leq t_s$. Since Π satisfies responsiveness under this condition, pick an arbitrary execution E where this condition holds, in addition to any other condition that may be required for responsiveness (such as corruptions below a certain threshold). That is, where we may let S_b denote the set of honest parties who have input b and the set of remaining parties who are corrupt be named R giving us:

$$R, \quad S_0, \quad S_1,$$

where $|S_b| \leq t_s$ for $b \in \{0, 1\}$.

As Π is responsive for all adversaries with probability p when these conditions hold, we may assume that we restricted to an adversary that delivers all messages within an arbitrarily small δ. We additionally assume that the adversary immediately corrupts all parties in R and lets them stay silent. Now, let f be the polynomial such that $\delta \cdot f(\kappa)$ upper bounds the running time of Π with probability p under these conditions, and pick $\delta < \Delta/f(\kappa)$. Thus, all honest parties in $S_0 \cup S_1$ finish the execution before one network delay bound Δ has passed, without hearing from the parties in R. Let b^* be the output of Π for some honest party (by the Agreement property this is the same for all parties).

We now want to reach a contradiction by considering a new execution E^* in which we corrupt all parties in S_{b^*}, give all (now honest) parties in R input $1-b^*$ and make the view of some party in S_{1-b^*} identically distributed to its view in E. Since we have at most t_s parties in S_{b^*}, we may corrupt all of them. It is also simple to make the views of all parties in S_{1-b^*} identically distributed to their views in E, as we may let the corrupted parties in S_{b^*} follow the protocol (with the same input as before) and delay all messages from the honest parties in R by Δ, such that they arrive after parties in S_{1-b^*} have decided their output.

In order for this to give us a contradiction, we just need to establish that both sets S_0 and S_1 are non-empty.

To this end, first note that the feasibility of synchronously secure BA under t_s corruptions is known [DLP$^+$86] to imply $n > 2t_s$. Now, assume for contradiction that $|R| = 0$. Then, as the sets form a partition of the n parties, $|S_0 \cup S_1| = n > 2t_s$, which violates the assumption that both sets were of size at most t_s. As all parties in R were corrupted in execution E, which was assumed to be responsive: Π can be responsive against some non-zero number of corruptions $t_r \geq |R| > 0$.

Now, due to Theorem 6 (which also applies to BA) we have that $n > t_s + 2t_r$. I.e., that $n > t_s + t_r + t_r \geq t_s + t_r + 1$ which as $t_r \geq |R|$ implies that $|S_0 \cup S_1| \geq 1 + t_s$. So, if one of the two sets S_0 and S_1 would be empty, then the other would contain at least $t_s + 1$ parties, violating the assumption that no bit was input by that many parties in execution E.

This gives us that both sets are non-empty, and thus that at least one honest party is contained in S_{1-b^*}. This honest parties has identically distributed views in the two executions E and E^*, and thus must (assuming it does not violate agreement in E) output b^* in both, which violates validity in E^* where all honest parties input $1 - b^*$.

6 Conclusion and Future Directions

In this work, we unify two previously distinct directions in consensus, network agnostic security and optimistic responsiveness, by constructing the first Byzantine agreement (BA) and validated Byzantine agreement (VBA) protocols that achieve both properties simultaneously. A central technical contribution is the relocation of the logical switch between synchronous and asynchronous reasoning from the global protocol structure to the step level. This step-level modularity enables responsiveness without sacrificing network-agnostic guarantees and is key to achieving optimal trade-offs across all three regimes. We complement our constructions with matching impossibility results, showing that the resilience tradeoffs are tight.

An interesting direction for future work is to extend this protocol design paradigm to other core distributed computing problems, including multi-valued Byzantine agreement (MVBA), verifiable secret sharing (VSS), multi-party computation (MPC), and state machine replication (SMR). A central open question is whether these tasks can simultaneously achieve dual-threshold (network-agnostic) security and optimistic responsiveness within a unified framework. To the best of our knowledge, MVBA in particular has not even been explored in a network-agnostic setting, making it an open and promising direction for future research as it is used as a building block in communication efficient constructions of MPC and SMR in the computational setting.

References

ABKL22. Andreea B. Alexandru, Erica Blum, Jonathan Katz, and Julian Loss. State machine replication under changing network conditions. In Shweta Agrawal and Dongdai Lin, editors, *ASIACRYPT 2022, Part I*, volume 13791 of *LNCS*, pages 681–710, Taipei, Taiwan, December 5–9, 2022. Springer, Cham, Switzerland.

ACC22. Ananya Appan, Anirudh Chandramouli, and Ashish Choudhury. Perfectly-secure synchronous MPC with asynchronous fallback guarantees. In Alessia Milani and Philipp Woelfel, editors, *41st ACM PODC*, pages 92–102, Salerno, Italy, July 25–29, 2022. ACM.

ACCI25. Gilad Asharov, Anirudh Chandramouli, Ran Cohen, and Yuval Ishai. Peeking into the future: MPC resilient to super-rushing adversaries. In *EUROCRYPT (5)*, volume 15605 of *Lecture Notes in Computer Science*, pages 390–420. Springer, 2025.

AMN+20. Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync HotStuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy*, pages 106–118, San Francisco, CA, USA, May 18–21, 2020. IEEE Computer Society Press.

AMS19. Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In Peter Robinson and Faith Ellen, editors, *38th ACM PODC*, pages 337–346, Toronto, ON, Canada, July 29 – August 2, 2019. ACM.

ANS21. Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. Optimal good-case latency for rotating leader synchronous BFT. In *OPODIS*, volume 217 of *LIPIcs*, pages 27:1–27:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

BCLZL23. Renas Bacho, Daniel Collins, Chen-Da Liu-Zhang, and Julian Loss. Network-agnostic security comes (almost) for free in DKG and MPC. In Helena Handschuh and Anna Lysyanskaya, editors, *CRYPTO 2023, Part I*, volume 14081 of *LNCS*, pages 71–106, Santa Barbara, CA, USA, August 20–24, 2023. Springer, Cham, Switzerland.

BKL19. Erica Blum, Jonathan Katz, and Julian Loss. Synchronous consensus with optimal asynchronous fallback guarantees. In Dennis Hofheinz and Alon Rosen, editors, *TCC 2019, Part I*, volume 11891 of *LNCS*, pages 131–150, Nuremberg, Germany, December 1–5, 2019. Springer, Cham, Switzerland.

BKL21. Erica Blum, Jonathan Katz, and Julian Loss. Tardigrade: An atomic broadcast protocol for arbitrary network conditions. In Mehdi Tibouchi and Huaxiong Wang, editors, *ASIACRYPT 2021, Part II*, volume 13091 of *LNCS*, pages 547–572, Singapore, December 6–10, 2021. Springer, Cham, Switzerland.

BZL20. Erica Blum, Chen-Da Liu Zhang, and Julian Loss. Always have a backup plan: Fully secure synchronous MPC with asynchronous fallback. In Daniele Micciancio and Thomas Ristenpart, editors, *CRYPTO 2020, Part II*, volume 12171 of *LNCS*, pages 707–731, Santa Barbara, CA, USA, August 17–21, 2020. Springer, Cham, Switzerland.

CDGW25. Andrei Constantinescu, Marc Dufay, Diana Ghinea, and Roger Wattenhofer. Validity in network-agnostic byzantine agreement. In *DISC*, volume 356 of *LIPIcs*, pages 24:1–24:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

CGG+23. Pierre Civit, Seth Gilbert, Rachid Guerraoui, Jovan Komatovic, and Manuel Vidigueira. On the validity of consensus. In Rotem Oshman, Alexandre Nolin, Magnús M. Halldórsson, and Alkida Balliu, editors, *42nd ACM PODC*, pages 332–343, Orlando, FL, USA, June 19–23, 2023. ACM.

CGWW24. Andrei Constantinescu, Diana Ghinea, Roger Wattenhofer, and Floris Westermann. Convex consensus with asynchronous fallback. In *DISC*, volume 319 of *LIPIcs*, pages 15:1–15:23. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2024.

CKPS01. Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In Joe Kilian, editor, *CRYPTO 2001*, volume 2139 of *LNCS*, pages 524–541, Santa Barbara, CA, USA, August 19–23, 2001. Springer Berlin Heidelberg, Germany.

CKS00. Christian Cachin, Klaus Kursawe, and Victor Shoup. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography (extended abstract). In Gil Neiger, editor, *19th ACM PODC*, pages 123–132, Portland, OR, USA, July 16–19, 2000. ACM.

CR93. Ran Canetti and Tal Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *25th ACM STOC*, pages 42–51, San Diego, CA, USA, May 16–18, 1993. ACM Press.

DE24. Giovanni Deligios and Mose Mizrahi Erbes. Closing the efficiency gap between synchronous and network-agnostic consensus. In Marc Joye and Gregor Leander, editors, *EUROCRYPT 2024, Part V*, volume 14655 of *LNCS*, pages 432–461, Zurich, Switzerland, May 26–30, 2024. Springer, Cham, Switzerland.

DHLZ21. Giovanni Deligios, Martin Hirt, and Chen-Da Liu-Zhang. Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In Kobbi Nissim and Brent Waters, editors, *TCC 2021, Part I*, volume 13042 of *LNCS*, pages 623–653, Raleigh, NC, USA, November 8–11, 2021. Springer, Cham, Switzerland.

DLP+86. Danny Dolev, Nancy A. Lynch, Shlomit S. Pinter, Eugene W. Stark, and William E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33(3):499–516, 1986.

DY81. Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols (extended abstract). In *22nd FOCS*, pages 350–357, Nashville, TN, USA, October 28–30, 1981. IEEE Computer Society Press.

DZJR23. Xiaohai Dai, Bolin Zhang, Hai Jin, and Ling Ren. ParBFT: Faster asynchronous BFT consensus with a parallel optimistic path. In Weizhi Meng, Christian Damsgaard Jensen, Cas Cremers, and Engin Kirda, editors, *ACM CCS 2023*, pages 504–518, Copenhagen, Denmark, November 26–30, 2023. ACM Press.

FLZL21. Matthias Fitzi, Chen-Da Liu-Zhang, and Julian Loss. A new way to achieve round-efficient byzantine agreement. In Avery Miller, Keren Censor-Hillel, and Janne H. Korhonen, editors, *40th ACM PODC*, pages 355–362, Virtual Event, Italy, July 26–30, 2021. ACM.

FM97. Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM J. Comput.*, 26(4):873–933, 1997.

GGLZ22. Diana Ghinea, Vipul Goyal, and Chen-Da Liu-Zhang. Round-optimal byzantine agreement. In Orr Dunkelman and Stefan Dziembowski, editors, *EUROCRYPT 2022, Part I*, volume 13275 of *LNCS*, pages 96–119, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

GLZW22. Diana Ghinea, Chen-Da Liu-Zhang, and Roger Wattenhofer. Optimal synchronous approximate agreement with asynchronous fallback. In Alessia Milani and Philipp Woelfel, editors, *41st ACM PODC*, pages 70–80, Salerno, Italy, July 25–29, 2022. ACM.

GP90. Oded Goldreich and Erez Petrank. The best of both worlds: Guaranteeing termination in fast randomized byzantine agreement protocols. *Inf. Process. Lett.*, 36(1):45–49, 1990.

Kam25. Simon Holmgaard Kamp. Short paper: A new way to achieve round-efficient asynchronous byzantine agreement. In *FC (2)*, volume 15752 of *Lecture Notes in Computer Science*, pages 160–169. Springer, 2025.

LLTW20. Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-MVBA: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In Yuval Emek and Christian Cachin, editors, *39th ACM PODC*, pages 129–138, Virtual Event, Italy, August 3–7, 2020. ACM.

LM18. Julian Loss and Tal Moran. Combining asynchronous and synchronous byzantine agreement: The best of both worlds. Cryptology ePrint Archive, Report 2018/235, 2018.

LSP82. Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Trans. Program. Lang. Syst.*, 4(3):382–401, jul 1982.

LZLM+20. Chen-Da Liu-Zhang, Julian Loss, Ueli Maurer, Tal Moran, and Daniel Tschudi. MPC with synchronous security and asynchronous responsiveness. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 92–119, Daejeon, South Korea, December 7–11, 2020. Springer, Cham, Switzerland.

MCK20. Atsuki Momose, Jason Paul Cruz, and Yuichi Kaji. Hybrid-BFT: Optimistically responsive synchronous consensus with optimal latency or resilience. Cryptology ePrint Archive, Report 2020/406, 2020.

MR21. Atsuki Momose and Ling Ren. Optimal communication complexity of authenticated byzantine agreement. In *DISC*, volume 209 of *LIPIcs*, pages 32:1–32:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021.

PS17. Rafael Pass and Elaine Shi. Hybrid consensus: Efficient consensus in the permissionless model. In *DISC*, volume 91 of *LIPIcs*, pages 39:1–39:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

PS18. Rafael Pass and Elaine Shi. Thunderella: Blockchains with optimistic instant confirmation. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 3–33, Tel Aviv, Israel, April 29 – May 3, 2018. Springer, Cham, Switzerland.

SARN20. Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the optimality of optimistic responsiveness. In Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna, editors, *ACM CCS 2020*, pages 839–857, Virtual Event, USA, November 9–13, 2020. ACM Press.