Closing the Efficiency Gap between Synchronous and Network-Agnostic Consensus[⋆]

Giovanni Deligios and Mose Mizrahi Erbes

ETH Zurich, Zürich, Switzerland
{gdeligios,mmizrahi}@ethz.ch

Abstract. In the consensus problem, n parties want to agree on a common value, even if some of them are corrupt and arbitrarily misbehave. If the parties have a common input m, then they must agree on m.

Protocols solving consensus assume either a *synchronous* communication network, where messages are delivered within a known time, or an *asynchronous* network with arbitrary delays. Asynchronous protocols only tolerate $t_a < n/3$ corrupt parties. Synchronous ones can tolerate $t_s < n/2$ corruptions with setup, but their security completely breaks down if the synchrony assumptions are violated.

Network-agnostic consensus protocols, as introduced by Blum, Katz, and Loss [TCC'19], are secure regardless of network conditions, tolerating up to t_s corruptions with synchrony and t_a without, under provably optimal assumptions $t_a \leq t_s$ and $2t_s + t_a < n$. Despite efforts to improve their efficiency, all known network-agnostic protocols fall short of the asymptotic complexity of state-of-the-art purely synchronous protocols.

In this work, we introduce a novel technique to compile *any* synchronous and *any* asynchronous consensus protocols into a network-agnostic one. This process only incurs a small constant number of overhead rounds, so that the compiled protocol matches the optimal round complexity for synchronous protocols. Our compiler also preserves under a variety of assumptions the asymptomatic communication complexity of state-of-the-art synchronous and asynchronous protocols. Hence, it closes the current efficiency gap between synchronous and network-agnostic consensus.

As a plus, our protocols support ℓ-bit inputs, and can be extended to achieve communication complexity $\mathcal{O}(n^2\kappa + \ell n)$ under the assumptions for which this is known to be possible for purely synchronous protocols.

1 Introduction

1.1 Motivation

Consensus, or byzantine agreement, is a fundamental problem in distributed computing and cryptography. A consensus protocol enables n parties with inputs to agree on a common output by communicating via bilateral channels, even when some parties maliciously deviate from the protocol. Pre-agreement among honest parties on a common input must be preserved by a consensus protocol.

[⋆] This is the full version of a paper appearing in Eurocrypt 2024.

The problem has a decades-long research history [27,13,8,10] and consensus protocols serve as building blocks for more complex tasks, such as distributed key generation and multi-party computation. In the last decade, the emergence of blockchain applications [34,26] sparked renewed interest in the problem, and the number of consensus protocols implemented and deployed is now higher than ever. Interestingly, the security of deployed protocols relies on assumptions on the underlying communication network which are, in practice, not always satisfied, leading to serious security failures [23].

The two dominant communication abstractions are known as the *synchronous* model and the *asynchronous* model. In the synchronous model, messages are delivered within some publicly known time Δ after being sent, and parties have access to a global clock. This is in contrast to the asynchronous setting, where messages are delivered with arbitrary (finite) delays decided by an adversary, and parties' local clocks need not be synchronized.

Protocol design is significantly simpler in the synchronous model: communication can proceed in rounds where each party waits for messages from *all* other parties before computing their next message. Consensus protocols in this model can with setup achieve statistical or cryptographic security against less than $\frac{n}{2}$ malicious parties, but their security completely breaks down if even a *single* message is dropped or delayed.

The unpredictability of the asynchronous model requires a more involved protocol design, as one cannot differentiate between a corrupt party not sending a message and an honest party whose message is delayed. Given the minimal assumptions of this model, the achievable security guarantees are unsurprisingly weaker: asynchronous consensus protocols can tolerate less than $\frac{n}{3}$ malicious parties. On the positive side, in the real world, asynchronous protocols are resilient to very adverse network conditions.

Starting with [6], a recent line of works tries to marry the better resiliency of synchronous protocols with the tolerance of asynchronous protocols to adverse network conditions. The goal is to design *network-agnostic* protocols in which parties are unaware of the network conditions at execution time. If synchrony is satisfied throughout the execution, then the protocol should be secure if up to a threshold t_s of parties misbehave. However, even if some of the synchrony assumptions are violated during the execution, the protocol should still tolerate a lower threshold t_a of corruptions.

In this setting, consensus (with setup) is possible if and only if $t_a \leq t_s$ and $2t_s + t_a < n$ [6]. Observe that if the threshold t_a is set to 0, one has the optimal resilience of synchronous consensus protocols. Furthermore, even if the network is asynchronous, the protocol still achieves security when all parties honestly follow it. This is not true for purely synchronous protocols, where an adversary who controls the network can typically disrupt the execution of a protocol by simply delaying a *single* message.

Currently, known network-agnostic protocols for consensus and multi-party computation mostly follow some variation of the following approach: 1) Identify a synchronous and an asynchronous protocol for the task at hand, 2) Enhance both so that the synchronous protocol provides *some* weak guarantees even when the network is not synchronous and the asynchronous protocol provides *some* stronger guarantees (than it normally would) if synchrony holds, and finally 3) Run some clever combination of the two enhanced protocols to obtain a network-agnostic protocol with full security.

Despite proving quite effective, this design approach has two major downsides. First, if one wishes to replace the synchronous or asynchronous components with different protocols (because more efficient constructions are discovered, or because for a specific application the cryptographic primitives or setup assumptions required are not available), then the enhancement step 2) above must be carried out from scratch. This step is typically the most technically involved, and even if successful, new security proofs are required. Second, solving the design challenges of step 2) typically incurs a large complexity overhead.

These observations naturally lead to the following question:

> *Is there a complexity-preserving way to combine in a black-box fashion synchronous and asynchronous consensus protocols into a network-agnostic protocol?*

For round complexity, we answer this question affirmatively with a compiler for network-agnostic consensus which makes only a single *black-box* use of *any* synchronous and asynchronous consensus protocols. The compiler only incurs a *constant* overhead in the number of rounds, and it does *not* run the asynchronous protocol if the network happens to be synchronous. Hence, we close up to a constant number of rounds any round complexity gap between purely synchronous and network-agnostic consensus protocols, including the asymptotic gap that remained open between [20] and [11].

Our construction is the most efficient known, and it preserves the asymptomatic communication complexity of the best-known synchronous and asynchronous protocols when they are used in the compiler. As a plus, it supports long inputs natively and more efficiently than previous network-agnostic protocols, despite only invoking consensus protocols for single-bit inputs. For security we rely on the (provably necessary) trade-off $2t_s + t_a < n$ and the existence of unforgeable signatures, which can be instantiated from any setup necessary for honest majority synchronous consensus.

There is no known result or heuristic evidence implying that network-agnostic protocols should be inherently less efficient than purely synchronous protocols *for the same task*, assuming network synchrony holds. Recent work [4] shows that network-agnostic distributed key generation incurs no such loss. Our work goes in the same direction, and shows that no inherent efficiency loss is incurred for the network-agnostic security of consensus.

1.2 Technical Overview

Starting Point. A consensus protocol is run among n parties: each party holds an input m in some alphabet $\mathcal{M}$. A consensus protocol achieves *consistency* if all honest parties (parties following the prescribed protocol) output the same value. It achieves *validity* if whenever all honest parties have the same input m, they all output m (informally, we say that pre-agreement is preserved).

If a protocol achieves a property, for example consistency, only if the assumptions of the synchronous model are satisfied, we say that it achieves *synchronous consistency*. We say it simply achieves *consistency* if it achieves consistency regardless of the assumptions on the network (in particular, in the weakest model possible, the asynchronous model). Similarly, we say it achieves t-*consistency* if consistency holds only if at most t parties deviate from the protocol.

Protocols with synchronous t_s-validity and synchronous t_s-consistency (that we denote by SBA) exist for all $t_s < \frac{n}{2}$ assuming setup [13], while consensus protocols with t_a-validity and t_a-consistency *even in asynchronous networks* (which we denote by ABA) only exist for $t_a < \frac{n}{3}$, even with setup [10].

Blum, Katz, and Loss [6] first proposed a way to adapt such SBA and ABA protocols to obtain a network-agnostic protocol. Assuming $2t_s + t_a < n$ (which they prove to be necessary) they obtain a protocol HBA with the following properties 1) synchronous t_s-validity 2) t_a-validity 3) synchronous t_s-consistency, and 4) t_a-consistency.

The paradigm they use has been adopted in later works [11,4]. The idea is to run SBA and ABA in succession. First, the input to HBA is used as input to SBA, and then the output from SBA as input to ABA. Finally, the output from ABA is taken to be the output from HBA. However, without modifications, this simple procedure does not provide the wanted guarantees. The first problem is that SBA provides no security guarantees whatsoever if the network is not synchronous. In particular, it does not preserve pre-agreement between honest parties; so, t_a-validity does not hold in the overall protocol HBA. The second problem is that when the network is synchronous but $t_s > t_a$, protocol ABA provides no security guarantees. This means that synchronous t_s-validity and synchronous t_s-consistency do not hold for HBA.

Their solution is clever. Consider the following weaker notion of validity, which we call *fallback validity*: if all honest parties have the same input $m \in \mathcal{M}$, then they all output m or abort the protocol. Assume the existence of a protocol SBA^* which, in addition to the properties of SBA, achieves t_a-fallback validity.

Now, run in succession SBA^* and ABA, but any party who aborts in SBA^* uses their original input as input to ABA. Now, even if the network is asynchronous, pre-agreement on an input m among honest parties is preserved: thanks to t_a-fallback validity, all parties output m from SBA^* or abort. Either way, they all input m to ABA, and pre-agreement is preserved by the t_a-validity of ABA.

To fix the second problem, assume the existence of a protocol ABA^* which, in addition to the properties of ABA achieves synchronous t_s-validity,[1] and replace ABA with ABA^* in HBA. Observe that, if the network is synchronous, by the

[1] The protocol ABA^* should also have certain termination properties if the network is synchronous, but such details are not needed to appreciate this technical overview.

synchronous t_s-consistency of SBA^*, honest parties are always in pre-agreement before executing ABA^* (either on the pre-agreed upon input, or on some arbitrary value if no pre-agreement was present before executing SBA^*). Therefore, the synchronous t_s-validity of ABA^* suffices for the security of HBA.

Constructing protocols SBA^* and ABA^* with the required properties is a significant design challenge which fundamentally exploits the assumption $2t_s + t_a < n$. In [6], Dolev-Strong broadcast [13] and the asynchronous consensus protocol from [31] are used as starting points. The resulting network-agnostic protocol requires setup for unique threshold signatures and runs in $\mathcal{O}(n)$ rounds when the network is synchronous. In [11], Deligios and Hirt design a new SBA^* by modifying the synchronous protocol from [15]. Again, unique threshold signatures are needed, but the new protocol runs in a constant (in n) number of rounds when the network is synchronous, with an error probability $\mathcal{O}(c^{-r})$ in r rounds for some constant $c > 1$. At the time, this matched the asymptotic round complexity of the most efficient purely-synchronous protocols known. However, more recently, [20] showed an SBA protocol which under appropriate assumptions can achieve an error probability of $\mathcal{O}((c \cdot r)^{-r})$ in r rounds for some constant $c > 0$, which has long been known to be optimal [24], thus reopening the gap between synchronous and network-agnostic protocols. This gap remained, until now, open.

Closing the round-efficiency gap. Design-wise, it is clearly sub-optimal to repeat the process of enhancing SBA with t_a-validity and rewriting security proofs to obtain a corresponding SBA^* whenever *any new synchronous protocol* is published. To make matters worse, certain SBA protocols (including [20]) seem to be inherently less friendly to such adaptations.

For this reason, we propose a new enhancing technique to obtain a protocol with the properties required from SBA^* that invokes *any* SBA protocol in a black-box fashion, and only introduces a constant overhead in the number of rounds required. We stress that this construction can be instantiated with any SBA protocol, including [20]. We begin by adding t_a-fallback validity to a weaker agreement primitive (a flavor of *graded consensus*), whose security only relies on an abstract form of digital signatures (which can also be instantiated with information theoretic security [38]). Our graded-consensus intuitively provides the validity guarantees required from SBA^* when synchrony does not hold, so that the role of the underlying SBA is reduced to providing full agreement when the network is synchronous. This allows us to use SBA in a black-box way.

However, plugging the SBA from [20] into our construction for SBA^* does not suffice: to achieve network-agnostic security, protocol HBA requires running both the SBA^* and the ABA^* sub-protocols regardless of the network conditions. Unfortunately, a classical lower bound shows that in r rounds, an asynchronous consensus protocol can only achieve consistency and validity with error probability $\mathcal{O}(c^{-r})$ for some constant $c > 1$ [3]. This means that even running an *optimal* ABA^* protocol when the network is synchronous would increase the error of the overall protocol from $\mathcal{O}((c \cdot r)^{-r})$ to $\mathcal{O}(c^{-r})$ in r rounds.

To fix this, we propose a new way to enhance *any* ABA protocol to fulfill the properties of ABA^* that only requires black-box access to the underlying ABA protocol. We observe that when the network is synchronous, the SBA^* protocol guarantees that honest parties are always in pre-agreement upon entering the protocol ABA^*. We exploit this by designing a termination procedure that, when the network is synchronous, is triggered within a small constant number of rounds, causing ABA^* to terminate *without* running the underlying ABA. This means that the underlying ABA protocol is only run when the network is asynchronous, which in turn allows us to use it in a black-box way.

Multi-valued inputs. Our protocols actually support inputs from any alphabet $\mathcal{M}$. When $\mathcal{M} = \{0, 1\}$, validity guarantees that the common output of honest parties is the input of at least one honest party. For large input spaces, this desirable property is unattainable unless $t \cdot |\mathcal{M}| < n$. Instead, it is common to only require that the common output is either the input of some honest party or a special symbol $\bot$: this property is called intrusion tolerance. Since it requires no extra work, our consensus protocols guarantee that the common output is the input of at least δn honest parties whenever $2t_s + t_a \leq (1 - \delta)n$ and δn is a positive integer. This is without loss of generality, as the requirement that t_a, t_s and n are integers allows us to simply take δn to be $n - 2t_s - t_a$.

Overview of our construction. Our novel SBA^* construction has two components: the first is *any* SBA protocol, and the second is a weaker agreement primitive we call SGC^2 (2-graded consensus), which provides the network-agnostic security guarantees. We build increasingly strong agreement primitives towards SGC^2. All these primitives achieve the validity and fallback validity properties, but provide increasingly strong consistency guarantees. SGC^2 is sufficiently strong so that combining it with SBA one finally obtains full security. We then mimic this approach for our ABA^* protocol.

Protocol SWC***.*** We begin with SWC (synchronous weak consensus). The protocol is simple: in the first round parties sends their signed inputs (with some signature scheme) to everyone. Any party receiving less than $n - t_s$ messages is *sure* that the network is asynchronous (or they would have received messages from all honest parties) and it simply aborts the protocol. Otherwise, in the second round, if a party has received at least $t_s + \delta n$ copies of the same validly signed value m from different parties, it sets its tentative output to m, combines the signatures on m into a *certificate*, and sends the certificate to everyone in an effort to prevent inconsistencies. Upon seeing a certificate on a value different from its tentative output, a party changes its output to $\bot$. In a synchronous network the protocol achieves t_s-validity and a weaker notion of t_s-consistency: there is some $m \in \mathcal{M}$ such that each honest party outputs either m or $\bot$, because before outputting some $m \in \mathcal{M}$, a party sends a certificate on m to everyone, preventing other parties from outputting any $m' \in \mathcal{M} \setminus \{m\}$. Notice that the protocol also achieves t_a-fallback validity: informally, if the network is asynchronous but honest parties have pre-agreement on m, an honest party who does not abort in the first round has received at least $n - t_s$ validly signed messages, and of these, at most t_a come from corrupted parties. Therefore, the assumption $2t_s + t_a \leq (1 - \delta)n$ guarantees that at least $n - t_s - t_a \geq t_s + \delta n$ of the signed inputs hail from honest parties, who all sign the input m.

Protocol* SProp*. The next building block for SGC^2 is the protocol SProp (synchronous proposal). In a proposal protocol, honest parties have inputs in some common set $S = \{x, \bot\} \subseteq \mathcal{M}^{\bot} = \mathcal{M} \cup \{\bot\}$, and they output either x, $\bot$, or the set $\{x, \bot\}$. The protocol description and the security properties achieved are similar to SWC: pre-agreement on an input should be preserved, and it should not happen that an honest party outputs x while another outputs $\bot$. However, it is allowed that an honest party outputs x or $\bot$ while another outputs $\{x, \bot\}$. Indeed, the mapping $(x, \{x, \bot\}, \bot) \longrightarrow (0, \bot, 1)$ *actually* yields a weak-consensus protocol for binary inputs. The crucial difference which we exploit is that for proposal, the set $S = \{x, \bot\}$ doesn't have to be known a priori: honest parties with the input $\bot$ do not need to know what x is.

Protocols* SGC^1 *and* SGC^2*. Protocols SGC^1 and SGC^2 are different flavors of a primitive called graded consensus. They invoke SWC and SProp as sub-protocols. In graded consensus, each party has an input $x \in \mathcal{M}^{\bot}$ and outputs a value $y \in \mathcal{M}^{\bot}$, together with a grade $g \in (0, k)$. Intuitively, the grade measures "how certain" a party is of its output. The grades of honest parties must differ by *at most* 1, and if an honest party has a non-zero grade, then all honest parties must have the same value y (graded consistency). In addition, if the honest parties are in pre-agreement on a value y, then all honest parties must output y with the *maximum grade* $g = k$ (graded validity). The higher k is, the harder it is to achieve these properties.

Protocol SGC^1 has a maximum grade $k = 1$. First, the parties run an instance of SWC on their inputs. If their outputs here matches their inputs, they repeat their inputs to an instance of SProp. Otherwise, they input $\bot$ to SProp. Finally, the mapping $(m, \{m, \bot\}, \bot) \longrightarrow ((m, 1), (m, 0), (\bot, 0))$ is applied to the SProp outputs. Protocol SWC guarantees that the inputs of honest parties are valid inputs for SProp, and then SProp provides graded consistency. Indeed, if a party outputs a value x with grade 1, this means that it output x from SProp, and no honest party has output $\bot$ from SProp. This shows all honest parties output x with grade 1 or 0. Other properties, including t_a-fallback validity, are inherited from the sub-protocols in a straightforward way.

Protocol SGC^2 has a maximum grade $k = 2$. The protocol is similar to SGC^1, but invokes SGC^1 instead of SWC and SWC instead of SProp. The output *value* of SGC^2 is simply taken to be the output value from SGC^1, but an instance of SWC is run on the output *grades* from SGC^1 in order to increase k from 1 to 2 via the mapping $(0, \bot, 1) \longrightarrow (0, 1, 2)$ on the outputs of SWC. The weak consistency of SWC guarantees that the grades of honest parties differ by at most by 1, and the validity of SWC guarantees that non-zero SGC^2 grades only occur if some honest parties obtained the grade 1 from SGC^1, implying agreement on the output values.[2].

Protocol SBA^****.*** We combine SGC^2 together with any fixed-round synchronous binary consensus protocol SBA to obtain our enhanced synchronous consensus protocol SBA^*. The construction is simple. First, the parties run SGC^2 on their inputs, and then they run SBA to decide whether agreement has been reached, inputting 1 to SBA if they have non-zero grades. The parties with the grade 2 just ignore SBA and output their SGC^2 output values. From this follows validity and fallback validity. The remaining parties use SBA for consistency. If SBA outputs 1, then they output their SGC^2 output values; otherwise, they output $\perp$. For synchronous consistency, there are three scenarios to consider. If some party has the grade 2, then by graded consistency, every party has the grades 2 or 1, and therefore, by validity, SBA outputs 1, leading to everyone outputting the common SGC^2 output value. If every party has the grade 0, then by validity SBA outputs 0, and so everyone outputs $\perp$. Finally, if some parties have the grade 1 while others the grade 0, then either SBA outputs 1 and everyone outputs the common SGC^2 output value, or SBA outputs 0 and everyone outputs $\perp$.

Protocol AProp. Our AProp (asynchronous proposal) protocol is an adaption of $\Pi^{t_s}_{\mathsf{prop}}$ from [6]. For simplicity, here we only consider the case $\delta n = 1$; if $\delta n > 1$, some care must be taken to achieve $(t_s, \delta n)$-intrusion tolerance. As in SProp, the honest parties have inputs in a set $S = \{x, \perp\} \in \mathcal{M}^{\perp}$. They start by simply sending their inputs to everyone (AProp does not need signatures). If a party receives an input v from $t_s + 1$ parties, then it learns that v is the input of an honest party, and therefore sends the input v to everyone, even if v is not its own input. If a party P_i receives an input v from $n - t_s$ parties, then it adds v to a set V_i. Upon adding a first value v to V_i, party P_i *proposes* to everyone that they should output v, and upon adding a second value to V_i, party P_i outputs $V_i = S$. Alternatively, a party will output v upon receiving from $n - t_s$ parties proposals to output v. If everyone has a common input v, then v will be the unique value everyone will add to V_i; therefore, everyone will propose and output v. A standard quorum-intersection argument on proposals shows t_a-consistency. The trickiest property is t_a-liveness, meaning that all parties obtain output. Since $n - t_a > 2t_s$, there exists an input held by at least $t_s + 1$ parties. Furthermore, if $t_s + 1$ parties send everyone an input v, then every party P_i sends everyone the input v and adds v to V_i; thus, there exists an input that every party P_i adds to V_i. Finally, if some P_i adds v to V_i, then at least $n - t_s - t_a \geq t_s + 1$ parties must have sent everyone the input v, meaning that every P_j adds v to V_j. Therefore, either every party P_i adds both x and $\perp$ to V_i and can output $\{x, \perp\}$, or there is a unique v which every party P_i adds to V_i, proposes, and outputs. Protocol AProp is non-terminating; it is designed to be run forever. Termination is guaranteed by the outer protocol ABA^*.

[2] Grades 0 and 1 suffice for SBA^* with binary inputs. Expanding the grade range is only necessary for multi-valued inputs, but incurs no asymptotic round or communication complexity overhead, which is why we do not consider the cases separately.

Protocol* AWC*. The asynchronous weak consensus protocol AWC is the counterpart of SWC. In theory, AProp can be used as a weak-consensus protocol for binary inputs via the mapping $(x, \{x, \bot\}, \bot) \longrightarrow (0, \bot, 1)$. This provides a simple way to obtain weak consensus on ℓ-bit messages by simply running ℓ parallel instances of AProp, one instance per bit. Then, any honest party that obtained $\bot$ from any instance would output $\bot$, and any honest party that obtained bits from all instances would output the concatenation of the bits. This design would increase message complexity by a multiplicative ℓ-factor, and the messages would need $(\log \ell)$-bit tags that indicate which AProp instance they belong to. To keep the complexity low, we parallelize the AProp instances in a more refined way, by batching/combining messages of different instances.

Protocols* AGC^1 *and* AGC^2*. These graded consensus protocols are simple message-driven adaptations of their synchronous counterparts SGC^1 and SGC^2. They invoke AWC and AProp as sub-protocols rather than SWC and SProp.

Protocol* ABA^**. Our asynchronous consensus protocol with t_s-validity ABA^* combines AGC^2 and *any* asynchronous binary consensus protocol ABA secure against t_a corruptions. The composition is similar to that in SBA^*, but since we can no longer rely on having a fixed running time, we need to rethink termination. We avoid the termination technique of sending certificates on tentative outputs (as done in some previous work [6,11,4]), and instead opt for an approach similar to Bracha's classical protocol from [8]. This keeps the communication complexity quadratic rather than cubic, and makes ABA^* a signature-free reduction of asynchronous multi-valued consensus to binary consensus [33]. To keep the communication complexity low, we ensure that the honest parties do not send ABA messages when the network is synchronous. We do so by requiring that when the network is synchronous, the honest parties know a common input $m \in \mathcal{M}$ by some publicly known time $r_s \cdot \Delta$, which triggers the termination rules and guarantees the termination of ABA^* within a few additional rounds, by some publicly known time T. Before local time T, honest parties do not send ABA messages. Hence they terminate without ever sending ABA messages if the network is synchronous.

Protocol* HBA*. We obtain HBA by composing SBA^* and ABA^* as described previously. The t_a-fallback validity of SBA^* and the synchronous t_s-validity (with termination) of ABA^* make the composition sound.

1.3 Contributions

Theorem 1. *Let* SBA *be any synchronous consensus protocol for binary inputs achieving error probability ϵ in k rounds, and* ABA *be any asynchronous consensus protocol for binary inputs. Under the provably optimal assumptions $2t_s + t_a < n$ and $t_a \leq t_s$, there exists a network-agnostic consensus protocol* HBA *for any inputs which invokes* SBA *and* ABA *in a black-box way and that, when the network is synchronous, achieves error probability ϵ in $k + 13$ rounds.*

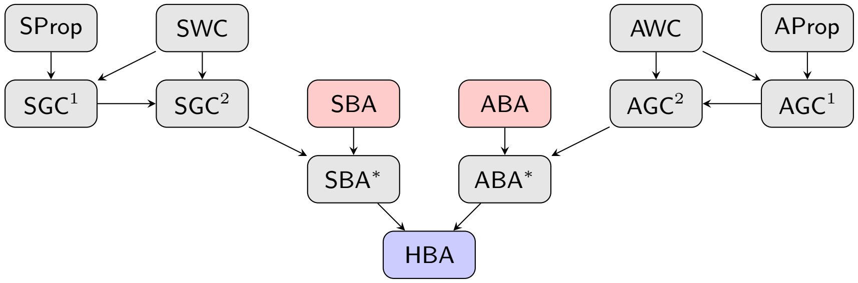

Fig. 1. An overview of how sub-protocols are composed for HBA.

Theorem 1 reduces the synchronous round complexity of network-agnostic consensus protocols to the round complexity of purely synchronous consensus protocols, up to a small additive constant. Concretely, HBA can take full advantage of a round-optimal λ-round SBA with an error probability decreasing super-exponentially with λ [20], improving over the state-of-the-art [11] (in which the error decreases exponentially) and finally matching the known lower bound for purely synchronous protocols [24]. Note that to achieve consensus for honest-majority when the network is synchronous, some sort of setup is *provably* necessary [27]. Our protocol also needs setup. Concretely, it can be instantiated from a variety of setup assumptions such as threshold signatures, bulletin-PKI or even correlated randomness for information theoretic pseudo-signatures.

Below, we compare the communication complexity (in terms of bits) of our construction with that of previous network-agnostic protocols, when we use the most efficient SBA and ABA protocols at hand. We denote with κ a computational security parameter and with ε a positive constant. Since [4] only has an ABA* component, we pair it with our SBA*. For fairness, we consider variants of previous protocols optimized to take full advantage of threshold signatures.

Table 1. Communication complexities of previous network-agnostic consensus protocols and ours, under assumptions that have been considered for network-agnostic consensus.

Assumptions Protocols	Bulletin-PKI	Bulletin-PKI & $2t_s \leq (1-\varepsilon)n$	Threshold Signatures
Blum, Katz, Loss [6]	—	—	$\mathcal{O}(n^4\kappa)$ (Bit Consensus)
Deligios, Hirt, Liu-Zhang [11]	—	—	$\mathcal{O}(n^2\kappa)$ (Bit Consensus)
Bacho, Collins, Liu-Zhang, Loss [4]	$\mathcal{O}(n^3\kappa + \ell n^3)$	$\mathcal{O}(n^3\kappa + \ell n^3)$	$\mathcal{O}(n^2\kappa + \ell n^3)$
Our Work	$\mathcal{O}(n^3\kappa + \ell n^2)$	$\mathcal{O}(n^2\kappa + \ell n^2)$ if network synchronous, else $\mathcal{O}(n^3\kappa + \ell n^2)$	$\mathcal{O}(n^2\kappa + \ell n^2)$

As a starting point, our protocol HBA has the communication complexity $\mathcal{O}(n^3\kappa + \ell n^2 + \mathsf{CC}_{\mathsf{SBA}} + \mathsf{CC}_{\mathsf{ABA}})$, where $\mathsf{CC}_{\mathsf{SBA}}$ and $\mathsf{CC}_{\mathsf{ABA}}$ are the communication complexities of SBA and ABA respectively. The $\mathcal{O}(n^3\kappa)$ overhead term can be reduced to $\mathcal{O}(n^2\kappa)$ by assuming a trusted setup for threshold signatures, or by slightly lowering the allowed corruptions to $2t_s \leq (1-\varepsilon)n$ for a positive constant ε. Furthermore, if the network is synchronous, then ABA messages are not sent and the term $\mathsf{CC}_{\mathsf{ABA}}$ is eliminated.

To minimize communication, we instantiate SBA with the protocol from [30] which in its base form achieves the communication complexity $\mathcal{O}(n^3\kappa)$, but can achieve the complexity $\mathcal{O}(n^2\kappa)$ with threshold signatures if they are available, or with expander graphs if $2t_s \leq (1-\varepsilon)n$. As for ABA, we instantiate it with the state-of-the-art cubic protocol from [19],[3] or, if setup for unique threshold signatures is available, with [31] using the coin protocol[4] from [9] to achieve quadratic complexity. When these instantiations are used, the asymptotic communication complexity of HBA for binary inputs matches that the complexity of its state-of-the-art[5] SBA component if the network is synchronous, and the complexity of its its state-of-the-art ABA component otherwise.

If bulletin-PKI is available and $2t_s \leq (1-\varepsilon)n$, then we achieve the complexity $O(n^2\kappa + \ell n^2)$ with expander graphs. We do so with 3-round variants of SWC and SProp, inspired by the graded consensus protocol in [30]. We present these variants in the appendix.

When considering ℓ-bit inputs, the $\mathcal{O}(\ell n^2)$ term is already a strict improvement over the $\mathcal{O}(\ell n^3)$ term from [4] and over any straightforward adaptation of known protocols. Using techniques from the literature on extension protocols together with some novel ideas, it is possible to bring this all the way down to $\mathcal{O}(\ell n)$, which is the best possible even for purely synchronous protocols [18]. We discuss network-agnostic consensus extension protocols later in more detail, and we present our extension protocols in the appendix appendix.

The efficiency improvements are facilitated by our new black-box construction of HBA which allows us to instantiate the sub-protocols SBA and ABA with the most efficient known protocols from the literature. We consider this new approach to be a contribution of independent interest, and hope that analogous constructions will unlock similar efficiency gains for other network-agnostic distributed tasks.

1.4 Related Work

Network-agnostic protocols were first considered by Blum, Katz and Loss [6]; in this work, the authors showed the first network-agnostic consensus protocol. The first network-agnostic full multi-party computation protocol (MPC) was shown in

[3] The ABA in [19] is secure statically, or adaptively with a one-time CRS.

[4] This coin protocol is secure in the random oracle model.

[5] An SBA protocol concurrent with our work uses threshold signatures to achieve $\mathcal{O}(nf\kappa)$ complexity, where $f \leq t_s \leq \frac{(1-\varepsilon)n}{2}$ is the actual number of malicious parties [14]. Our work only considers the worst case $f = t_s$.

[7]. There has since been a significant interest in the field. Other network-agnostic MPC protocols include [2,12]. Network-agnostic approximate agreement has been investigated in [21,22]. Most related to our work are [11], which contains an efficient network-agnostic consensus protocol which at the time matched the round complexity of the best known synchronous protocols, and [4], that deals with distributed-key-generation but also constructs an ABA^* counterpart.

The round complexity of consensus protocols has a long research history. Without setup, consensus among n parties is possible (both in synchronous and asynchronous networks) if less than $n/3$ parties are corrupted [27,8]. In this setting, the first synchronous consensus protocol with a number of rounds independent from n was [15], and the first asynchronous one was [10]. Constant-round protocols, regardless of the network assumptions, cannot be deterministic [16,13], and they fail with negligible probability. Assuming setup (like a PKI, or correlated randomness) consensus tolerating up to $n/2$ corruptions is possible [13]. One can also obtain constant round constructions in this setting [25]. Until recently, constant-round consensus protocols in any corruption setting failed with probability at least c^{-r} in r rounds for some constant $c > 1$. A long standing lower bound from [24] shows that, even in synchronous networks and assuming setup, one cannot hope to reduce the failure probability to less than $(c \cdot r)^{-r}$ in r rounds. The first synchronous protocol matching this lower bound is [20], and in this work we match its optimal round complexity when synchrony holds, while also ensuring consensus if the network is asynchronous. Our construction must not (and does not) run an asynchronous consensus protocol if the network is asynchronous, in order to circumvent another lower bound [3] which shows that error c^{-r} in r rounds is actually optimal for asynchronous protocols.

2 Preliminaries

2.1 Model

Adversary. We consider n parties $P_1, P_2, \ldots, P_n$ who communicate over a complete network of point-to-point authenticated channels. We consider an active threshold adversary bound by the integer thresholds t_s and t_a such that $t_a \leq t_s$ and $2t_a + t_s \leq (1 - \delta)n$, where δn is a positive integer.[6] The adversary may corrupt up to t_s parties in an adaptive fashion (depending on information learned during the execution) if the network is synchronous and t_a parties if the network is asynchronous, making them deviate arbitrarily from the prescribed protocol in a coordinated and malicious manner. We call a party who is never corrupted throughout the execution of a protocol *honest*.

Network. We consider different network models. If the network is *synchronous*, then all messages sent by honest parties must be delivered within a fixed time bound Δ, known to all honest parties. Subject to this rule, the adversary can

[6] Since $2t_s + t_a < n$ is required, this assumption is without loss of generality. One can simply consider $\delta = (n - 2t_s - t_a)/n$.

arbitrarily schedule the delivery of messages. Furthermore, we assume that the honest parties have synchronized local clocks, which means that they can start a protocol simultaneously and that their local clocks progress at the same rate. In this setting, the adversary is allowed to corrupt up to t_s parties for a fixed threshold t_s known to all parties. If the network is *asynchronous*, then the adversary can arbitrarily schedule the delivery of messages, with the restriction that messages sent by honest parties must eventually be delivered. Additionally, honest parties need not have synchronized local clocks. In this setting, the adversary is allowed to corrupt up to t_a parties for a fixed threshold t_a known to all parties. Finally, in the network-agnostic setting, the network may be synchronous or asynchronous. Honest parties do not know the network condition. Depending on the network type, the rules for the synchronous setting or the asynchronous setting are in effect, and the parameters Δ, t_s and t_a are all known to all parties. This is the setting in which we analyze our protocols.

Some of our protocols are *round-based.* The round r should be understood to be the local time interval between $(r-1)\Delta$ and $r\Delta$. In round r, each honest party sends messages at time $(r-1)\Delta$, listens to messages throughout the round, and makes decisions depending on received messages at time $r\Delta$. The scheduling powers of the adversary make it *rushing*, which means that it can choose its round r messages depending on the honest parties' round r messages.

In our protocols, something we very commonly direct parties to do is to send a message m to *all* parties. We call this "multicasting the message m."

2.2 Building Blocks

Security Parameter. We denote by κ a security parameter.

Signatures and Bulletin-PKI. For a bulletin-PKI setup, before the execution of the protocol, each party P_i generates a key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$, where sk_i is for signing messages and vk_i is for verifying them. Party P_i keeps sk_i private, and posts vk_i on a public board. If P_i is corrupted before the execution of a protocol, then it can freely choose its public vk_i; e.g. it can duplicate the key vk_j of an honest party P_j. We denote with $\sigma = \mathsf{Sgn}_{\mathsf{sk}_i}(m)$ that σ is a signature of length $\mathcal{O}(\kappa)$ obtained by signing m with sk_i, and say $\mathsf{Vfy}_{\mathsf{vk}_i}(m, \sigma) = 1$ if σ is a valid signature on m with respect to vk_i. We idealize the signature scheme (consisting of efficient key generation, signing and verification algorithms) to have perfect existential unforgeability, so that for any honestly generated key pair $(\mathsf{sk}_i, \mathsf{vk}_i)$, an adversary can't forge a signature σ such that $\mathsf{Vfy}_{\mathsf{vk}_i}(m, \sigma) = 1$ without sk_i.

Certificates. A certificate is a collection of valid signatures on a message. We formally define a certificate on a message $m \in \mathcal{M}$ to be a pair (m, L), where L is a list $(l_1, \dots, l_n)$ such that either $l_i = \sigma_i$ and $\mathsf{Vfy}_{\mathsf{vk}_i}(m, \sigma_i) = 1$ for some signature σ_i, or $l_i = \bot$, where $\bot$ is a special value which indicates the absence of a signature. A k-certificate on m contains at least k signatures on m.

Common Coins. Given an instance number k and a corruption threshold t, an idealized common coin protocol emulates a trusted entity which, upon receiving the message $(\mathbf{coin\ request}, k)$ from $t+1$ distinct parties, samples the bit $\mathbf{coin}_k$ uniformly at random and sends the message $(k, \mathbf{coin}_k)$ to all parties.

Unique Threshold Signatures. Appropriate trusted setup makes it possible for a party to compress any f-certificate (where e.g. $f = t_s + 1$) into a single threshold signature of length $\mathcal{O}(\kappa)$. This provides complexity savings whenever the party needs to send a certificate. Uniqueness is the desirable property that all f-certificates on a message m compress into the same threshold signature. While not needed for certificate compression, it is essential for the construction of a common coin protocol against up to $t < \frac{n}{2}$ corruptions with $\mathcal{O}(n^2)$ messages of length $\mathcal{O}(\kappa)$ in the random oracle model (wherein one models a cryptographic hash function as a random function from its domain to its range) [9].

2.3 Definitions of Primitives and Security Properties

Notions of validity. For multi-valued consensus, one can consider different notions of validity.

Validity: The most typical notion is that if all honest parties have the same input m, then all honest parties must output m.

Strong Validity: A stronger notion of validity is that the common output m must be the input of some honest party. While this is equivalent to regular validity for binary inputs, it is indeed stronger for multi-valued consensus: if two honest parties run consensus with the distinct inputs m_1 and m_2, then only regular validity permits the common output to be $m_3 \notin \{m_1, m_2\}$. Unfortunately, to achieve strong validity, even in the synchronous setting and for computational security, one needs $t < \frac{n}{2^\ell}$ [17]. Intuitively, the problem is that if all honest parties have different inputs, there is no secure way to choose the input of an honest party against even a single corruption.

Intrusion Tolerance: An alternative weakening of strong validity is that the common output should either be the input of some honest party, or a special output $\perp$ outside the domain of inputs. This property has been considered both explicitly [32,4] and implicitly [18,35,5], and it can be achieved together with optimal resilience for consensus. One can obtain it trivially by having all parties output $\perp$, but not when (regular) validity is also required.

Protocol Syntax. We primarily concern ourselves with protocols to reach consensus on ℓ-bit inputs, where ℓ is a publicly known length parameter. Hence, we define $\mathcal{M} = \{0,1\}^\ell$ to be the input space, and $\mathcal{M}^\perp = \mathcal{M} \cup \{\perp\}$ to be the input space with a special value $\perp$ added to it.

Security Properties. We consider property-based security for our protocols. Recall that we use the prefix "t-" to mean that a property only holds if at most t parties can be corrupted, and the prefix "synchronous" to mean that a property only holds if the network is synchronous. For round complexity, we sometimes also use the prefix "r-round". With it, we mean that if the network is synchronous and all honest parties know their inputs by some time $k\Delta$, then the guarantees of the prefixed property are achieved by time $(k+r)\Delta$.

With notational conventions out of the way, let us first define some properties shared by many primitives.

- **t-termination:** If all honest parties participate in the protocol with input, and they don't stop participating until termination, then all honest parties terminate the protocol with output.
- **t-liveness:** If all honest parties participate in the protocol with input without ever stopping, then all honest parties obtain output from the protocol.
- **t-robustness:** No honest party aborts the protocol.

Our round-based protocols all have fixed numbers of rounds, so that honest parties who do not abort trivially terminate. Because in this case termination follows from robustness, we only consider robustness for these protocols. Also, we do not consider robustness for protocols where honest parties cannot abort.

We define some agreement primitives together with relevant security properties below. The "fallback validity" properties refer to honest parties who provide input. This is because our round-based protocols permit honest parties to not have any input and abort immediately if the network is asynchronous.

Note that our parameterized notion of intrusion tolerance is related to the differential validity studied in [17]. Against t corruptions, a consensus protocol with (t,q)-intrusion tolerance automatically has $(n-2q)$-differential validity.

Weak Consensus. In a weak consensus protocol, each honest party P_i has an input $m_i \in \mathcal{M}$ and outputs some $y_i \in \mathcal{M}^{\perp}$. Below are the relevant properties:

- **$(\mathbf{t},\mathbf{q})$-intrusion tolerance:** Suppose less than q honest parties have the input m for some $m \in \mathcal{M}$. Then no honest party outputs m.
- **t-validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$. Then honest parties can only output m.
- **t-weak consistency:** If some honest party outputs some $m \in \mathcal{M}$, then honest parties can only output m or $\perp$.
- **t-fallback validity:** Suppose all honest parties who provide input have the same input $m \in \mathcal{M}$. Then honest parties either abort or output m.
- **t-validity with liveness:** Suppose all honest parties participate in the protocol with a common input $m \in \mathcal{M}$, and they never stop participating. Then all honest parties output m.

Proposal. In a proposal protocol, there exists some set $S = \{x, \perp\} \subseteq \mathcal{M}^{\perp}$ such that each honest party P_i has an input $v_i \in S$, and each honest party P_i outputs some $y_i \in \{x, \{x,\perp\}, \perp\}$. The relevant properties are the following:

- **$(\mathbf{t}, \mathbf{q})$-intrusion tolerance:** Suppose less than q honest parties have the input m for some $m \in \mathcal{M}$. Then no honest party outputs m or $\{m, \bot\}$. Note that for any proposal protocol, this property with $q \geq 1$ implies that honest parties can only obtain outputs in $\{x, \{x, \bot\}, \bot\}$, as required above.
- **t-validity:** Suppose all honest parties have the same input $v \in S$. Then honest parties can only output v.
- **t-weak consistency:** If some honest party outputs x, then no honest party outputs $\bot$.
- **t-fallback validity:** Suppose all honest parties who provide input have the same input $v \in S$. Then honest parties either abort or output v.
- **t-validity with liveness:** Suppose all honest parties participate in the protocol with a common input $v \in S$, and they never stop participating. Then all honest parties output v.

k-Graded Consensus. In a k-graded consensus protocol, each honest party P_i has an input $m_i \in \mathcal{M}$ and outputs a tuple (y_i, g_i), where $g_i \in \{0, 1, \ldots, k\}$. The relevant properties are the following:

- **$(\mathbf{t}, \mathbf{q})$-intrusion tolerance:** Suppose less than q honest parties have the input m for some $m \in \mathcal{M}$. Then no honest party P_i obtains $y_i = m$.
- **t-graded validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$. Then honest parties can only output (m, k).
- **t-graded consistency:** Suppose honest parties P_i and P_j output (y_i, g_i) and (y_j, g_j). Then, $|g_i - g_j| \leq 1$, and if $g_i \geq 1$, then $y_j = y_i$.
- **t-fallback graded validity:** If all honest parties who provide input have the same input $m \in \mathcal{M}$, then honest parties either abort or output (m, k).
- **t-graded validity with liveness:** Suppose all honest parties participate in the protocol with a common input $m \in \mathcal{M}$, and they never stop participating. Then all honest parties output (m, k).

Consensus. In a consensus protocol, each honest party P_i has an input $m_i \in \mathcal{M}$ and outputs some $y_i \in \mathcal{M}^{\bot}$. The relevant properties are the following:

- **$(\mathbf{t}, \mathbf{q})$-intrusion tolerance:** Suppose less than q honest parties have the input m for some $m \in \mathcal{M}$. Then no honest party outputs m.
- **t-validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$. Then honest parties can only output m.
- **t-consistency:** Honest parties do not obtain different outputs in $v \in \mathcal{M}^{\bot}$.
- **t-fallback validity:** Suppose all honest parties who provide input have the same input $m \in \mathcal{M}$. Then honest parties either abort or output m.
- **t-validity with termination:** Suppose all honest parties participate in the protocol with a common input $m \in \mathcal{M}$, and they never stop participating until termination. Then all honest parties terminate with the output m.

3 Synchronous Consensus with Fallback Validity

In this section, we show how to compile *any* fixed-round consensus protocol SBA for binary inputs with synchronous t_s-robustness, synchronous t_s-validity and synchronous t_s-consistency, into a fixed-round multi-valued consensus protocol SBA^* with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-consistency and t_a-fallback validity.

3.1 Synchronous Weak Consensus (SWC)

We begin with SWC: an adaptation of a protocol from [11], modified to support multi-valued inputs and have increased intrusion tolerance. The security of the protocol is captured by the lemma below it. All missing proofs can be found in the appendix.

Protocol SWC

Input: Party P_i has an input $m_i \in \mathcal{M}$ which it knows at the beginning, or it may have no input if the network is asynchronous.

Output: Party P_i either aborts or outputs $y_i \in \mathcal{M}^{\perp}$.

Initialization: If P_i has input, then P_i sets $y_i = \perp$. Else, P_i aborts.

Round 1: P_i computes $\sigma_i = \mathsf{Sgn}_{\mathsf{sk}_i}(m_i)$ and multicasts (m_i, σ_i). At the end of the round,

- If P_i hasn't received validly signed messages from $n - t_s$ parties, P_i aborts.
- Else, if the messages P_i received are so that P_i can form a $(t_s+\delta n)$-certificate on a unique $m \in \mathcal{M}$, then P_i sets $y_i = m$.

Round 2: If P_i possesses a $(t_s + \delta n)$-certificate on a unique $m \in \mathcal{M}$, then P_i multicasts a $(t_s + \delta n)$-certificate on m. At the end of the round, if P_i has seen a $(t_s + \delta n)$-certificate on some $m \neq y_i$, then P_i sets $y_i = \perp$.

Lemma 1 (Security of SWC). *If $t_a \leq t_s$ and $2t_s + t_a \leq (1 - \delta)n$, then SWC is a weak consensus protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-weak consistency and t_a-fallback validity.*

Complexity of SWC: The message complexity is $\mathsf{MC}_{\mathsf{SWC}} = \mathcal{O}(n^2)$, and the communication complexity is $\mathsf{CC}_{\mathsf{SWC}} = \mathcal{O}(n^3\kappa + \ell n^2)$.

3.2 Synchronous Proposal (SProp)

We continue with SProp, which we obtain by modifying SWC to fit the mold of a proposal protocol. Proposal protocols require there to exist some set $S = \{x, \perp\} \subseteq \mathcal{M}^{\perp}$ such that honest parties can only have inputs in S. We assume such a set $S = \{x, \perp\}$ exists, and use it in our description of SProp below.

Protocol SProp

Input: Party P_i has an input $v_i \in S$ which it knows at the beginning, or it may have no input if the network is asynchronous

Output: Party P_i either aborts or outputs $y_i \in \{x, \{x, \bot\}, \bot\}$.

Initialization: If P_i has input, then P_i sets $y_i = \bot$. Else, P_i aborts.

Round 1: If $v_i = \bot$, then P_i multicasts $\bot$. Else, P_i computes $\sigma_i = \mathsf{Sgn}_{\mathsf{sk}_i}(v_i)$ and multicasts (v_i, σ_i). At the end of the round,

- If P_i hasn't received valid messages from $n - t_s$ parties, then P_i aborts.
- Else, if the messages P_i received are so that P_i can form a $(t_s + \delta n)$-certificate on a unique $m \in \mathcal{M}$, then P_i sets $y_i = m$.

Round 2: If P_i possesses a $(t_s + \delta n)$-certificate on a unique $m \in \mathcal{M}$, then P_i multicasts it. At the end of the round, if P_i had $v_i = \bot$ but received a $(t_s + \delta n)$-certificate on some $m \in \mathcal{M}$, then P_i sets $y_i = \{m, \bot\}$.

Lemma 2 (Security of SProp). *If $t_a \leq t_s$ and $2t_s + t_a \leq (1 - \delta)n$, then SProp is a proposal protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-weak consistency and t_a-fallback validity.*

Complexity of SProp: The message and communication complexities of SProp are also respectively $\mathsf{MC}_{\mathsf{SProp}} = \mathcal{O}(n^2)$ and $\mathsf{CC}_{\mathsf{SProp}} = \mathcal{O}(n^3\kappa + \ell n^2)$.

3.3 Synchronous 1-Graded Consensus (SGC^1)

Now that we have SWC and SProp, we compose them into SGC^1. When instantiated with the protocols SWC and SProp above, SGC^1 runs in 4 rounds.

Protocol SGC^1

Input: Party P_i has an input $m_i \in \mathcal{M}$ which it knows at the beginning, or it may have no input if the network is asynchronous.

Output: Party P_i either aborts or outputs (y_i, g_i), where $y_i \in \mathcal{M}^{\bot}$ is the output value and $g_i \in \{0, 1\}$ is the output grade.

Phase 1: P_i participates in a common SWC instance with the input m_i. If P_i aborts SWC, P_i aborts the protocol. Else, let v_i be P_i's output.

Phase 2: P_i participates in a common SProp instance with the input v_i if $v_i = m_i$, and the input $\bot$ otherwise. If P_i aborts SProp, P_i aborts the protocol. Else, let z_i be P_i's output.

- If $z_i = m$, then P_i outputs $(m, 1)$.
- If $z_i = \{m, \bot\}$, then P_i outputs $(m, 0)$.
- If $z_i = \bot$, then P_i outputs $(\bot, 0)$.

Lemma 3 (Security of SGC^1). *Let SWC and SProp respectively be a weak consensus protocol and a proposal protocol, both with fixed numbers of rounds, such that they both have synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-weak consistency and t_a-fallback validity. Furthermore, suppose SProp has $(t_s, \delta n)$-intrusion tolerance. Then, SGC^1 is a 1-graded consensus protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-graded validity, synchronous t_s-graded consistency and t_a-fallback graded validity.*

Complexity of SGC^1: We have $\mathsf{MC}_{\mathsf{SGC}^1} = \mathsf{MC}_{\mathsf{SWC}} + \mathsf{MC}_{\mathsf{SProp}} = \mathcal{O}(n^2)$ and $\mathsf{CC}_{\mathsf{SGC}^1} = \mathsf{CC}_{\mathsf{SWC}} + \mathsf{CC}_{\mathsf{SProp}} = \mathcal{O}(n^3\kappa + \ell n^2)$.

3.4 Synchronous 2-Graded Consensus (SGC^2)

Using SGC^1 as a basis and SWC as a weak consensus protocol for binary inputs, we construct SGC^2. When instantiated with sub-protocols from previous sections, the protocol runs in 6 rounds when the network is synchronous.

Protocol SGC^2

Input: Party P_i has an input $m_i \in \mathcal{M}$ which it knows at the beginning, or it may have no input if the network is asynchronous.

Output: Party P_i either aborts or outputs (y_i, g_i), where $y_i \in \mathcal{M}^{\perp}$ is the output value and $g_i \in \{0, 1, 2\}$ is the output grade.

Phase 1: P_i participates in a common SGC^1 instance with the input m_i. If P_i aborts SGC^1, P_i aborts the protocol. Else, let (z_i, h_i) be P_i's output. P_i sets $y_i = z_i$.

Phase 2: P_i participates in a common SWC instance with the input h_i. If P_i aborts SWC, then P_i aborts the protocol. Else, let v_i be P_i's output.

- If $v_i = 0$, then P_i sets $g_i = 0$.
- If $v_i = \perp$, then P_i sets $g_i = 1$.
- If $v_i = 1$, then P_i sets $g_i = 2$.

Lemma 4 (Security of SGC^2). *Let SGC^1 and SWC respectively be a 1-graded consensus protocol and a weak binary consensus protocol, both with fixed numbers of rounds, such that SGC^1 has $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-graded validity, synchronous t_s-graded consistency and t_a-fallback graded validity, and SWC has synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-weak consistency and t_a-fallback validity. Then, SGC^2 is a 2-graded consensus protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-graded validity, synchronous t_s-graded consistency and t_a-fallback graded validity.*

Complexity of SGC^2: We have $\mathsf{MC}_{\mathsf{SGC}^2} = \mathsf{MC}_{\mathsf{SGC}^1} + \mathsf{MC}_{\mathsf{SWC}} = \mathcal{O}(n^2)$ and $\mathsf{CC}_{\mathsf{SGC}^2} = \mathsf{CC}_{\mathsf{SGC}^1} + \mathsf{CC}_{\mathsf{SWC}} = \mathcal{O}(n^3\kappa + \ell n^2)$.

3.5 Synchronous Consensus (SBA^*)

Finally, we take *any* fixed-round binary consensus protocol SBA and combine it with SGC^2 to obtain SBA^*: the synchronous component of HBA. Again, SBA^* is a two-phase protocol. The first phase lasts sufficiently long for SGC^2 (6 rounds with our constructions). The number of rounds for the second phase depends on the specific SBA protocol chosen.

Protocol SBA^*

Input: Party P_i has an input $m_i \in \mathcal{M}$ which it knows at the beginning, or it may have no input if the network is asynchronous.

Output: Party P_i either aborts or outputs $y_i \in \mathcal{M}^\perp$.

Phase 1: P_i participates in a common SGC^2 instance with the input m_i. If P_i aborts SGC^2, P_i aborts the protocol. Else, let (z_i, g_i) be P_i's output.

Phase 2: P_i participates in a common SBA instance with the input 1 if $g_i \in \{1, 2\}$ and 0 otherwise. If P_i fails to obtain output from SBA for any reason, then P_i outputs z_i and terminates. Else, let h_i be P_i's output.

- If $g_i = 2$ or $h_i = 1$, then P_i outputs z_i.
- Otherwise, P_i outputs $\perp$.

Theorem 2 (Security of SBA^*). *Let SGC^2 and SBA respectively be 2-graded consensus protocol and a binary consensus protocol, both with fixed numbers of rounds, such that SGC^2 has $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-graded validity, synchronous t_s-graded consistency and t_a-fallback graded validity, and SBA has synchronous t_s-robustness, synchronous t_s-validity and synchronous t_s-consistency. Then, SBA^* is a consensus protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-consistency and t_a-fallback validity.*

Complexity of SBA^*: We have $\mathsf{MC}_{\mathsf{SBA}^*} = \mathsf{MC}_{\mathsf{SGC}^2} + \mathsf{MC}_{\mathsf{SBA}} = \mathcal{O}(\mathsf{MC}_{\mathsf{SBA}} + n^2)$ and $\mathsf{CC}_{\mathsf{SBA}^*} = \mathsf{CC}_{\mathsf{SGC}^2} + \mathsf{CC}_{\mathsf{SBA}^*} = \mathcal{O}(\mathsf{CC}_{\mathsf{SBA}} + n^3\kappa + \ell n^2)$. Note that if SBA is designed to be run synchronously, then it might suffer from increased complexity when run asynchronously. This is not an issue for any protocol that we cite.

The $\mathcal{O}(n^3\kappa)$ term in the complexity is the consequence of certificate multicasting in SWC and SProp. One straightforward option to reduce this term by a factor of n is to assume a $(t_s + \delta n - 1)$-threshold signature setup. Then, individual parties' signatures are replaced by signature shares, and $(t_s + \delta n)$-certificates by signatures of size $\mathcal{O}(\kappa)$. Another option, only assuming a bulletin-PKI setup, is to use expander graph based techniques [30]. One imposes an n-vertex expander graph on the parties, with each party corresponding to a vertex. In SWC and SProp parties just send the $\mathcal{O}(n)$ sized certificates to their (constant sized) set of neighbors in the graph. This modification breaks weak consistency, which can be regained with the introduction of an additional round. However, with expander graphs, security requires the slightly stronger assumption $2t_s \leq (1-\varepsilon)n$ for a constant $\varepsilon > 0$. In the appendix, we present 3-round expander-based variants of SWC and SProp with which the communication complexity of SBA^* can be reduced to $\mathcal{O}(\mathsf{CC}_{\mathsf{SBA}} + n^2\kappa + \ell n^2)$ when $2t_s \leq (1-\varepsilon)n$ for a constant $\varepsilon > 0$.

4 Asynchronous Consensus with High Validity

We now leave round-based protocols behind, and switch to message-driven protocols. In this section we take a (possibly non-terminating) consensus protocol ABA with t_a-validity, t_a-consistency and t_a-liveness, and combine it with our AGC^2 protocol to obtain a protocol ABA^* with $(t_s, \delta n)$-intrusion tolerance, t_s-validity[7], synchronous t_s-validity with termination, t_a-consistency and t_a-termination.

4.1 Asynchronous Proposal (AProp)

We begin with AProp. The "Conflict Echoing" rule is asymmetrically biased to make the output $\perp$ more likely than x or $\{x, \perp\}$. This lets us obtain $(t_s, \delta n)$-intrusion tolerance rather than $(t_s, \lceil \frac{\delta n}{2} \rceil)$-intrusion tolerance.

As AProp is a proposal protocol, we assume there exists some set $S = \{x, \perp\} \in \mathcal{M}^{\perp}$ such that honest parties have inputs in S.

Protocol AProp

Input: Party P_i has an input $v_i \in S$ which it can eventually learn.

Output: Party P_i outputs $y_i \in \{x, \{x, \perp\}, \perp\}$. P_i only outputs once, so P_i only takes into account its first outputting directive.

Initialization: P_i lets $V_i = \varnothing$.

Input Acquisition: When P_i knows v_i, P_i multicasts $(\mathbf{input}, v_i)$.

Conflict Echoing: Upon receiving the message $(\mathbf{input}, v)$ for some $v \neq v_i$ for the first time from some party,

- If $v = \perp$ and P_i has received the message $(\mathbf{input}, v)$ from exactly $t_s + 1$ parties, P_i multicasts $(\mathbf{input}, \perp)$.
- If $v \in \mathcal{M}$ and P_i has received the message $(\mathbf{input}, v)$ from exactly $t_s + \delta n$ parties, P_i multicasts $(\mathbf{input}, v)$.

Value Rule: Upon receiving the message $(\mathbf{input}, v)$ for some v from exactly $n - t_s$ parties, P_i adds v to V_i. Then,

- If this rule has been activated for the first time, P_i multicasts $(\mathbf{propose}, v)$.
- If this rule has been activated for the second time, P_i outputs V_i.

Certify Rule: Upon receiving the message $(\mathbf{propose}, v)$ for some v from exactly $n - t_s$ parties, P_i outputs v.

[7] Actually, t_a-validity from ABA^* suffices for HBA.

Lemma 5 (Security of AProp). *If $t_a \leq t_s$ and $2t_s + t_a \leq (1-\delta)n$, then AProp is a proposal protocol with $(t_s, \delta n)$-intrusion tolerance, 2-round t_s-validity with liveness, t_a-weak consistency and t_a-liveness.*

Complexity of AProp: AProp is designed so that honest parties only send **input** messages on inputs held by honest parties. As the honest parties have inputs in $S = \{x, \perp\}$, they multicast at most two **input** messages. Furthermore, by the design of AProp, honest parties may only multicast a single **propose** message. So, honest parties multicast at most three messages (two **input** messages and one **propose** message), and all of these messages are of size $\mathcal{O}(\ell)$. Therefore, the message complexity is $\mathsf{MC}_{\mathsf{AProp}} = \mathcal{O}(n^2)$ and the communication complexity is $\mathsf{CC}_{\mathsf{AProp}} = \mathcal{O}(\ell n^2)$.

4.2 Asynchronous Weak Consensus (AWC)

Now, we present our weak consensus protocol AWC. To improve efficiency, we parallelize ℓ instances of AWC in a refined way. We batch together **input** messages from the "Input Acquisition" rule and **propose** messages from the "Value Rule." The "Conflict Echoing" rule is not amenable to such batching; therefore, any message sent as a result of the rule affects all instances. Although this introduces dependency among instances, we still get security for AWC *without* intrusion tolerance, which we do not need in any case.

Protocol AWC

Input: Party P_i has an input $m_i \in \mathcal{M}$ which it can eventually learn.

Output: Party P_i outputs $y_i \in \mathcal{M}^\perp$. P_i only outputs once, so P_i only takes into account its first outputting directive.

Initialization: P_i lets $V_i^1, V_i^2, \ldots, V_i^\ell = \varnothing, \varnothing, \ldots, \varnothing$.

Input Acquisition: When P_i knows m_i, P_i multicasts $(\textbf{input}, m_i)$.

Conflict Echoing: Upon receiving from exactly $t_s + 1$ parties inputs with the k^{th} bit $1 - b_i^k$ or the message **conflict**, P_i multicasts **conflict**.

Value Rule: For each k, upon receiving from exactly $n - t_s$ parties inputs with the k^{th} bit b or the message **conflict**, P_i adds b to V_i^k. Then,

- If each of the sets $V_i^1, V_i^2, \ldots, V_i^\ell$ contain exactly one bit, then P_i crafts the ℓ-bit message m which as its k^{th} bit has the unique bit in V_i^k, and P_i multicasts $(\textbf{propose}, m)$.
- If $V_i^k = \{0, 1\}$, then P_i outputs $\perp$.

Certify Rule: Upon receiving the message $(\textbf{propose}, m)$ for some $m \in \mathcal{M}$ from exactly $n - t_s$ parties, P_i outputs m.

Theorem 3 (Security of AWC). *If $t_a \le t_s$ and $2t_s + t_a < n$, then AWC is a weak consensus protocol with 2-round t_s-validity with liveness, t_a-weak consistency and t_a-liveness.*

Complexity of AWC: Honest parties multicast at most three messages (one **input** message, one **conflict** message and one **propose** message), and all of these messages are of size $\mathcal{O}(\ell)$. Therefore, the message complexity is $\mathsf{MC}_{\mathsf{AWC}} = \mathcal{O}(n^2)$ and the communication complexity is $\mathsf{CC}_{\mathsf{AWC}} = \mathcal{O}(\ell n^2)$.

4.3 Asynchronous Graded Consensus

Recall that in the previous section, we obtained SGC^1 by composing SWC and SProp, and then obtained SGC^2 by composing SGC^1 and SWC. These compositions translate very well to the asynchronous setting: we can easily compose AWC and AProp to obtain the 1-graded consensus protocol AGC^1, and then compose AGC^1 and AWC to obtain the 2-graded consensus protocol AGC^1.

Since AGC^1 and AGC^2 are almost identical (although message-driven) copies of SGC^1 and SGC^2, here we only state the security guarantees they achieve. The full protocols AGC^1 and AGC^2 can be found in the appendix.

Lemma 6 (Security of AGC^1). *Let AProp and AWC respectively be a proposal protocol and a weak consensus protocol such that AProp has $(t_s, \delta n)$-intrusion tolerance, r_p-round t_s-validity with liveness, t_a-weak consistency and t_a-liveness, and AWC has r_w-round t_s-validity with liveness, t_a-weak consistency and t_a-liveness. Then, AGC^1 is a 1-graded consensus protocol with $(t_s, \delta n)$-intrusion tolerance, $(r_w + r_p)$-round t_s-graded validity with liveness, t_a-graded consistency and t_a-liveness.*

Lemma 7 (Security of AGC^2). *Let AGC^1 and AWC respectively be a 1-graded consensus protocol and a weak binary consensus protocol such that AGC^1 has $(t_s, \delta n)$-intrusion tolerance, r_g-round t_s-graded validity with liveness, t_a-graded consistency and t_a-liveness, and AWC has r_w-round t_s-graded validity with liveness, t_a-weak consistency and t_a-liveness. Then, AGC^2 is a 2-graded consensus protocol with $(t_s, \delta n)$-intrusion tolerance, $(r_g + r_w)$-round t_s-graded validity with liveness, t_a-graded consistency and t_a-liveness.*

Complexity of AGC^1 and AGC^2: Both composed protocols involve no messages other than those of their sub-protocols. Thus, we have $\mathsf{MC}_{\mathsf{AGC}^1} = \mathsf{MC}_{\mathsf{AWC}} + \mathsf{MC}_{\mathsf{AProp}} = \mathcal{O}(n^2)$, $\mathsf{CC}_{\mathsf{AGC}^1} = \mathsf{CC}_{\mathsf{AWC}} + \mathsf{CC}_{\mathsf{AProp}} = \mathcal{O}(\ell n^2)$, $\mathsf{MC}_{\mathsf{AGC}^2} = \mathsf{MC}_{\mathsf{AGC}^1} + \mathsf{MC}_{\mathsf{AWC}} = \mathcal{O}(n^2)$, and $\mathsf{CC}_{\mathsf{AGC}^2} = \mathsf{CC}_{\mathsf{AGC}^1} + \mathsf{CC}_{\mathsf{AWC}} = \mathcal{O}(\ell n^2)$.

4.4 Asynchronous Consensus (ABA^*)

To conclude this section, we construct ABA^* by composing AGC^2 and a binary consensus protocol ABA with just t_a-validity, t_a-consistency and t_a-liveness.

The protocol ABA actually does not even need to provide termination, as this is guaranteed by a separate termination procedure in ABA^*. For maximum generality, we let r_g be the number of rounds in which AGC^2 achieves t_s-validity with liveness; the r_g-round t_s-graded validity with liveness of AGC^2 lets us prove synchronous $(r_g + 1)$-round t_s-validity with termination for ABA^*. Note that if AGC^2 is based on our AProp and AWC protocols, then $r_g = 6$.

Protocol ABA^*

Start Round: The parties have an agreed upon "start round" r_s.

Input: Party P_i has an input $m_i \in \mathcal{M}$ which it can eventually learn.

Output: Party P_i outputs $y_i \in \mathcal{M}^\perp$.

Initialization: Party P_i starts participating in a common instance of AGC^2 and a common instance of ABA. Before time $(r_s + r_g + 1)\varDelta$, P_i runs ABA passively, not processing the messages it receives.

Input Acquisition: When P_i knows m_i, P_i sets it as its input for AGC^2.

Graded Output: Upon outputting (z_i, g_i) from AGC^2,

- If $g_i = 2$, then P_i multicasts $(\textbf{commit}, z_i)$ if it hasn't done so previously.
- If $g_i \in \{2, 1\}$, P_i sets its ABA input to 1. Else, P_i sets its ABA input to 0.

Late Output: Upon outputting (z_i, g_i) from AGC^2 where $g_i \neq 2$ and outputting h_i from ABA, P_i sets x_i to z_i if $h_i = 1$, and to $\perp$ otherwise. Then, P_i multicasts $(\textbf{commit}, x_i)$ if it hasn't done so previously.

Commit Processing: Upon receiving $(\textbf{commit}, x)$ for some $x \in \mathcal{M}^\perp$,

- If P_i has received $(\textbf{commit}, x)$ from exactly $t_s{+}1$ parties, then P_i multicasts $(\textbf{commit}, x)$ if it hasn't done so previously.
- If P_i has received $(\textbf{commit}, x)$ from exactly $n{-}t_s$ parties, then P_i terminates with the output x if $(r_s{+}r_g)\varDelta$ time has passed, and sets itself up to terminate with the output x at time $(r_s + r_g)\varDelta$ otherwise.

Theorem 4 (Security of ABA^*). *Let $t_a \leq t_s$ and $2t_s + t_a \leq (1 - \delta)n$, and suppose honest parties must know their inputs by time $r_s\varDelta$ when the network is synchronous. Let AGC^2 and ABA respectively be a 2-graded consensus protocol and a binary consensus protocol such that AGC^2 has $(t_s, \delta n)$-intrusion tolernace, r_g-round t_s-graded validity with liveness, t_a-graded consistency and t_a-liveness, and ABA has t_a-validity, t_a-consistency and t_a-liveness. Then, ABA^* is a consensus protocol with $(t_s, \delta n)$-intrusion tolerance, t_s-validity, t_a-consistency, t_a-termination and synchronous $(r_g + 1)$-round t_s-validity with termination.*

Complexity of ABA^*: One can prove that honest parties commit at most once. With that in mind, the complexity depends on the network type.

- If the network is asynchronous, then we sum up the complexities of AGC^2 and ABA together with the complexities arising from the **commit** messages. Hence, we get $\mathsf{MC}^{\mathsf{Async}}_{\mathsf{ABA}^*} = \mathsf{MC}_{\mathsf{ABA}} + \mathsf{MC}_{\mathsf{AGC}^2} + \mathcal{O}(n^2) = \mathcal{O}(\mathsf{MC}_{\mathsf{ABA}} + n^2)$ and $\mathsf{CC}^{\mathsf{Async}}_{\mathsf{ABA}^*} = \mathsf{CC}_{\mathsf{ABA}} + \mathsf{CC}_{\mathsf{AGC}^2} + \mathcal{O}(\ell n^2) = \mathcal{O}(\mathsf{CC}_{\mathsf{ABA}} + \ell n^2)$.
- If the network is synchronous, then the synchronous $(r_g + 1)$-round t_s-validity of ABA^* ensures that all honest terminate by time $(r_s + r_g + 1)\Delta$ without ever sending ABA messages. Thus, we get the reduced $\mathsf{MC}^{\mathsf{Sync}}_{\mathsf{ABA}^*} = \mathsf{MC}_{\mathsf{AGC}^2} + \mathcal{O}(n^2) = \mathcal{O}(n^2)$ and $\mathsf{CC}^{\mathsf{Sync}}_{\mathsf{ABA}^*} = \mathsf{CC}_{\mathsf{AGC}^2} + \mathcal{O}(\ell n^2) = \mathcal{O}(\ell n^2)$.

5 The Network-Agnostic Protocol (HBA)

Finally, in this section we compose SBA^* and ABA^* to construct our network-agnostic consensus protocol HBA. Intrusion tolerance here is a challenge: SBA^* allows honest parties to output $\bot$, but ABA^* doesn't allow them to input $\bot$. Thus, we extend inputs for ABA^*, making them $(\ell + 1)$ bits long rather than ℓ bits long; the first bit is reserved to indicate whether the SBA^* output is $\bot$ or not. Notationally, we write $b \,\|\, m$ to represent $m \in \mathcal{M}$ with the bit b prepended.

We present HBA as a two-phase protocol. The first phase begins at time 0 and lasts sufficiently long for SBA^*; the second phase begins after the first and lasts indefinitely.

Protocol HBA

Input: Party P_i has an input $m_i \in \mathcal{M}$ which it must know at the beginning of the protocol if the network is synchronous.

Output: Party P_i outputs $y_i \in \mathcal{M}^{\bot}$.

Initialization: P_i starts participating in a common instance of ABA^*, set to achieve consensus on $(\ell + 1)$-bit inputs. Also, P_i sets $v_i = \bot$.

Phase 1: If P_i knows its input m_i, then P_i participates in a common instance of SBA^* with the input m_i. If P_i doesn't abort it, let z_i be the output P_i obtains from it. If $z_i = \bot$, then P_i sets $v_i = (0, 0, \ldots, 0) \in \{0, 1\}^{\ell+1}$. Otherwise, P_i sets $v_i = 1 \,\|\, z_i$.

Phase 2: If $v_i \neq \bot$, then P_i sets v_i as its input for ABA^*. Else, P_i sets $1 \,\|\, m_i$ as its ABA^* input once P_i knows m_i. Upon terminating ABA^* with the output $x_i = (x_i^1, x_i^2, \ldots, x_i^{\ell+1})$,

- If $x_i^1 = 0$, then P_i outputs $\bot$ and terminates.
- If $x_i^1 = 1$, then P_i outputs $(x_i^2, \ldots, x_i^{\ell+1}) \in \mathcal{M}$ and terminates.

Note that to use the particular ABA^* protocol presented in Section 4.4, we would need to set its "start round r_s" to be the round count of SBA^*.

Theorem 5 (Security of HBA). *Let SBA^* be an r_s-round consensus protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous*

t_s-validity, synchronous t_s-consistency and t_a-fallback validity, and let ABA^* be a consensus protocol with $(t_s, \delta n)$-intrusion tolerance, t_a-validity, synchronous r_a-round t_s-validity with termination, t_a-consistency and t_a-termination. Then, HBA is a consensus protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-validity, synchronous t_s-consistency, synchronous $(r_s + r_a)$-round t_s-termination, t_a-validity, t_a-consistency and t_a-termination.

Proof. We use the synchronous t_s-robustness of SBA^* implicitly.

- $(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$**-intrusion tolerance:** Suppose less than δn honest parties have some $m \in \mathcal{M}$ as input. By the $(t_s, \delta n)$-intrusion tolerance of SBA^*, honest parties do not output m from SBA^*. This ensures that only honest parties who abort SBA^* and have the input m may provide the input $1 \parallel m$ to ABA^*. As there can only be less than δn such honest parties, by the $(t_s, \delta n)$-intrusion tolerance of ABA^*, honest parties do not output $1 \parallel m$ from ABA^*. This implies that honest parties do not output m.
- **synchronous properties:** Suppose the network is synchronous, and suppose the adversary can corrupt at most t_s parties. Since the network is synchronous, all honest participate in SBA^* with input. They all terminate it by time r_s, with some common output $z \in \mathcal{M}^\perp$ by the synchronous t_s-consistency of SBA^*. Afterwards, all honest parties participate in ABA^* with the common input z', where $z' = (0, 0, \ldots, 0) \in \{0,1\}^{\ell+1}$ if $z = \perp$, and $z' = 1 \parallel z$ otherwise. By time $(r_s + r_a)\Delta$, the synchronous r_a-round t_s-validity with termination of ABA^* guarantees that they terminate ABA^* with the common output z' and hence terminate HBA with a common output. This gives us synchronous t_s-consistency and synchronous $(r_s + r_a)$-round t_s-termination. Additionally, suppose all honest parties have the same input $m \in \mathcal{M}$. Then, the synchronous t_s-validity of SBA^* implies that $z = m$ and hence that $z' = 1 \parallel m$, which means that the honest parties all terminate with the output m. Thus, we also obtain synchronous t_s-validity.
- $\mathbf{t_a}$**-termination and** $\mathbf{t_a}$**-consistency:** All honest parties eventually provide ABA^* input, and run it until they terminate it with consistent outputs. Thus, HBA inherits the t_a-termination and the t_a-consistency of ABA^*.
- $\mathbf{t_a}$**-validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$. By the t_a-fallback validity of SBA^*, any honest party that doesn't abort SBA^* outputs m from it. This ensures that all honest parties participate in ABA^* with the input $1 \parallel m$. Finally, by the t_a-validity and the t_a-termination of ABA^*, all honest parties terminate ABA^* with the output $1 \parallel m$ and therefore terminate HBA with the output m.

Complexity of HBA: Note that the extension of ABA^* inputs to $\ell + 1$ bits doesn't affect asymptotic complexity.

- If the network is synchronous, then the message complexity is $\mathsf{MC}_{\mathsf{HBA}}^{\mathsf{Sync}} = \mathsf{MC}_{\mathsf{SBA}^*} + \mathsf{MC}_{\mathsf{ABA}^*}^{\mathsf{Sync}} = \mathcal{O}(\mathsf{MC}_{\mathsf{SBA}} + n^2)$, and if the network is asynchronous, then it is $\mathsf{MC}_{\mathsf{HBA}}^{\mathsf{Async}} = \mathsf{MC}_{\mathsf{SBA}^*} + \mathsf{MC}_{\mathsf{ABA}^*}^{\mathsf{Async}} = \mathcal{O}(\mathsf{MC}_{\mathsf{SBA}} + \mathsf{MC}_{\mathsf{ABA}} + n^2)$.

– As discussed in the technical overview, if threshold signatures are available or if $2t_s < (1-\varepsilon)n$ for a constant $\varepsilon > 0$, then we get $\mathsf{CC}_{\mathsf{HBA}} = \mathcal{O}(\mathsf{CC}_{\mathsf{SBA}} + \mathsf{CC}_{\mathsf{ABA}} + n^2\kappa + \ell n^2)$. Else, we get $\mathsf{CC}_{\mathsf{HBA}} = \mathcal{O}(\mathsf{CC}_{\mathsf{SBA}} + \mathsf{CC}_{\mathsf{ABA}} + n^3\kappa + \ell n^2)$. Finally, if the network is synchronous, then the term $\mathsf{CC}_{\mathsf{ABA}}$ is eliminated.

– We also concretely state the synchronous round complexity of HBA. Recall that we build SBA^* by composing SGC^2 (6 rounds) and a fixed-round synchronous consensus protocol SBA (k rounds), so that SBA^* requires $k+6$ rounds. Protocol AGC^2 achieves 6-round t_s-validity with liveness, and hence ABA^* attains synchronous 7-round t_s-validity with termination. Combining the round complexities of SBA^* and ABA^*, we conclude that HBA terminates in at most $k+13$ rounds when the network is synchronous.

Reducing $\mathcal{O}(\ell n^2)$ ***to*** $\mathcal{O}(\ell n)$. For ℓ-bit consensus, the optimal communication complexity is $\mathcal{O}(\ell n + \dots)$ [18]. There is a rich literature *extending* consensus protocols to handle ℓ-bit inputs in clever ways to meet this bar [18,37,35,5].

To the best of our knowledge, [35] presents the state-of-the-art adaptively secure consensus extension protocol.[8] The protocol requires the parties to agree on κ-bit values in an intrusion-tolerant manner. Intrusion tolerance is achieved via two instances of consensus (one to decide the a κ-bit value, the other to decide if this value was the input of some honest party), but we observe that this is superfluous if the underlying consensus protocol is already intrusion-tolerant. In the appendix, we present an adaptation of the protocol which makes only one black-box invocation of HBA on κ-bit inputs, and has an overhead of two rounds when the network is synchronous, $\mathcal{O}(n^2)$ messages, and $\mathcal{O}(n^2\kappa + \ell n)$ or $\mathcal{O}(n^2\kappa \log n + \ell n)$ bits of communication depending on the availability of trusted setup. We follow it with a novel setup-free expander-based extension protocol which makes full use of $(t_s, \delta n)$-intrusion tolerance to reduce the overhead $\mathcal{O}(n^2\kappa \log n + \ell n)$ to $\mathcal{O}(n^2\kappa + \ell n)$ when $\delta = \Theta(1)$.

Note that the $\mathcal{O}(\ell n^2)$ term in the communication complexity of HBA (as opposed to the $\mathcal{O}(\ell n^3)$ in [4]) permits extensions for long inputs with the total complexity $\mathcal{O}(n^2\kappa+\ell n)$ whenever HBA achieves the complexity $\mathcal{O}(n^2\kappa+\ell n^2)$. Thus, with HBA, one can match the complexity of state-of-the-art synchronous protocols.

References

1. Alon, N.: Explicit expanders of every degree and size. Combinatorica **41**(4), 447–463 (2021). `https://doi.org/10.1007/s00493-020-4429-x`
2. Appan, A., Chandramouli, A., Choudhury, A.: Perfectly-secure synchronous mpc with asynchronous fallback guarantees. In: Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing. pp. 92–102 (2022)
3. Attiya, H., Censor, K.: Lower bounds for randomized consensus under a weak adversary. In: Bazzi, R.A., Patt-Shamir, B. (eds.) 27th ACM Symposium Annual on Principles of Distributed Computing. pp. 315–324. Association for Computing Machinery (Aug 2008). `https://doi.org/10.1145/1400751.1400793`

[8] Adaptively secure sub-quadratic extension is possible in the atomic-send model [5].

4. Bacho, R., Collins, D., Liu-Zhang, C.D., Loss, J.: Network-agnostic security comes (almost) for free in DKG and MPC. In: Handschuh, H., Lysyanskaya, A. (eds.) Advances in Cryptology – CRYPTO 2023, Part I. Lecture Notes in Computer Science, vol. 14081, pp. 71–106. Springer, Heidelberg (Aug 2023). https://doi.org/10.1007/978-3-031-38557-5_3
5. Bhangale, A., Liu-Zhang, C.D., Loss, J., Nayak, K.: Efficient adaptively-secure byzantine agreement for long messages. In: Agrawal, S., Lin, D. (eds.) Advances in Cryptology – ASIACRYPT 2022, Part I. Lecture Notes in Computer Science, vol. 13791, pp. 504–525. Springer, Heidelberg (Dec 2022). https://doi.org/10.1007/978-3-031-22963-3_17
6. Blum, E., Katz, J., Loss, J.: Synchronous consensus with optimal asynchronous fallback guarantees. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019: 17th Theory of Cryptography Conference, Part I. Lecture Notes in Computer Science, vol. 11891, pp. 131–150. Springer, Heidelberg (Dec 2019). https://doi.org/10.1007/978-3-030-36030-6_6
7. Blum, E., Zhang, C.D.L., Loss, J.: Always have a backup plan: Fully secure synchronous MPC with asynchronous fallback. In: Micciancio, D., Ristenpart, T. (eds.) Advances in Cryptology – CRYPTO 2020, Part II. Lecture Notes in Computer Science, vol. 12171, pp. 707–731. Springer, Heidelberg (Aug 2020). https://doi.org/10.1007/978-3-030-56880-1_25
8. Bracha, G.: Asynchronous byzantine agreement protocols. Information and Computation **75**(2), 130–143 (1987). https://doi.org/https://doi.org/10.1016/0890-5401(87)90054-X
9. Cachin, C., Kursawe, K., Shoup, V.: Random oracles in constantipole: Practical asynchronous byzantine agreement using cryptography (extended abstract). In: Proceedings of the Nineteenth Annual ACM Symposium on Principles of Distributed Computing. p. 123–132. PODC '00, Association for Computing Machinery, New York, NY, USA (2000). https://doi.org/10.1145/343477.343531
10. Canetti, R., Rabin, T.: Fast asynchronous byzantine agreement with optimal resilience. In: 25th Annual ACM Symposium on Theory of Computing. pp. 42–51. ACM Press (May 1993). https://doi.org/10.1145/167088.167105
11. Deligios, G., Hirt, M., Liu-Zhang, C.D.: Round-efficient byzantine agreement and multi-party computation with asynchronous fallback. In: Nissim, K., Waters, B. (eds.) TCC 2021: 19th Theory of Cryptography Conference, Part I. Lecture Notes in Computer Science, vol. 13042, pp. 623–653. Springer, Heidelberg (Nov 2021). https://doi.org/10.1007/978-3-030-90459-3_21
12. Deligios, G., Liu-Zhang, C.D.: Synchronous perfectly secure message transmission with optimal asynchronous fallback guarantees. Cryptology ePrint Archive, Report 2022/1397 (2022), https://eprint.iacr.org/2022/1397
13. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM Journal on Computing **12**(4), 656–666 (1983). https://doi.org/10.1137/0212045
14. Elsheimy, F., Tsimos, G., Papamanthou, C.: Deterministic byzantine agreement with adaptive $o(n \cdot f)$ communication. Cryptology ePrint Archive, Paper 2023/1723 (2023), https://eprint.iacr.org/2023/1723
15. Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: 20th Annual ACM Symposium on Theory of Computing. pp. 148–161. ACM Press (May 1988). https://doi.org/10.1145/62212.62225
16. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Journal of the ACM (JACM) **32**(2), 374–382 (1985)

17. Fitzi, M., Garay, J.A.: Efficient player-optimal protocols for strong and differential consensus. In: Borowsky, E., Rajsbaum, S. (eds.) 22nd ACM Symposium Annual on Principles of Distributed Computing. pp. 211–220. Association for Computing Machinery (Jul 2003). https://doi.org/10.1145/872035.872066
18. Fitzi, M., Hirt, M.: Optimally efficient multi-valued Byzantine agreement. In: Ruppert, E., Malkhi, D. (eds.) 25th ACM Symposium Annual on Principles of Distributed Computing. pp. 163–168. Association for Computing Machinery (Jul 2006). https://doi.org/10.1145/1146381.1146407
19. Gao, Y., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Efficient asynchronous byzantine agreement without private setups. In: 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS). pp. 246–257 (2022). https://doi.org/10.1109/ICDCS54860.2022.00032
20. Ghinea, D., Goyal, V., Liu-Zhang, C.D.: Round-optimal byzantine agreement. In: Dunkelman, O., Dziembowski, S. (eds.) Advances in Cryptology – EUROCRYPT 2022, Part I. Lecture Notes in Computer Science, vol. 13275, pp. 96–119. Springer, Heidelberg (May / Jun 2022). https://doi.org/10.1007/978-3-031-06944-4_4
21. Ghinea, D., Liu-Zhang, C.D., Wattenhofer, R.: Optimal synchronous approximate agreement with asynchronous fallback. In: Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing. pp. 70–80 (2022)
22. Ghinea, D., Liu-Zhang, C.D., Wattenhofer, R.: Multidimensional approximate agreement with asynchronous fallback. Cryptology ePrint Archive (2023)
23. Heilman, E., Kendler, A., Zohar, A., Goldberg, S.: Eclipse attacks on bitcoin's peer-to-peer network. In: Jung, J., Holz, T. (eds.) USENIX Security 2015: 24th USENIX Security Symposium. pp. 129–144. USENIX Association (Aug 2015)
24. Karlin, A., Yao, A.: Probabilistic lower bounds for byzantine agreement. Unpublished document (1986)
25. Katz, J., Koo, C.Y.: On expected constant-round protocols for byzantine agreement. In: Dwork, C. (ed.) Advances in Cryptology – CRYPTO 2006. Lecture Notes in Computer Science, vol. 4117, pp. 445–462. Springer, Heidelberg (Aug 2006). https://doi.org/10.1007/11818175_27
26. King, S., Nadal, S.: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. self-published paper, August **19**(1) (2012)
27. Lamport, L., Shostak, R., Pease, M.: Concurrency: The Works of Leslie Lamport. Association for Computing Machinery, New York, NY, USA (2019), edited by Dahlia Malkhi
28. Mehlhorn, K., Sun, H.: Great ideas in theoretical computer science (2013), https://resources.mpi-inf.mpg.de/departments/d1/teaching/ss13/gitcs/lecture7.pdf
29. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Pomerance, C. (ed.) Advances in Cryptology – CRYPTO'87. Lecture Notes in Computer Science, vol. 293, pp. 369–378. Springer, Heidelberg (Aug 1988). https://doi.org/10.1007/3-540-48184-2_32
30. Momose, A., Ren, L.: Optimal communication complexity of authenticated byzantine agreement. In: Gilbert, S. (ed.) 35th International Symposium on Distributed Computing (DISC 2021). Leibniz International Proceedings in Informatics (LIPIcs), vol. 209, pp. 32:1–32:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.DISC.2021.32
31. Mostéfaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous binary byzantine consensus with $t < n/3$, $\mathcal{O}(n^2)$ messages, and $\mathcal{O}(1)$ expected time. J. ACM **62**(4) (2015). https://doi.org/10.1145/2785953

32. Mostéfaoui, A., Raynal, M.: Signature-free broadcast-based intrusion tolerance: Never decide a byzantine value. In: Lu, C., Masuzawa, T., Mosbah, M. (eds.) Principles of Distributed Systems. pp. 143–158. Springer Berlin Heidelberg, Berlin, Heidelberg (2010)
33. Mostéfaoui, A., Raynal, M.: Signature-free asynchronous byzantine systems: From multivalued to binary consensus with $t < n/3$, $\mathcal{O}(n^2)$ messages, and constant time. Acta Inf. **54**(5), 501–520 (2017). https://doi.org/10.1007/s00236-016-0269-y
34. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized business review (2008)
35. Nayak, K., Ren, L., Shi, E., Vaidya, N.H., Xiang, Z.: Improved extension protocols for byzantine broadcast and agreement. In: Attiya, H. (ed.) 34th International Symposium on Distributed Computing (DISC 2020). Leibniz International Proceedings in Informatics (LIPIcs), vol. 179, pp. 28:1–28:17. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2020). https://doi.org/10.4230/LIPIcs.DISC.2020.28
36. Nguyen, L.: Accumulators from bilinear pairings and applications. In: Menezes, A. (ed.) Topics in Cryptology – CT-RSA 2005. Lecture Notes in Computer Science, vol. 3376, pp. 275–292. Springer, Heidelberg (Feb 2005). https://doi.org/10.1007/978-3-540-30574-3_19
37. Patra, A., Rangan, C.P.: Communication optimal multi-valued asynchronous byzantine agreement with optimal resilience. In: Fehr, S. (ed.) ICITS 11: 5th International Conference on Information Theoretic Security. Lecture Notes in Computer Science, vol. 6673, pp. 206–226. Springer, Heidelberg (May 2011). https://doi.org/10.1007/978-3-642-20728-0_19
38. Pfitzmann, B., Waidner, M.: Information-theoretic pseudosignatures and byzantine agreement for $t \geq n/3$. IBM Research, Armonk, NY, USA (1996)
39. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the Society for Industrial and Applied Mathematics **8**(2), 300–304 (1960). https://doi.org/10.1137/0108018

Appendix

A Additional Preliminaries

Collision-Resistant Hash Functions: Given a message m, one can compute the cryptographic hash of m of length $\mathcal{O}(\kappa)$, denoted $\mathsf{Hash}(m)$. We say that Hash is *collision-resistant* if it is computationally infeasible to find distinct messages m_1 and m_2 such that $\mathsf{Hash}(m_1) = \mathsf{Hash}(m_2)$. The difficulty of finding collisions should apply not only for the adversary, but for honest parties as well. For example, the protocol $\mathsf{HBAX}^{(\delta)}$ is secure assuming the adversary can't find collisions and that the honest parties have non-colliding inputs. To be precise, when we use collision-resistant hash functions in our consensus extension protocols, we prove security with the idealized assumption that collisions can never be found.

Cryptographic Accumulators: We present a slightly modified version of the definition by Nayak et al. [35]. Intuitively, the cryptographic accumulation of a set S is a short value c_{acc} such that for any value v, if and only if $v \in S$, one can efficiently compute a witness w which together with c_{acc} proves that $v \in S$. Formally, given n, the accumulation scheme specifies the following efficient algorithms:

- $\mathsf{KGen}(1^\kappa)$: Given the computational security parameter κ, this algorithm generates an accumulator key ak that can be used to securely accumulate sets of size up to n. If the scheme requires a key, then we assume that a trusted dealer generates and distributes ak. Otherwise, we say $\mathsf{ak} = \perp$.
- $\mathsf{Eval}_{\mathsf{ak}}(S)$: This is a deterministic algorithm which outputs the accumulation c_{acc} of a set S of size up to n.
- $\mathsf{Wit}_{\mathsf{ak}}(S, v)$: Given a set S of size up to n and any $v \in S$, this algorithm outputs a witness w_i which proves that $v \in S$.
- $\mathsf{Vfy}_{\mathsf{ak}}(c_{\mathsf{acc}}, v, w)$: Given v, w and an accumulation c_{acc} which equals $\mathsf{Eval}_{\mathsf{ak}}(S)$ for some set S of size up to n, this algorithm outputs 1 if w proves that $v \in S$, and it outputs 0 otherwise.

For any set S of size up to n, the following properties must hold:

- *Correctness:* For all $v \in S$, $\mathsf{Vfy}_{\mathsf{ak}}(\mathsf{Eval}_{\mathsf{ak}}(S), v, \mathsf{Wit}_{\mathsf{ak}}(S, v)) = 1$.
- *Collision Resistance:* For any $v \notin S$, it is computationally infeasible to compute a witness w such that $\mathsf{Vfy}_{\mathsf{ak}}(\mathsf{Eval}_{\mathsf{ak}}(S), v, w) = 1$.

Two pertinent accumulators are the bilinear accumulator [36], based on bilinear pairings, and the Merkle tree [29], based on collision-resistant hash functions. Both have accumulations of size $\mathcal{O}(\kappa)$. The Merkle tree is restricted in that it can only handle indexed sets of size n (of the form $\{(1, v_1), (2, v_2), \dots, (n, v_n)\}$), though this suffices for us. Meanwhile, the bilinear is more flexible in that it uses not S but only $\mathsf{c}_{\mathsf{acc}}$ for the computation of witnesses. For us, the relevant advantage of the bilinear accumulator is that it has $\mathcal{O}(\kappa)$-sized witnesses; whereas the Merkle tree has $\mathcal{O}(\kappa \log n)$-sized witnesses. On the other hand, the Merkle tree (unlike the bilinear accumulator) doesn't use a key, which enables its use without trusted setup.

Error Correcting Codes: We use Reed-Solomon error correcting codes [39] in our extension protocols. Given $b \leq n$, we use the code which encodes an input tuple of b symbols in the Galois Field $GF(2^a)$ into a codeword of n symbols in $GF(2^a)$, where $2^a - 1 \geq n$. The codeword can later be decoded back into the input tuple, with resilience against errors and erasures. Concretely, we get the following two functionalities:

- $\mathsf{Encode}_b(m)$: Given the message m of length $\ell = a \cdot b$, split it into a tuple of b symbols $(m_1, \ldots, m_b)$ in $GF(2^a)$, and then use the code to output the codeword $(s_1, \ldots, s_n)$.
- Decode_b: Given a codeword $(s_1, \ldots, s_n)$, decode it into $(m_1, \ldots, m_b)$ and output m. The decoding procedure can tolerate up to c errors and d erasures in the codeword symbols if $n - b \geq 2c + d$.

We use error correcting codes to reduce communication complexity. The key observation is that when we use a code to encode a message m of length $\ell = a \cdot b$ with $b \geq \delta n$ for a positive constant δ, we get $a = \frac{\ell}{b} \leq \frac{\ell}{\delta n} = \mathcal{O}(\frac{\ell}{n})$.

When we encode a message m of an arbitrary length ℓ, we need to pad m so that it reaches a length $\ell' = a \cdot b$, where $a \geq \log_2(n+1)$. This can require a padding of one bit per symbol if ℓ is sufficiently large, and $\mathcal{O}(\log n)$ bits per symbol otherwise. In our extension protocols, the parties send $\mathcal{O}(n^2)$ symbols in total, and hence the communication complexity overhead of padding is $\mathcal{O}(n^2 \log n)$. By the natural assumption $\kappa = \Omega(\log n)$, we simplify this to $\mathcal{O}(n^2 \kappa)$.

Expanders: Expanders are graphs with good connectivity properties despite being sparse, with small sets of vertices guaranteed to have large neighborhoods. We use them, as Momose and Ren did [30], to reduce protocol complexity when the corruption thresholds are below their optimal values by at least εn for some positive constant ε. We impose an n-vertex expander graph on the parties, with each party corresponding to a vertex. Then, the parties send their messages to their neighbors in the expander instead of multicasting them. Since the expander has a constant maximum degree dependent only on ε, this shaves a factor of $O(n)$ from the message and communication complexities. Below, we formally define expanders, and then prove their existence with a deterministic construction.

Lemma 8 (Existence of Expanders). *Given n, α and β, we call a graph $G = (V, E)$ an (n, α, β)-expander if $|V| = n$ and for all $S \subseteq V$ with $|S| \geq \alpha n$, the open neighborhood of S contains more than βn vertices. For all $0 < \alpha, \beta < 1$, there exists some d depending on α and β such that for all n, there exists a d-regular (n, α, β)-expander.*

Proof. The theorem can be straightforwardly proven via the probabilistic method. Momose and Ren present such a proof [30], which we could adapt to show that for some sufficiently large d depending on α and β, the union of d random perfect matchings is an (n, α, β)-expander except with probability decaying exponentially with n. However, we prefer to present a proof that also gives us explicit expander constructions with which we can deterministically instantiate our protocols.

In the following, we allow graphs to have loops and multi-edges, and we use the convention that a loop edge (v, v) adds 1 to the degree of the vertex v. Also, we assume that n is sufficiently large so that $\alpha n - 1 > 0$; this is the case if $n > \frac{1}{\alpha}$.

Let us call a graph $G = (V, E)$ with n vertices a (K, A)-vertex expander if in G, all sets $S \subseteq V$ of size at most K have open neighborhoods of size at least $A \cdot |S|$. If we set $K = \alpha n$ and $A \geq \frac{\beta n}{\alpha n - 1}$, then sets of size $\lfloor \alpha n \rfloor \leq \alpha n$ have at least $\frac{\lfloor \alpha n \rfloor \beta n}{\alpha n - 1} > \beta n$ neighbors, and therefore G is an (n, α, β)-expander. Now, we have the following:

- (Alon, [1]): For every constant degree $d \geq 3$, there exists a deterministic $\mathsf{poly}(n)$ algorithm which, for all sufficiently large $n \geq n_0(d)$ with nd even, outputs a d-regular n-vertex graph G with spectral expansion $\lambda \leq \frac{2\sqrt{d-1}+1}{d}$. Here, letting $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ be a descending list of the eigenvalues (with repetition) of the adjacency matrix of G, the spectral expansion is defined to be the value $\frac{\max\{|\lambda_2|, |\lambda_n|\}}{d}$ [28].
- ([28]): Let G be an n-vertex graph with spectral expansion λ. For all α where $0 < \alpha < 1$, G is an $(\alpha n, \frac{1}{(1-\alpha)\lambda^2 + \alpha})$-vertex expander.

Combining these results, we get our desired explicit constructions. Let d be an even integer such that $d \geq 5$. We have $2\sqrt{d-1} + 1 \leq \sqrt{5d}$, and so, assuming $n \geq n_0(d)$ for some constant $n_0(d)$ that depends only on d, we can in $\mathsf{poly}(n)$ time construct a d-regular n-vertex graph with spectral expansion $\sqrt{5/d}$.

Now, we show that for sufficiently large n, choosing d appropriately depending on α and β makes the constructed graph G an (n, α, β)-expander. Solving $\frac{1}{(1-\alpha)\lambda^2+\alpha} \geq \frac{\beta n}{\alpha n - 1}$ for λ^2 gives us $\lambda^2 \leq \frac{\alpha(1-\beta)n-1}{\beta(1-\alpha)n}$. To eliminate extraneous solutions we also require that $\lambda^2 \geq 0$; this is compatible with the previous inequality if we assume $n \geq \frac{1}{\alpha(1-\beta)}$. Hence, assuming $n \geq \frac{1}{\alpha(1-\beta)}$, if $\lambda^2 \leq \frac{\alpha(1-\beta)n-1}{\beta(1-\alpha)n}$, then G is an $(\alpha n, A)$-vertex expander with $A \geq \frac{\beta n}{\alpha n - 1}$, which as we discussed previously makes it an (n, α, β)-expander. Finally, since $\lambda \leq \sqrt{5/d}$, we can ensure $\lambda^2 \leq \frac{5}{d} \leq \frac{\alpha(1-\beta)n-1}{\beta(1-\alpha)n}$ by setting $d \geq \frac{5\beta(1-\alpha)n}{\alpha(1-\beta)n-1}$. Note that we actually need $n > \frac{1}{\alpha(1-\beta)}$ to avoid division by 0 in the last inequality.

We do not want d to depend on n. Fortunately, if we assume $n \geq \frac{6}{\alpha(1-\beta)}$, then we get $\frac{6\beta(1-\alpha)}{\alpha(1-\beta)} \geq \frac{5\beta(1-\alpha)n}{\alpha(1-\beta)n-1}$, and so it suffices to choose d to be the least even integer such that $d \geq \frac{6\beta(1-\alpha)}{\alpha(1-\beta)}$.

To conclude, we also need to consider the case where n is not sufficiently large. Then, we can simply let G be a complete graph with loops, ensuring that every non-empty set of vertices has the open neighborhood V. The sufficiency threshold for n depends not only on α and β, but also on d. However, as d itself only depends on α and β, the threshold ultimately only depends on α and β. Hence, the proof isn't adversely impacted.

B Proofs for Section 3

Lemma 1 (Security of SWC). *If $t_a \leq t_s$ and $2t_s + t_a \leq (1-\delta)n$, then SWC is a weak consensus protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-weak consistency and t_a-fallback validity.*

Proof.

- **$(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$-intrusion tolerance:** Suppose less than δn honest parties have some $m \in \mathcal{M}$ as input. Less than δn honest parties sign m, which means that there is never a $(t_s + \delta n)$-certificate on m. This implies that honest parties do not output m.
- **synchronous $\mathbf{t_s}$-robustness:** When the network is synchronous and the adversary is able to corrupt t_s parties, no honest party aborts. This is because there are at least $n - t_s$ honest parties who all multicast validly signed messages in the first round, which all get delivered in time since the network is synchronous. We implicitly use this property in the proofs of synchronous t_s-validity and synchronous t_s-weak consistency.
- **synchronous $\mathbf{t_s}$-validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$. In Round 1, all honest parties multicast signed m messages, leading to $n - t_s \geq t_s + \delta n$ signed m messages getting multicast. Hence, all honest parties are able to form $(t_s + \delta n)$-certificates on m by the end of the first round. Meanwhile, for any $m' \in \mathcal{M} \setminus \{m\}$, no honest party ever signs m', so there is never a $(t_s + \delta n)$-certificate on m'. All of this means that every honest party P_i sets $y_i = m$ at the end of Round 1 and then never changes y_i again.
- **synchronous $\mathbf{t_s}$-weak consistency:** Suppose some honest party P_i outputs $m \in \mathcal{M}$. Then it was able to form a $(t_s + \delta n)$-certificate on m and only m at the end of Round 1. P_i multicasts a $(t_s + \delta n)$-certificate on m in Round 2, and every honest party P_j sees it and outputs either m or $\perp$.
- **$\mathbf{t_a}$-fallback validity:** Suppose all honest parties with input have the same input $m \in \mathcal{M}$. Consider an honest party P_i with input. At the end of Round 1,
  - If P_i hasn't seen validly signed messages from $n - t_s$ parties, P_i aborts.
  - Suppose P_i has seen validly signed messages from $n - t_s$ parties. At most t_a of these come from corrupt parties, so P_i has seen at least $n - t_s - t_a \geq t_s + \delta n$ signed m messages from honest parties. Meanwhile, for any $m' \in \mathcal{M} \setminus \{m\}$, no honest party ever signs m', so there is never a $(t_s + \delta n)$-certificate on m'. All of this means that P_i sets $y_i = m$ at the end of Round 1 and then never changes y_i again.

Lemma 2 (Security of SProp). *If $t_a \leq t_s$ and $2t_s + t_a \leq (1-\delta)n$, then SProp is a proposal protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-weak consistency and t_a-fallback validity.*

Proof.

- **$(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$-intrusion tolerance:** Suppose less than δn honest parties have some $m \in \mathcal{M}$ as input. Less than δn honest parties sign m, which means that there is never a $(t_s + \delta n)$-certificate on m. This implies that honest parties do not output m or $\{m, \bot\}$.
- **synchronous $\mathbf{t_s}$-robustness:** When the network is synchronous and the adversary is able to corrupt t_s parties, no honest party aborts. This is because there are at least $n - t_s$ honest parties who all multicast valid messages in the first round, which all get delivered in time since the network is synchronous. We implicitly use this property in the proofs of synchronous t_s-validity and synchronous t_s-weak consistency.
- **synchronous $\mathbf{t_s}$-validity:** Suppose all honest parties have the same input $v \in S$.
  - If $v = \bot$, then by the $(t_s, \delta n)$-intrusion tolerance of SProp, all honest parties output $\bot$.
  - If $v = x$, then in Round 1 all honest parties multicast signed x messages, leading to $n - t_s \geq t_s + \delta n$ signed x messages getting multicast. Hence, all honest parties are able to form $(t_s + \delta n)$-certificates on x by the end of the first round. Meanwhile, for any $x' \in \mathcal{M} \setminus \{x\}$, no honest party ever signs x', so there is never a $(t_s + \delta n)$-certificate on x'. All of this means that every honest party P_i sets $y_i = x$ at the end of Round 1 and then never changes y_i again.
- **synchronous $\mathbf{t_s}$-weak consistency:** Suppose some honest party P_i outputs x. Then it was able to form a $(t_s + \delta n)$-certificate on x and only x at the end of Round 1. P_i multicasts a $(t_s + \delta n)$-certificate on x in Round 2, and every honest party P_j sees it and outputs a non-$\bot$ value.
- **$\mathbf{t_a}$-fallback validity:** Suppose all honest parties with input have the same input $v \in S$. If $v = \bot$, then by the $(t_s, \delta n)$-intrusion tolerance of SProp, all honest parties with input output $\bot$ or abort. Now suppose $v = x$. Consider an honest party P_i with input. At the end of Round 1,
  - If P_i hasn't seen valid messages from $n - t_s$ parties, then P_i aborts.
  - Suppose P_i has seen valid messages from $n - t_s$ parties. At most t_a of these come from corrupt parties, so P_i has seen at least $n - t_s - t_a \geq t_s + \delta n$ signed x messages from honest parties. Meanwhile, for any $x' \in \mathcal{M} \setminus \{x\}$, no honest party ever signs x', so there is never a $(t_s + \delta n)$-certificate on x'. All of this means that P_i sets $y_i = x$ at the end of Round 1 and then never changes y_i again.

Lemma 3 (Security of SGC^1). *Let SWC and SProp respectively be a weak consensus protocol and a proposal protocol, both with fixed numbers of rounds,* *such that they both have synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-weak consistency and t_a-fallback validity. Furthermore, suppose* SProp *has $(t_s, \delta n)$-intrusion tolerance. Then,* SGC^1 *is a 1-graded consensus protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-graded validity, synchronous t_s-graded consistency and t_a-fallback graded validity.*

Proof. We use the synchronous t_s-robustness of SWC and SProp implicitly.

- $\mathbf{(t_s, \delta n)}$**-intrusion tolerance:** Suppose less than δn honest parties have some $m \in \mathcal{M}$ as input. By design, honest parties do not provide SProp the input m unless they have the input m for SGC^1. Hence, less than δn honest parties provide SProp the input m, and so by the $(t_s, \delta n)$-intrusion tolerance of AProp, no honest party P_i obtains $z_i = m$ or $z_i = \{m, \bot\}$. This implies that an honest party P_i never obtains $y_i = m$.
- **synchronous $\mathbf{t_s}$-robustness and synchronous $\mathbf{t_s}$-graded consistency:** By the synchronous t_s-weak consistency of SWC, there exists some $m \in \mathcal{M}$ such that all honest parties output m or $\bot$ from SWC. This leads to all honest parties running SProp with inputs in $\{m, \bot\}$. By the synchronous t_s-weak consistency of SProp, one of the following happens:
  - All honest parties output m or $\{m, \bot\}$ from SProp. Then, every honest party P_i obtains $y_i = m$.
  - All honest parties output $\{m, \bot\}$ or $\bot$ from SProp. Then, every honest party P_i obtains $g_i = 0$.
- **synchronous $\mathbf{t_s}$-graded validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$. By the synchronous t_s-validity of SWC, all honest parties output m from SWC. Then, as all honest parties have the input m for SGC^1, they all use the input m for SProp. And so, by the synchronous t_s-validity of SProp, they all output m from SProp, and thus output $(m, 1)$ from SGC^1.
- $\mathbf{t_a}$**-fallback graded validity:** Suppose all honest parties with input have the same input $m \in \mathcal{M}$. By the t_a-fallback validity of SWC, all honest parties with input either abort or output m from SWC. Then, all remaining honest parties with input use the input m for SProp, and so by the t_a-fallback validity of SProp, they either abort or output m from SProp as well. Therefore, all honest parties with input either abort or output $(m, 1)$.

Lemma 4 (Security of SGC^2). *Let* SGC^1 *and* SWC *respectively be a 1-graded consensus protocol and a weak binary consensus protocol, both with fixed numbers of rounds, such that* SGC^1 *has $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-graded validity, synchronous t_s-graded consistency and t_a-fallback graded validity, and* SWC *has synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-weak consistency and t_a-fallback validity. Then,* SGC^2 *is a 2-graded consensus protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-graded validity, synchronous t_s-graded consistency and t_a-fallback graded validity.*

Proof. We use the synchronous t_s-robustness of SGC^1 and SWC implicitly.

- $(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$**-intrusion tolerance:** Suppose less than δn honest parties have some $m \in \mathcal{M}$ as input. By the $(t_s, \delta n)$-intrusion tolerance of SGC^1, no honest party P_i obtains $z_i = m$. This implies that an honest party P_i never obtains $y_i = m$.
- **synchronous $\mathbf{t_s}$-robustness and synchronous $\mathbf{t_s}$-graded consistency:** The synchronous t_s-graded consistency of SGC^1 guarantees that the honest parties either obtain the output grade 0 from SGC^1 or obtain some common output value $m \in \mathcal{M}^{\perp}$ from SGC^1.
  - If the former happens, then all honest parties use the input 0 for SWC, and so by the synchronous t_s-validity of SWC, output 0 from it. Therefore, all honest parties obtain the output grade 0.
  - If the latter happens, then all honest parties obtain the output value m. In addition, by the synchronous t_s-weak consistency of SWC, there exists a bit b such that all honest parties obtain outputs in $\{b, \perp\}$ from SWC. If $b = 0$, then all honest parties obtain the output grades 0 or 1, and if $b = 1$, then that all honest parties obtain the output grades 1 or 2.
- **synchronous $\mathbf{t_s}$-graded validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$. By the synchronous t_s-graded validity of SGC^1, all honest parties output $(m, 1)$ from SGC^1. Then, all honest parties use the input 1 for SWC, and so by the synchronous t_s-validity of SWC, they output 1 from it. Therefore, all honest parties output $(m, 2)$.
- $\mathbf{t_a}$**-fallback graded validity:** Suppose all honest parties with input have the same input $m \in \mathcal{M}$. By the t_a-fallback graded validity of SGC^1, all honest parties either abort or output $(m, 1)$ from SGC^1. Then, all remaining honest parties with input use the input 1 for SWC, and so by the t_a-fallback validity of SWC, they either abort or output 1 from it. Therefore, all honest parties with input either abort or output $(m, 2)$.

Theorem 2 (Security of SBA^*). *Let SGC^2 and SBA respectively be 2-graded consensus protocol and a binary consensus protocol, both with fixed numbers of rounds, such that SGC^2 has $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-graded validity, synchronous t_s-graded consistency and t_a-fallback graded validity, and SBA has synchronous t_s-robustness, synchronous t_s-validity and synchronous t_s-consistency. Then, SBA^* is a consensus protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-consistency and t_a-fallback validity.*

Proof.

- $(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$**-intrusion tolerance:** Suppose less than δn honest parties have some $m \in \mathcal{M}$ as input. By the $(t_s, \delta n)$-intrusion tolerance of SGC^2, no honest party P_i obtains $z_i = m$. This implies that honest parties do not output m.
- **synchronous $\mathbf{t_s}$-robustness:** This follows from synchronous t_s-robustness of SGC^2. We implicitly use the synchronous t_s-robustness of SBA^* and SGC^2 in the proofs of synchronous t_s-validity and synchronous t_s-consistency.

- **synchronous $\mathbf{t_s}$-validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$. By the synchronous t_s-graded validity of SGC^2, all honest parties output $(m, 2)$ from SGC^2. Therefore, all honest parties output m.
- **synchronous $\mathbf{t_s}$-consistency:** Using the synchronous t_s-robustness of SBA implicitly, we have three cases to consider:
  - Suppose some honest party P_i outputs $(m, 2)$ from SGC^2 for some $m \in \mathcal{M}^\perp$. By the synchronous t_s-graded consistency of SGC^2, all honest parties output $(m, 2)$ or $(m, 1)$ from SGC^2. This means that all honest parties use the input 1 for SBA, and by the synchronous t_s-validity of SBA, output 1 from it. At the end, every honest party P_i obtains $z_i = m$ and $h_i = 1$ and therefore outputs m.
  - Suppose some honest party outputs $(m, 1)$ from SGC^2 for some $m \in \mathcal{M}^\perp$, and no honest party obtains the output grade 2 from SGC^2. Then, the synchronous t_s-graded consistency of SGC^2 implies that all honest parties output $(m, 1)$ or $(m, 0)$ from it. Afterwards, by the synchronous t_s-consistency of SBA, either all honest output 1 from SBA or all honest parties output 0 from SBA. If the former happens, then all honest parties output m. If the latter happens, then all honest parties output $\perp$.
  - Suppose all honest parties obtain the output grade 0 from SGC^2. Then, all honest parties use the input 0 for SBA, and by the synchronous t_s-validity of SBA, output 0 from it. At the end, every honest party P_i obtains $g_i = 0$ and $h_i = 0$, so every honest party outputs $\perp$.
- **$\mathbf{t_a}$-fallback validity:** Suppose all honest parties with input have the same input $m \in \mathcal{M}$. By the t_a-fallback graded validity of SGC^2, all honest parties with input either abort or output $(m, 2)$ from SGC^2. Therefore, all honest parties with input either abort or output m, no matter what output they are able to obtain from SBA, if any.

C Expander-Based Weak Consensus and Proposal

In this section we present $\mathsf{SWC}^{(\varepsilon)}$ and $\mathsf{SProp}^{(\varepsilon)}$, the expander-based variants of SWC and SProp. These variants are 3-round drop-in replacements of their namesakes, and they use an $(n, \varepsilon, 1-\varepsilon)$-expander G_ε imposed on the parties to reduce communication complexity from $\mathcal{O}(n^3\kappa + \ell n^2)$ to $\mathcal{O}(n^2\kappa + \ell n^2)$. The price they pay for this reduction is that they require $2t_s \leq (1-\varepsilon)n$ for a positive constant ε rather than just $2t_s < n$.

C.1 SWC – Expander Variant

In $\mathsf{SWC}^{(\varepsilon)}$, the parties no longer multicast certificates in the second round, but instead send them to their neighbors in G_ε. The sparsity of G_ε reduces the communication complexity. Although the change removes weak consistency, the expansion property of G_ε ensures that all but less than εn honest parties receive the pertinent certificates they would have received if G_ε were complete. This lets us regain weak consistency via a third voting round, as done in [30].

Protocol $\mathsf{SWC}^{(\varepsilon)}$

Input: Party P_i has an input $m_i \in \mathcal{M}$ which it knows at the beginning, or it may have no input if the network is asynchronous

Output: Party P_i either aborts or outputs $y_i \in \mathcal{M}^{\perp}$.

Initialization: If P_i has input, then P_i sets $y_i = \perp$. Else, P_i aborts.

Round 1: P_i computes $\sigma_i = \mathsf{Sgn}_{\mathsf{sk}_i}(m_i)$ and multicasts (m_i, σ_i). At the end of the round,

- If P_i hasn't received validly signed messages from $n - t_s$ parties, P_i aborts.
- Else, if the messages P_i received are so that P_i can form a $(t_s+\delta n)$-certificate on a unique $m \in \mathcal{M}$, then P_i sets $y_i = m$.

Round 2: If P_i possesses a $(t_s + \delta n)$-certificate on a unique $m \in \mathcal{M}$, then P_i sends it to its neighbors in G_ε. At the end of the round, if P_i has received a $(t_s + \delta n)$-certificate on some $m \neq y_i$ from a neighbor in G_ε, then P_i sets $y_i = \perp$.

Round 3: P_i multicasts y_i. At the end of the round, if P_i has received from more than t_s parties messages which differ from y_i, P_i sets $y_i = \perp$.

Lemma 9. *If $t_a \leq t_s$, $2t_s \leq (1-\varepsilon)n$ and $2t_s + t_a \leq (1-\delta)n$, then $\mathsf{SWC}^{(\varepsilon)}$ is a weak consensus protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-weak consistency and t_a-fallback validity.*

Proof.

- $(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$**-intrusion tolerance:** Suppose less than δn honest parties have some $m \in \mathcal{M}$ as input. Less than δn honest parties sign m, which means there is never a $(t_s + \delta n)$-certificate on m. This implies that honest parties do not output m.
- **synchronous** $\mathbf{t_s}$**-robustness:** When the network is synchronous and the adversary is able to corrupt t_s parties, no honest party aborts. This is because there are at least $n - t_s$ honest parties who all multicast validly signed messages in the first round, which all get delivered in time since the network is synchronous. We implicitly use this property in the proofs of synchronous t_s-validity and synchronous t_s-weak consistency.
- **synchronous** $\mathbf{t_s}$**-validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$, and let P_i be an honest party. In Round 1, all honest parties multicast signed m messages, leading to $n - t_s \geq t_s + \delta n$ signed m messages getting multicast. Hence, P_i is able to form a $(t_s+\delta n)$-certificate on m by the

end of the first round. Meanwhile, for any $m' \in \mathcal{M} \setminus \{m\}$, no honest party ever signs m', so there is never a $(t_s + \delta n)$-certificate on m'. All of this means that P_i sets $y_i = m$ and keeps it unchanged until the beginning of Round 3. Finally, in Round 3, only corrupt parties send any message other than m, so P_i keeps y_i unchanged as it sees at most t_s messages that differ from m.

- **synchronous $\mathbf{t_s}$-weak consistency:** We consider two scenarios:
  - Suppose that for some $m \in \mathcal{M}$, at least εn honest parties P_i have $y_i = m$ at the end of Round 1. In Round 2, all of these parties send $(t_s + \delta n)$-certificates on m to their neighbors in G_ε. By the expansion property of G_ε, more than $n - \varepsilon n$ parties are able to receive the certificates, more than $n - \varepsilon n - t_s \geq t_s$ of which are honest. All of these receiving honest parties obtain the tentative outputs m or $\perp$ at the end of Round 2, and so they multicast m or $\perp$ in the beginning of Round 3. In Round 3, every honest party P_i who has $y_i = m'$ for some $m' \in \mathcal{M} \setminus \{m\}$ in the beginning of the round receives from more than t_s honest parties the messages m or $\perp$, and therefore sets $y_i = \perp$. So, we conclude that every honest party outputs either m or $\perp$.
  - Suppose that for all $m \in \mathcal{M}$, less than εn honest parties have the tentative output m at the end of Round 1. Then this state of affairs is preserved until the end of Round 2, as during Round 2 honest parties may only change their tentative outputs to $\perp$. For all $m \in \mathcal{M}$, more than $n - \varepsilon n - t_s \geq t_s$ honest parties multicast messages other than m in Round 3, so every honest party P_i who has $y_i = m$ in the beginning of the round receives from more than t_s honest parties messages other than m, and therefore sets $y_i = \perp$. From this, we conclude that all honest parties output $\perp$.
- **$\mathbf{t_a}$-fallback validity:** Suppose all honest parties with input have the same input $m \in \mathcal{M}$. Consider an honest party P_i with input. At the end of Round 1,
  - If P_i hasn't seen validly signed messages from $n - t_s$ parties, P_i aborts.
  - Suppose P_i has seen validly signed messages from $n - t_s$ parties. At most t_a of these come from corrupt parties, so P_i has seen at least $n - t_s - t_a \geq t_s + \delta n$ signed m messages from honest parties. Meanwhile, for any $m' \in \mathcal{M} \setminus \{m\}$, no honest party ever signs m', so there is never a $(t_s + \delta n)$-certificate on m'. All of this means that P_i sets $y_i = m$ and keeps it unchanged until the beginning of Round 3. Finally, in Round 3, only corrupt parties send any message other than m, so P_i keeps y_i unchanged as it sees at most t_s messages that differ from m.

Complexity of $\mathsf{SWC}^{(\varepsilon)}$: The message complexity is $\mathsf{MC}_{\mathsf{SWC}^{(\varepsilon)}} = \mathcal{O}(n^2)$ due to the multicasting. In the first round the multicast messages are of length $\mathcal{O}(\ell + \kappa)$ and in the third round the multicast messages are of length $\mathcal{O}(\ell)$, so from these rounds we get the communication complexity $\mathcal{O}(n^2\kappa + \ell n^2)$. In the second round, each party possibly sends a certificate of size $\mathcal{O}(\ell + n\kappa)$ to $\mathcal{O}(1)$ neighbors in G_ε, so from this round we get the communication complexity $\mathcal{O}(n^2\kappa + \ell n)$. In total, we get the communication complexity $\mathsf{CC}_{\mathsf{SWC}^{(\varepsilon)}} = \mathcal{O}(n^2\kappa + \ell n^2)$.

C.2 SProp – Expander Variant

Again in $\mathsf{SProp}^{(\varepsilon)}$, the parties send certificates instead of multicasting them, and weak consistency is regained via a third round. As we did for SProp, we assume there exists some set $S = \{x, \bot\} \in \mathcal{M}^{\bot}$ such that honest parties can only have inputs in S, and use it in our description of $\mathsf{SProp}^{(\varepsilon)}$ below.

Protocol $\mathsf{SProp}^{(\varepsilon)}$

Input: Party P_i has an input $v_i \in S$ which it knows at the beginning, or it may have no input if the network is asynchronous.

Output: Party P_i either aborts or outputs $y_i \in \{x, \{x, \bot\}, \bot\}$.

Initialization: If P_i has input, then P_i sets $y_i = \bot$. Else, P_i aborts.

Round 1: If $v_i = \bot$, then P_i multicasts $\bot$. Else, P_i computes $\sigma_i = \mathsf{Sgn}_{\mathsf{sk}_i}(v_i)$ and multicasts (v_i, σ_i). At the end of the round,

- If P_i hasn't received valid messages from $n - t_s$ parties, then P_i aborts.
- Else, if the messages P_i received are so that P_i can form a $(t_s + \delta n)$-certificate on a unique $m \in \mathcal{M}$, then P_i sets $y_i = m$.

Round 2: If P_i possesses a $(t_s + \delta n)$-certificate on y_i, then P_i sends it to its neighbors in G_ε. At the end of the round, if P_i had $y_i = \bot$ but received a $(t_s + \delta n)$-certificate on some $m \in \mathcal{M}$ from a neighbor in G_ε, then P_i sets $y_i = \{m, \bot\}$.

Round 3: P_i multicasts y_i. At the end of the round, if P_i has received from more than t_s parties messages which differ from y_i, then

- If $y_i \in \mathcal{M}$, then P_i sets $y_i = \{m, \bot\}$.
- If $y_i = \{m, \bot\}$ for some $m \in \mathcal{M}$, then P_i keeps y_i unchanged.
- If $y_i = \bot$, P_i inspects the messages it received to find some $m \in \mathcal{M}$ such that more than t_s of the messages are either m or $\{m, \bot\}$, and sets $y_i = \{m, \bot\}$.

Lemma 10. *If $t_a \leq t_s$, $2t_s \leq (1-\varepsilon)n$ and $2t_s + t_a \leq (1-\delta)n$, then $\mathsf{SProp}^{(\varepsilon)}$ is a proposal protocol with $(t_s, \delta n)$-intrusion tolerance, synchronous t_s-robustness, synchronous t_s-validity, synchronous t_s-weak consistency and t_a-fallback validity.*

Proof.

- **$(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$-intrusion tolerance:** Suppose less than δn honest parties have some $m \in \mathcal{M}$ as input. Less that δn honest parties sign m, which means there is never a $(t_s + \delta n)$-certificate on m. This implies that honest parties do not switch their tentative outputs to m or $\{m, \bot\}$ in the first two rounds. In the third round, no honest party multicasts m or $\{m, \bot\}$, so no honest party receives the messages m or $\{m, \bot\}$ from more than t_s parties. This implies that honest parties do not switch their outputs to m or $\{m, \bot\}$ in the third round either.

- **synchronous $\mathbf{t_s}$-robustness:** When the network is synchronous and the adversary is able to corrupt t_s parties, no honest party aborts. This is because there are at least $n - t_s$ honest parties who all multicast valid messages in the first round, which all get delivered in time since the network is synchronous. We implicitly use this property in the proofs of synchronous t_s-validity and synchronous t_s-weak consistency.
- **synchronous $\mathbf{t_s}$-validity:** Suppose all honest parties have the same input $v \in S$, and let P_i be an honest party.
  - If $v = \perp$, then by $(t_s, \delta n)$-intrusion tolerance, P_i outputs $\perp$.
  - If $v = x$, then in Round 1 all honest parties multicast signed x messages, leading to $n - t_s \geq t_s + \delta n$ signed x messages getting multicast. Hence, P_i is able to form a $(t_s + \delta n)$-certificate on x by the end of the first round. Meanwhile, for any $x' \in \mathcal{M} \setminus \{x\}$, no honest party ever signs x', so there is never a $(t_s + \delta n)$-certificate on x'. All of this means that P_i sets $y_i = x$ and keeps it unchanged until the beginning of Round 3. Finally, in Round 3, only corrupt parties send any message other than x, so P_i keeps y_i unchanged as it sees at most t_s messages that differ from x.
- **synchronous $\mathbf{t_s}$-weak consistency:** Firstly, note that we have already shown in the proof of $(t_s, \delta n)$-intrusion tolerance that for any $x' \in \mathcal{M} \setminus \{x\}$, no $(t_s + \delta n)$-certificate on x' ever exists, and no honest party P_i ever sets $y_i = x'$ or $y_i = \{x', \perp\}$. Now, we consider two scenarios.
  - Suppose at least εn honest parties have $(t_s + \delta n)$-certificates on x at the end of Round 1. In Round 2, all of these parties send $(t_s + \delta n)$-certificates on x to their neighbors in G_ε. By the expansion property of G_ε, more than $n - \varepsilon n$ parties are able to receive the certificates, more than $n - \varepsilon n - t_s \geq t_s$ of which are honest. All of these receiving honest parties obtain the tentative outputs x or $\{x, \perp\}$ at the end of Round 2, and so they multicast x or $\{x, \perp\}$ in the beginning of Round 3. In Round 3, every honest party P_i who has $y_i = \perp$ in the beginning of the round sets $y_i = \{x, \perp\}$ at the end of the round, because during the round, P_i receives from more than t_s honest parties the messages x or $\{x, \perp\}$, and for any $x' \in \mathcal{M} \setminus \{x\}$, receives only from corrupt parties (hence from at most t_s parties) the messages x' or $\{x', \perp\}$. So, we conclude that every honest party outputs either x or $\{x, \perp\}$.
  - Suppose less than εn honest parties have $(t_s + \delta n)$-certificates on x at the end of Round 1. An honest party P_i never sets $y_i = x$ unless it has a $(t_s + \delta n)$-certificate on x at the end of Round 1, so less than εn honest parties have the tentative output x at the end of Round 2. In Round 3, more than $n - \varepsilon n - t_s \geq t_s$ honest parties multicast messages other than x, so every honest party P_i with $y_i = x$ in the beginning of the round sets y_i to $\{x, \perp\}$. From this, we conclude that no honest party outputs x.
- **$\mathbf{t_a}$-fallback validity:** Suppose all honest parties with input have the same input $v \in S$. If $v = \perp$, then by the $(t_s, \delta n)$-intrusion tolerance of $\mathsf{SProp}^{(\varepsilon)}$, all

honest parties with input output $\bot$ or abort. Now suppose $v = x$. Consider an honest party P_i with input. At the end of Round 1,

- If P_i hasn't seen valid messages from $n - t_s$ parties, then P_i aborts.
- Suppose P_i has seen valid messages from $n - t_s$ parties. At most t_a of these come from corrupt parties, so P_i has seen at least $n - t_s - t_a \geq t_s + \delta n$ signed x messages from honest parties. Meanwhile, for any $x' \in \mathcal{M} \setminus \{x\}$, no honest party ever signs x', so there is never a $(t_s + \delta n)$-certificate on x'. All of this means that P_i sets $y_i = x$ and keeps it unchanged until the beginning of Round 3. Finally, in Round 3, only corrupt parties send any message other than x, so P_i keeps y_i unchanged as it sees at most t_s messages that differ from x.

Complexity of $\mathsf{SProp}^{(\varepsilon)}$: The message and communication complexities of $\mathsf{SProp}^{(\varepsilon)}$ are also respectively $\mathsf{MC}_{\mathsf{SProp}^{(\varepsilon)}} = \mathcal{O}(n^2)$ and $\mathsf{CC}_{\mathsf{SProp}^{(\varepsilon)}} = \mathcal{O}(n^2\kappa + \ell n^2)$, for the same reasons that $\mathsf{SWC}^{(\varepsilon)}$ attains these complexities.

D Additional Material for Section 4

D.1 Asynchronous Graded Consensus

Protocol AGC^1

Input: Party P_i has an input $m_i \in \mathcal{M}$ which it can eventually learn.

Output: Party P_i outputs (y_i, g_i), where $y_i \in \mathcal{M}^{\bot}$ is the output value and $g_i \in \{0, 1\}$ is the output grade.

Initialization: Party P_i starts participating in a common instance of AWC and a common instance of AProp.

Input Acquisition: When P_i knows m_i, P_i sets it as its input for AWC.

Prop Rule: Upon outputting x_i from AWC, P_i checks if $x_i = m_i$. If so, then P_i sets x_i as its AProp input. Otherwise, P_i sets $\bot$ its AProp input.

Output Rule: Upon outputting z_i from AProp,

– If $z_i = m$, then P_i outputs $(m, 1)$.

– If $z_i = \{m, \bot\}$, then P_i outputs $(m, 0)$.

– If $z_i = \bot$, then P_i outputs $(\bot, 0)$.

Lemma 6 (Security of AGC^1). *Let AProp and AWC respectively be a proposal protocol and a weak consensus protocol such that AProp has $(t_s, \delta n)$-intrusion tolerance, r_p-round t_s-validity with liveness, t_a-weak consistency and t_a-liveness, and AWC has r_w-round t_s-validity with liveness, t_a-weak consistency and t_a-liveness. Then, AGC^1 is a 1-graded consensus protocol with $(t_s, \delta n)$-intrusion* *tolerance, $(r_w + r_p)$-round t_s-graded validity with liveness, t_a-graded consistency and t_a-liveness.*

Proof.

- **$(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$-intrusion tolerance:** Suppose less than δn honest parties have some $m \in \mathcal{M}$ as input. The "Prop Rule" is designed so that honest parties do not provide AProp the input m unless they have the input m for AGC^1. Hence, less than δn honest parties provide AProp the input m, and so by the $(t_s, \delta n)$-intrusion tolerance of AProp, no honest party P_i obtains $z_i = m$ or $z_i = \{m, \perp\}$. This implies that an honest party P_i never obtains $y_i = m$.
- **$(\mathbf{r_w} + \mathbf{r_p})$-round $\mathbf{t_s}$-graded validity with liveness:** Suppose all honest parties have the same input $m \in \mathcal{M}$. By the r_w-round t_s-validity with liveness of AWC, all honest parties output m from AWC. Then, as all honest parties have the input m for AGC^1, they all provide AProp the input m. And so, by the r_p-round t_s-validity with liveness of AProp, all honest parties output m from AProp as well. This means that all honest parties output $(m, 1)$. Furthermore, if the network is synchronous and all honest parties know their inputs by time k, then all honest parties output m from AWC and therefore provide AProp the input m by time $k + r_w\Delta$, and then, all honest parties output m from AProp and therefore output $(m, 1)$ from AGC^1 by time $k + r_w\Delta + r_p\Delta = k + (r_w + r_p)\Delta$.
- **$\mathbf{t_a}$-liveness and $\mathbf{t_a}$-graded consistency:** All honest parties eventually provide input to AWC, and thus, by the t_a-weak consistency and t_a-liveness of AWC, obtain outputs in $\{m, \perp\}$ from it for some $m \in \mathcal{M}$. This leads to them providing inputs in $\{m, \perp\}$ to AProp. Finally, by the t_a-weak consistency and t_a-liveness of AProp, one of the following happens:
  - All honest parties output m or $\{m, \perp\}$ from AProp. In this case, all honest parties obtain the same output value.
  - All honest parties output $\{m, \perp\}$ or $\perp$ from AProp. In this case, all honest parties obtain the output grade 0.

Protocol AGC^2

Input: Party P_i has an input m_i which it can eventually learn.

Output: Party P_i outputs (y_i, g_i), where y_i is the output value and g_i is the output grade.

Initialization: Party P_i starts participating in a common instance of AGC^1 and a common instance of AWC.

Input Acquisition: When P_i knows m_i, P_i sets it as its input for AGC^1.

1-Grade Rule: Upon outputting (z_i, h_i) from AGC^1, P_i sets h_i as its input for AWC.

Output Rule: Upon outputting (z_i, h_i) from AGC^1 and v_i from AWC, P_i sets $y_i = z_i$ and outputs (y_i, g_i) after setting g_i as follows:

- If $v_i = 0$, then P_i sets $g_i = 0$.
- If $v_i = \bot$, then P_i sets $g_i = 1$.
- If $v_i = 1$, then P_i sets $g_i = 2$.

Lemma 7 (Security of AGC^2). *Let AGC^1 and AWC respectively be a 1-graded consensus protocol and a weak binary consensus protocol such that AGC^1 has $(t_s, \delta n)$-intrusion tolerance, r_g-round t_s-graded validity with liveness, t_a-graded consistency and t_a-liveness, and AWC has r_w-round t_s-graded validity with liveness, t_a-weak consistency and t_a-liveness. Then, AGC^2 is a 2-graded consensus protocol with $(t_s, \delta n)$-intrusion tolerance, $(r_g + r_w)$-round t_s-graded validity with liveness, t_a-graded consistency and t_a-liveness.*

Proof.

- $(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$**-intrusion tolerance:** Suppose less than δn honest parties have some $m \in \mathcal{M}$ as input. By the $(t_s, \delta n)$-intrusion tolerance of AGC^1, no honest party P_i obtains $z_i = m$. This implies that an honest party P_i never obtains $y_i = m$.
- $(\mathbf{r_g} + \mathbf{r_w})$**-round** $\mathbf{t_s}$**-graded validity with liveness:** Suppose all honest parties have the same input $m \in \mathcal{M}$. By the r_g-round t_s-graded validity with liveness of AGC^1, all honest parties output $(m, 1)$ from AGC^1. Then, all honest parties provide AWC the input 1, and so by the 2-round t_s-validity with liveness of AWC, they output 1 from it. Therefore, all honest parties output $(m, 2)$. Furthermore, if the network is synchronous and all honest parties know their inputs by time k, then all honest parties output $(m, 1)$ from AWC and therefore provide AWC the input 1 by time $k + r_g\Delta$, and then, all honest parties output 1 from AWC and therefore output $(m, 2)$ from AGC^2 by time $k + r_g\Delta + r_w\Delta = k + (r_g + r_w)\Delta$.
- $\mathbf{t_a}$**-liveness and** $\mathbf{t_a}$**-graded consistency:** All honest parties eventually provide input to AGC^1, and thus, by the t_a-liveness of AGC^1, obtain output from it. By the t_a-graded consistency of AGC^1, either all honest parties obtain some common output value $y \in \mathcal{M}^\bot$ from AGC^1, or all honest parties obtain the output grade 0 from AGC^1.
    - Suppose all honest parties obtain some common output value y from AGC^1. All honest parties provide input to AWC, and so by the t_a-liveness and t_a-weak consistency of AWC, either all honest parties output 1 or $\bot$ from it, or all honest parties output $\bot$ or 0 from it. If the former happens, then all honest parties output $(y, 2)$ or $(y, 1)$. If the latter happens, then all honest parties output $(y, 1)$ or $(y, 0)$.
    - Suppose all honest parties obtain the output grade 0 from AGC^1. All honest parties provide AWC the input 0, and so by the t_s-graded validity

with liveness of AWC, they output 0 from it. This means that all honest parties obtain the output grade 0.

D.2 Skipped Proofs

Lemma 5 (Security of AProp). *If $t_a \leq t_s$ and $2t_s + t_a \leq (1 - \delta)n$, then AProp is a proposal protocol with $(t_s, \delta n)$-intrusion tolerance, 2-round t_s-validity with liveness, t_a-weak consistency and t_a-liveness.*

Proof.

- **$(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$-intrusion tolerance:** Suppose less than δn honest parties have some $m \in \mathcal{M}$ as input. Less than δn honest parties send the message $(\mathbf{input}, m)$ via the "Input Acquisition" rule. This implies that honest parties do not send the message $(\mathbf{input}, m)$ via the "Conflict Echoing" rule. Suppose otherwise, for the sake of contradiction. The first time some honest party P_i sends $(\mathbf{input}, m)$ via the "Conflict Echoing" rule, it needs to have received the message $(\mathbf{input}, m)$ from $t_s + \delta n$ parties. Among these $t_s + \delta n$ parties, at least δn must be honest parties who sent the message via the "Input Acquisition" rule, which gives us a contradiction. And so, we conclude that less than δn honest parties send the message $(\mathbf{input}, m)$ via any rule. This means that no honest party receives the message $(\mathbf{input}, m)$ from $t_s + \delta n$ parties. As $t_s + \delta n \leq n - t_s$, honest parties do not activate the "Value Rule" with the message $(\mathbf{input}, m)$, and so no honest party P_i adds m to V_i or multicasts $(\mathbf{propose}, m)$. Hence, honest parties do not output sets containing m, and since they receive the message $(\mathbf{propose}, m)$ from at most $t_s < n - t_s$ parties, they do not output m either.
- **V-validity lemma:** Suppose for some $v \in \mathcal{M}^{\perp}$ that no honest party has the input v, and suppose the adversary can corrupt at most t_s parties. Then no honest party P_i adds v to V_i, and no honest party P_i outputs v or a set containing v. The proof of this lemma proceeds similarly to the proof of $(t_s, \delta n)$-intrusion tolerance. No honest party sends the message $(\mathbf{input}, v)$ via the "Input Acquisition rule." This implies that no honest party sends the message $(\mathbf{input}, v)$ via the "Conflict Echoing" rule either. Suppose otherwise, for the sake of contradiction. The first time some honest party P_i sends $(\mathbf{input}, v)$ via the "Conflict Echoing" rule, it needs to have received the message $(\mathbf{input}, v)$ from $t_s + 1$ parties. Among these $t_s + 1$ parties, at least one must be an honest party who sent the message via the "Input Acquisition" rule, which gives us a contradiction. And so, we conclude that no honest party sends the message $(\mathbf{input}, v)$ via any rule. Then, as honest parties receive the message $(\mathbf{input}, v)$ from at most $t_s < n - t_s$ parties, they do not activate the "Value Rule" with the message $(\mathbf{input}, v)$, and so no honest party P_i adds v to V_i or multicasts $(\mathbf{propose}, v)$. Hence, honest parties do not output sets containing v, and since they receive the message $(\mathbf{propose}, v)$ from at most $t_s < n - t_s$ parties, they do not output v either.

- **$\mathbf{2}$-round $\mathbf{t_s}$-validity with liveness:** Suppose all honest parties have the same input $v \in S$. By the V-validity lemma, honest parties may only output v; this gives us t_s-validity. Meanwhile,
  - All honest parties multicast $(\textbf{input}, v)$. This leads to each honest party P_i receiving the message $(\textbf{input}, v)$ from $n - t_s$ parties and therefore adding v to V_i.
  - Whenever an honest party P_i adds v to V_i, it multicasts $(\textbf{propose}, v)$. This is because P_i would not do so only if it had added some $v' \in \mathcal{M}^{\perp} \setminus \{v\}$ to V_i earlier, which by the V-validity lemma can't happen. And so, since every honest party P_i adds v to V_i and multicasts $(\textbf{propose}, v)$ when it does so, all honest parties receive the message $(\textbf{propose}, v)$ from $n - t_s$ parties and therefore become able to output v.

  Furthermore, if the network is synchronous and all honest parties know their inputs by time k, then in the reasoning chain above, all honest parties multicast $(\textbf{input}, v)$ by time k; all honest parties receive all honest party $(\textbf{input}, v)$ messages and therefore multicast $(\textbf{propose}, v)$ by time $k + \Delta$, and finally, all honest parties receive all honest party $(\textbf{propose}, v)$ messages and therefore become able to output v by time $k + 2\Delta$.
- **V-consistency lemma:** Suppose the adversary is able to corrupt t_a parties. If some honest party P_i adds some $v \in \mathcal{M}^{\perp}$ to V_i, then every honest party P_j adds v to V_j. To see why, note that if P_i adds v to V_i, it does so because it has received the message $(\textbf{input}, v)$ from $n - t_s$ parties. At least $n - t_s - t_a \geq t_s + \delta n$ of these parties are honest, which means that there are at least $t_s + \delta n$ honest parties who multicast $(\textbf{input}, v)$. These multicast messages ensure via the "Conflict Echoing" rule that all honest parties multicast $(\textbf{input}, v)$, whether $v = \perp$ or not. And so, it is ensured that every honest party P_j receives the message $(\textbf{input}, v)$ from $n - t_s$ parties and therefore adds v to V_j.
- **V-adding lemma:** Suppose the adversary is able to corrupt t_a parties. Then every honest party P_i adds some $v \in S$ to V_i. To see why, let H_x and $H_{\perp}$ be the sets of honest parties with the input x and with the input $\perp$, respectively. We have $|H_x| + |H_{\perp}| \geq n - t_a$, which by $n - t_a \geq 2t_s + \delta n$ means that at least one of the inequalities $|H_x| \geq t_s + \delta n$ and $|H_{\perp}| > t_s$ must hold.
  - If $|H_x| \geq t_s + \delta n$, then at least $t_s + \delta n$ honest parties multicast $(\textbf{input}, x)$, which as we showed in the proof of the V-consistency lemma leads to every honest party P_i adding x to V_i.
  - If $|H_{\perp}| > t_s$, then at least $t_s + 1$ honest parties multicast $(\textbf{input}, \perp)$. This leads to all honest parties receiving the message $(\textbf{input}, \perp)$ from at least $t_s + 1$ parties and therefore multicasting $(\textbf{input}, \perp)$. And so, it is ensured that every honest party P_j receives the message $(\textbf{input}, \perp)$ from $n - t_s$ parties and therefore adds $\perp$ to V_j.
- **$\mathbf{t_a}$-liveness:** By the V-validity lemma, no honest party P_i adds anything but x and $\perp$ to V_i. Furthermore, by the V-adding lemma, eventually every honest party P_i adds some $v \in S$ to V_i. We have two cases to consider:

- Suppose some honest party P_i adds x to V_i while some honest party P_j adds $\perp$ to V_j. By the V-consistency lemma, every honest party P_k eventually adds both x and $\perp$ to V_k and thus becomes able to output S.
- Suppose there exists some $v \in S$ such that no honest party P_i adds v to V_i. Let v' be the other value in S. For every honest party P_i, v' is the unique value that P_i adds to V_i, which means that P_i multicasts $(\textbf{propose}, v')$. The fact that all honest parties multicast $(\textbf{propose}, v')$ ensures that all honest parties receive the message $(\textbf{propose}, v)$ from $n - t_s$ parties and therefore become able to output v'.

– **$\mathbf{t_a}$-weak consistency:** Suppose some honest party P_i outputs x while some other honest party P_i outputs $\perp$. Then at least $n - t_s$ parties must have sent the message $(\textbf{propose}, x)$ to P_i, and at least $n - t_s$ parties must have sent the message $(\textbf{propose}, \perp)$ to P_j. The number of double-proposers, that is, the number of parties that sent $(\textbf{propose}, x)$ to P_i and $(\textbf{propose}, \perp)$ to P_j, must therefore be at least $n - 2t_s \geq t_a + 1$. But this means that there is at least one double-proposing honest party, which is a contradiction.

Theorem 3 (Security of AWC). *If $t_a \leq t_s$ and $2t_s + t_a < n$, then AWC is a weak consensus protocol with 2-round t_s-validity with liveness, t_a-weak consistency and t_a-liveness.*

Proof.

– **conflict-validity lemma:** Suppose all honest parties have the same input $m = (b_1, \ldots, b_\ell) \in \mathcal{M}$, and suppose the adversary can corrupt at most t_s parties. Then no honest party multicasts **conflict**. Suppose otherwise, for the sake of contradiction. The first time some honest party P_i multicasts **conflict**, it needs to have received from $t_s + 1$ parties inputs other than m or the message **conflict**. Since P_i is assumed to be the first honest party to send the message **conflict**, among these $t_s + 1$ parties, at least one must be an honest party who sent an input other than m. This contradicts the fact that all honest parties have the input m.

– **V-validity lemma:** Suppose all honest parties have the same input $m \in \mathcal{M}$ where $m = (b_1, \ldots, b_\ell)$, and suppose the adversary can corrupt at most t_s parties. Then for all k, no honest party P_i adds $1 - b_k$ to V_i^k, and furthermore, honest parties can only output m. Let k be a bit index and P_i be an honest party. To see why the lemma is true, note that honest parties do not send inputs with the k^{th} bit $1 - b_k$, and by the conflict-validity lemma, they do not send **conflict** messages either. This means P_i receives inputs with the k^{th} bit $1 - b_k$ or the message **conflict** from at most $t_s < n - t_s$ parties and therefore does not add $1 - b_k$ to V_i^k, and that makes b_k is the unique bit which P_i may add to V_i^k. Hence, we see that P_i doesn't output $\perp$ or multicast $(\textbf{propose}, m')$ for any $m' \in \mathcal{M}$ that disagrees with m at any bit. Finally, for any $m' \in \mathcal{M} \setminus \{m\}$, as no honest party multicasts $(\textbf{propose}, m')$, no honest party receives the message $(\textbf{propose}, m')$ from $n - t_s > t_s$ parties, and so no honest party outputs m'.

- **2-round $\mathbf{t_s}$-validity with liveness:** Suppose all honest parties have the same input $m = (b_1, \ldots, b_\ell) \in \mathcal{M}$. By the V-validity lemma, honest parties may only output m; this gives us t_s-validity. Meanwhile,
    - All honest parties multicast $(\mathbf{input}, m)$. For all k, this leads to each honest party P_i receiving an input with the k^{th} bit b_k from $n - t_s$ parties and therefore adding b_k to V_i^k.
    - By the V-validity lemma, for all k, b_k is the unique bit which an honest party P_i can add to V_i^k. This, combined with the previous bullet point, means that every honest party P_i eventually observes $V_i^1, V_i^2, \ldots V_i^\ell = \{b_1\}, \{b_2\}, \ldots, \{b_\ell\}$ and therefore multicasts $(\mathbf{propose}, m)$. This leads to all honest parties receiving the message $(\mathbf{propose}, m)$ from $n - t_s$ parties and therefore becoming able to output m.

  Furthermore, if the network is synchronous and all honest parties know their inputs by time k, then in the reasoning chain above, all honest parties multicast $(\mathbf{input}, m)$ by time k; all honest parties receive all honest party $(\mathbf{input}, m)$ messages and therefore multicast $(\mathbf{propose}, m)$ by time $k + \Delta$, and finally, all honest parties receive all honest party $(\mathbf{propose}, m)$ messages and therefore become able to output m by time $k + 2\Delta$.
- **V-consistency lemma:** Suppose the adversary is able to corrupt t_a parties. For all k, if some honest party P_i adds the bit b to V_i^k, then every honest party P_j adds b to V_j^k. To see why, note that if P_i adds b to V_i^k, it does so because it has received inputs with the k^{th} bit b or the message **conflict** from $n - t_s$ parties. At least $n - t_s - t_a \geq t_s + 1$ of these parties are honest, which means that there are at least $t_s + 1$ honest parties who multicast inputs with the k^{th} bit b or the message **conflict**. These multicast messages ensure that every honest party P_j either multicasts an input with the k^{th} bit b if it has such an input, or multicasts **conflict** otherwise. And so, it is ensured that every honest party P_j receives inputs with the k^{th} bit b or the message **conflict** from $n - t_s$ parties and therefore adds b to V_j^k.
- **V-adding lemma:** Suppose the adversary is able to corrupt t_a parties. Then for all k, there exists a bit b such that every honest party P_i adds b to V_i^k. To see why, for some arbitrary k, let H_0 and H_1 be the sets of honest parties who have inputs with the k^{th} bit 0 and the k^{th} bit 1, respectively. We have $|H_0| + |H_1| \geq n - t_a$, which by $2t_s + t_a < n$ means that at least one of the inequalities $|H_0| > t_s$ and $|H_1| > t_s$ must hold. Without loss of generality, suppose $|H_0| > t_s$. Then at least $t_s + 1$ honest parties multicast inputs with the k^{th} bit 0, which as we showed in the proof of the V-consistency lemma leads to every honest party P_i adding 0 to V_i^k.
- $\mathbf{t_a}$**-liveness:** We have two cases to consider:
    - Suppose there exists some k such that some honest party P_i adds 0 to V_i^k while some other honest party P_j adds 1 to V_j^k. By the V-consistency lemma, every honest party P_q eventually adds both 0 and 1 to V_q^k and therefore becomes able to output $\perp$.

- Suppose there exists no such k. Then, by the V-adding lemma, for all k, there exists a unique bit b_k such that every honest party P_i adds b_k and not $1 - b_k$ to V_i^k. Each honest party P_i therefore eventually observes $V_i^1, V_i^2, \dots, V_i^k = \{b_1\}, \{b_2\}, \dots, \{b_\ell\}$ and multicasts $(\textbf{propose}, m)$, where $m = (b_1, b_2, \dots, b_\ell) \in \mathcal{M}$. This leads to all honest parties receiving the message $(\textbf{propose}, m)$ from $n - t_s$ parties and therefore becoming able to output m.

– $\mathbf{t_a}$**-weak consistency:** Suppose that for some distinct $m, m' \in \mathcal{M}$, some honest party P_i outputs m while some other honest party P_j outputs m'. Then at least $n - t_s$ parties must have sent the message $(\textbf{propose}, m)$ to P_i, and at least $n - t_s$ parties must have sent the message $(\textbf{propose}, m')$ to P_j. The number of double-proposers, that is, the number of parties that sent $(\textbf{propose}, m)$ to P_i and $(\textbf{propose}, m')$ to P_j, must therefore be at least $n - 2t_s \geq t_a + 1$. But this means that there is at least one double-proposing honest party, which is a contradiction.

Theorem 4 (Security of ABA^*). *Let $t_a \leq t_s$ and $2t_s + t_a \leq (1 - \delta)n$, and suppose honest parties must know their inputs by time $r_s\Delta$ when the network is synchronous. Let AGC^2 and ABA respectively be a 2-graded consensus protocol and a binary consensus protocol such that AGC^2 has $(t_s, \delta n)$-intrusion tolernace, r_g-round t_s-graded validity with liveness, t_a-graded consistency and t_a-liveness, and ABA has t_a-validity, t_a-consistency and t_a-liveness. Then, ABA^* is a consensus protocol with $(t_s, \delta n)$-intrusion tolerance, t_s-validity, t_a-consistency, t_a-termination and synchronous $(r_g + 1)$-round t_s-validity with termination.*

Proof. For the sake of brevity, we say that an honest party commits to $m \in \mathcal{M}^\perp$ if it multicasts $(\textbf{commit}, m)$.

– **commit safety lemma:** Suppose the adversary can corrupt at most t_s parties, and suppose for some $x \in \mathcal{M}^\perp$ that no honest party commits to x via the "Graded Output" or the "Late Output" rules. Then no honest party commits to x via the "Commit Processing" rule either. Suppose otherwise, for the sake of contradiction. The first time some honest party P_i commits to x via the "Commit Processing" rule, it needs to have received the message $(\textbf{commit}, x)$ from $t_s + 1$ parties. Among these $t_s + 1$ parties, at least one must be an honest party who committed to x via the "Graded Output" or the "Late Output" rules, which is a contradiction.

– $(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$**-intrusion tolerance:** Suppose less than δn honest parties have some $m \in \mathcal{M}$ as input. By the $(t_s, \delta n)$-intrusion tolerance of AGC^2, no honest party obtains the output value m from AGC^2. This implies that honest parties do not commit to m via the "Graded Output" or the "Late Output" rules. By the commit safety lemma, honest parties do not commit to m via the "Commit Processing" rule either. Finally, as no honest party commits to m, no honest party receives the message $(\textbf{commit}, m)$ from $n - t_s > t_s$ parties, and so no honest party outputs m.

- $\mathbf{t_a}$**-commit unanimity lemma:** Suppose the adversary can corrupt at most t_a parties. Then there is a unique $m \in \mathcal{M}^\perp$ to which honest parties may commit. Firstly, we show that there is a unique $m \in \mathcal{M}^\perp$ to which honest parties may commit via the "Graded Output" or the "Late Commit" rules. Considering only these rules, we have three cases to consider:
  - Suppose some honest parties output $(m, 2)$ from AGC^2 for some $m \in \mathcal{M}^\perp$. Then, by the t_a-graded consistency of AGC^2, honest parties can only output $(m, 2)$ or $(m, 1)$ from AGC^2. This means that honest parties can only provide ABA the input 1, which by the t_a-validity of ABA means that honest parties can only output 1 from ABA. And so, honest parties can only commit to m, either because they output $(m, 2)$ from AGC^2 or because they output $(m, 1)$ from AGC^2 and 1 from ABA.
  - Suppose some honest party outputs $(m, 1)$ from AGC^2 for some $m \in \mathcal{M}^\perp$, and suppose no honest party obtains the output grade 2 from AGC^2. Then no honest party commits to anything via the "Graded Output" rule, and the t_a-graded consistency of AGC^2 implies that honest parties can only output $(m, 1)$ or $(m, 0)$ from AGC^2. As for ABA, its t_a-consistency means that there is a common bit b that honest parties can obtain as output from it. If $b = 1$, then honest parties can only commit to m via the "Late Output" rule, and if $b = 0$, then honest parties can only commit to $\perp$ via the "Late Output" rule.
  - Suppose no honest party obtains the output grades 2 or 1 from AGC^2. Then no honest party commits to anything via the "Graded Output" rule, and no honest party provides ABA the input 1. By the t_a-validity of ABA, honest parties can only output 0 from ABA and therefore can only commit to $\perp$ via the "Late Output" rule.

  By the above, let m be the unique value in $\mathcal{M}^\perp$ to which honest parties may commit via the "Graded Output" or the "Late Commit" rules. Then for any $m' \in \mathcal{M}^\perp \setminus \{m\}$, the commit safety lemma guarantees that no honest party commits to m' via any rule. This makes m the unique value in $\mathcal{M}^\perp$ to which honest parties may commit via any rule.
- **commit validity lemma:** Suppose the adversary can corrupt at most t_s parties, and suppose there is a unique $m \in \mathcal{M}^\perp$ to which honest parties may commit. Then honest parties may only output m. To see why, suppose an honest party outputs some $m' \in \mathcal{M}^\perp$. Then it must have received the message $(\textbf{commit}, m')$ from $n - t_s$ parties, at least $n - 2t_s \geq 1$ of which must be honest. Since honest parties can only commit to m, we conclude that $m' = m$.
- $\mathbf{t_a}$**-consistency:** By the t_a-commit unanimity lemma, there is a unique $m \in \mathcal{M}^\perp$ to which honest parties may commit. Therefore, by the commit validity lemma, honest parties can only output m.
- $\mathbf{t_a}$**-unanimous termination lemma:** Suppose the adversary can corrupt at most t_a parties. If some honest party terminates, then all honest parties terminate. To see why, suppose some honest party P_i terminates. For some

$m \in \mathcal{M}^{\perp}$, it must have received the message $(\mathbf{commit}, m)$ from $n - t_s$ parties, at least $n - t_s - t_a \geq t_s + 1$ of which must be honest. Since there are $t_s + 1$ honest parties who commit to m and since $t_s + 1 \leq n - t_s$, every honest party eventually receives the message $(\mathbf{commit}, m)$ from $t_s + 1$ parties while still running and therefore commits to m via the "Commit Processing" rule, assuming it hasn't done so previously. Finally, since all honest parties commit to m, all honest parties can eventually receive the message $(\mathbf{commit}, m)$ from $n - t_s$ parties and therefore terminate.

- $\mathbf{t_a}$**-termination:** If some honest party terminates, then by the t_a-unanimous termination lemma, all honest parties terminate. Thus, it suffices to prove that some honest party terminates. For the sake of contradiction, suppose otherwise. All honest parties provide input to AGC^2, and thus, by the t_a-liveness of AGC^2, obtain output from it. Hence, they provide input to ABA, any the t_a-liveness of ABA, obtain output from it. Now let $m \in \mathcal{M}^{\perp}$ be the unique value to which honest parties may commit, which exists by the t_a-commit unanimity lemma. All honest parties eventually obtain output from both AGC^2 and ABA. Thus, all honest parties eventually commit to m, leading to them all receiving the message $(\mathbf{commit}, m)$ from $n - t_s$ parties and therefore terminating. This contradicts the assumption that no honest party terminates.
- $\mathbf{t_s}$**-validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$. By the t_s-graded validity of AGC^2, honest parties can only output $(m, 2)$ from AGC^2. This means that honest parties can't commit to anything other than m via the "Graded Output" or the "Late Output" rules. By the commit safety lemma, m is the unique value in $\mathcal{M}^{\perp}$ to which honest parties may commit. Therefore, by the commit validity lemma, honest parties can only output m.
- **synchronous** $(\mathbf{r_g + 1})$**-round** $\mathbf{t_s}$**-validity with termination:** Suppose the network is synchronous, and suppose all honest parties have some common input $m \in \mathcal{M}$ which they all know by time $r_s \Delta$, as required by the protocol description when the network is synchronous. By the t_s-validity of ABA^*, m is the unique value in $\mathcal{M}^{\perp}$ which honest parties may output. Furthermore, as honest parties can't terminate before time $(r_s + r_g)\Delta$, the r_g-round t_s-graded-validity with liveness of AGC^2 guarantees that all honest parties output $(m, 2)$ from AGC^2 and therefore commit to m by time $(r_s + r_g)\Delta$. And so, by time $(r_s + r_g + 1)\Delta$, all honest parties receive the message $(\mathbf{commit}, m)$ from $n - t_s$ parties and therefore terminate.

E Network-Agnostic Extension Protocols

In this section we present our network-agnostic consensus extension protocols HBAX and $\mathsf{HBAX}^{(\delta)}$. Both of them extend an underlying intrusion-tolerant network-agnostic protocol with two additional rounds so that consensus on ℓ-bit inputs is reached with a communication complexity $\mathcal{O}(\ell n + \dots)$. The protocol HBAX is an adaptation from [35]; its communication complexity overhead is

$\mathcal{O}(n^2\kappa + \ell n)$ bits with trusted setup, or $\mathcal{O}(n^2\kappa \log n + \ell n)$ bits without. The novel extension protocol $\mathsf{HBAX}^{(\delta)}$ can use an expander to achieve a complexity overhead of $\mathcal{O}(n^2\kappa + \ell n)$ bits without trusted setup, but for this it requires for some $\delta = \Theta(1)$ that $2t_s + t_a \leq (1-\delta)n$ and that the underlying consensus protocol has $(t_s, \delta n)$-intrusion tolerance.

E.1 Extended Agreement (HBAX)

In HBAX, the parties encode their inputs into n symbols $s_1, \ldots, s_n$ with an error correcting code set to tolerate t_s erasures, and then compute cryptographic accumulations of their indexed codeword symbols. Then, they agree on a common cryptographic accumulation c_{acc} via an intrusion-tolerant consensus instance.[9] If $c_{\mathsf{acc}} = \bot$, then the final output is $\bot$. Otherwise, each party who had the input c_{acc} sends each party P_j the pair (s_j, w_j), where s_j is the j^{th} reconstruction symbol and w_j is a witness which proves s_j correct with respect to c_{acc}. Upon receiving (s_j, w_j), the party P_j multicasts (s_j, w_j). Eventually, every honest party P_i multicasts (s_i, w_i), and so every honest party receives $n - t_s$ witnessed symbols, which together suffice for output reconstruction.

Protocol HBAX

Input: Party P_i has an input $m_i \in \mathcal{M}$ which it must know at the beginning of the protocol if the network is synchronous.

Output: Party P_i outputs $y_i \in \mathcal{M}^{\bot}$.

- **Initialization:** P_i starts participating in a common instance of HBA and lets $A_i = \varnothing$. Once P_i knows its input m_i,
  - P_i computes the codeword $(s_1, s_2, \ldots, s_n) = \mathsf{Encode}_{n-t_s}(m_i)$.
  - P_i lets S be the indexed set $\{(1, s_1), (2, s_2), \ldots, (n, s_n)\}$ and computes the cryptographical accumulator $c_i = \mathsf{Acc}_{\mathsf{ak}}(S)$.
  - P_i sets c_i as its input for HBA.
- **Base Output:** Upon terminating HBA with the output c_{acc},
  - If $c_{\mathsf{acc}} = \bot$, then P_i outputs $\bot$ and terminates.
  - Otherwise, if $c_{\mathsf{acc}} = c_i$, then for each party P_j, P_i computes the witness $w_j = \mathsf{Wit}_{\mathsf{ak}}(S, (j, s_j))$ and sends the message $(\textbf{personal}, s_j, w_j)$ to P_j.
- **Symbol Casting:** Upon terminating HBA with the output $c_{\mathsf{acc}} \neq \bot$ and for the first time receiving a message $(\textbf{personal}, s, w)$ such that $\mathsf{Vfy}_{\mathsf{ak}}(c_{\mathsf{acc}}, (i, s), w) = 1$, P_i multicasts $(\textbf{global}, s, w)$.

[9] In [35], the underlying consensus protocol is not assumed to be intrusion-tolerant. Hence, after the parties agree on c_{acc}, they ensure intrusion tolerance with a separate binary consensus instance.

- **Symbol Accepting:** Upon receiving from a party P_j a message $(\mathbf{global}, s_j, w_j)$ where $\mathsf{Vfy}_{\mathsf{ak}}(c_{\mathsf{acc}}, (j, s_j), w_j) = 1$, P_i adds (j, s_j) to A_i.
- **Extended Termination:** Upon terminating HBA with the output $c_{\mathsf{acc}} \neq \bot$, running the "Base Output" and "Symbol Casting" rules and observing that $|A_i| \geq n - t_s$, P_i reconstructs an output $m \in \mathcal{M}$ using the accepted indexed symbols in A_i as input for Decode_{n-t_s}, and then terminates with the output m.

Theorem 6. *Let $t_a \leq t_s$, and suppose honest parties must know their inputs at time 0 when the network is synchronous. Let HBA be a consensus protocol with*

- *$(t_s, \delta n)$-intrusion tolerance;*
- *synchronous t_s-validity, synchronous t_s-consistency and synchronous r_h-round t_s-termination;*
- *t_a-validity, t_a-consistency and t_a-termination.*

Then HBAX is a consensus protocol with

- *$(t_s, \delta n)$-intrusion tolerance;*
- *synchronous t_s-validity, synchronous t_s-consistency and synchronous $(r_h + 2)$-round t_s-termination;*
- *t_a-validity, t_a-consistency and t_a-termination.*

Proof.

- **accumulator lemma:** Suppose the adversary can corrupt at most t_s parties. Let P_i be an honest party who has encoded its input m_i into the codeword $(s_1, s_2, \dots, s_n)$ and computed the cryptographic accumulation c_i of the indexed set $S = \{(1, s_1), (2, s_2), \dots, (n, s_n)\}$. Suppose $c_{\mathsf{acc}} = c_i$, where c_{acc} is the common output of HBA. Then no honest party verifies any indexed symbol except those in the set S, and no honest party outputs anything except m_i.
  - No honest party verifies any indexed symbol except those in the set S, because for any $v \notin S$, it is computationally infeasible for there to exist a witness w such that $\mathsf{Vfy}_{\mathsf{ak}}(c_{\mathsf{acc}}, v, w) = 1$.
  - No honest party outputs $\bot$, because $c_{\mathsf{acc}} \neq \bot$.
  - Suppose an honest party P_j outputs. Then it does so via the "Extended Termination" rule upon observing that A_j contains $n - t_s$ verified indexed symbols. All of these symbols must be elements of S, which means by the correctness of the error correcting code that P_j correctly reconstructs m_i and outputs m_i.
- **termination and consistency:** All honest parties participate in HBA with input, and keep on participating until termination. By the termination and consistency properties of HBA, all honest parties terminate it with some

common output c_{acc}. If $c_{\mathsf{acc}} = \perp$, then all honest parties terminate with the output $\perp$. Suppose $c_{\mathsf{acc}} \neq \perp$. By the $(t_s, \delta n)$-intrusion tolerance of HBA, $c_{\mathsf{acc}} = c_i$ for some honest party P_i who has encoded its input m_i into the codeword $(s_1, s_2, \ldots, s_n)$ and computed the accumulation c_i for the indexed set $S = \{(1, s_1), (2, s_2), \ldots, (n, s_n)\}$. The accumulator lemma implies that honest parties can only output m_i; this gives us consistency. To each party P_j, P_i sends the message $(\mathbf{personal}, s_j, w_j)$. This makes every honest party P_j able to verify s_j and therefore multicast $(\mathbf{global}, s_j, w_j)$. Finally, as every honest party P_j multicasts $(\mathbf{global}, s_j, w_j)$, all honest parties are able to collect witnessed indexed symbols from $n - t_s$ parties and therefore terminate with the output m_i.

Now, suppose the network is synchronous. The synchronous r_h-round t_s-termination of HBA implies that all honest parties terminate it by time $r_h \Delta$. Then, one of following happens:

- $c_{\mathsf{acc}} = \perp$. In this case, all honest parties terminate by time $r_h \Delta$.
- $c_{\mathsf{acc}} \neq \perp$. In this case, by time $r_h \Delta$, some honest party P_i who has $c_i = c_{\mathsf{acc}}$ sends each party P_j the message $(\mathbf{personal}, s_j, w_j)$. Then, by time $(r_h + 1)\Delta$, every honest party P_j receives the message $(\mathbf{personal}, s_j, w_j)$ and therefore multicasts $(\mathbf{global}, s_j, w_j)$. Finally, by time $(r_h + 2)\Delta$, all honest parties are able to collect witnessed indexed symbols from $n - t_s$ parties and therefore terminate with the output m_i.

– **validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$. All honest parties encode m into the symbols $(s_1, s_2, \ldots, s_n)$, compute the accumulation c_{acc} for the indexed set $S = \{(1, s_1), (2, s_2), \ldots, (n, s_n)\}$, participate in HBA with the input c_{acc}, and keep on participating until termination. By the termination and validity properties of HBA, all honest parties terminate HBA with the output c_{acc}. The accumulator lemma ensures that honest parties can output nothing but m.

– $(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$**-intrusion tolerance:** Suppose the honest parties don't output $\perp$. Then they must have obtained a common output $c_{\mathsf{acc}} \neq \perp$ from HBA, which by the $(t_s, \delta n)$-intrusion tolerance of HBA must have been provided to it as input by δn honest parties $P_{a_1}, P_{a_2}, \ldots, P_{a_{\delta n}}$. Let $m_{a_1}, m_{a_2}, \ldots, m_{a_{\delta n}}$ be the respective inputs of these honest parties. By the accumulator lemma, for all $k \in \{a_1, a_2, \ldots a_{\delta n}\}$, honest parties can output nothing but m_k. Therefore, if an honest party outputs some $m \in \mathcal{M}$, then $m = m_{a_1} = m_{a_2} = \cdots = m_{a_{\delta n}}$, which makes it an input held by at least δn honest parties.

Complexity of HBAX: For the complexity analysis assume $2t_s < n$, which is in any case required for synchronous consensus. Let $\mathsf{MC}_{\mathsf{HBA}}(\kappa)$ and $\mathsf{CC}_{\mathsf{HBA}}(\kappa)$ respectively be the message and communication complexities of running HBA on κ-bit messages, and let w be the size of accumulator witnesses. As $n - t_s = \Theta(n)$, the error correction symbols $s_1, s_2, \ldots, s_n$ each have size $\mathcal{O}(\frac{\ell}{n} + \log n) = \mathcal{O}(\frac{\ell}{n} + \kappa)$. Each party may send each party a **personal** symbol and a witness; this respectively induces the message and communication complexity costs $\mathcal{O}(n^2)$

and $\mathcal{O}(n^2(\frac{\ell}{n} + \kappa + w)) = \mathcal{O}(\ell n + n^2(\kappa + w))$. In addition, each party may multicast a **global** symbol and a witness, again with the previous complexity costs. Therefore, in total, we obtain the complexities $\mathsf{MC}_{\mathsf{HBAX}} = \mathcal{O}(\mathsf{MC}_{\mathsf{HBA}}(\kappa) + n^2)$ and $\mathsf{CC}_{\mathsf{HBAX}} = \mathcal{O}(\mathsf{CC}_{\mathsf{HBA}}(\kappa) + \ell n + n^2(w + \kappa))$.

- If we assume trusted setup, then we can use the bilinear accumulator, and therefore get $w = \mathcal{O}(\kappa)$. This gives us $\mathsf{CC}_{\mathsf{HBAX}} = \mathcal{O}(\mathsf{CC}_{\mathsf{HBA}}(\kappa) + \ell n + n^2\kappa)$.
- If we don't assume trusted setup, then we can set $\mathsf{ak} = \perp$ and use the Merkle tree. This gives us $w = \mathcal{O}(\kappa \log n)$ and therefore $\mathsf{CC}_{\mathsf{HBAX}} = \mathcal{O}(\mathsf{CC}_{\mathsf{HBA}}(\kappa) + \ell n + n^2\kappa \log n)$.

E.2 Extended Agreement – Expander Variant ($\mathsf{HBAX}^{(\delta)}$)

For $\mathsf{HBAX}^{(\delta)}$, we assume for some $\delta = \Theta(1)$ that $2t_s + t_a \leq (1 - \delta)n$ and that the underlying consensus protocol has $(t_s, \delta n)$-intrusion tolerance. This allows us to impose an $(n, \delta, 1 - \frac{\delta}{2})$-expander G_δ on the parties and achieve the complexity $\mathcal{O}(\mathsf{CC}_{\mathsf{HBA}}(\kappa) + \ell n + n^2\kappa)$ without trusted setup. In $\mathsf{HBAX}^{(\delta)}$ we eschew symbol verification, and instead use error correction deal with adversary-provided incorrect symbols. The parties hash their inputs, and then they agree on a common hash h. If $h = \perp$, then the final output is $\perp$. Otherwise, honest parties who have inputs which hash to h (at least δn parties) send their inputs to their neighbors in G_δ. Consequently, all but less than $\frac{\delta n}{2}$ honest parties acquire inputs which hash to h. An honest party P_i who acquires an input m which hashes to h computes $(s_1, s_2, \dots, s_n) = \mathsf{Encode}_{\lceil \delta n/2 \rceil}(m)$ and multicasts s_i. For proper reconstruction when the network is synchronous, a party only runs the decoding procedure after local time $(r_h + 2)\Delta$, where r_h is the synchronous round count of HBA. This way, if the network is synchronous, a party only reconstructs after receiving the symbols of all but less than $\frac{\delta n}{2}$ honest parties. The correctness of the protocol crucially relies on the assumption that if $h \neq \perp$, then the execution does not contain some $m \neq m'$ such that $\mathsf{Hash}(m) = h = \mathsf{Hash}(m')$.

Protocol $\mathsf{HBAX}^{(\delta)}$

Input: Party P_i has an input $m_i \in \mathcal{M}$ which it must know at the beginning of the protocol if the network is synchronous.

Output: Party P_i outputs $y_i \in \mathcal{M}^{\perp}$.

- **Initialization:** P_i starts participating in a common instance of HBA and lets $A_i = \varnothing$. Once P_i knows its input m_i, P_i computes $h_i = \mathsf{Hash}(m_i)$ and sets h_i as its input for HBA.
- **Base Output:** Upon terminating HBA with the output h,
  - If $h = \perp$, then P_i outputs $\perp$ and terminates.
  - Otherwise, if $h = h_i$, then P_i sends the message $(\textbf{input}, m_i)$ to its neighbors in G_δ.

- **Symbol Casting:** Upon terminating HBA with the output $h \neq \bot$ and receiving a message $(\mathbf{input}, m)$ such that $\mathsf{Hash}(m) = h$ from a neighbor in G_δ for the first time, P_i computes $(s_1, s_2, \ldots, s_n) = \mathsf{Encode}_{\lceil \delta n/2 \rceil}(m)$ and multicasts $(\mathbf{symbol}, s_i)$.
- **Symbol Accepting:** Upon receiving from a party P_j a message $(\mathbf{symbol}, s_j)$, P_i adds (j, s_j) to A_i.
- **Extended Termination:** After running the protocol for $(r_h + 2)\Delta$ time, if P_i has terminated HBA, run the "Base Output" rule and observed that $|A_i| \geq n - t_s - \lfloor \frac{\delta n}{2} \rfloor$,
  - P_i reconstructs an output $m \in \mathcal{M}$ using the the indexed symbols in A_i as input for $\mathsf{Decode}_{\lceil \delta n/2 \rceil}$.
  - If P_i hasn't done so already via the "Symbol Casting" rule, P_i computes $(s_1, s_2, \ldots, s_n) = \mathsf{Encode}_{\lceil \delta n/2 \rceil}(m)$ and multicasts $(\mathbf{symbol}, s_i)$.
  - P_i terminates with the output m.

Theorem 7. *Let $t_a \leq t_s$ and $2t_s + t_a \leq (1 - \delta)n$, and suppose honest parties must know their inputs at time 0 when the network is synchronous. Let HBA be a consensus protocol with*

- *$(t_s, \delta n)$-intrusion tolerance;*
- *synchronous t_s-validity, synchronous t_s-consistency and synchronous r_h-round t_s-termination;*
- *t_a-validity, t_a-consistency and t_a-termination.*

Then, $\mathsf{HBAX}^{(\delta)}$ is a consensus protocol with

- *$(t_s, \delta n)$-intrusion tolerance;*
- *synchronous t_s-validity, synchronous t_s-consistency and synchronous $(r_h + 2)$-round t_s-termination;*
- *t_a-validity, t_a-consistency and t_a-termination.*

Proof. In the proof, we denote with h the common output of HBA.

- **hash lemma:** Suppose $h = \mathsf{Hash}(m_i)$, where m_i is the input of some honest party P_i. Let $(s_1, s_2, \ldots, s_n) = \mathsf{Encode}_{\lceil \delta n/2 \rceil}(m_i)$. If an honest party P_j multicasts $(\mathbf{symbol}, s'_j)$ via the "Symbol Casting" rule, then $s'_j = s_j$. The reason for this is that P_j multicasts $(\mathbf{symbol}, s'_j)$ after seeing a message $(\mathbf{input}, m')$ such that $h = \mathsf{Hash}(m')$ and computing $(s'_1, s'_2, \ldots, s'_n) = \mathsf{Encode}_{\lceil \delta n/2 \rceil}(m')$. Assuming no hash collisions, the fact that $\mathsf{Hash}(m') = h = \mathsf{Hash}(m)$ implies that $m' = m$ and therefore that $s'_j = s_j$.
- **synchronous symbol lemma:** Suppose the network is synchronous, and suppose $h \neq \bot$. Then at time $(r_h + 2)\Delta$, for every honest party P_i, the set

A_i contains indexed symbols from all but at most $\lfloor\frac{\delta n}{2}\rfloor$ honest parties. To see why, begin by noting that all honest parties terminate HBA by time r_h. As $h \neq \perp$, by the $(t_s, \delta n)$-intrusion tolerance of HBA, there exist at least δn honest parties P_j such that $h = h_j$. All these honest parties send their inputs to their neighbors in G_δ, which ensures that by time $(r_h + 1)\Delta$, all but at most $\lfloor\frac{\delta n}{2}\rfloor$ honest parties receive input messages that hash to h and therefore multicast indexed symbols via the "Symbol Casting" rule. Finally, by time $(r_h + 2)\Delta$, all of these multicast indexed symbols are delivered, and so every honest party P_j is able to insert the indexed symbols of all but at most $\lfloor\frac{\delta n}{2}\rfloor$ honest parties to A_j.

- **safety lemma:** Let P_i be an honest party who has computed $h_i = \mathsf{Hash}(m_i)$, and suppose $h = h_i$. Let $(s_1, \ldots, s_n) = \mathsf{Encode}_{\lceil\delta n/2\rceil}(m_i)$. The following holds:

  - Honest parties can output nothing but m_i.
  - If an honest party P_j multicasts $(\mathbf{symbol}, s'_j)$, then $s'_j = s_j$.

  For the former bullet point, as $h \neq \perp$, we already know that honest parties don't output $\perp$ via the "Base Output" rule. For the latter bullet point, we know by the hash lemma that if an honest party P_j multicasts $(\mathbf{symbol}, s'_j)$ via the "Symbol Casting" rule, then $s'_j = s_j$. And so, we see that honest parties may only violate this lemma via the "Extended Termination" rule. For the sake of contradiction, suppose some honest party P_j violates it for the first time. Let us say that A_j contains an incorrect symbol from a party P_k if there is some $(k, s'_k) \in A_j$ such that $s'_k \neq s_k$, and let us say that A_j is missing a symbol from a party P_k if there is no tuple $(k, s'_k) \in A_j$. As P_j is assumed to be the first lemma violator, A_j contains no incorrect symbols from honest parties. And so, for the "Extended Termination" rule, we have the following two scenarios:

  - Suppose the network is asynchronous. Then, the adversary can corrupt at most t_a parties. P_j reconstructs m_i with at most $t_s + \lfloor\frac{\delta n}{2}\rfloor$ missing symbols, and with at most t_a incorrect symbols. As $2t_a + t_s + \lfloor\frac{\delta n}{2}\rfloor \leq n - \lceil\frac{\delta n}{2}\rceil$, the reconstruction of m_i is correct.
  - Suppose the network is synchronous. Then, the adversary can corrupt at most t_s parties. By the synchronous symbol lemma, at time $(r_h + 2)$, A_j can lack at most $\lfloor\frac{\delta n}{2}\rfloor$ honest party symbols. Meanwhile, for up to t_s corrupt parties, A_j can be missing symbols, or even worse, contain incorrect symbols. For reconstruction, the worst case is that all corrupt party symbols are incorrect. In this case, there are at most t_s incorrect symbols, and at most $\lfloor\frac{\delta n}{2}\rfloor$ missing symbols. As $2t_s + \lfloor\frac{\delta n}{2}\rfloor \leq n - \lceil\frac{\delta n}{2}\rceil$, the reconstruction of m_i is correct.

  We see that no matter the network type, P_j reconstructs m_i correctly in the "Extended Termination" rule. And so, no matter the network type, P_j can only output m_i, and P_j can only multicast $(\mathbf{symbol}, s_i)$. This contradicts the assumption that P_j is a lemma violator.

- **termination and consistency:** All honest parties participate in HBA with input, and keep on participating until termination. By the termination and consistency properties of HBA, all honest parties terminate it with some common output h, by time $r_h\varDelta$ if the network is synchronous. If $h = \bot$, then this directly causes all honest parties to terminate with the output $\bot$. Now suppose $h \neq \bot$. By the $(t_s, \delta n)$-intrusion tolerance of HBA, there must exist at least δn honest parties who have provided HBA the input h. Assuming no hash collisions, this implies that all of these honest parties have some common input $m \in \mathcal{M}$ such that $h = \mathsf{Hash}(m)$. The safety lemma implies that honest parties can only output m; this gives us consistency. Now let $(s_1, s_2, \ldots, s_n) = \mathsf{Encode}_{\lceil \delta n/2 \rceil}(m)$. We have the following:
  - If the network is synchronous, then by the synchronous symbol lemma, at time $(r_h + 2)\varDelta$, all honest parties possess indexed symbols from all but at most $\lfloor \frac{\delta n}{2} \rfloor$ honest parties. This means that every honest party is able to terminate at time $(r_h + 2)\varDelta$, having terminated HBA earlier and possessing at least $n - t_s - \lfloor \frac{\delta n}{2} \rfloor$ indexed symbols.
  - Suppose the network is asynchronous. At least δn honest parties with the input m send this input to their neighbors in G_δ, and this enables a set H of all but at most $\lfloor \frac{\delta n}{2} \rfloor$ honest parties to receive m and multicast their indexed symbols via the "Symbol Casting" rule, unless they multicast earlier via the "Extended Termination" rule. As $|H| \geq n - t_a - \lfloor \frac{\delta n}{2} \rfloor \geq n - t_s - \lfloor \frac{\delta n}{2} \rfloor$, every honest party is able to eventually receive at least $n - t_s - \lfloor \frac{\delta n}{2} \rfloor$ indexed symbols and therefore terminate.
- **validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$. All honest parties participate in HBA with the input $h = \mathsf{Hash}(m)$, and keep on participating until termination. By the termination and validity properties of HBA, all honest parties terminate HBA with the output h. The safety lemma ensures that honest parties can output nothing but m.
- $(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$**-intrusion tolerance:** Suppose the common output is not $\bot$. Then the honest parties must have obtained a common output $h \neq \bot$ from HBA, which by the $(t_s, \delta n)$-intrusion tolerance of HBA must have been provided to it as input by at least δn honest parties. Assuming no hash collisions, all of these honest parties must have had some common input $m \in \mathcal{M}$ such that $h = \mathsf{Hash}(m)$. By the safety lemma, we get that honest parties can only output m, an input held by at least δn honest parties.

Complexity of $\mathsf{HBAX}^{(\delta)}$: Assume $\delta = \varTheta(1)$, which is not required for security but required for the communication complexity to be as follows. The communication in $\mathsf{HBAX}^{(\delta)}$, above that of HBA, consists of the following:

- Via the "Base Output" rule, each party P_i may send the message $(\textbf{input}, m_i)$ to $\mathcal{O}(1)$ neighbors in G_δ. As the messages are of size $\mathcal{O}(\ell)$, the message and communication complexity costs incurred are $\mathcal{O}(n)$ and $\mathcal{O}(\ell n)$, respectively.
- Via the "Symbol Casting" and the "Extended Termination" rules, each party P_i may multicast the message $(\textbf{symbol}, s_i)$. As $\lceil \frac{\delta n}{2} \rceil = \varTheta(n)$, this would

be a message of size $\mathcal{O}(\frac{\ell}{n} + \log n) = \mathcal{O}(\frac{\ell}{n} + \kappa)$. Therefore, the message and communication complexity costs incurred are $\mathcal{O}(n^2)$ and $\mathcal{O}(n^2(\frac{\ell}{n} + \kappa)) = \mathcal{O}(\ell n + n^2\kappa)$, respectively.

Overall, we get the message complexity $\mathsf{MC}_{\mathsf{HBAX}^{(\delta)}} = \mathcal{O}(\mathsf{CC}_{\mathsf{HBA}}(\kappa) + n^2)$ and the communication complexity $\mathsf{CC}_{\mathsf{HBAX}^{(\delta)}} = \mathcal{O}(\mathsf{CC}_{\mathsf{HBA}}(\kappa) + n^2\kappa + \ell n)$.