# 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 asymptotic 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 $\ell$-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

A full version of this paper is available at https://eprint.iacr.org/2024/317.

© International Association for Cryptologic Research 2024
M. Joye and G. Leander (Eds.): EUROCRYPT 2024, LNCS 14655, pp. 432–461, 2024.
https://doi.org/10.1007/978-3-031-58740-5_15

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.

The problem has a decades-long research history [7,9,12,26] 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 [25,31] 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 [22].

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 $\Delta$ 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 [5], 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 $2t_s + t_a < n$ [5]. 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 [10,19].

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 [3] 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 $\mathsf{SBA}$) exist for all $t_s < \frac{n}{2}$ assuming setup [12], while consensus protocols with $t_a$-validity and $t_a$-consistency *even in asynchronous networks* (which we denote by $\mathsf{ABA}$) only exist for $t_a < \frac{n}{3}$, even with setup [9].

Blum, Katz, and Loss [5] first proposed a way to adapt such $\mathsf{SBA}$ and $\mathsf{ABA}$ protocols to obtain a network-agnostic protocol. Assuming $2t_s + t_a < n$ (which they prove to be necessary) they obtain a protocol $\mathsf{HBA}$ 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 [3,10]. The idea is to run $\mathsf{SBA}$ and $\mathsf{ABA}$ in succession. First, the input to $\mathsf{HBA}$ is used as input to $\mathsf{SBA}$, and then the output from $\mathsf{SBA}$ as input to $\mathsf{ABA}$. Finally, the output from $\mathsf{ABA}$ is taken to be the output from $\mathsf{HBA}$. However, without modifications, this simple procedure does not provide the wanted guarantees. The first problem is that $\mathsf{SBA}$ 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 $\mathsf{HBA}$. The second problem is that when the network is synchronous but $t_s > t_a$, protocol $\mathsf{ABA}$ provides no security guarantees. This means that synchronous $t_s$-validity and synchronous $t_s$-consistency do not hold for $\mathsf{HBA}$.

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 $\mathsf{SBA}^*$ which, in addition to the properties of $\mathsf{SBA}$, achieves $t_a$-fallback validity.

Now, run in succession $\mathsf{SBA}^*$ and $\mathsf{ABA}$, but any party who aborts in $\mathsf{SBA}^*$ uses their original input as input to $\mathsf{ABA}$. 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 $\mathsf{SBA}^*$ or abort. Either way, they all input $m$ to $\mathsf{ABA}$, and pre-agreement is preserved by the $t_a$-validity of $\mathsf{ABA}$.

To fix the second problem, assume the existence of a protocol $\mathsf{ABA}^*$ which, in addition to the properties of $\mathsf{ABA}$ achieves synchronous $t_s$-validity,[1] and replace $\mathsf{ABA}$ with $\mathsf{ABA}^*$ in $\mathsf{HBA}$. Observe that, if the network is synchronous, by the synchronous $t_s$-consistency of $\mathsf{SBA}^*$, honest parties are always in pre-agreement before executing $\mathsf{ABA}^*$ (either on the pre-agreed upon input, or on some arbitrary value if no pre-agreement was present before executing $\mathsf{SBA}^*$). Therefore, the synchronous $t_s$-validity of $\mathsf{ABA}^*$ suffices for the security of $\mathsf{HBA}$.

Constructing protocols $\mathsf{SBA}^*$ and $\mathsf{ABA}^*$ with the required properties is a significant design challenge which fundamentally exploits the assumption $2t_s + t_a < n$. In [5], Dolev-Strong broadcast [12] and the asynchronous consensus protocol from [28] 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 [10], Deligios and Hirt design a new $\mathsf{SBA}^*$ by modifying the synchronous protocol from [14]. 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, [19] showed an $\mathsf{SBA}$ 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 [23], 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 $\mathsf{SBA}$ with $t_a$-validity and rewriting security proofs to obtain a corresponding $\mathsf{SBA}^*$ whenever *any new synchronous protocol* is published. To make matters worse, certain $\mathsf{SBA}$ protocols (including [19]) 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 $\mathsf{SBA}^*$ that invokes *any* $\mathsf{SBA}$ 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 $\mathsf{SBA}$ protocol, including [19]. 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 [34]). Our graded-consensus intuitively provides the validity guarantees required from $\mathsf{SBA}^*$ when synchrony does not hold, so that the role of the underlying $\mathsf{SBA}$ is reduced to providing full agreement when the network is synchronous. This allows us to use $\mathsf{SBA}$ in a black-box way.

However, plugging the $\mathsf{SBA}$ from [19] into our construction for $\mathsf{SBA}^*$ does not suffice: to achieve network-agnostic security, protocol $\mathsf{HBA}$ requires running both the $\mathsf{SBA}^*$ and the $\mathsf{ABA}^*$ sub-protocols regardless of the network conditions. Unfortunately, a classical lower bound shows that in $r$ rounds, an asynchronous

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

consensus protocol can only achieve consistency and validity with error probability $\mathcal{O}(c^{-r})$ for some constant $c > 1$ [2]. This means that even running an *optimal* $\mathsf{ABA}^*$ 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* $\mathsf{ABA}$ protocol to fulfill the properties of $\mathsf{ABA}^*$ that only requires black-box access to the underlying $\mathsf{ABA}$ protocol. We observe that when the network is synchronous, the $\mathsf{SBA}^*$ protocol guarantees that honest parties are always in pre-agreement upon entering the protocol $\mathsf{ABA}^*$. We exploit this by designing a termination procedure that, when the network is synchronous, is triggered within a small constant number of rounds, causing $\mathsf{ABA}^*$ to terminate *without* running the underlying $\mathsf{ABA}$. This means that the underlying $\mathsf{ABA}$ 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 $\delta n$ honest parties whenever $2t_s + t_a \leq (1 - \delta)n$ and $\delta n$ 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 $\delta n$ to be $n - 2t_s - t_a$.

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

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

***Protocols* $\mathsf{SGC}^1$ *and* $\mathsf{SGC}^2$.** Protocols $\mathsf{SGC}^1$ and $\mathsf{SGC}^2$ 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 $x \in \mathcal{M}^{\perp}$ and outputs a value $y \in \mathcal{M}^{\perp}$, 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 $\mathsf{SGC}^1$ 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 $\perp$ to SProp. Finally, the mapping $(m, \{m, \perp\}, \perp) \longrightarrow ((m, 1), (m, 0), (\perp, 0))$ 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 $\perp$ 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 $\mathsf{SGC}^2$ has a maximum grade $k = 2$. The protocol is similar to $\mathsf{SGC}^1$, but invokes $\mathsf{SGC}^1$ instead of SWC and SWC instead of SProp. The output *value* of $\mathsf{SGC}^2$ is simply taken to be the output value from $\mathsf{SGC}^1$, but an instance of SWC is run on the output *grades* from $\mathsf{SGC}^1$ in order to increase $k$ from 1 to 2 via the mapping $(0, \perp, 1) \longrightarrow (0, 1, 2)$ 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 $\mathsf{SGC}^2$ grades only occur if some honest parties obtained the grade 1 from $\mathsf{SGC}^1$, implying agreement on the output values.[2]

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

***Protocol*** $\mathsf{AProp}$. Our $\mathsf{AProp}$ (asynchronous proposal) protocol is an adaption of $\Pi^{t_s}_{\mathsf{prop}}$ from [5]. 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 $\mathsf{SProp}$, the honest parties have inputs in a set $S = \{x, \perp\} \in \mathcal{M}^{\perp}$. They start by simply sending their inputs to everyone ($\mathsf{AProp}$ 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 $\mathsf{AProp}$ is non-terminating; it is designed to be run forever. Termination is guaranteed by the outer protocol $\mathsf{ABA}^*$.

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

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

***Protocol*** $\mathsf{ABA}^*$***.*** Our asynchronous consensus protocol with $t_s$-validity $\mathsf{ABA}^*$ combines $\mathsf{AGC}^2$ and *any* asynchronous binary consensus protocol $\mathsf{ABA}$ secure against $t_a$ corruptions. The composition is similar to that in $\mathsf{SBA}^*$, 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 [3,5,10]), and instead opt for an approach similar to Bracha's classical protocol from [7]. This keeps the communication complexity quadratic rather than cubic, and makes $\mathsf{ABA}^*$ a signature-free reduction of asynchronous multi-valued consensus to binary consensus [30]. To keep the communication complexity low, we ensure that the honest parties do not send $\mathsf{ABA}$ 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 $\mathsf{ABA}^*$ within a few additional rounds, by some publicly known time $T$. Before local time $T$, honest parties do not send $\mathsf{ABA}$ messages. Hence they terminate without ever sending $\mathsf{ABA}$ messages if the network is synchronous.

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

## 1.3 Contributions

**Theorem 1.** *Let* $\mathsf{SBA}$ *be any synchronous consensus protocol for binary inputs achieving error probability* $\epsilon$ *in* $k$ *rounds, and* $\mathsf{ABA}$ *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* $\mathsf{HBA}$ *for any inputs which invokes* $\mathsf{SBA}$ *and* $\mathsf{ABA}$ *in a black-box way and that, when the network is synchronous, achieves error probability* $\epsilon$ *in* $k + 13$ *rounds.*

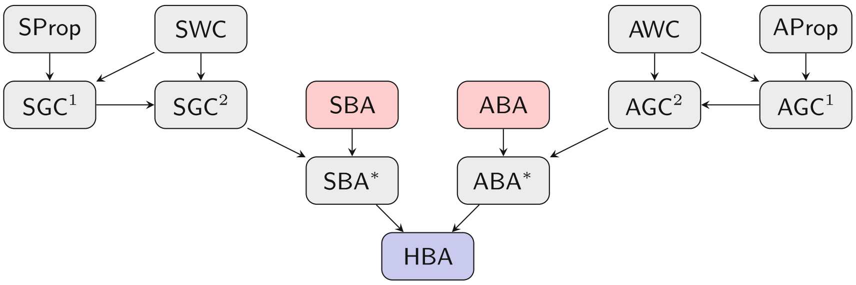

**Fig. 1.** An overview of how sub-protocols are composed for HBA.

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 $\lambda$-round SBA with an error probability decreasing super-exponentially with $\lambda$ [19], improving over the state-of-the-art [10] (in which the error decreases exponentially) and finally matching the known lower bound for purely synchronous protocols [23]. Note that to achieve consensus for honest-majority when the network is synchronous, some sort of setup is *provably* necessary [26]. 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 $\kappa$ a computational security parameter and with $\varepsilon$ a positive constant. Since [3] 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 [5] | — | — | $\mathcal{O}(n^4\kappa)$ (Bit Consensus) |
| Deligios, Hirt, Liu-Zhang [10] | — | — | $\mathcal{O}(n^2\kappa)$ (Bit Consensus) |
| Bacho, Collins, Liu-Zhang, Loss [3] | $\mathcal{O}(n^3\kappa + \ell n^3)$ | $\mathcal{O}(n^3\kappa + \ell n^3)$ | $\mathcal{O}(n^2\kappa + \ell n^3)$ |
| **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 $\varepsilon$. 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 [27] 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 [18],[3] or, if setup for unique threshold signatures is available, with [28] using the coin protocol[4] from [8] 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 [27]. Due to a lack of space, we only present these variants in the full version.

When considering $\ell$-bit inputs, the $\mathcal{O}(\ell n^2)$ term is already a strict improvement over the $\mathcal{O}(\ell n^3)$ term from [3] 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 [17]. We discuss network-agnostic consensus extension protocols later in more detail, and we present such extension protocols in the full version.

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 [5]; in this work, the authors showed the first network-agnostic consensus protocol. The first network-agnostic full multi-party computation protocol (MPC)

[3] The ABA in [18] 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 [13]. Our work only considers the worst case $f = t_s$.

was shown in [6]. There has since been a significant interest in the field. Other network-agnostic MPC protocols include [1,11]. Network-agnostic approximate agreement has been investigated in [20,21]. Most related to our work are [10], which contains an efficient network-agnostic consensus protocol which at the time matched the round complexity of the best known synchronous protocols, and [3], that deals with distributed-key-generation but also constructs an $\mathsf{ABA}^*$ 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 [7,26]. In this setting, the first synchronous consensus protocol with a number of rounds independent from $n$ was [14], and the first asynchronous one was [9]. Constant-round protocols, regardless of the network assumptions, cannot be deterministic [12,15], and they fail with negligible probability. Assuming setup (like a PKI, or correlated randomness) consensus tolerating up to $n/2$ corruptions is possible [12]. One can also obtain constant round constructions in this setting [24]. 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 [23] 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 [19], 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 [2] 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 $\delta n$ 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

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

bound $\Delta$, known to all honest parties. Subject to this rule, the adversary can 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 $\Delta$, $t_s$ 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 $\kappa$ 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 $\mathsf{sk}_i$ is for signing messages and $\mathsf{vk}_i$ is for verifying them. Party $P_i$ keeps $\mathsf{sk}_i$ private, and posts $\mathsf{vk}_i$ on a public board. If $P_i$ is corrupted before the execution of a protocol, then it can freely choose its public $\mathsf{vk}_i$; e.g. it can duplicate the key $\mathsf{vk}_j$ of an honest party $P_j$. We denote with $\sigma = \mathsf{Sgn}_{\mathsf{sk}_i}(m)$ that $\sigma$ is a signature of length $\mathcal{O}(\kappa)$ obtained by signing $m$ with $\mathsf{sk}_i$, and say $\mathsf{Vfy}_{\mathsf{vk}_i}(m, \sigma) = 1$ if $\sigma$ is a valid signature on $m$ with respect to $\mathsf{vk}_i$. 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 $\sigma$ such that $\mathsf{Vfy}_{\mathsf{vk}_i}(m, \sigma) = 1$ without $\mathsf{sk}_i$.

***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 $(l_1, \ldots, l_n)$ such that either $l_i = \sigma_i$ and $\mathsf{Vfy}_{\mathsf{vk}_i}(m, \sigma_i) = 1$ for some signature $\sigma_i$, or $l_i = \perp$, where $\perp$ 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) [8].

## 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 $t < \frac{n}{2^\ell}$ [16]. 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 [3,29] and implicitly [4,17,32], 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 $\ell$-bit inputs, where $\ell$ 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 [16]. 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:

- **(t,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:

- **(t,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 $\{m, \perp\}$. Note that for any proposal protocol, this property with $q \geq 1$ implies that honest parties can only obtain outputs in $\{x, \{x, \perp\}, \perp\}$, 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 $g_i \in \{0, 1, \dots, k\}$. The relevant properties are the following:

- **(t,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:

- **(t,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 $\mathsf{SBA}$ 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 $\mathsf{SBA}^*$ 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 [10], 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 full version of this paper.

**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 $(m_i, \sigma_i)$. 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 $\mathsf{SProp}$**

*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 $(v_i, \sigma_i)$. 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 $\mathsf{SProp}$).** *If $t_a \le t_s$ and $2t_s + t_a \le (1-\delta)n$, then $\mathsf{SProp}$ 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 $\mathsf{SProp}$:** The message and communication complexities of $\mathsf{SProp}$ 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 ($\mathsf{SGC}^1$)

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

**Protocol $\mathsf{SGC}^1$**

*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 $\mathsf{SWC}$ instance with the input $m_i$. If $P_i$ aborts $\mathsf{SWC}$, $P_i$ aborts the protocol. Else, let $v_i$ be $P_i$'s output.

**Phase 2:** $P_i$ participates in a common $\mathsf{SProp}$ instance with the input $v_i$ if $v_i = m_i$, and the input $\bot$ otherwise. If $P_i$ aborts $\mathsf{SProp}$, $P_i$ 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 $\mathsf{SGC}^1$).** *Let $\mathsf{SWC}$ and $\mathsf{SProp}$ 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 $\mathsf{SProp}$ has $(t_s, \delta n)$-intrusion tolerance. Then, $\mathsf{SGC}^1$ 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 $\mathsf{SGC}^1$:** 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 ($\mathsf{SGC}^2$)

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

**Protocol $\mathsf{SGC}^2$**

*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 $\mathsf{SGC}^1$ instance with the input $m_i$. If $P_i$ aborts $\mathsf{SGC}^1$, $P_i$ 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 $\mathsf{SWC}$ instance with the input $h_i$. If $P_i$ aborts $\mathsf{SWC}$, 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 $\mathsf{SGC}^2$).** *Let $\mathsf{SGC}^1$ and $\mathsf{SWC}$ respectively be a 1-graded consensus protocol and a weak binary consensus protocol, both with fixed numbers of rounds, such that $\mathsf{SGC}^1$ 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 $\mathsf{SWC}$ has synchronous $t_s$-robustness, synchronous $t_s$-validity, synchronous $t_s$-weak consistency and $t_a$-fallback validity. Then, $\mathsf{SGC}^2$ 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 $\mathsf{SGC}^2$:** 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 ($\mathsf{SBA}^*$)

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

**Protocol $\mathsf{SBA}^*$**

*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 $\mathsf{SGC}^2$ instance with the input $m_i$. If $P_i$ aborts $\mathsf{SGC}^2$, $P_i$ aborts the protocol. Else, let $(z_i, g_i)$ be $P_i$'s output.

**Phase 2:** $P_i$ participates in a common $\mathsf{SBA}$ instance with the input 1 if $g_i \in \{1, 2\}$ and 0 otherwise. If $P_i$ fails to obtain output from $\mathsf{SBA}$ 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 $\mathsf{SBA}^*$).** *Let $\mathsf{SGC}^2$ and $\mathsf{SBA}$ respectively be 2-graded consensus protocol and a binary consensus protocol, both with fixed numbers of rounds, such that $\mathsf{SGC}^2$ 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 $\mathsf{SBA}$ has synchronous $t_s$-robustness, synchronous $t_s$-validity and synchronous $t_s$-consistency. Then, $\mathsf{SBA}^*$ 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 $\mathsf{SBA}^*$:** 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 $\mathsf{SBA}$ 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 $\mathsf{SWC}$ and $\mathsf{SProp}$. 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 [27]. One imposes an $n$-vertex expander graph on the parties, with each party corresponding to a vertex. In $\mathsf{SWC}$ and $\mathsf{SProp}$ 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 full version, we present 3-round expander-based variants of SWC and SProp with which the communication complexity of $\mathsf{SBA}^*$ 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 $\mathsf{AGC}^2$ protocol to obtain a protocol $\mathsf{ABA}^*$ 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 $(\textbf{input}, v_i)$.

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

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

**Value Rule:** Upon receiving the message $(\textbf{input}, v)$ 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 $(\textbf{propose}, v)$.
- If this rule has been activated for the second time, $P_i$ outputs $V_i$.

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

[7] Actually, $t_a$-validity from $\mathsf{ABA}^*$ 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 $S = \{x, \bot\}$, 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 $\ell$ instances of AProp 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 $y_i \in \mathcal{M}^{\bot}$. $P_i$ 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 $k^{\text{th}}$ 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 $k^{\text{th}}$ 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 $\ell$-bit message $m$ which as its $k^{\text{th}}$ 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 $\bot$.

**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 $\mathsf{AWC}$).** *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.*

**Complexity of $\mathsf{AWC}$:** 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 $\mathsf{SGC}^1$ by composing $\mathsf{SWC}$ and $\mathsf{SProp}$, and then obtained $\mathsf{SGC}^2$ by composing $\mathsf{SGC}^1$ and $\mathsf{SWC}$. These compositions translate very well to the asynchronous setting: we can easily compose $\mathsf{AWC}$ and $\mathsf{AProp}$ to obtain the 1-graded consensus protocol $\mathsf{AGC}^1$, and then compose $\mathsf{AGC}^1$ and $\mathsf{AWC}$ to obtain the 2-graded consensus protocol $\mathsf{AGC}^1$.

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

**Lemma 6 (Security of $\mathsf{AGC}^1$).** *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, $\mathsf{AGC}^1$ 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 $\mathsf{AGC}^2$).** *Let $\mathsf{AGC}^1$ and AWC respectively be a 1-graded consensus protocol and a weak binary consensus protocol such that $\mathsf{AGC}^1$ 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, $\mathsf{AGC}^2$ 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 $\mathsf{AGC}^1$ and $\mathsf{AGC}^2$:** 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 ($\mathsf{ABA}^*$)

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

**Protocol $\mathsf{ABA}^*$**

*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 $\mathsf{AGC}^2$ and a common instance of $\mathsf{ABA}$. Before time $(r_s + r_g + 1)\Delta$, $P_i$ runs $\mathsf{ABA}$ passively, not processing the messages it receives.

**Input Acquisition:** When $P_i$ knows $m_i$, $P_i$ sets it as its input for $\mathsf{AGC}^2$.

**Graded Output:** Upon outputting $(z_i, g_i)$ from $\mathsf{AGC}^2$,

- 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 $\mathsf{ABA}$ input to 1. Else, $P_i$ sets its $\mathsf{ABA}$ input to 0.

**Late Output:** Upon outputting $(z_i, g_i)$ from $\mathsf{AGC}^2$ where $g_i \neq 2$ and outputting $h_i$ from $\mathsf{ABA}$, $P_i$ 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 $(r_s + r_g)\Delta$ time has passed, and sets itself up to terminate with the output $x$ at time $(r_s + r_g)\Delta$ otherwise.

**Theorem 4 (Security of $\mathsf{ABA}^*$).** *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 $\mathsf{AGC}^2$ and $\mathsf{ABA}$ respectively be a 2-graded consensus protocol and a binary consensus protocol such that $\mathsf{AGC}^2$ 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 $\mathsf{ABA}$ has $t_a$-validity, $t_a$-consistency and $t_a$-liveness. Then, $\mathsf{ABA}^*$ 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 $\mathsf{ABA}^*$:** 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 $\mathsf{AGC}^2$ and $\mathsf{ABA}$ 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 $\mathsf{ABA}^*$ ensures that all honest terminate by time $(r_s + r_g + 1)\Delta$ without ever sending $\mathsf{ABA}$ 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 $\mathsf{SBA}^*$ and $\mathsf{ABA}^*$ to construct our network-agnostic consensus protocol $\mathsf{HBA}$. Intrusion tolerance here is a challenge: $\mathsf{SBA}^*$ allows honest parties to output $\perp$, but $\mathsf{ABA}^*$ doesn't allow them to input $\perp$. Thus, we extend inputs for $\mathsf{ABA}^*$, making them $(\ell+1)$ bits long rather than $\ell$ bits long; the first bit is reserved to indicate whether the $\mathsf{SBA}^*$ output is $\perp$ or not. Notationally, we write $b \parallel m$ to represent $m \in \mathcal{M}$ with the bit $b$ prepended.

We present $\mathsf{HBA}$ as a two-phase protocol. The first phase begins at time 0 and lasts sufficiently long for $\mathsf{SBA}^*$; 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 $y_i \in \mathcal{M}^{\perp}$.

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

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

**Phase 2:** If $v_i \neq \perp$, then $P_i$ sets $v_i$ as its input for $\mathsf{ABA}^*$. Else, $P_i$ sets $1 \parallel m_i$ as its $\mathsf{ABA}^*$ input once $P_i$ knows $m_i$. Upon terminating $\mathsf{ABA}^*$ 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 $\perp$ 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 $\mathsf{ABA}^*$ protocol presented in Sect. 4.4, we would need to set its "start round $r_s$" to be the round count of $\mathsf{SBA}^*$.

**Theorem 5 (Security of $\mathsf{HBA}$).** *Let $\mathsf{SBA}^*$ 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 $\mathsf{ABA}^*$ 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, $\mathsf{HBA}$ 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 $\mathsf{SBA}^*$ implicitly.

– $(\mathbf{t_s}, \boldsymbol{\delta}\mathbf{n})$**-intrusion tolerance:** Suppose less than $\delta n$ honest parties have some $m \in \mathcal{M}$ as input. By the $(t_s, \delta n)$-intrusion tolerance of $\mathsf{SBA}^*$, honest parties do not output $m$ from $\mathsf{SBA}^*$. This ensures that only honest parties who abort $\mathsf{SBA}^*$ and have the input $m$ may provide the input $1 \| m$ to $\mathsf{ABA}^*$. As there can only be less than $\delta n$ such honest parties, by the $(t_s, \delta n)$-intrusion tolerance of $\mathsf{ABA}^*$, honest parties do not output $1 \| m$ from $\mathsf{ABA}^*$. 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 $\mathsf{SBA}^*$ 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 $\mathsf{SBA}^*$. Afterwards, all honest parties participate in $\mathsf{ABA}^*$ with the common input $z'$, where $z' = (0, 0, \ldots, 0) \in \{0, 1\}^{\ell+1}$ if $z = \perp$, and $z' = 1 \| z$ otherwise. By time $(r_s + r_a)\Delta$, the synchronous $r_a$-round $t_s$-validity with termination of $\mathsf{ABA}^*$ guarantees that they terminate $\mathsf{ABA}^*$ with the common output $z'$ and hence terminate $\mathsf{HBA}$ 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 $\mathsf{SBA}^*$ implies that $z = m$ and hence that $z' = 1 \| m$, 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 $\mathsf{ABA}^*$ input, and run it until they terminate it with consistent outputs. Thus, $\mathsf{HBA}$ inherits the $t_a$-termination and the $t_a$-consistency of $\mathsf{ABA}^*$.
– $\mathbf{t_a}$**-validity:** Suppose all honest parties have the same input $m \in \mathcal{M}$. By the $t_a$-fallback validity of $\mathsf{SBA}^*$, any honest party that doesn't abort $\mathsf{SBA}^*$ outputs $m$ from it. This ensures that all honest parties participate in $\mathsf{ABA}^*$ with the input $1 \| m$. Finally, by the $t_a$-validity and the $t_a$-termination of $\mathsf{ABA}^*$, all honest parties terminate $\mathsf{ABA}^*$ with the output $1 \| m$ and therefore terminate $\mathsf{HBA}$ with the output $m$.

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

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

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

***Reducing*** $\mathcal{O}(\ell n^2)$ ***to*** $\mathcal{O}(\ell n)$. For $\ell$-bit consensus, the optimal communication complexity is $\mathcal{O}(\ell n + \dots)$ [17]. There is a rich literature *extending* consensus protocols to handle $\ell$-bit inputs in clever ways to meet this bar [4,17,32,33].

To the best of our knowledge, [32] presents the state-of-the-art adaptively secure consensus extension protocol.[8] The protocol requires the parties to agree on $\kappa$-bit values in an intrusion-tolerant manner. Intrusion tolerance is achieved via two instances of consensus (one to decide the a $\kappa$-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 full version, we present an adaptation of the protocol which makes only one black-box invocation of $\mathsf{HBA}$ on $\kappa$-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 $\mathsf{HBA}$ (as opposed to the $\mathcal{O}(\ell n^3)$ in [3]) permits extensions for long inputs with the total complexity $\mathcal{O}(n^2\kappa + \ell n)$ whenever $\mathsf{HBA}$ achieves the complexity $\mathcal{O}(n^2\kappa + \ell n^2)$. Thus, with $\mathsf{HBA}$, one can match the complexity of state-of-the-art synchronous protocols.

## References

1. 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)
2. 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 [4].

3. 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. LNCS, vol. 14081, pp. 71–106. Springer, Heidelberg (Aug 2023). https://doi.org/10.1007/978-3-031-38557-5_3
4. 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. LNCS, vol. 13791, pp. 504–525. Springer, Heidelberg (Dec 2022). https://doi.org/10.1007/978-3-031-22963-3_17
5. Blum, E., Katz, J., Loss, J.: Synchronous consensus with optimal asynchronous fallback guarantees. In: Hofheinz, D., Rosen, A. (eds.) TCC 2019. LNCS, vol. 11891, pp. 131–150. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-36030-6_6
6. Blum, E., Liu-Zhang, C.-D., Loss, J.: Always have a backup plan: fully secure synchronous MPC with asynchronous fallback. In: Micciancio, D., Ristenpart, T. (eds.) CRYPTO 2020. LNCS, vol. 12171, pp. 707–731. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-56880-1_25
7. Bracha, G.: Asynchronous byzantine agreement protocols. Inf. Comput. **75**(2), 130–143 (1987). https://doi.org/10.1016/0890-5401(87)90054-X
8. 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
9. 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
10. 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. LNCS, vol. 13042, pp. 623–653. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90459-3_21
11. 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
12. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM J. Comput. **12**(4), 656–666 (1983). https://doi.org/10.1137/0212045
13. 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
14. 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
15. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. J. ACM (JACM) **32**(2), 374–382 (1985)
16. 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
17. 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

18. 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
19. 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
20. 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)
21. Ghinea, D., Liu-Zhang, C.D., Wattenhofer, R.: Multidimensional approximate agreement with asynchronous fallback. Cryptology ePrint Archive (2023)
22. 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)
23. Karlin, A., Yao, A.: Probabilistic lower bounds for byzantine agreement. Unpublished document (1986)
24. 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
25. King, S., Nadal, S.: Ppcoin: Peer-to-peer crypto-currency with proof-of-stake. self-published paper, August **19**(1) (2012)
26. 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
27. 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
28. 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
29. 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)
30. 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
31. Nakamoto, S.: Bitcoin: A peer-to-peer electronic cash system. Decentralized business review (2008)
32. 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

33. 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
34. Pfitzmann, B., Waidner, M.: Information-theoretic pseudosignatures and byzantine agreement for $t \geq n/3$. IBM Research, Armonk, NY, USA (1996)