Any Friend of Yours: Asynchronous Consensus with Unknown Participants from Local Trust

Ignacio Amores-Sesar[1], Simon Holmgaard Kamp[2], and Juan Villacis[3]

[1] Aarhus University
[2] Ruhr University Bochum
[3] University of Bern

Abstract. Byzantine agreement protocols typically assume that all processes already agree on who participates and which failures must be tolerated. In an open system, reaching agreement on this information is itself a consensus problem. We avoid this circularity by presenting asynchronous protocols in which each process acts only on its own local trust. Each process declares quorums over the identities it knows and communicates by authenticated multicast. Processes need not know the number of participants or the trust assumptions of others, and we assume no trusted dealer, common reference string, or bound on the number of identities, including Sybil identities. Unlike Stellar and prior asymmetric-trust protocols, our protocols do not require a closed set of mutually trusting correct processes. Instead, their guarantees are characterized by depth, a fine-grained, failure-dependent measure of how well each process's trust matches the actual failures.

We construct validated reliable broadcast, Byzantine agreement, agreement on a core set, and state machine replication in this model. The main technical ingredient for randomized termination is an approximate common coin based only on local trust.

1 Introduction

Reaching agreement between parties in distributed systems is a key problem in computer science that has been studied for decades [33]. Many solutions are known, under varying models and assumptions [7]. However, most of them are intended for settings with a fixed and known set of participants, also referred to as permissioned [25]. A known participant set, combined with a failure assumption, allows the definition of *quorums* [29], sets of processes whose joint messages can serve as certificates in protocols. For example, in a system with n participants and a threshold failure assumption of f faults with $n \geq 3f+1$, each quorum of $n-f$ participants satisfies several useful properties. It contains at least $n-2f \geq f+1$ honest members, it intersects any other quorum in at least $f+1$ processes and hence in at least one honest process, and there is always at least one such quorum containing solely honest members.

If, instead, the set of participants is fixed but unknown, a model introduced by Cavin *et al.* [9] and denoted as the *unknown participants* setting, quorums

can only be defined locally, relative to what each participant knows or trusts. A set that one process could accept as a quorum need not be a quorum for another. The useful properties mentioned above are no longer automatic.

This model can be further characterized by the knowledge level of the participants. If partial knowledge of other processes in the system is assumed, it is possible to solve problems such as asynchronous consensus, as long as there is a globally known threshold f on the number of faults and additional conditions are satisfied by the graph representing the knowledge relationship between processes [1]. If participants only know about their own existence, reliable broadcast, approximate agreement, and consensus are solvable as long as there is a synchronous network and at most 1/3 of the processes are faulty [24]. The authors further show that synchrony is necessary; if processes know only their own identifiers, consensus is impossible under asynchrony. Both works in this model rely crucially on the global threshold assumption being respected.

A failure assumption of this kind counts identities, and is therefore only as strong as the assumption that identities are scarce. An adversary able to craft many malicious participants causes global assumptions to fail and quorums to lose their properties, and can gain control over the system [14]. The common solution, pioneered by Nakamoto [32], moves the assumption from the adversary having limited identities to the adversary having limited control over an underlying resource, such as computational power [32], economic capital [19,12], or storage space [31]. Such assumptions, however, simply replace one global quantity with another; the majority of the underlying resource must be held by honest participants. This resource-restricted model [18] carries its own costs, from the electricity consumed by proof-of-work [36] to the formal barriers facing incentive design in longest-chain proof of stake, where deviating from the protocol destroys no resource and therefore carries no inherent cost [6]. Moreover, many such protocols rely on synchronous communication for safety, liveness, or both [17].

An alternative approach relies on trust relationships between participants. Assumptions of this kind, in which each process holds its own failure assumption rather than sharing a global one, and which are also referred to as *asymmetric*, were initially proposed by Damgård *et al.* [13]. Their use as a Sybil-resistance and quorum-building mechanism in open networks came with Ripple [34], was developed most fully in Stellar [30,27], and was further advanced by SybilQuorum [35]. In such systems, processes declare which other participants they trust, typically in the form of a quorum system. Nothing is counted, and so minting identities is of no use to an adversary unless someone can be induced to trust them; Sybil resistance follows from the structure of trust rather than from the scarcity of a resource. What is given up is uniformity of guarantees. Correct processes that trust an adequate set of participants get the protocol's guarantees, while correct processes that misjudge may not. Determining what actually comprises a good trust assumption is an ongoing question [3].

Asymmetric trust assumptions introduce challenges that do not arise under a single global assumption, even when the participant set is fixed and known. Protocols developed for the latter setting usually do not transfer to the asymmet-

ric one [5], and additional assumptions on mutual trust, vacuous in symmetric environments, are usually required [30,2]. Unknown participants add a further difficulty: defining a sufficiently good quorum system when processes hold distinct and partial views of who is present.

Traditional Byzantine quorum systems must satisfy two properties: consistency, that all quorums intersect in a correct process, and availability, that at least one quorum contains no faulty process [29]. Availability, together with the universality of the quorum system, gives rise to a structure that is easy to overlook: a set of correct processes containing a quorum for each of its members. Being a direct consequence of availability, its role is implicit under a global assumption and becomes visible only when trust is asymmetric. It is nonetheless central to liveness. The set is a fixpoint of the trust relation, and it is what permits its members infinite cycles of mutual trust, which matters for consensus in particular: since every asynchronous protocol tolerating one crash admits an infinite execution in which no process decides [15], so a liveness argument must rest on something that survives arbitrarily long runs. We call this set the *trust fixpoint*.

Under asymmetric trust the fixpoint is not immediate, and emerges only when the failure pattern leaves enough correct processes with local quorums that sufficiently trust one another. This restricts the failure patterns that the resulting protocols can tolerate. Most work in the area recognized the utility of the fixpoint and gave liveness guarantees only in executions containing one; the condition recurs under different names (guilds [2], intact set [30], consensus clusters [28], strong availability [26], leagues [8]). The Stellar Consensus Protocol [30] partially addressed this by constructing quorums so as to promote the appearance of the fixpoint. Rather than defining local quorums directly, processes define *slices*, which are also sets of processes; slices are exchanged, and a quorum for a process is any set containing one of its slices together with a slice for each member. A quorum for a process is then automatically a quorum for each of its members, which is exactly the fixpoint. This technique also aids consistency, since quorums contain elements drawn from the views of many processes, which promotes, though it does not guarantee, adequate intersection.

It was recently shown [3] that, in the asymmetric permissioned setting, the fixpoint is not necessary for tasks such as reliable broadcast or consensus, opening the way to protocols that work under less restrictive conditions and tolerate more failure patterns. The authors characterize the liveness of an individual process by its *depth*, a measure of how well its trust assumptions match the actual failures. Every process has depth -1 by convention, and a process that is correct and has a quorum all members of which have depth $d-1$ has depth d; in particular, every correct process has depth at least 0. Depth thus measures, in a fine-grained way, the level of liveness a process can expect. For the fixed and known participant setting, they give a reliable broadcast protocol that guarantees safety and liveness for all processes of depth three, and an asynchronous consensus protocol for processes of depth ten. Members of the trust fixpoint have infinite depth, precisely because of the cycles of mutual trust described above.

Contribution. The Stellar Consensus Protocol [30] shows that it is possible to solve agreement problems in open environments by relying on local trust. They provide a protocol for partially-synchronous networks with an unknown and varying participant set. However, their protocol requires the trust fixpoint; it only guarantees safety and liveness for intact processes, which limits the fault patterns that can be tolerated. Two natural questions arise from this.

1. Is it possible to solve asynchronous consensus in the unknown participants model without relying on global fault thresholds and in environments prone to Sybil attacks?
2. Can a protocol based on local trust solve the problem without relying on trust fixpoints?

We answer both questions positively. We first adapt the definition of a quorum system to environments without a trust fixpoint and with an unknown participant set (Section 3). We then introduce protocols for reliable consensus, validated reliable broadcast (Section 4), proxcensus (Section 5), validated graded gather (Section 7), verifiable secret sharing (Section 8), and approximate common coin (Section 9) based on local trust relationships but without requiring any trust fixpoint. Using these protocols, we solve binary agreement (Section 10) and agreement on a core set (Section 11) in the unknown participants model under asynchrony and without trust fixpoints. The common coin relies on a contribution from a correct dealer in the common core, so its security does not depend on the number of identities controlled by the adversary.

Additionally, we generalize the impossibility of asynchronous consensus in the unknown participants model with no knowledge of processes [24] to the asymmetric model and to settings in which processes know participants other than themselves. We show that consensus remains impossible if two disjoint groups are each guaranteed to decide without hearing from the rest of the system. Our quorum assumptions exclude this case and allow consensus (Appendix A). Table 1 compares our results with prior work in similar models.

1.1 Related Work

Consensus with unknown participants. Work in the unknown participants model [9] follows two directions. In the first, each process has a partial view of the participants, given by a participant detector, which induces a knowledge connectivity graph, and processes know the fault threshold. Cavin *et al.* [10] solve consensus with crash faults using a perfect failure detector. Alchieri *et al.* [1] extend this to Byzantine faults, requiring at least $3f + 1$ processes in the sink component. Heydari *et al.* [20] remove the need to know the threshold under partial synchrony, at the cost of a stronger connectivity condition. In the second direction, each process initially knows only its own identifier. Here, consensus for $f < n/3$ [24] and Byzantine broadcast for $f < n$ with a certificate authority [11] are solvable under synchrony, but consensus is impossible under asynchrony or partial synchrony [24]. Our model is closer to the first direction:

trust relationships give each process partial knowledge of the participants. Unlike both directions, however, we do not bound the fraction of faulty identities and therefore must address Sybil attacks.

Permissionless consensus. When the participant set is neither known nor fixed [25], Sybil resistance [14] is usually obtained by bounding the adversary's share of a resource such as computational power [32], storage [31], or stake [12], the resource-restricted model [18]. Like these works, we allow the adversary to control any fraction of the identities. Unlike them, we do not bound its resources and derive Sybil resistance from local trust instead. To our knowledge, this yields the first asynchronous consensus protocol with unknown participants that assumes neither a global threshold nor a bound on adversarial resources.

Asymmetric trust. Byzantine quorum systems [29] generalize threshold assumptions to explicit sets of trusted processes. Damgård *et al.* [13] let each process choose its own quorum system, yielding asymmetric quorum systems, but assume a fixed participant set and globally known trust assumptions. Subsequent work studies necessary and sufficient conditions, and protocols in this setting [2,26,28,3,4]. Ripple [34] and Stellar [30] first used asymmetric trust for Sybil resistance in open systems. Since no process can build a universal quorum system without knowing the complete set of participants, each builds its own. Stellar's partially synchronous protocol, built from slices, guarantees safety and liveness only for intact processes. Cachin *et al.* [8] adapt fail-prone systems to permissionless settings by letting processes make assumptions about other processes' assumptions, which encourages quorum intersection between processes that do not know each other. They also give a reliable broadcast protocol in this model.

2 Preliminaries

2.1 Model

We consider asynchronous protocols for a fixed but initially unknown set of n processes $\mathcal{P} = \{p_1, \ldots, p_n\}$. Processes exchange messages and execute the event-based programs of Cachin *et al.* [7]. The adversary may corrupt processes adaptively and permanently, obtaining their complete local state and controlling their subsequent behavior. For an execution, let F be the set of all processes corrupted during that execution. A process is *correct* if it is outside F and follows its program throughout the execution; every process in F is *faulty* or *Byzantine*, including before its corruption. Depth and quorum assumptions are evaluated with respect to this set F.

Processes have stable identifiers and communicate by asynchronous authenticated multicast. A process can multicast a message without enumerating the participants, as indicated by *send to all* in the pseudocode. Every correct process eventually receives each message multicast by a correct sender, and recipients authenticate the original sender of each message. The adversary may read messages and schedule their delivery in any order, and a Byzantine sender may send

different messages to different recipients or omit recipients altogether. Choi *et al.* [11] use the synchronous analogue, in which a correct sender's message reaches all parties active in the same round.

To send a message to a particular process p_j, the sender includes p_j's identifier in the multicast message, and every other correct process ignores the payload. For example, multicast can be implemented by flooding signed messages over reliable links, with each process using its signature verification key as its identifier. Flooding guarantees delivery if every pair of correct processes is connected by a path of correct relays.

Initially, each process knows its own quorum system over authenticated identifiers. Encryption keys used by the sharing protocol are generated and disseminated during execution. There is no bound on the faulty fraction: the adversary may populate $\mathcal{P}$ with arbitrarily many processes, subject to any explicit participant bound.

Cryptographic security statements quantify over probabilistic polynomial-time adversaries and experiments whose process set, explicit quorum descriptions, protocol invocations, and total computation have size polynomial in the security parameter λ. The asynchronous model places no bound on message delays; liveness assumes that enabled local actions are eventually executed and messages between correct processes are eventually delivered.

We say an environment is *permissioned* if the set of participants is fixed and known at all times. We use *open* to refer to any environment that is not permissioned. In particular, the unknown participants model [9] (the model in which we work) and the fully permissionless model [25], where participants can join and leave at any time, are examples of open environments.

2.2 Overview of Permissioned Quorum Systems

Given a set of participants $\mathcal{P}$, a *fail-prone system* $\mathcal{F} \subseteq 2^{\mathcal{P}}$ is a family of sets, each containing processes that may fail together in some execution [29]. For a system $\mathcal{A} \subseteq 2^{\mathcal{P}}$, the notation $\mathcal{A}^*$ denotes the collection of all subsets of the sets in $\mathcal{A}$, that is, $\mathcal{A}^* = \{A' \mid A' \subseteq A,\ A \in \mathcal{A}\}$. Classically, all processes share a single such assumption, and a Byzantine quorum system for $\mathcal{F}$ exists if and only if $\mathcal{F}$ satisfies the Q^3-condition: no three fail-prone sets cover $\mathcal{P}$ [29,21]. This generalizes the threshold condition $n > 3f$.

Asymmetric trust [13,2] lets every process choose its own assumption. An *asymmetric fail-prone system* is an array $\mathbb{F}^p = [\mathcal{F}_1, \ldots, \mathcal{F}_n]$, where $\mathcal{F}_i$ is the failure assumption of p_i.

Definition 1 (Asymmetric Byzantine quorum system [2]). *An* asymmetric Byzantine quorum system *for $\mathbb{F}^p$ is an array $\mathbb{Q}^p = [\mathcal{Q}_1, \ldots, \mathcal{Q}_n]$ with $\mathcal{Q}_i \subseteq 2^{\mathcal{P}}$; each $Q \in \mathcal{Q}_i$ is a* quorum for p_i. *It satisfies:*

Consistency: $\forall i, j \in [1,n],\ \forall Q_i \in \mathcal{Q}_i,\ \forall Q_j \in \mathcal{Q}_j,\ \forall F_{ij} \in \mathcal{F}_i^* \cap \mathcal{F}_j^* :\quad Q_i \cap Q_j \not\subseteq F_{ij}$.

Availability: $\forall i \in [1,n],\ \forall F_i \in \mathcal{F}_i :\quad \exists Q_i \in \mathcal{Q}_i :\ F_i \cap Q_i = \emptyset$.

Such a system exists if and only if $\mathbb{F}$ satisfies the B^3-condition, the asymmetric analogue of Q^3 [13,2]:

$$\forall i, j \in [1, n],\ \forall F_i \in \mathcal{F}_i,\ \forall F_j \in \mathcal{F}_j,\ \forall F_{ij} \in \mathcal{F}_i^* \cap \mathcal{F}_j^* : \quad \mathcal{P} \not\subseteq F_i \cup F_j \cup F_{ij}.$$

A *kernel* of p_i is a set of processes intersecting every quorum of p_i; we write $\mathcal{K}_i$ for the kernels of p_i [2].

In the symmetric setting the shared quorum system covers the actual set of faulty processes F by assumption. Under asymmetric trust this need not hold for every correct process, and the resulting heterogeneity is graded by the notion of *depth* [3].

Definition 2 (Depth of a process [3]). *Fix an execution with set of faulty processes F. Every process has depth at least -1. Additionally, a correct process p_i has depth d if*

$$\exists Q \in \mathcal{Q}_i :\ \forall p_j \in Q,\ p_j \text{ has depth at least } d-1.$$

When we refer to the depth of a process, we refer to its maximum depth.

A process has depth infinity if and only if there exists a set $\mathcal{G}$ containing that process such that all processes in $\mathcal{G}$ are honest and for each $p_i \in \mathcal{G}$, there is a $Q \in \mathcal{Q}_i$ such that $Q \subseteq \mathcal{G}$. This is an example of a trust fixpoint, which is known in the asymmetric trust model [2] as a *guild*.

Throughout the protocol statements, a process of depth d means a process of depth at least d. A quorum is *live* in an execution with faulty set F if all its members are correct, that is, $Q \cap F = \emptyset$. A correct process has depth at least 1 exactly when it has a live quorum.

Definition 3 (Universal kernel depth). *Fix an execution and its set of faulty processes. The quorum system has* universal kernel depth at least d *if there is a set S of processes of depth at least d that intersects every quorum of every correct process. Equivalently, the set of all processes of depth at least d is a kernel for every correct process. We also write* universal kernel depth d *for this lower-bound condition.*

Protocols in the asymmetric trust literature cannot guarantee all properties to all processes because the guarantees depend on the depths of the processes. For instance, the reliable broadcast and consensus protocols of Alpos *et al.* [2] guarantee all properties only to processes with infinite depth. Subsequently, Amores-Sesar *et al.* [3] improved this and guaranteed all properties of reliable broadcast and consensus to processes with depth three and ten respectively.

2.3 Local Validity Predicates

For an execution prefix τ, let $\mathsf{state}_i(\tau)$ denote the local state of party p_i after τ.

Definition 4 (Local validity predicate). *A* local validity predicate *for p_i is a function $\mathsf{V}_i : \{0,1\}^* \times \mathsf{State}_i \to \{\mathsf{false}, \mathsf{true}\}$; at prefix τ, we abbreviate $\mathsf{V}_i(x, \mathsf{state}_i(\tau)) = \mathsf{true}$ by $\mathsf{V}_i(x)$. It is* monotone *if, for every x, once true it remains true at all extensions of the execution prefix. Write $\mathsf{V} = (\mathsf{V}_i)_{p_i \in \mathcal{P}}$ for the family of local predicates.*

3 Modeling Local Trust with Unknown Participants

Unlike prior work on asymmetric trust [13,2], which specifies both an asymmetric fail-prone system and an asymmetric quorum system, we specify only the quorum system. This corresponds to assuming that each process tolerates precisely the fault sets disjoint from at least one of its quorums, as is common in the literature [3,4]. It also suits open systems, where processes know little about the participant set [30]. As a consequence, the properties of the quorum system depend on the failures that actually occur in an execution. We also drop the availability property and replace it with *depth*, a per-process characterization of the execution rather than a property of the quorum system. With this information, we re-define a quorum system as follows. To distinguish it from the definition for permissioned settings, we refer to it as an open quorum system.

Definition 5 (Open Quorum System). *We consider an array of collections of sets* $\mathbb{Q} = [\mathcal{Q}_1, \cdots, \mathcal{Q}_n]$ *where* $\emptyset \neq \mathcal{Q}_i \subseteq 2^{\mathcal{P}}$ *for* $i \in [1, |\mathcal{P}|]$ *and a failure scenario* $F \subseteq \mathcal{P}$*. Based on these, and for each* $p_i \in \mathcal{P}$*, we define its depth* $depth(p_i)$ *as follows*

- *Any process* p_i *has depth -1*
- *Additionally, a process* p_i *has depth* d *if* $p_i \notin F$ *and there is a quorum* $Q \in \mathcal{Q}_i$ *such that for all* $p \in Q$, $depth(p) \geq d - 1$

We say $\mathbb{Q}$ *is an* open Byzantine quorum system *for* F *if the following property holds, which we denote* consistency

- *For any two parties* p_i, p_j *with* $depth(p_i) \geq 0$ *and* $depth(p_j) \geq 0$*, for any* $Q_i \in \mathcal{Q}_i$, $Q_j \in \mathcal{Q}_j$, $Q_i \cap Q_j \neq \emptyset$
- *For any two parties* p_i, p_j *with* $depth(p_i) \geq 1$ *and* $depth(p_j) \geq 1$*, for any* $Q_i \in \mathcal{Q}_i$, $Q_j \in \mathcal{Q}_j$, $\exists p \in Q_i \cap Q_j$ *such that* $depth(p) \geq 0$

For each integer $d \geq 0$, write

$$D_d := \{p_i \in \mathcal{P} : \ \text{depth}(p_i) \geq d\}.$$

Given this definition, kernels and universal kernel depth (Definition 3) are defined in the same manner as for the asymmetric quorum systems.

We note that since the definition is recursive, a process can have multiple depths. For clarity, when we refer to the depth of a process we refer to its maximal depth. As in the permissioned setting, having a process with infinite depth implies the existence of a set of correct processes that contains a quorum for each member, i.e., a trust fixpoint. In this work, we strive to develop protocols that guarantee properties to processes that are not part of such structures, that is, we strive to be able to give all the guarantees of a protocol to processes having some bounded depth d. As will be seen later, this parameter varies between problems.

4 Reliable Consensus and Validated Reliable Broadcast

Every protocol instance has a distinct tag, and every message carries the tag of its instance. A correct process proposes at most one message in each reliable consensus instance and continues processing protocol messages after delivery.

Definition 6 (Depth-characterized asymmetric Byzantine reliable consensus). *A protocol for* asymmetric Byzantine reliable consensus *with validity depth vd, safety depth sd, and liveness depth ld, shortened as RC[vd, sd, ld], defined through the events rc-propose(m) and rc-deliver(m) satisfies the following properties:*

- ***Validity:*** *If a process with depth sd rc-delivers a message m, then m was previously rc-proposed by a party with depth at least vd.*
- ***Consistency:*** *If a process with depth sd rc-delivers m and another process with depth sd rc-delivers m', then $m = m'$.*
- ***Integrity:*** *Every process of depth at least sd rc-delivers at most once.*
- ***Totality 1:*** *If all processes with depth ld rc-propose the same message, all processes with depth ld eventually rc-deliver.*
- ***Totality 2:*** *If a process with depth ld rc-delivers, all processes with depth ld eventually rc-deliver.*

Theorem 1. *Algorithm 1 implements $RC[d, 0, 3]$, assuming a universal kernel depth $d \geq 4$.*

The proof appears in Appendix C.1.

4.1 Validated Reliable Broadcast

Reliable broadcast is defined as reliable consensus, except that it has a designated sender p_s who will broadcast its message to the remaining parties. The first totality property is changed so that termination is ensured whenever p_s is honest and invokes broadcast, while validity now demands that outputs should be broadcast by the designated sender. Otherwise, the definition matches Definition 6. Some applications additionally require delivered values to satisfy a monotone validity predicate V. The asymmetric validated reliable-broadcast variant defined in Definition 7 buffers the designated sender's value at process p_i until $\mathsf{V}_i(m)$ holds and only then invokes reliable consensus, which is the point at which the echo is sent. Its external validity property guarantees that every delivered value passed V in the view of a correct process of depth vd. With the reliable consensus implementation from Theorem 1, this process has depth d. Totality for a correct sender additionally requires that the predicate eventually hold at every correct process of the stated liveness depth.

Let $\mathsf{V} = (\mathsf{V}_i)_{p_i \in \mathcal{P}}$ be a monotone local validity predicate on broadcast messages. The designated sender invokes broadcast at most once in each instance.

Algorithm 1 Asymmetric reliable consensus (code for p_i).

State
1: $proposed \leftarrow \mathsf{false}$, $delivered \leftarrow \mathsf{false}$
2: $ready \leftarrow \emptyset$, $amplified \leftarrow \emptyset$

3: **upon invocation** *rc-propose*(m) **and** $\neg$*proposed* **do**
4: $proposed \leftarrow \mathsf{true}$
5: send $[\text{ECHO}, m]$ to all $p_j \in \mathcal{P}$

6: **upon** receiving $[\text{ECHO}, m]$ from a quorum in $\mathcal{Q}_i$ **and** $m \notin ready$ **do**
7: $ready \leftarrow ready \cup \{m\}$
8: send $[\text{READY}, m]$ to all $p_j \in \mathcal{P}$

9: **upon** receiving $[\text{READY}, m]$ from a quorum in $\mathcal{Q}_i$ **and** $\neg$*delivered* **do**
10: $delivered \leftarrow \mathsf{true}$
11: **output** *rc-deliver(m)*

12: **upon** receiving $[\text{READY}, m]$ from a kernel in $\mathcal{K}_i$ **and** $m \notin amplified$ **do**
13: $amplified \leftarrow amplified \cup \{m\}$
14: send $[\text{READY-AMPLIFY}, m]$ to all $p_j \in \mathcal{P}$

15: **upon** receiving $[\text{READY-AMPLIFY}, m]$ from a quorum in $\mathcal{Q}_i$ **and** $m \notin ready$ **do**
16: $ready \leftarrow ready \cup \{m\}$
17: send $[\text{READY}, m]$ to all $p_j \in \mathcal{P}$

Definition 7 (Validated reliable broadcast). *Fix sender p_s, validity depth vd, safety depth sd, liveness depth ld, and monotone local validity predicate V. We write $VRB_{\mathsf{V}}[vd, sd, ld]$ for* asymmetric validated reliable broadcast *with these parameters. Its interface consists of events* vrb-broadcast(m) *and* vrb-deliver(m)*, and it satisfies the following properties:*

Validity. *If p_s is correct and a process of depth sd vrb-delivers m, then p_s previously vrb-broadcast m.*

External validity. *If a process of depth sd vrb-delivers m, then $\mathsf{V}_j(m)$ previously held for some process p_j of depth at least vd.*

Consistency. *If two processes of depth sd vrb-deliver m and m', respectively, then $m = m'$.*

Integrity. *Every process of depth at least sd vrb-delivers at most once.*

Totality 1. *If p_s is correct, vrb-broadcasts m, and $\mathsf{V}_i(m)$ eventually holds at every process p_i of depth at least ld, then all processes of depth ld eventually vrb-deliver m.*

Totality 2. *If a process of depth ld vrb-delivers, then all processes of depth ld eventually vrb-deliver.*

Corollary 1. *Algorithm 2 implements $VRB_{\mathsf{V}}[d, 0, 3]$, assuming a universal kernel depth $d \geq 4$.*

Algorithm 2 Asymmetric validated reliable broadcast with sender p_s (code for p_i).

State
1: $pending \leftarrow \emptyset$, $proposed \leftarrow \mathsf{false}$, $broadcast \leftarrow \mathsf{false}$

2: **upon** *vrb-broadcast*(m) **and** $\neg$*broadcast* **do**
3: $broadcast \leftarrow \mathsf{true}$
4: send message $[\textsc{send}, m]$ to all $p_j \in \mathcal{P}$

5: **upon** receiving $[\textsc{send}, m]$ from p_s **do**
6: $pending \leftarrow pending \cup \{m\}$

7: **upon** $m \in pending$, $\mathsf{V}_i(m)$, and $\neg proposed$ **do**
8: $proposed \leftarrow \mathsf{true}$ and invoke *rc-propose*(m)

9: **upon** *rc-deliver*(m) **do**
10: **output** *vrb-deliver(m)*

The proof appears in Appendix C.1.

5 Proxcensus

We show how to build proxcensus [16] from reliable consensus. Binary agreement uses proxcensus to expand its inputs, and the approximate common coin uses it to compute weights.

Definition 8 (Depth-characterized proxcensus). *A protocol for* asymmetric Byzantine proxcensus *with maximum grade M, validity depth $vd \geq 0$, safety depth $sd \geq 0$, and liveness depth $ld \geq 1$, written $\mathrm{Prox}[vd, sd, ld]$, has input events* prox-propose(b) *for $b \in \{0, 1\}$ and output events* prox-deliver(g) *for $g \in \{0, \ldots, M\}$. Each correct process may propose each bit once, including a second bit after its first proposal. The protocol satisfies the following properties:*

Graded agreement. *If processes in D_{sd} deliver g and g', then $|g - g'| \leq 1$.*
Validity. *If a process in D_{sd} delivers $g \neq bM$, then some process in D_{vd} previously proposed $1 - b$.*
Integrity. *Every process in D_{sd} delivers at most once.*
Liveness. *For $b \in \{0, 1\}$, let S_b contain the correct processes that eventually invoke* prox-propose(b)*. If some bit b satisfies $S_b \in \mathcal{K}_i$ for every $p_i \in D_{ld}$, then every process in D_{ld} eventually delivers.*

The liveness condition holds, in particular, when every process in D_{ld} eventually proposes the same bit and the universal kernel depth is at least ld. It also holds when every correct process proposes some bit (Lemma 25 in Appendix C.7). Accepting both bits lets a process forward inputs as they become available.

The protocol follows justified proxcensus [22], replacing threshold signatures with reliable consensus. For initial justification, a process rc-proposes a value once it receives that value from one of its kernels. Delivery of the value then plays the role of a "$t + 1$" certificate: some process of depth vd saw the value from a kernel, so a correct process input it. Totality of reliable consensus among processes in D_{ld} replaces transferability of the certificate.

Each round has three RC instances: one for primary votes and one for secondary votes of each parity. A process casts a secondary vote for v by proposing v to the instance of v's parity. Its first secondary vote in a round also proposes v to the primary instance. A primary output v justifies $2v$ in the next round, and secondary outputs v and $v + 1$ justify $2v + 1$.

Each RC instance is tagged by the proxcensus invocation and its purpose: initial justification of a bit, the primary vote of a round, or the secondary vote of a given parity in a round. These tags keep instances distinct when a process proposes both bits. Processes ignore messages whose type, round $r \in \{1, \ldots, k-1\}$, or parity do not match their instance, and votes outside $\{0, \ldots, 2^{r-1}\}$.

Definition 9 (Justified Votes). *We say that a vote v is justified for round r in the view of process p_i if $v \in justified[r]$.*

Theorem 2 (Proxcensus). *Assume universal kernel depth $d \geq 4$ and let $M = 2^{k-1}$. If every tagged reliable consensus instance implements $RC[d, 0, 3]$, then Algorithm 3 implements $Prox[d-1, 0, 3]$ with outputs in $\{0, \ldots, M\}$.*

The proof appears in Appendix C.2.

6 Announce and Eventually Consistent Quorum Systems

The protocols in the following sections share a single long-lived instance of *announce*, in which each process broadcasts its quorum system once through validated reliable broadcast. This binds faulty processes to a single, consistent choice of advertised quorum system. The validity check ensures that every announced quorum contains a process of sufficient depth.

An *advertised quorum system* $\widehat{\mathcal{Q}}_j$ is an encoded nonempty family of nonempty sets of process identifiers attributed to p_j. A correct p_j advertises $\mathcal{Q}_j$. Each process invokes announce exactly once. The output at p_i is a partial map $\widehat{\mathbb{Q}}_i : p_j \longmapsto \widehat{\mathcal{Q}}_j$, called the *eventually consistent quorum system* of p_i. It grows as announcements are delivered, but its entries are never replaced.

Definition 10 (Eventually consistent quorum system). *At announce validity depth d, the output maps satisfy the following properties.*

Monotonicity. *If $\widehat{\mathbb{Q}}_i[p_j]$ is defined, it remains defined with the same value for the rest of the execution.*

Validity. *If p_j is correct and $\widehat{\mathbb{Q}}_i[p_j]$ is defined at a correct process, then $\widehat{\mathbb{Q}}_i[p_j] = \mathcal{Q}_j$.*

Algorithm 3 Proxcensus with $M+1$ grades for $M = 2^{k-1}$ (code for p_i)

State
1: $\mathit{proposed} \leftarrow \emptyset$
2: $\mathit{justified}, \mathit{handled} \leftarrow [\emptyset]^k$

3: **upon** *prox-propose*(v) with $v \in \{0,1\} \setminus \mathit{proposed}$ **do**
4: $\mathit{proposed} \leftarrow \mathit{proposed} \cup \{v\}$
5: send $[\textsc{input}, v]$ to all $p_j \in \mathcal{P}$

6: **upon** receiving $[\textsc{input}, v]$ from each p_j in some $K \in \mathcal{K}_i$ for the first time **do**
7: *rc-propose*$([\textsc{justify}, v])$ with tag v

8: **upon** *rc-deliver*$([\textsc{justify}, v])$ with tag v **do**
9: $\mathit{justified}[1] \leftarrow \mathit{justified}[1] \cup \{v\}$

10: **upon** $\mathit{justified}[r] \setminus \mathit{handled}[r] \neq \emptyset \ \wedge r \neq k$ **do**
11: choose $v \in \mathit{justified}[r] \setminus \mathit{handled}[r]$
12: **if** $\mathit{handled}[r] = \emptyset$ **then**
13: *rc-propose*$([1\textsc{st}, v])$ with tag r
14: *rc-propose*$([2\textsc{nd}, v])$ with tag $(r, v \mod 2)$
15: $\mathit{handled}[r] \leftarrow \mathit{handled}[r] \cup \{v\}$

16: **upon** *rc-deliver*$([1\textsc{st}, v])$ with tag r **do**
17: $\mathit{justified}[r+1] \leftarrow \mathit{justified}[r+1] \cup \{2 \cdot v\}$

18: **upon** *rc-deliver*$([2\textsc{nd}, v])$ with tag $(r, 0)$ and *rc-deliver*$([2\textsc{nd}, v'])$ with tag $(r, 1)$ **do**
19: $\mathit{justified}[r+1] \leftarrow \mathit{justified}[r+1] \cup \{v + v'\}$

20: **upon** $\mathit{justified}[k] \neq \emptyset \ \wedge \ \mathit{handled}[k] = \emptyset$ **do**
21: choose $v \in \mathit{justified}[k]$
22: $\mathit{handled}[k] \leftarrow \mathit{handled}[k] \cup \{v\}$
23: **output** *prox-deliver(v)*

Consistency. *If $\widehat{\mathbb{Q}}_i[p_j]$ and $\widehat{\mathbb{Q}}_k[p_j]$ are defined at correct processes, then they are equal.*

Eventual consistency. *If $\widehat{\mathbb{Q}}_i[p_j]$ is defined at a process in D_3, then the same entry is eventually defined at every process in D_3.*

Correct-announcement liveness. *If a correct process p_j invokes announce, then $\widehat{\mathbb{Q}}_i[p_j] = \mathcal{Q}_j$ is eventually defined at every process $p_i \in D_3$.*

Certified depth. *Every quorum in an entry defined at a process in D_3 contains a process in D_{d-1}.*

Definition 11 (Certified quorum). *At process p_i, a set R is a* certified quorum of p_j *if $\widehat{\mathbb{Q}}_i[p_j]$ is defined and $R \in \widehat{\mathbb{Q}}_i[p_j]$.*

For any quorum system $\mathcal{A}$, local or advertised, define its kernels as

$$\mathcal{K}(\mathcal{A}) := \{R \subseteq \mathcal{P} : \ \forall Q \in \mathcal{A},\ R \cap Q \neq \emptyset\}, \tag{1}$$

extending the notation of Section 2.2 with $\mathcal{K}_i = \mathcal{K}(\mathcal{Q}_i)$. Membership $R \in \mathcal{K}(\mathcal{A})$ can be checked from R and the description of $\mathcal{A}$.

Process p_i accepts a well-formed announcement $\widehat{\mathcal{Q}}_j$ if the following predicate holds and p_j is the designated sender of the broadcast instance:

$$\mathsf{AnnounceValid}_i(p_j, \widehat{\mathcal{Q}}_j) := \forall R \in \widehat{\mathcal{Q}}_j : \ R \in \mathcal{K}_i.$$

The predicate depends only on the fixed $\mathcal{Q}_i$. It is therefore monotone, and p_i can evaluate it as soon as the announcement arrives. Validated reliable broadcast may still deliver an announcement at p_i for which the predicate fails. Since each process has a single announce instance, validated reliable broadcast consistency fixes at most one quorum system per identity.

The notation $\widehat{\mathbb{Q}}_i[p_j]$ denotes a locally available entry, while $\widehat{\mathcal{Q}}_j$ denotes the advertised system stored in that entry. Consistency permits the receiver index to be omitted when referring to a value delivered at correct processes. At a process in D_3, a certified quorum contains a process in D_{d-1} by certified depth. To check a certified quorum R against a specified quorum system $\mathcal{A}$, a process tests $R \in \mathcal{K}(\mathcal{A})$.

Algorithm 4 Announce protocol (code for p_i).

State
1: $\widehat{\mathbb{Q}}_i \leftarrow \emptyset$

2: **upon** *announce*$(\mathcal{Q}_i)$ **do**
3: *vrb-broadcast*$([\text{ANNOUNCE}, p_i, \mathcal{Q}_i])$

4: **upon** *vrb-deliver*$([\text{ANNOUNCE}, p_j, \widehat{\mathcal{Q}}_j])$ **do**
5: **if** $\widehat{\mathbb{Q}}_i[p_j]$ is undefined **then**
6: $\widehat{\mathbb{Q}}_i[p_j] \leftarrow \widehat{\mathcal{Q}}_j$
7: make the updated quorum system $\widehat{\mathbb{Q}}_i$ available locally

Each sender uses exactly one instance of $VRB_{\mathsf{AnnounceValid}}[d, 0, 3]$. Validated reliable broadcast Totality 2 propagates every delivered entry. Protocols may use the current map immediately and may buffer a message that references an entry not yet present.

Lemma 1 (Announce). *Assume the open quorum consistency conditions of Definition 5 and universal kernel depth $d \geq 4$. Then Algorithm 4 implements an eventually consistent quorum system satisfying Definition 10.*

The proof appears in Appendix C.3.

7 Validated Graded Gather

We generalize the graded gather primitive from [23] to local trust and multiple output layers. Let $\mathsf{V} = (\mathsf{V}_i)_{p_i \in \mathcal{P}}$ be a monotone local validity predicate on inputs tagged with a corresponding sender (p_j, x_j).

Definition 12 (Depth-characterized validated graded gather). *For a number of layers $\ell \geq 2$, a protocol for* validated graded gather *with validity depth vd, safety depth sd, and liveness depth ld is denoted by $gg^{\ell}_{\mathsf{V}}[vd, sd, ld]$. A process proposes an input through gg-propose(x) and outputs sets $S_1, \ldots, S_\ell$ through gg-deliver$(S_1, \ldots, S_\ell)$. The protocol satisfies the following properties:*

Common core. *If a correct process delivers, there exist processes p_i and p_j, a quorum $Q_i \in \mathcal{Q}_i$, and a common set S_0 such that p_i has depth at least $vd-1$, $p_j \in Q_i$ has depth at least vd, S_0 contains a tagged value (p_l, x_l) for every $p_l \in Q_i$, and $S_0 \subseteq S_{1,k}$ for every process p_k of depth sd that delivers. For each correct $p_l \in Q_i$, the value x_l was proposed by p_l.*

Boundedness. *If processes p_i and p_j of depth sd deliver output sets, then $S_{r,i} \subseteq S_{r+1,j}$ for every $r \in \{1, \ldots, \ell - 1\}$.*

Validity. *If a process of depth sd delivers $(S_1, \ldots, S_\ell)$ with $(p_k, x_k) \in S_\ell$ and p_k correct, then p_k previously invoked gg-propose(x_k).*

External validity. *Every tagged input (p_i, x_i) in S_ℓ delivered by a process of depth sd satisfies external validity: $\mathsf{V}_j(p_i, x_i)$ previously held at some $p_j \in D_{vd}$.*

Agreement. *If two processes of depth sd deliver sets S_ℓ and S'_ℓ containing (p_k, x_k) and (p_k, x'_k), respectively, then $x_k = x'_k$.*

Totality. *If every process in D_{ld} invokes gg-propose and every input x_j proposed by a process $p_j \in D_{ld}$ eventually satisfies $\mathsf{V}_k(p_j, x_j)$ at every $p_k \in D_{ld}$, then every process in D_{ld} eventually invokes gg-deliver with ℓ output sets.*

Construction. We instantiate the interface with $\ell = 4$, using the eventually consistent quorum systems of Section 6 with announce validity depth five. The protocol takes a common upper bound $N \geq |\mathcal{P}|$ as a parameter; the identities and number of participants remain unknown. It uses $L := \lceil \log_2 N \rceil + 2$ collection levels. For security parameter λ we set $N = 2^\lambda$, so $L = \lambda + 2$.

Every message is sent through its own instance of validated reliable broadcast with depths $[5, 0, 3]$, identified by the gather instance, message type, level, and sender. A proposal from p_j at level 1 must have payload exactly $\{(p_j, x_j)\}$ and satisfy $\mathsf{V}_i(p_j, x_j)$ at verifier p_i. All other messages use the structural predicates described below.

Collection levels. A process invoking *gg-propose*(x_i) broadcasts $\{(p_i, x_i)\}$ as its relay at level 1. At each level $r < L$, a process:

1. collects level-r relays from all members of one of its own quorums and broadcasts their union as a level-r *collection*;
2. collects level-r collections from all members of any certified quorum and broadcasts their union as a level-$(r+1)$ *aggregation*;

3. relays the first valid level-$(r+1)$ aggregation it delivers.

Consistency between the collectors' own quorums ensures that any two level-r relay payloads contain a common level-$(r-1)$ relay payload. After L levels, this descent yields a single proposal set contained in every final relay payload, without increasing the required universal kernel depth.

Candidate and output layers. The candidate layer uses collection, aggregation, and relay; the output layer uses collection and aggregation.

- *Candidates.* Collecting level-L relays yields an inner set (their intersection) and an outer set (their union). Aggregation intersects the inner sets and unites the outer sets.
- *Outputs.* Collecting candidate relays yields four sets: the intersection and union of the inner sets, and the intersection and union of the outer sets. Aggregation applies intersection, union, intersection, and union, respectively.

A process outputs the payload of the first valid output aggregation it delivers.

Validation. Every collection and aggregation names its supporting quorum and references the message it used from each member. A collection must use a quorum from the sender's own announced system, so quorum consistency applies between correct senders. An aggregation may use a certified quorum (Definition 11) of any process; by certified depth, this quorum contains a process in D_4. Each relay references an aggregation of the same level and reproduces its payload.

A receiver waits for the relevant announce entry and referenced messages, then checks quorum membership and recomputes the claimed set operations. Every derived payload can therefore be traced back to the original proposals. The predicates are monotone: later deliveries can only supply missing references, and announce and validated reliable broadcast consistency fix the contents of references already delivered. The protocol is given in Algorithms 5 and 6.

Theorem 3 (Validated graded gather). *Assume the open quorum consistency conditions of Definition 5, the announce protocol from Lemma 1 instantiated at validity depth five, universal kernel depth five, and a common upper bound $N \geq |\mathcal{P}|$. Then Algorithms 5 and 6 implements $\mathrm{gg}_{\mathsf{V}}^{4}[5,0,3]$ using $L = \lceil \log_2 N \rceil + 2$ collection levels. Whenever a correct process delivers, the common core contains a tagged value for every member of a quorum of a process in D_4 and contains an input proposed by a process in D_5.*

The proof appears in Appendix C.4.

8 Verifiable Sharing for the Coin

We construct a verifiable sharing protocol from announce and validated reliable broadcast (VRB). The dealer commits to additive sharings indexed by advertised quorum systems, and local intersection checks certify the existence of a recipient set of sufficient depth for reconstruction. For independently sampled

Algorithm 5 Collection levels of validated graded gather (code for p_i).

Convention
Execute each trigger at most once for each message type and level.

1: **upon invocation** *gg-propose*(x_i) **do**
2: *vrb-broadcast*$([\text{RELAY}, 1, p_i, \{(p_i, x_i)\}])$

3: **upon** $r < L$ and delivery of level-r relays from every member of some $Q_i \in \mathcal{Q}_i$ **do**
4: wait until $\widehat{\mathbb{Q}}_i[p_i]$ is defined
5: $T_i^{\text{col},r} \leftarrow \bigcup_{p_j \in Q_i} \text{payload}([\text{RELAY}, r, p_j])$
6: *vrb-broadcast*$([\text{COLLECTION}, r, p_i, Q_i, T_i^{\text{col},r}])$

7: **upon** $r < L$ and delivery of level-r collections from every member of a certified quorum R of p_j **do**
8: $T_i^{\text{agg},r+1} \leftarrow \bigcup_{p_k \in R} \text{payload}([\text{COLLECTION}, r, p_k])$
9: *vrb-broadcast*$([\text{AGGREGATION}, r+1, p_i, p_j, R, T_i^{\text{agg},r+1}])$

10: **upon first delivery of a level-r aggregation with payload T do**
11: *vrb-broadcast*$([\text{RELAY}, r, p_i, T])$

coin contributions, Section 9 establishes the direct probability bound used by binary agreement.

Fix a dealer p_D and a finite additive group $\mathbb{G}$ of secrets and shares. The sharing phase exposes the events *vss-share*(s) and *vss-share-complete*(Γ). For the coin, *vss-share-random* instead requests a fresh uniform input sampled after the recipient description has been fixed. The reconstruction phase exposes *vss-reconstruct* and *vss-deliver*(s). Within each instance, a correct dealer invokes at most one of *vss-share*(s) and *vss-share-random*, and each correct process invokes *vss-reconstruct* at most once and only after its local *vss-share-complete* event.

Definition 13 (Depth-characterized verifiable sharing). *A VSS protocol must provide the following properties, except with negligible probability in the security parameter.*

Sharing consistency. *If two correct processes complete the sharing phase, they complete it for the same transcript Γ.*

Sharing liveness. *If the dealer has depth at least* 3*, and invokes* vss-share(s) *or* vss-share-random*, then every process of depth at least* 3 *eventually completes the sharing phase.*

Sharing totality. *If a process of depth at least* 3 *completes the sharing phase, then every process of depth at least* 3 *eventually completes it.*

Dealer correctness. *If a correct dealer shares s and a correct process completes the sharing phase for Γ, then $s_\Gamma = s$.*

Algorithm 6 Candidate and output layers of validated graded gather (code for p_i).

upon delivery of level-L relays from every member of some $Q_i \in \mathcal{Q}_i$ **do**
  wait until $\widehat{\mathbb{Q}}_i[p_i]$ is defined
  $A_i^{\text{col}} \leftarrow \bigcap_{p_j \in Q_i} \text{payload}([\text{RELAY}, L, p_j])$
  $B_i^{\text{col}} \leftarrow \bigcup_{p_j \in Q_i} \text{payload}([\text{RELAY}, L, p_j])$
  vrb-broadcast$([\text{CANDIDATE-COLLECTION}, p_i, Q_i, A_i^{\text{col}}, B_i^{\text{col}}])$

upon delivery of candidate collections from every member of a certified quorum R of p_j **do**
  $A_i^{\text{agg}} \leftarrow \bigcap_{p_k \in R} A_k^{\text{col}}$ and $B_i^{\text{agg}} \leftarrow \bigcup_{p_k \in R} B_k^{\text{col}}$
  vrb-broadcast$([\text{CANDIDATE-AGGREGATION}, p_i, p_j, R, A_i^{\text{agg}}, B_i^{\text{agg}}])$

upon first delivery of a candidate aggregation with payload (A, B) **do**
  vrb-broadcast$([\text{CANDIDATE-RELAY}, p_i, A, B])$

upon delivery of candidate relays from every member of some $Q_i \in \mathcal{Q}_i$ **do**
  wait until $\widehat{\mathbb{Q}}_i[p_i]$ is defined
  $S_{1,i}^{\text{col}} \leftarrow \bigcap_{p_j \in Q_i} A_j$ and $S_{2,i}^{\text{col}} \leftarrow \bigcup_{p_j \in Q_i} A_j$
  $S_{3,i}^{\text{col}} \leftarrow \bigcap_{p_j \in Q_i} B_j$ and $S_{4,i}^{\text{col}} \leftarrow \bigcup_{p_j \in Q_i} B_j$
  vrb-broadcast$([\text{OUTPUT-COLLECTION}, p_i, Q_i, S_{1,i}^{\text{col}}, S_{2,i}^{\text{col}}, S_{3,i}^{\text{col}}, S_{4,i}^{\text{col}}])$

upon delivery of output collections from every member of a certified quorum R of p_j **do**
  $S_{1,i}^{\text{agg}} \leftarrow \bigcap_{p_k \in R} S_{1,k}^{\text{col}}$ and $S_{2,i}^{\text{agg}} \leftarrow \bigcup_{p_k \in R} S_{2,k}^{\text{col}}$
  $S_{3,i}^{\text{agg}} \leftarrow \bigcap_{p_k \in R} S_{3,k}^{\text{col}}$ and $S_{4,i}^{\text{agg}} \leftarrow \bigcup_{p_k \in R} S_{4,k}^{\text{col}}$
  vrb-broadcast$([\text{OUTPUT-AGGREGATION}, p_i, p_j, R, S_{1,i}^{\text{agg}}, S_{2,i}^{\text{agg}}, S_{3,i}^{\text{agg}}, S_{4,i}^{\text{agg}}])$

upon first delivery of an output aggregation with payload (S_1, S_2, S_3, S_4) **do**
  gg-deliver(S_1, S_2, S_3, S_4)

Commitment. *If a correct process completes the sharing phase for Γ, then Γ determines a unique value s_Γ. Every value delivered by a correct process during reconstruction equals s_Γ.*

Reconstruction liveness. *Suppose the underlying VRB instance has validity depth $d \geq 5$. If a process in D_3 completes sharing and every process in D_3 invokes reconstruction after local sharing completion, then every process in D_3 eventually delivers s_Γ.*

8.1 Preliminaries and cryptographic primitives

Let $(\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ be an IND-CPA public-key encryption scheme with message space containing $\mathbb{G}$. We require perfect correctness for every allowed key-

generation tape η, message m, and encryption tape r:

$$(\mathsf{pk}, \mathsf{sk}) = \mathsf{KeyGen}(\eta) \quad \Longrightarrow \quad \mathsf{Dec}_{\mathsf{sk}}(\mathsf{Enc}_{\mathsf{pk}}(m; r)) = m.$$

Decryption is unique on generated keys: secret keys associated with the same public key agree on the decryption of well-formed ciphertexts. Encryption is a total polynomial-time algorithm on syntactically admissible public-key strings, with rejection represented by a canonical ciphertext if necessary. Correctness is required for every key in the image of KeyGen, whereas IND-CPA security is required for honestly generated keys. Keys supplied by Byzantine processes may be chosen adversarially. The construction uses the authenticated identifiers of Section 2 and generates all encryption keys during execution.

We use an auxiliary-input black-box concurrently zero-knowledge argument for NP, with completeness, negligible soundness error under polynomially many interleaved sessions, and a polynomial-time simulator. The simulator has a polynomial running-time bound on all statements, including false statements used in the encryption hybrids of Appendix C.6. The concrete instantiation below uses the perfectly hiding commitment variant of Prabhakaran, Rosen, and Sahai, whose explicit simulator has this bound independently of statement truth. The same argument system implements both relations below in the plain model. Each proof session identifies its statement, prover, verifier, invocation, and purpose. The prover and verifier address each proof message to the other process using the multicast convention of Section 2.

Concrete instantiation. For the coin group $\mathbb{G} = \mathbb{Z}_{2^{2\lambda}}$, a concrete instantiation uses bitwise ElGamal encryption in a fixed family of efficiently represented cyclic groups $\mathbb{H}_\lambda = \langle g \rangle$ of prime order q, under the decisional Diffie–Hellman assumption. The group family, its generator, canonical encodings, and efficient membership test are part of the public algorithm description. No party generates common parameters with a secret trapdoor. Write $\ell = 2\lambda$ and $m = \sum_{t=0}^{\ell-1} 2^t m_t$ for the canonical binary representation of a share. Key generation chooses an exponent $a \in \mathbb{Z}_q$ and returns $(Y, a) = (g^a, a)$. Encryption with independent exponents $r_t \in \mathbb{Z}_q$ is

$$\mathsf{Enc}_Y(m; (r_t)_t) = \big(u_t, v_t\big)_{t=0}^{\ell-1} = \big(g^{r_t}, Y^{r_t} g^{m_t}\big)_{t=0}^{\ell-1}.$$

Decryption computes v_t / u_t^a, interprets 1 as zero and g as one, and returns $\perp$ if any component has neither value or fails canonical group membership. Public-key syntax requires canonical group membership, and encryption on any rejected key string returns a fixed dummy ciphertext. For bounded binary-tape implementations of key generation and encryption, put $n = \lceil \log_2 q \rceil$ and obtain each exponent by reducing an independent $(n + 2\lambda)$-bit tape modulo q. This changes its uniform distribution by at most $2^{-2\lambda}$ and leaves correctness exact for every tape. Every exponent has a corresponding tape, so the opening relation below is equivalently $\exists a : Y = g^a \wedge \mathsf{Dec}_a(c) = x$.

For every generated key and every encryption tape, $v_t / u_t^a = g^{m_t}$. Moreover, $g^a = g^{a'}$ implies $a = a'$ in $\mathbb{Z}_q$, so generated secret keys give identical decryption, including on malformed ciphertexts. A standard bitwise hybrid reduces

IND-CPA security to DDH, with only a polynomial number of hybrids and the negligible sampling difference above. This encoding avoids requiring a discrete logarithm to recover a share. All equations, bit constraints, and sums modulo $2^{2\lambda}$ in the sharing and opening relations have polynomial-size verification circuits.

For these circuits, use the concurrent zero-knowledge protocol of Prabhakaran, Rosen, and Sahai, Theorem 1 and Sections 3–4 of Concurrent Zero Knowledge with Logarithmic Round-Complexity, with its Hamiltonicity reduction and superlogarithmically many preamble iterations. Taking λ iterations suffices for polynomial-size statements and executions. Instantiate its preamble commitments bitwise by interactive Pedersen commitments. The commitment receiver first sends a fresh $h = g^b \neq 1$, and the committer checks canonical group membership and $h \neq 1$ before sending $g^m h^r$ for a bit m and fresh $r \in \mathbb{Z}_q$. The receiver generates this base within the proof session and keeps its state. The commitment is perfectly hiding for every permitted receiver choice and computationally binding against a malicious committer when the receiver is honest, since distinct openings yield $\log_g h$. This receiver message precedes the preamble's initial commitment and adds only a constant number of messages. For the Hamiltonicity stage use Naor's statistically binding commitment from a length-tripling pseudorandom generator, obtained, for example, by applying the Håstad–Impagliazzo–Levin–Luby construction to a discrete-logarithm one-way function. These ingredients require no independent hardness assumption beyond DDH. All commitment parameters are generated within their sessions. We analyze these commitment protocols with exact uniform scalar sampling. Replacing their scalar samplers by bounded samplers within statistical distance $2^{-2\lambda}$ changes every polynomially bounded execution and simulator execution by negligible statistical distance. Perfect hiding refers to the exact-sampling version, and the bounded implementation inherits the required guarantees through this statistical comparison.

The recursive simulation schedule of Sections 3.2 and 4.1 has a polynomial bound even on false statements. Its simulation applies with arbitrary polynomial-size auxiliary verifier state by incorporating that state into the verifier it rewinds. For the coin bound in Appendix C.6, the simulator replaces only the sharing proofs of one dealer after its transcript Γ_D has been fixed. The experiment stops before that dealer or a member of a chosen quorum of that dealer is corrupted, or a member of that quorum decrypts a ciphertext from the dealer's sharing. All other proof sessions are executed normally, so the states of uncorrupted participants are available if they are later corrupted. Separate keys for each dealer and invocation allow the reduction to recover the other dealers' committed contributions without the secret key corresponding to the challenge public key.

8.2 Sharing transcripts and verification

Recipient keys. Fix an invocation identifier σ and a dealer p_D. Upon first learning of this sharing or receiving a key request, an uncorrupted process p_k

independently samples $\eta_{\sigma,D,k}$, computes

$$(\mathsf{pk}_{\sigma,D,k}, \mathsf{sk}_{\sigma,D,k}) = \mathsf{KeyGen}(\eta_{\sigma,D,k}),$$

and broadcasts $(\sigma, p_D, p_k, \mathsf{pk}_{\sigma,D,k})$ through a uniquely tagged VRB instance with sender p_k. The local validity predicate checks syntax and the invocation, dealer, and sender identifiers. Process p_k keeps the entire key-generation state and uses this key pair exclusively for dealer p_D in invocation σ. Key requests and their associated broadcasts are processed independently of the holder's own sharing and coin-start events. VRB consistency fixes at most one delivered key per triple (σ, p_D, p_k). Totality 1 disseminates every correct sender's key throughout D_3, irrespective of the sender's depth, and Totality 2 propagates any key delivered in D_3 throughout D_3.

Recipient sets. Write $\widehat{\mathbb{Q}}_i[p_j]$ for the announce entry for p_j at p_i. The dealer selects a *commitment quorum* $C \in \mathcal{Q}_D$ and a set R of holders whose key broadcasts it has delivered. It requires defined announce entries for all processes in $C \cup \{p_D\}$ and, writing $\widehat{\mathcal{Q}}_j = \widehat{\mathbb{Q}}_D[p_j]$, requires

$$A_{j,Q} := Q \cap R \in \mathcal{K}(\mathcal{Q}_D) \qquad \text{for every } p_j \in C \text{ and } Q \in \widehat{\mathcal{Q}}_j. \tag{2}$$

All recipient sets $A_{j,Q}$ must be nonempty. The advertised quorums identify the holders to which the dealer sends key requests. It requests keys from holders in all advertised quorums encountered while considering its candidate commitment quorums and continues considering every candidate as announcements and keys arrive. Upon finding suitable C, R, it fixes these sets, the advertised systems, and the referenced key broadcasts. For input *vss-share*(s), it uses the supplied secret. For input *vss-share-random*, it samples $s \leftarrow\!\!\$\ \mathbb{G}$ independently after fixing this description. The coin instantiates $\mathbb{G} = \mathbb{Z}_{2^{2\lambda}}$.

Sharing relation. A sharing transcript has the form

$$\Gamma = \left(\sigma, p_D, C, R, \{\widehat{\mathcal{Q}}_j^{\Gamma}\}_{p_j \in C \cup \{p_D\}}, \{\mathsf{pk}_{\sigma,D,k}\}_{p_k \in R}, \{c_{j,Q,k}\}_{j,Q,k}\right),$$

where the key entries include references to their broadcasts and the ciphertext indices range over $p_j \in C$, $Q \in \widehat{\mathcal{Q}}_j^{\Gamma}$, and $p_k \in A_{j,Q} = Q \cap R$. A correct dealer sets $\widehat{\mathcal{Q}}_D^{\Gamma} = \mathcal{Q}_D$ and $\widehat{\mathcal{Q}}_j^{\Gamma} = \widehat{\mathbb{Q}}_D[p_j]$ for each $p_j \in C$. The sharing relation holds for Γ if there exist $s \in \mathbb{G}$ and shares and randomness $\{s_{j,Q,k}, r_{j,Q,k}\}_{j,Q,k}$ satisfying

$$c_{j,Q,k} = \mathsf{Enc}_{\mathsf{pk}_{\sigma,D,k}}(s_{j,Q,k}; r_{j,Q,k}), \qquad \sum_{p_k \in A_{j,Q}} s_{j,Q,k} = s \quad \text{in } \mathbb{G}$$

for every indicated index. Thus each recipient set carries an additive sharing of the same witness s. The dealer generates these sharings independently, encrypts their shares with independent randomness, and proves the relation to each process using Γ as the common NP statement. The interactive proof sessions are identified separately from the common sharing transcript, and their transcripts may be observed by the adversary.

Local validity. The predicate $\mathsf{Valid}_i(\Gamma)$ holds if and only if all the following conditions hold:

1. The invocation and designated dealer agree with the VRB instance.
2. For every $p_j \in C \cup \{p_D\}$, the entry $\widehat{\mathbb{Q}}_i[p_j]$ is defined and equals $\widehat{\mathcal{Q}}_j^{\Gamma}$.
3. $C \in \widehat{\mathcal{Q}}_D^{\Gamma}$ and $C \in \mathcal{K}_i$.
4. For every $p_j \in C$ and $Q \in \widehat{\mathcal{Q}}_j^{\Gamma}$, the set $A_{j,Q} = Q \cap R$ is nonempty and belongs to $\mathcal{K}(\widehat{\mathcal{Q}}_D^{\Gamma})$.
5. Each referenced key broadcast has been delivered locally and agrees with the transcript's public key for the exact triple (σ, p_D, p_k).
6. The transcript contains exactly one ciphertext for each prescribed index (p_j, Q, p_k), and p_i has accepted the sharing argument for this exact Γ.

The predicate is monotone as announce entries, key broadcasts, and proof acceptances become available. The condition on C ensures that it contains a member of a live quorum of the validating process. The conditions on $A_{j,Q}$ ensure that each recipient set intersects every live quorum of the dealer. These two intersection requirements yield reconstruction and the coin-specific security bound, respectively.

Opening relation. For a ciphertext $c_{j,Q,k}$ in a committed transcript Γ, an opening to $x \in \mathbb{G}$ proves

$$\exists \eta, \mathsf{sk}: \quad \mathsf{KeyGen}(\eta) = (\mathsf{pk}_{\sigma,D,k}, \mathsf{sk}) \quad \wedge \quad \mathsf{Dec}_{\mathsf{sk}}(c_{j,Q,k}) = x.$$

The public statement consists of Γ, the ciphertext index (p_j, Q, p_k), and x. A correct holder uses its stored key-generation randomness and secret key as the witness. A process reconstructs from accepted openings for every member of any one recipient set $A_{j,Q}$ and returns their sum in $\mathbb{G}$. In the coin protocol, reconstruction is invoked only after the guard in Algorithm 8 has been satisfied.

8.3 The VSS protocol

For each pair (σ, p_D), use one tagged instance of $VRB_{\mathsf{Valid}}[d, 0, 3]$ with sender p_D and local validity predicate $\mathsf{Valid}_i(\Gamma)$ defined above. A correct prover answers requests from every authenticated process for its own generated sharing transcript or an opening it has already sent at line 14. Correct processes continue participating in proof sessions after completing sharing or delivering a reconstructed value.

An opening is valid at a process if its label occurs in the committed transcript and that process has accepted the holder's interactive correct-decryption argument for the opening. Opening messages and their proof messages received before local sharing completion are buffered until the committed transcript is available. Processes ignore duplicate opening messages and store at most one accepted opening for each index. VRB external validity ensures that every committed transcript passed Valid_i at a process of depth at least d.

Correctness. Sharing liveness at depth three follows from the availability of correct key holders in every required quorum intersection, as established in Lemma 16. Reconstruction of a completed sharing instead follows from the depth

Algorithm 7 Verifiable sharing with dealer p_D (code for p_i), using VRB with local predicate $\mathsf{Valid}_i(\Gamma)$.

Use $VRB_{\mathsf{Valid}}[d, 0, 3]$ with sender p_D, tag (σ, p_D), and local predicate $\mathsf{Valid}_i(\Gamma)$ defined above.

State
  $commit \leftarrow \bot$, $delivered \leftarrow \mathsf{false}$
  $openings \leftarrow \emptyset$

upon *vss-share*(s) or *vss-share-random* at dealer p_D **do**
  request fresh keys and consider all candidate commitment quorums as described above
  wait for C, R satisfying (2) and the required announce deliveries
  freeze C, R, the advertised systems, and the referenced key broadcasts
  for *vss-share-random*, sample $s \leftarrow\!\!\$\ \mathbb{G}$
  independently sample an additive sharing of s over every $A_{j,Q} = Q \cap R$
  encrypt under $\mathsf{pk}_{\sigma,D,k}$, construct Γ, and invoke *vrb-broadcast*(Γ)
  keep the sharing witness and answer proof requests for Γ

upon receiving the dealer's transcript Γ in the underlying VRB instance **do**
  request a sharing argument from p_D and participate as verifier

upon event *vrb-deliver*(Γ) in instance (σ, p_D) and $commit = \bot$ **do**
  $commit \leftarrow \Gamma$
  output *vss-share-complete*(Γ)

upon *vss-reconstruct* and $commit = \Gamma$ **do**
  for every $p_j \in C$, $Q \in \widehat{\mathcal{Q}}_j^{\Gamma}$ with $p_i \in A_{j,Q}$ **do**
    $x_{j,Q,i} \leftarrow \mathsf{Dec}_{\mathsf{sk}_{\sigma,D,i}}(c_{j,Q,i})$
    send $[\textsc{open}, \sigma, p_D, p_j, Q, p_i, x_{j,Q,i}]$ to all processes
    keep the opening witness and answer proof requests for this opening

upon receiving $[\textsc{open}, \sigma, p_D, p_j, Q, p_k, x]$ from p_k for an index in *commit* **do**
  request a correct-decryption argument from p_k and participate as verifier

upon accepting a correct-decryption argument for (p_j, Q, p_k, x) in *commit* **do**
  add (p_j, Q, p_k, x) to *openings*

upon having one valid opening for every $p_k \in A_{j,Q}$, for some $p_j \in C$ and $Q \in \widehat{\mathcal{Q}}_j^{\Gamma}$, and $\neg$*delivered* **do**
  $delivered \leftarrow \mathsf{true}$
  output *vss-deliver*$\big(\sum_{p_k \in A_{j,Q}} x_{j,Q,k}\big)$

of its validator and imposes no correctness or depth requirement on the dealer. By VRB external validity, some validator $p_i \in D_d$ accepted the sharing relation and all structural checks. For $d \geq 5$, choose a live quorum $H_i \subseteq D_{d-1}$ of this validator. The check $C \in \mathcal{K}_i$ gives $p_j \in C \cap H_i$, and this correct process has a

live quorum $Q^* \subseteq D_{d-2}$ in its announced system. Consequently,

$$p_i \in D_d \implies p_j \in C \cap H_i \subseteq D_{d-1} \implies \varnothing \neq A_{j,Q^*} \subseteq Q^* \subseteq D_{d-2} \subseteq D_3.$$

The recipient set A_{j,Q^*} fixes the shared value and allows reconstruction under the completion and invocation premises of Definition 13. Soundness of the opening relation and perfect encryption correctness ensure that every accepted reconstruction agrees with this value, including reconstructions using Byzantine holders.

Theorem 4. *Assume the encryption and concurrent zero-knowledge argument primitives specified in Section 8.1. Assume also the announce and VRB constructions from Lemma 1 and Corollary 1. With universal kernel depth $d \geq 5$, Algorithm 7 satisfies all properties of Definition 13, including reconstruction liveness.*

The proof appears in Appendix C.5. The coin-specific security argument appears in Appendix C.6.

9 Approximate Common Coin

We combine the VSS construction, validated graded gather from Section 7, and proxcensus into an approximate common coin. Validated graded gather identifies the VSS instances that may receive nonzero weight, and proxcensus assigns weights that differ by at most one between correct processes. The construction establishes termination, proximity, and a direct bound on the probability of falling near a threshold fixed by the protocol invoking the coin.

Set $M = 2^{2\lambda}$ and $\mathbb{G} = \mathbb{Z}_M$. For $x, y \in \mathbb{Z}_M$, let $\mathsf{dist}_M(x, y) = \min\{|\bar{x} - \bar{y}|, M - |\bar{x} - \bar{y}|\}$, where $\bar{x}, \bar{y} \in \{0, \ldots, M-1\}$ are the canonical representatives. To round modulo M, set $\mathsf{Round}_M(z) = \lfloor z + 1/2 \rfloor \bmod M$. This rounding rule commutes with adding an integer.

Definition 14 (Approximate common coin). *A Δ-approximate common coin over $\mathbb{Z}_M$ has safety depth sd and liveness depth ld. Its interface consists of* coin-start *and* coin-deliver(c) *events. The following properties hold except with negligible probability. The protocol invoking the coin accesses it through these events and may observe the public transcript. It does not read uncorrupted processes' internal VSS inputs, shares, encryption keys, or proof state. Its own randomness is independent of the fresh inputs sampled inside the coin. Let E be the event that a correct process enables reconstruction. At the first such event, define the reference value $c^* \in \mathbb{Z}_M$ by applying the output formula of Algorithm 8 to that process's committed VSS secrets and the weights it has fixed. For the security bound, let $q(\lambda)$ be a fixed polynomial upper bound on $|\mathcal{P}| \sum_{p_D \in \mathcal{P}} |\mathcal{Q}_D|$ over the experiment's possible initializations, as supplied by the explicit representation bound.*

Termination. *If every process in D_{ld} invokes* coin-start*, then every process in D_{ld} eventually delivers an output.*

Proximity. *If two processes in D_{sd} deliver c_i and c_j, then* $\mathsf{dist}_M(c_i, c_j) \leq \Delta$. *Every such output also satisfies* $\mathsf{dist}_M(c_i, c^*) \leq \Delta$.

Direct bound. *Suppose the protocol invoking the coin fixes a value $x_i \in \{0, \ldots, M\}$ at each process before that process starts the coin. Write x_* for this value at the first correct process to enable reconstruction. With B_Δ defined in* (3),

$$\Pr[E \wedge c^* \in B_\Delta(x_*)] \leq q(\lambda)\frac{4\Delta + 4}{M} + \mathsf{negl}(\lambda).$$

Executions in which E does not occur contribute zero to this probability.

Composition interface. For the coin invocation of validated graded gather, the monotone local predicate $\mathsf{CoinValid}_i(p_D, \mathsf{shared})$ holds exactly when p_i has output *vss-share-complete*(Γ_D) in dealer p_D's VSS instance. Dealer p_D proposes the fixed value shared only after completing its own sharing phase. A validated graded gather output consists of four sets of tagged inputs (p_D, shared), which we identify with the corresponding sets $S_{1,i}, S_{2,i}, S_{3,i}, S_{4,i}$ of dealer identities at process p_i. External validity and VSS sharing totality imply that every dealer in $S_{4,i}$ has a completed sharing phase that eventually becomes available to every process of depth at least 3. The set $S_{3,i}$ determines the weights and secrets needed for the coin.

We use $S_{r,i} \subseteq S_{r+1,j}$ for all correct processes p_i, p_j that deliver graded-gather outputs and every $r \in \{1, 2, 3\}$, totality, and the common-core guarantee of Theorem 3. In particular, the common core S_0 contains a quorum $A^* \in \mathcal{Q}_i$ of a process p_i of depth at least 4, and $S_0 \subseteq S_{1,j}$ for every correct process p_j that delivers a graded-gather output. These inclusions ensure that the weight instances terminate and that every potentially positive contribution belongs to every correct set S_3.

For each dealer p_D, we use an independent binary-input proxcensus instance with outputs in $\{0, \ldots, M\}$. Besides liveness, we use graded agreement $|g_i[D] - g_j[D]| \leq 1$ and validity from Definition 8: if no correct process proposes 0, every correct output is M, while if no correct process proposes 1, they output 0.

Protocol. Each process p_i starts its VSS sharing preparation at coin-start. After fixing its available recipient sets and their keys, it samples $s_i \leftarrow\!\!\$\, \mathbb{Z}_M$ independently and constructs its encrypted sharings. Once p_i completes its own sharing phase, it proposes shared to validated graded gather.

After obtaining $(S_{1,i}, S_{2,i}, S_{3,i}, S_{4,i})$, process p_i proposes 1 to the proxcensus instance of every dealer in $S_{2,i}$ and proposes 0 to the instance of every dealer in $S_{4,i} \setminus S_{1,i}$. Each bit is proposed at most once per instance, and an instance indexed by a dealer in $S_{2,i} \setminus S_{1,i}$ receives both proposals from p_i.

Process p_i enables reconstruction once it has started the coin, obtained its gather output, obtained weights for every dealer in $S_{3,i}$, and locally completed sharing for every dealer in $S_{3,i}$. From that point on, it invokes reconstruction in every VSS instance in $S_{3,i}$, including instances whose local weight is zero. This ensures that a process sends openings even when another process assigns

a positive weight to an instance with local weight zero. It delivers after reconstructing every instance to which it assigns positive weight. Processes handle subprotocol messages before invoking *coin-start* and after delivering, but they enable reconstruction only after their own start event.

Algorithm 8 Approximate common coin over $\mathbb{Z}_M$ (code for p_i).

State
1: $started \leftarrow \mathsf{false}$, $gatherProposed \leftarrow \mathsf{false}$, $gathered \leftarrow \mathsf{false}$
2: $open \leftarrow \mathsf{false}$, $delivered \leftarrow \mathsf{false}$
3: $S_{1,i}, S_{2,i}, S_{3,i}, S_{4,i} \leftarrow \emptyset$, $weight, secret \leftarrow \emptyset$

4: **upon** *coin-start* and $\neg started$ **do**
5: $started \leftarrow \mathsf{true}$
6: invoke *vss-share-random* in the instance with dealer p_i

7: **upon** *vss-share-complete*(Γ_i) for dealer p_i, and $\neg gatherProposed$ **do**
8: $gatherProposed \leftarrow \mathsf{true}$ and invoke *gg-propose*(shared)

9: **upon** *gg-deliver*($\widetilde{S}_{1,i}, \widetilde{S}_{2,i}, \widetilde{S}_{3,i}, \widetilde{S}_{4,i}$) and $\neg gathered$ **do**
10: **for every** $r \in \{1,2,3,4\}$ **do** $S_{r,i} \leftarrow \{p_D : (p_D, \mathsf{shared}) \in \widetilde{S}_{r,i}\}$
11: $gathered \leftarrow \mathsf{true}$
12: **for every** $p_D \in S_{2,i}$ **do**
13: invoke *prox-propose*$_D(1)$
14: **for every** $p_D \in S_{4,i} \setminus S_{1,i}$ **do**
15: invoke *prox-propose*$_D(0)$

16: **upon** *prox-deliver*$_D(g)$ and $weight[D]$ undefined **do**
17: $weight[D] \leftarrow g$

upon $started$, $gathered$, $\neg open$, and $weight[D]$ defined for every $p_D \in S_{3,i}$
18: and local VSS sharing completed for every $p_D \in S_{3,i}$ **do**
19: $open \leftarrow \mathsf{true}$
20: **for every** $p_D \in S_{3,i}$ **do**
21: invoke *vss-reconstruct*$_D$

22: **upon event** *vss-deliver*$_D(s_D)$ **do**
23: $secret[D] \leftarrow \bar{s}_D \in \{0, \ldots, M-1\}$

24: **upon** $open$, $\neg delivered$, and $secret[D]$ defined for every $p_D \in S_{3,i}$
with $weight[D] > 0$ **do**
25: $delivered \leftarrow \mathsf{true}$
26: $X_i \leftarrow \sum_{p_D \in S_{3,i}:\, weight[D]>0} \frac{weight[D]}{M} secret[D]$
27: **output** *coin-deliver*($\mathsf{Round}_M(X_i)$)

Each process keeps the first output of every weight instance and fixes all weights in $S_{3,i}$ before enabling reconstruction at line 18. Proxcensus ensures that their weights for each dealer differ by at most one, and a dealer omitted from any correct set S_3 has weight zero at every correct process. Since each secret is represented by an integer smaller than M, a weight difference of one changes its contribution to the sum by less than one. Summing these differences gives the proximity bound, including the distance from later outputs to the reference value fixed when the first correct process enables reconstruction (Lemma 20 in Appendix C.6).

The common core also contains a correct dealer whose weight is M at every correct process that fixes its weights, so its secret enters each sum with coefficient one. The recovery computation of Lemma 22 and the experiment analyzed in Lemma 23 establish the direct bound, including adaptive corruption choices.

Theorem 5. *Assume universal kernel depth $d \geq 5$. Use graded gather from Theorem 3, VSS from Theorem 4, and Prox$[d-1, 0, 3]$ with maximum grade M from Theorem 2. Then Algorithm 8 implements a 2^{λ}-approximate common coin over $\mathbb{Z}_{2^{2\lambda}}$ with safety depth 0 and liveness depth 3 for all sufficiently large λ.*

Proof. Lemmas 20 and 24 establish termination and proximity. The direct probability bound follows from Theorem 8.

The proxcensus construction is instantiated with $k = 2\lambda + 1$, giving its two endpoints 0 and $M = 2^{2\lambda}$.

10 Binary Agreement by Expansion and Extraction

We expand binary proposals to a large range using proxcensus, then use the approximate common coin to choose a threshold in that range. Proxcensus places the correct processes' grades within one of each other, while the coin places their thresholds within 2^{λ} of each other on a circle of size $2^{2\lambda}$. Disagreement is therefore possible only when the coin falls close to the expanded grade or to the point where the circle wraps around.

We use the strong termination formulation of [3, Definition 18], expressing strong liveness through kernels of correct processes proposing a common bit. Allowing a process to propose both bits supports applications in which additional inputs become available during the execution.

Definition 15 (Depth-characterized binary Byzantine agreement). *Fix validity depth $vd \geq 0$, safety depth $sd \geq 0$, and liveness depth $ld \geq 1$. We write $BA[vd, sd, ld]$ for binary Byzantine agreement with these depths. Its interface consists of events* ba-propose(b) *and* ba-decide(b) *for $b \in \{0, 1\}$. Each correct process may propose each bit once, including a second bit after its first proposal. The protocol satisfies the following, except with negligible probability in λ:*

Validity. *If a process in D_{sd} decides b, then some process in D_{vd} previously proposed b.*

Agreement. *No two processes in D_{sd} decide different bits.*
Integrity. *Every process in D_{sd} decides at most once.*
Strong liveness. *Let S_b contain the correct processes that eventually propose b. Every process in D_{ld} eventually decides if some $b \in \{0,1\}$ satisfies $S_b \in \mathcal{K}_i$ for every $p_i \in D_{ld}$.*
Liveness. *If every correct process eventually proposes at least one bit, then every process in D_{ld} eventually decides.*

Strong liveness implies liveness by Lemma 25 in Appendix C.7, using the argument from [3].

For composition, strong liveness also applies when some bit is eventually proposed by every process in D_{ld} and the universal kernel depth is at least ld.

Protocol. Use one proxcensus instance with maximum grade $M = 2^{2\lambda}$ and one approximate common coin over $\mathbb{Z}_M$ with proximity bound $\Delta = 2^{\lambda}$. The expansion instance is distinct from the proxcensus instances used internally by the coin, and each agreement invocation uses fresh subprotocol instances and coin randomness. At system initialization, every correct process invokes *announce*$(\mathcal{Q}_i)$ once; all agreement invocations share the resulting growing announce map. Processes handle messages for an agreement instance when they first learn its identifier, even before receiving a local proposal.

Every proposed bit is forwarded to the expansion instance. When process p_i obtains its grade x_i, it fixes that grade and starts the coin. For the coin output c_i, it decides 1 precisely when its canonical representative $\bar{c}_i$ is smaller than x_i. In particular, grade 0 always gives decision 0, and grade M always gives decision 1. Starting the coin only after fixing the grade ensures that the first correct process to enable coin reconstruction already has a fixed expansion grade.

Algorithm 9 Binary agreement from proxcensus and an approximate common coin (code for p_i).

1: **State**
  $proposed \leftarrow \emptyset$, $grade \leftarrow \perp$, $decided \leftarrow$ false

2: **upon invocation** *ba-propose*(b) with $b \in \{0,1\} \setminus proposed$ **do**
  $proposed \leftarrow proposed \cup \{b\}$
  invoke *prox-propose*(b) in the expansion instance

3: **upon event** *prox-deliver*(x) in the expansion instance, and $grade = \perp$ **do**
  $grade \leftarrow x$
  invoke *coin-start*

4: **upon event** *coin-deliver*(c), with $grade \neq \perp$ and $\neg decided$ **do**
  $decided \leftarrow$ true
  if $\bar{c} < grade$ **then output** *ba-decide*(1)
  else output *ba-decide*(0)

Processes continue handling subprotocol messages and forwarding a newly proposed bit after deciding. As in Algorithm 8, a process may handle the coin's subprotocol messages before its local *coin-start*, but reconstruction only after that event.

Extraction. Let p_i be the first correct process to enable coin reconstruction, and write $x_* = x_i$. Let c_* be the reference coin value fixed by its set $S_{3,i}$, weights, and committed VSS secrets, as in Definition 14. For $x \in \{0, \dots, M\}$, define

$$B_\Delta(x) = \{c \in \mathbb{Z}_M : \mathsf{dist}_M(c, 0) \le \Delta\} \cup \{c \in \mathbb{Z}_M : \mathsf{dist}_M(c, x \bmod M) \le \Delta + 1\}. \tag{3}$$

The direct coin bound applies to the already fixed grade x_* and these two neighborhoods.

Theorem 6. *Assume that expansion uses Prox$[vd, 0, ld]$, with $vd \ge 0$ and $ld \ge 1$. Assume also that the coin satisfies Definition 14 with safety depth 0 and liveness depth ld. Then Algorithm 9 implements BA$[vd, 0, ld]$ from Definition 15. The probability that any two correct decisions differ is at most $q(\lambda)(2^{2-\lambda} + 2^{2-2\lambda}) + \mathrm{negl}(\lambda)$, which is negligible in λ.*

The proof appears in Appendix C.7.

Corollary 2 (Binary agreement with unknown participants). *Assume the authenticated communication model of Section 2, the open quorum consistency conditions of Definition 5, universal kernel depth $d \ge 5$, and the encryption and interactive argument primitives specified in Section 8.1. Encryption keys are generated during execution, with Byzantine processes permitted to supply arbitrary keys. For polynomially bounded executions, instantiate the common upper bound on the number of processes as $N = 2^\lambda$, both proxcensus constructions with $k = 2\lambda + 1$, and the VSS secret group as $\mathbb{Z}_{2^{2\lambda}}$. Then the announce, validated graded gather, VSS, proxcensus, and approximate common coin protocols of this paper, composed as in Algorithm 9, implement BA$[d-1, 0, 3]$ for all sufficiently large λ.*

The proof appears in Appendix C.7.

11 Agreement on a Core Set

We construct validated *agreement on a core set* (ACS), in which processes agree on a set of valid proposed messages. Fix a monotone local validity predicate $\mathsf{V} = (\mathsf{V}_i)_{p_i \in \mathcal{P}}$ on tagged payloads (p_j, m_j). Each process proposes at most one payload, and an output is a set of tagged payloads.

Definition 16 (Depth-characterized validated agreement on a core set). *Fix validity depth $vd \ge 1$, safety depth $sd \ge 0$, and liveness depth $ld \ge 1$. We write $ACS_\mathsf{V}[vd, sd, ld]$ for validated agreement on a core set with events* acs-propose(m) *and* acs-deliver(S)*. The protocol satisfies the following properties, except with negligible probability in λ:*

Agreement. *If two processes in D_{sd} deliver sets S and S', then $S = S'$.*
Proposal validity. *If a process in D_{sd} delivers a set containing (p_j, m_j) and p_j is correct, then p_j previously proposed m_j.*
External validity. *Every tagged payload (p_j, m_j) in a set delivered by a process in D_{sd} satisfies $\mathsf{V}_k(p_j, m_j)$ at some process $p_k \in D_{vd}$.*
Core inclusion. *If a process in D_{sd} delivers S, then there exist $p_i \in D_{vd-1}$, $Q_i \in \mathcal{Q}_i$, and $p_j \in Q_i \cap D_{vd}$ such that S contains a tagged payload attributed to every member of Q_i.*
Integrity. *Every process in D_{sd} delivers at most once.*
Liveness. *If every process $p_j \in D_{ld}$ proposes a payload m_j and $\mathsf{V}_k(p_j, m_j)$ eventually holds at every $p_k \in D_{ld}$, then every process in D_{ld} eventually delivers.*

The construction uses one instance of validated graded gather, $gg^4_{\mathsf{V}}[5, 0, 3]$, with validity predicate V, and one binary-agreement instance BA_j for each process p_j that proposes an input. After receiving its four sets from validated graded gather, process p_i proposes 1 to BA_j for every $(p_j, m_j) \in S_{2,i}$ and proposes 0 for every $(p_j, m_j) \in S_{4,i} \setminus S_{1,i}$. An instance indexed by an input in $S_{2,i} \setminus S_{1,i}$ therefore receives both proposals from p_i. Process p_i waits for the binary-agreement decision of every sender in $S_{3,i}$ and outputs the tagged payloads whose instances decide 1.

Algorithm 10 Validated agreement on a core set (code for p_i).

State
1: *delivered* $\leftarrow$ false

2: **upon invocation** *acs-propose*(m_i) **do**
3: invoke *gg-propose*(m_i)

4: **upon** *gg-deliver*$(S_{1,i}, S_{2,i}, S_{3,i}, S_{4,i})$ **do**
5: **for every** $(p_j, m_j) \in S_{2,i}$ **do invoke** *ba-propose*$_j(1)$
6: **for every** $(p_j, m_j) \in S_{4,i} \setminus S_{1,i}$ **do invoke** *ba-propose*$_j(0)$

7: **upon** *ba-decide*$_j(b_j)$ for every $(p_j, m_j) \in S_{3,i}$ and $\neg$*delivered* **do**
8: *delivered* $\leftarrow$ true
9: **output** *acs-deliver*$(\{(p_j, m_j) \in S_{3,i} : b_j = 1\})$

Processes handle messages for every binary-agreement instance whose identifier they learn, including before their local graded-gather output and after their ACS output.

Theorem 7 (Validated agreement on a core set). *Assume universal kernel depth $d \geq 5$, validated graded gather from Theorem 3, binary agreement $BA[d-1, 0, 3]$ from Corollary 2. Then Algorithm 10 implements $ACS_{\mathsf{V}}[5, 0, 3]$.*

The proof appears in Appendix C.8.

Repeated ACS instances implement state machine replication. In instance r, each process proposes a batch of pending valid requests; after delivering the core set, every replica orders the tagged batches deterministically, removes duplicate requests, and applies the resulting sequence to the deterministic state machine. Agreement gives every correct replica the same log and state, termination lets every process in D_3 advance to instance $r + 1$, and a valid request kept in the proposals of every process in D_3 until it is executed is eventually included by the core inclusion property.

12 Conclusions & Future Work

We construct binary agreement and agreement on a core set in the unknown participants model using local trust and asynchronous authenticated multicast. Encryption keys are generated during execution, so the construction requires no predistributed encryption keys, trusted dealer, or common reference string. The assumptions impose neither a global failure threshold nor a bound on the adversary's share of a resource. Under universal kernel depth five, consistency, agreement, and integrity hold for every correct process, while liveness holds for processes of depth three under the stated invocation premises. Sharing completion for a correct dealer is guaranteed at depth three, whereas commitment and reconstruction of a completed sharing apply independently of the dealer's correctness or depth. In particular, the construction requires no process of infinite depth and hence no trust fixpoint, distinguishing it from prior approaches to local trust in open settings [30].

The main technical contribution is an approximate common coin built from graded gather, verifiable secret sharing, and proxcensus, all based on local trust. Its security relies on a contribution from a correct dealer in the common core and does not depend on the number of identities controlled by the adversary.

Our constructions assume a fixed quorum system. Extending them to allow processes to change their trust assumptions remains open. We show that universal kernel depth five suffices for safety and liveness, but whether this condition is necessary remains open.

References

1. Eduardo Adílio Pelinson Alchieri, Alysson Neves Bessani, Joni da Silva Fraga, and Fabíola Greve. Byzantine consensus with unknown participants. In Theodore P. Baker, Alain Bui, and Sébastien Tixeuil, editors, *Principles of Distributed Systems, 12th International Conference, OPODIS 2008, Luxor, Egypt, December 15-18, 2008. Proceedings*, volume 5401 of *Lecture Notes in Computer Science*, pages 22–40. Springer, 2008. doi:10.1007/978-3-540-92221-6_4.
2. Orestis Alpos, Christian Cachin, Björn Tackmann, and Luca Zanolini. Asymmetric distributed trust. *Distributed Comput.*, 37(3):247–277, 2024.
3. Ignacio Amores-Sesar, Christian Cachin, Simon Holmgaard Kamp, and Juan Villacis. Weaker assumptions for asymmetric trust. In *OPODIS*, volume 361 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2025.

4. Ignacio Amores-Sesar, Christian Cachin, Simon Holmgaard Kamp, and Juan Villacis. Symmetry all the way down. *CoRR*, abs/2607.04887, 2026. URL: `https://doi.org/10.48550/arXiv.2607.04887`, `arXiv:2607.04887`, `doi:10.48550/ARXIV.2607.04887`.
5. Ignacio Amores-Sesar, Christian Cachin, Juan Villacis, and Luca Zanolini. Dag-based consensus with asymmetric trust. In *PODC*, pages 151–161. ACM, 2025.
6. Jonah Brown-Cohen, Arvind Narayanan, Alexandros Psomas, and S. Matthew Weinberg. Formal barriers to longest-chain proof-of-stake protocols. In Anna R. Karlin, Nicole Immorlica, and Ramesh Johari, editors, *Proceedings of the 2019 ACM Conference on Economics and Computation, EC 2019, Phoenix, AZ, USA, June 24-28, 2019*, pages 459–473. ACM, 2019. `doi:10.1145/3328526.3329567`.
7. Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.
8. Christian Cachin, Giuliano Losa, and Luca Zanolini. Quorum systems in permissionless networks. In Eshcar Hillel, Roberto Palmieri, and Etienne Rivière, editors, *26th International Conference on Principles of Distributed Systems, OPODIS 2022, Brussels, Belgium, December 13-15, 2022*, volume 253 of *LIPIcs*, pages 17:1–17:22. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. URL: `https://doi.org/10.4230/LIPIcs.OPODIS.2022.17`, `doi:10.4230/LIPICS.OPODIS.2022.17`.
9. David Cavin, Yoav Sasson, and André Schiper. Consensus with unknown participants or fundamental self-organization. In Ioanis Nikolaidis, Michel Barbeau, and Evangelos Kranakis, editors, *Ad-Hoc, Mobile, and Wireless Networks: Third International Conference, ADHOC-NOW 2004, Vancouver, Canada, July 22-24, 2004. Proceedings*, volume 3158 of *Lecture Notes in Computer Science*, pages 135–148. Springer, 2004. `doi:10.1007/978-3-540-28634-9\_11`.
10. David Cavin, Yoav Sasson, and André Schiper. Reaching agreement with unknown participants in mobile self-organized networks in spite of process crashes. Technical Report IC/2005/026, EPFL - LSR, 2005.
11. Wonseok Choi, Ran Cohen, Juan Garay, Nikos Skoumios, and Vassilis Zikas. Byzantine broadcast with unknown participants. *IACR Cryptol. ePrint Arch.*, 2025:2133, 2025. URL: `https://eprint.iacr.org/2025/2133`.
12. Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In Ian Goldberg and Tyler Moore, editors, *Financial Cryptography and Data Security - 23rd International Conference, FC 2019, Frigate Bay, St. Kitts and Nevis, February 18-22, 2019, Revised Selected Papers*, volume 11598 of *Lecture Notes in Computer Science*, pages 23–41. Springer, 2019. `doi:10.1007/978-3-030-32101-7\_2`.
13. Ivan Damgård, Yvo Desmedt, Matthias Fitzi, and Jesper Buus Nielsen. Secure protocols with asymmetric trust. In *ASIACRYPT*, volume 4833 of *Lecture Notes in Computer Science*, pages 357–375. Springer, 2007.
14. John R. Douceur. The sybil attack. In Peter Druschel, M. Frans Kaashoek, and Antony I. T. Rowstron, editors, *Peer-to-Peer Systems, First International Workshop, IPTPS 2002, Cambridge, MA, USA, March 7-8, 2002, Revised Papers*, volume 2429 of *Lecture Notes in Computer Science*, pages 251–260. Springer, 2002. `doi:10.1007/3-540-45748-8\_24`.
15. Michael J. Fischer, Nancy A. Lynch, and Mike Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, 1985.
16. Matthias Fitzi, Chen-Da Liu-Zhang, and Julian Loss. A new way to achieve round-efficient byzantine agreement. In Avery Miller, Keren Censor-Hillel, and

Janne H. Korhonen, editors, *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 355–362. ACM, 2021. doi:10.1145/3465084.3467907.

17. Juan A. Garay, Aggelos Kiayias, and Nikos Leonardos. The bitcoin backbone protocol: Analysis and applications. In Elisabeth Oswald and Marc Fischlin, editors, *Advances in Cryptology - EUROCRYPT 2015 - 34th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Sofia, Bulgaria, April 26-30, 2015, Proceedings, Part II*, volume 9057 of *Lecture Notes in Computer Science*, pages 281–310. Springer, 2015. doi:10.1007/978-3-662-46803-6_10.
18. Juan A. Garay, Aggelos Kiayias, Rafail M. Ostrovsky, Giorgos Panagiotakos, and Vassilis Zikas. Resource-restricted cryptography: Revisiting MPC bounds in the proof-of-work era. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part II*, volume 12106 of *Lecture Notes in Computer Science*, pages 129–158. Springer, 2020. doi:10.1007/978-3-030-45724-2_5.
19. Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles, Shanghai, China, October 28-31, 2017*, pages 51–68. ACM, 2017. doi:10.1145/3132747.3132757.
20. Hasan Heydari, Robin Vassantlal, and Alysson Bessani. Knowledge connectivity requirements for solving BFT consensus with unknown participants and fault threshold. In *44th IEEE International Conference on Distributed Computing Systems, ICDCS 2024, Jersey City, NJ, USA, July 23-26, 2024*, pages 221–231. IEEE, 2024. doi:10.1109/ICDCS60910.2024.00029.
21. Martin Hirt and Ueli M. Maurer. Player simulation and general adversary structures in perfect multiparty computation. *J. Cryptol.*, 13(1):31–60, 2000.
22. Simon Holmgaard Kamp. A new way to achieve round-efficient asynchronous byzantine agreement. *IACR Cryptol. ePrint Arch.*, page 143, 2025. URL: https://eprint.iacr.org/2025/143.
23. Simon Holmgaard Kamp and Jesper Buus Nielsen. Byzantine agreement decomposed: Honest majority asynchronous atomic broadcast from reliable broadcast. Cryptology ePrint Archive, Paper 2023/1738, 2023. URL: https://eprint.iacr.org/2023/1738.
24. Pankaj Khanchandani and Roger Wattenhofer. Byzantine agreement with unknown participants and failures. In *35th IEEE International Parallel and Distributed Processing Symposium, IPDPS 2021, Portland, OR, USA, May 17-21, 2021*, pages 952–961. IEEE, 2021. doi:10.1109/IPDPS49936.2021.00104.
25. Andrew Lewis-Pye and Tim Roughgarden. Permissionless consensus. *CoRR*, abs/2304.14701, 2023. URL: https://doi.org/10.48550/arXiv.2304.14701, arXiv:2304.14701, doi:10.48550/ARXIV.2304.14701.
26. Xiao Li, Eric Chan, and Mohsen Lesani. Quorum subsumption for heterogeneous quorum systems. In *DISC*, volume 281 of *LIPIcs*, pages 28:1–28:19. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2023.
27. Marta Lokhava, Giuliano Losa, David Mazières, Graydon Hoare, Nicolas Barry, Eli Gafni, Jonathan Jove, Rafal Malinowsky, and Jed McCaleb. Fast and secure global payments with stellar. In *SOSP*, pages 80–96. ACM, 2019.
28. Giuliano Losa, Eli Gafni, and David Mazières. Stellar consensus by instantiation. In *DISC*, volume 146 of *LIPIcs*, pages 27:1–27:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

29. Dahlia Malkhi and Michael K. Reiter. Byzantine quorum systems. *Distributed Comput.*, 11(4):203–213, 1998.
30. David Mazieres. The stellar consensus protocol: A federated model for internet-level consensus. *Stellar Development Foundation*, 32:1–45, 2015. Available online from `https://stellar.org/learn/stellar-consensus-protocol`.
31. Andrew Miller, Ari Juels, Elaine Shi, Bryan Parno, and Jonathan Katz. Permacoin: Repurposing bitcoin work for data preservation. In *2014 IEEE Symposium on Security and Privacy, SP 2014, Berkeley, CA, USA, May 18-21, 2014*, pages 475–490. IEEE Computer Society, 2014. `doi:10.1109/SP.2014.37`.
32. Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, October 2008. White paper. URL: `https://bitcoin.org/bitcoin.pdf`.
33. Marshall C. Pease, Robert E. Shostak, and Leslie Lamport. Reaching agreement in the presence of faults. *J. ACM*, 27(2):228–234, 1980. `doi:10.1145/322186.322188`.
34. David Schwartz, Noah Youngs, and Arthur Britto. The Ripple protocol consensus algorithm. Ripple Labs Inc., 2014. Available at `https://ripple.com/files/ripple_consensus_whitepaper.pdf`.
35. Alberto Sonnino and George Danezis. Sybilquorum: Open distributed ledgers through trust networks. *CoRR*, abs/1906.12237, 2019. URL: `http://arxiv.org/abs/1906.12237`, `arXiv:1906.12237`.
36. Christian Stoll, Lena Klaaßen, and Ulrich Gallersdörfer. The carbon footprint of bitcoin. *Joule*, 3(7):1647–1661, 2019.

A Feasibility of Asynchronous Consensus with Unknown Participants under Generalized Assumptions

Khanchandani and Wattenhofer [24] show that asynchronous consensus is impossible when processes initially know only their own identifiers, even with probabilistic termination. Their partition argument separates two groups that cannot distinguish delayed communication from the absence of the other group. Our protocols begin with local quorum information and give their guarantees under consistency and depth assumptions. We show that the impossibility persists when two disjoint quorum-closed sets are each guaranteed to make progress while outsiders remain silent, subject to the quantitative requirement defined below. The consistency assumption used by our construction ensures that no such pair exists in a fault-free execution.

For this lower bound, fix a participant set $\mathcal{P}$ and arbitrary local quorum families $\mathbb{Q} = [\mathcal{Q}_1, \ldots, \mathcal{Q}_n]$, without assuming consistency. Consensus requires agreement and unanimous validity in fault-free executions, except with negligible probability: no two processes decide different bits, and if all processes propose the same bit, every decision equals that bit. Inputs are supplied locally after an input-independent initialization. All executions have the same participants and the same joint distribution of initial information. As λ varies, the participant indices, quorum incidences, and sets considered below are deterministic polynomial-time computable functions of λ. Initialization may sample identifiers and keys while preserving these incidences.

A nonempty set $S \subseteq \mathcal{P}$ is *quorum-closed* if, for every $p_i \in S$, there exists $Q_i \in \mathcal{Q}_i$ with $Q_i \subseteq S$.

Definition 17 (Quorum-closed termination). *A protocol satisfies* quorum-closed termination *if, for every such set S and every input assignment, all processes in S eventually decide when they are correct and all outsiders are faulty and silent, except with negligible probability.*

For computational security, we also require the following quantitative bound. For each S and $p_i \in S$, a uniform polynomial-time procedure given λ, the participant indices, quorum incidences, S, and p_i specifies an internal scheduling strategy and a deterministic polynomial cutoff $t_{S,i}(\lambda)$ so that, for every input assignment, p_i decides within the cutoff with probability at least $3/4$ over initialization and protocol randomness. The strategy schedules only local actions of members of S and message deliveries between them, using only the public initialization information and the scheduling history within S. Through the cutoff, the scheduler runs online in polynomial time, and protocol computation and communication are polynomially bounded on every random tape. The polynomial bound does not follow from eventual termination.

Lemma 2. *If $\mathbb{Q}$ has two quorum-closed sets X, Y that are disjoint for infinitely many security parameters, then no protocol satisfying Definition 17 and the quantitative bound above solves binary consensus in the asynchronous unknown participants model with negligible validity and agreement error, even when the execution violating agreement is fault-free.*

Proof. Fix a security parameter for which $X \cap Y = \emptyset$, and choose $p_i \in X$ and $p_j \in Y$ with the smallest indices in their respective sets. Let $\varepsilon_v(\lambda)$ bound the probability of violating unanimous validity in a fault-free execution. The quantitative bound gives internal scheduling strategies for X and Y, with cutoffs t_X and t_Y for these representatives.

Consider a fault-free execution in which every process proposes 0. Run the internal strategy for X for t_X steps, postponing all other actions and deliveries. Through the cutoff, the processes in X have the same distribution of local histories as when all outsiders are faulty and silent, since inputs and initialization are unchanged. The quantitative bound and unanimous validity therefore give

$$\Pr[p_i \text{ decides } 0 \text{ by step } t_X] \geq \frac{3}{4} - \varepsilon_v(\lambda).$$

Similarly, a fault-free all-one execution that runs the internal strategy for Y for t_Y steps gives

$$\Pr[p_j \text{ decides } 1 \text{ by step } t_Y] \geq \frac{3}{4} - \varepsilon_v(\lambda).$$

Now consider a fault-free execution in which processes in X propose 0, processes in Y propose 1, and all remaining processes propose an arbitrary bit. Run the internal strategies for X and Y successively for t_X and t_Y steps, scheduling only local actions of the current group and message deliveries within it. During its prefix, each group has the same distribution of local histories as in its

unanimous execution, because its inputs and initialization are unchanged and it receives no outside messages. The union bound therefore gives

$$\Pr[p_i \text{ decides } 0 \ \wedge \ p_j \text{ decides } 1] \geq \frac{1}{2} - 2\varepsilon_v(\lambda).$$

The combined prefix has polynomial computation and communication and is generated by a polynomial-time scheduler. After the fixed cutoffs, resume all processes and deliver pending and subsequent messages. If the sets are disjoint for infinitely many security parameters, a single scheduling strategy tests disjointness and applies this construction at those parameters, using an ordinary fair schedule otherwise. For negligible ε_v, its disagreement probability is nonnegligible, contradicting agreement.

Lemma 3. *If $\mathbb{Q}$ satisfies consistency for F, any two quorum-closed sets that each contain a correct process intersect. In particular, in a fault-free execution, the hypothesis of Lemma 2 cannot hold.*

Proof. Let X, Y be quorum-closed and choose correct processes $p_i \in X$ and $p_j \in Y$. There are quorums $Q_i \in \mathcal{Q}_i$ and $Q_j \in \mathcal{Q}_j$ with $Q_i \subseteq X$ and $Q_j \subseteq Y$. Both owners have depth at least 0, so the first consistency condition gives $Q_i \cap Q_j \neq \emptyset$ and hence $X \cap Y \neq \emptyset$.

For a quorum-closed set S with $F = \mathcal{P} \setminus S$, consistency also makes S a kernel for every correct process. Indeed, each quorum of an owner in S intersects that owner's quorum contained in S. Since all members of S have infinite depth, the universal kernel depth is at least d for every finite d. Under the remaining assumptions of Corollary 2, our construction therefore provides eventual progress in this execution.

B Comparison with Previous Work

	Participant set	Timing model	Sybil assumption	Liveness assumption	Problem solved
This work	A†	Asynchrony	None	Universal kernel depth 5	BA, ACS, SMR
Alchieri *et al.* [1]	A†	Asynchrony	Bounded	-	BA
Heydari *et al.* [20]	A†	Partial synchrony	Bounded	-	BA
Khanchandani and Wattenhofer [24]	B†	Synchrony	Bounded	-	BA
Choi *et al.* [11]	B†	Synchrony	$f < n$	-	BB
SCP [30]	C†	Partial synchrony	None	Intact set (Depth ∞)	SMR
Alpos *et al.* [2]	D†	Asynchrony	Permissioned	Guild (Depth ∞)	BA
Amores-Sesar *et al.* [3]	D†	Asynchrony	Permissioned	Depth 10	BA

Table 1. Comparison of different protocols related to our work. †The participant set is one of: **A**, unknown participants with partial knowledge, where each process initially knows some of the others; **B**, unknown participants with no knowledge, where each process initially knows only its own identifier; **C**, permissionless with partial knowledge, where the participant set may in addition change over time; **D**, known participants, where the participant set is fixed and globally known. *BA* denotes Byzantine agreement, *BB* denotes Byzantine broadcast, *ACS* agreement on a core set, and *SMR* state machine replication.

C Deferred Proofs

C.1 Reliable Consensus and Validated Reliable Broadcast

Theorem 1. *Algorithm 1 implements $RC[d, 0, 3]$, assuming a universal kernel depth $d \geq 4$.*

Proof. **Integrity:** A correct process rc-delivers only while *delivered* = false and sets *delivered* ← true upon delivery, so it rc-delivers at most once.

Validity: If an honest party P outputs m, then it received $[\text{READY}, m]$ from one of its quorums. Since we assume a universal kernel depth of at least $d \geq 4$, all honest quorums include parties with depth d. So, in particular a party P' of depth d sent $[\text{READY}, m]$ which by Lemma 4 implies that some party P'' with depth d sent $[\text{ECHO}, m]$. P'' follows the protocol and would only send $[\text{ECHO}, m]$ when they rc-propose m.

Consistency: If an honest party P outputs m, then it received $[\text{READY}, m]$ from one of its quorums. Since we assume a universal kernel depth of at least $d \geq 4$, all honest quorums include parties with depth 4. Let P'' be such a depth 4

party who sent $[\text{READY}, m]$ to P. Applying Lemma 4 we get that some party with depth at least 3 (either P'' itself or some other party with depth at least $4-1=3$) received $[\text{ECHO}, m]$ from one of their quorums. Applying the same argument for any honest party P' who outputs m', again some party with depth 3 received $[\text{ECHO}, m']$ from one of their quorums. Now, since two parties with depth 1 each received $[\text{ECHO}, \cdot]$ messages from a quorum, we may apply Lemma 5 to see that the intersection of these include an honest party. Honest parties send at most one message of the form $[\text{ECHO}, \cdot]$, thus $m = m'$.

Totality 1: Assume all parties with depth 3 rc-propose the same message m. This implies that all parties with depth 4 receive $[\text{ECHO}, m]$ from one of their quorums, and thus send $[\text{READY}, m]$. Since we assume universal kernel depth 4, this means that all honest parties receive $[\text{READY}, m]$ from one of their kernels, prompting them to send $[\text{READY-AMPLIFY}, m]$. Thus, all parties with depth 1 receive $[\text{READY-AMPLIFY}, m]$ from one of their quorums, prompting them to send $[\text{READY}, m]$. Thus, all parties with depth 2 receive $[\text{READY}, m]$ from one of their quorums, prompting them to rc-deliver m.

Totality 2: If a party with depth 1 outputs m, then it received $[\text{READY}, m]$ from one of its quorums. By Lemma 5, the honest parties in this quorum contains a kernel for all parties with depth 1. Thus, all parties with depth 1 will receive $[\text{READY}, m]$ from a kernel and send $[\text{READY-AMPLIFY}, m]$. This means that all parties with depth 2 will receive $[\text{READY-AMPLIFY}, m]$ from a quorum and send $[\text{READY}, m]$. Thus, all parties with depth 3 will receive $[\text{READY}, m]$ from one of their quorums, prompting them to rc-deliver m.

Lemma 4. *Assume universal kernel depth $d \geq 1$. If an honest party P sends $[\text{READY}, m]$, then either P itself or some other party with depth $d-1$ received $[\text{ECHO}, m]$ from one of their (own) quorums, and in particular: a party with depth d sent $[\text{ECHO}, m]$.*

Proof. An honest party P may send $[\text{READY}, m]$ because it received $[\text{ECHO}, m]$ from a quorum, or because it received $[\text{READY-AMPLIFY}, m]$ from a quorum. In the former case, the quorum intersects the universal kernel and thus includes a party with depth d.

In the latter case, consider the quorum of parties who sent $[\text{READY-AMPLIFY}, m]$ to P. This quorum again contains a party P'' from the universal kernel. As party P'' has depth $d \geq 1$, one of its quorums is comprised of parties with depth $d-1$, and thus all of its kernels include at least one party with depth $d-1$. Since P'' is honest and would only send $[\text{READY-AMPLIFY}, m]$ upon receiving $[\text{READY}, m]$ from a kernel, this means that some party P''' with depth $d-1$ must have already sent $[\text{READY}, m]$. Now, either P''' did so because it received $[\text{ECHO}, m]$ from a quorum, or we can keep applying the same logic to find another depth $d-1$ party who sent $[\text{READY}, m]$ until we find the first party with depth $d-1$ who sent $[\text{READY}, m]$. Each repetition moves to a strictly earlier send event in a finite execution prefix, so this first send must have been caused by a quorum of echo messages.

Lemma 5. *Consider an execution with faulty processes F, a process p_i such that depth$(p_i) \geq 1$ and one of its quorums $Q_i \in \mathcal{Q}_i$. For any other process p_j with depth$(p_j) \geq 1, \exists K_j \in \mathcal{K}_j$ such that $K_j \subseteq Q_i \setminus F$.*

Proof. Fix any quorum $Q_j \in \mathcal{Q}_j$. Because both quorum owners have depth at least 1, the second consistency condition of Definition 5 gives a correct process in $Q_i \cap Q_j$. Thus $Q_i \setminus F$ intersects every quorum of p_j and is itself a kernel for p_j.

Corollary 1. *Algorithm 2 implements $VRB_{\mathsf{V}}[d, 0, 3]$, assuming a universal kernel depth $d \geq 4$.*

Proof. If the sender is correct, it sends only the value with which it invoked *vrb-broadcast*, and reliable-consensus validity therefore implies validated reliable-broadcast validity. Reliable-consensus consistency and integrity give the corresponding validated reliable-broadcast properties. For Totality 1, every process in D_3 eventually receives the correct sender's value, passes the predicate, and proposes that value, so reliable-consensus Totality 1 applies. Reliable-consensus Totality 2 gives Totality 2. For external validity, reliable-consensus validity implies that any delivered m was proposed by a process p_j of depth at least d, which by line 7 occurs only after $\mathsf{V}_j(m)$ holds.

C.2 Proxcensus

Lemma 6 (Initial Justified Votes). *Assume universal kernel depth $d \geq 4$. If a vote v is justified for round 1 in the view of a correct process, then v was previously prox-proposed by a party with depth $d - 1$.*

Proof. If a vote v is justified for round 1 in the view of an honest party, then $[\textsc{justify}, v]$ was rc-delivered. By validity of the RC protocol, this means that $[\textsc{justify}, v]$ was rc-proposed by a party with depth d. An honest party only rc-proposes $[\textsc{justify}, v]$ if it received $[\textsc{input}, v]$ from one of its kernels. Since a party with depth d has a quorum exclusively made up of parties with depth $d - 1$, any kernel that could have sent it such a message includes at least one party with depth $d - 1$. Since $d - 1 \geq 4 - 1 \geq 0$, that party follows the protocol and must have prox-proposed v in order to send $[\textsc{input}, v]$.

Lemma 7 (Justified Graded Agreement). *If two votes v and v' are justified for round r in the view of correct processes p_i and p_j, respectively, then $|v - v'| \leq 1$.*

Proof. By Lemma 6, every vote justified in round 1 was prox-proposed and is therefore in $\{0, 1\}$, so the claim holds for round 1. Assume the claim for round r. All votes justified in that round are contained in two consecutive integers, so at most one has each parity. Each correct process handles each justified vote once and therefore proposes at most once to each secondary instance. A process proposes to the primary instance only when $handled[r] = \emptyset$ and then adds its vote

to this set, so it proposes at most once in that instance. Thus reliable-consensus validity and consistency apply to these instances. Let x and y be justified in round $r+1$. If both are derived by doubling an output of the single primary reliable consensus instance, consistency of reliable consensus gives $x = y$. If both are derived by summation, consistency of each secondary reliable consensus instance gives $x = y$. Otherwise, suppose without loss of generality that $x = 2a$ and $y = e + o$, where e and o come from the even and odd secondary instances. Reliable consensus validity and lines 10–14 imply that a, e, and o were justified in round r by honest parties. The induction hypothesis gives pairwise distance at most one, and the opposite parities of e and o give $|e - o| = 1$. Hence $a \in \{e, o\}$ and $|x - y| = |2a - (e + o)| = 1$.

Lemma 8 (Justified Validity). *Assume a universal kernel depth of $d \geq 4$. Fix an execution prefix in which no process in D_{d-1} has prox-proposed $v \in \{0, 1\}$. The only vote that may be justified in round r at a correct process in this prefix is $(1 - v)2^{r-1}$.*

Proof. In round 1, the claim follows from Lemma 6: justifying v would require a prior proposal of v by a process in D_{d-1}.

For the inductive step, assume the claim for round r and let x be justified in round $r+1$. If $x = 2a$ at line 17, reliable consensus validity and lines 10–13 imply that a was justified in round r by an honest party, so $x = (1-v)2^r$ by induction. Otherwise, line 19 gives $x = a + a'$ for values from the two secondary instances. Reliable consensus validity and lines 10–14 imply that both were justified in round r by honest parties, so induction makes them equal, contradicting their opposite parities.

Theorem 2 (Proxcensus). *Assume universal kernel depth $d \geq 4$ and let $M = 2^{k-1}$. If every tagged reliable consensus instance implements $RC[d, 0, 3]$, then Algorithm 3 implements $\text{Prox}[d - 1, 0, 3]$ with outputs in $\{0, \ldots, M\}$.*

Proof. Every output is justified in round k, and induction over the two derivation rules shows that every vote justified in round r lies in $\{0, \ldots, 2^{r-1}\}$. Graded agreement follows from Lemma 7. For validity, take the execution prefix ending at an output. If no process in D_{d-1} previously proposed $1 - b$, then Lemma 8 makes that output bM.

After its first output, a process has $\mathit{handled}[k] \neq \emptyset$, so it cannot output again.

For liveness, suppose that the correct processes eventually proposing some bit b form a kernel for every process in D_3. These correct proposers eventually send their input messages to every process. Every process in D_3 therefore rc-proposes $[\textsc{justify}, b]$, so reliable consensus Totality 1 makes all of them justify b in round 1. Reliable consensus Totality 2 ensures that any vote justified in a round by one depth-3 process is eventually justified by all depth-3 processes. By Lemma 7, the votes justified at processes in D_3 in a round are either one value x or two consecutive values x and $x + 1$. In the first case all depth-3 processes propose the same primary vote, so reliable consensus Totality 1 justifies $2x$ in the next round. In the second case they propose both secondary votes, so reliable

consensus Totality 1 justifies $2x+1$ in the next round. Induction therefore gives every depth-3 process a justified vote in round k, after which line 23 makes it prox-deliver.

C.3 Announce and Eventually Consistent Quorum Systems

Lemma 1 (Announce). *Assume the open quorum consistency conditions of Definition 5 and universal kernel depth $d \geq 4$. Then Algorithm 4 implements an eventually consistent quorum system satisfying Definition 10.*

Proof. Monotonicity follows because there is one validated reliable broadcast instance per sender and the code installs its output only once. Validated reliable broadcast validity gives validity, and validated reliable broadcast consistency gives consistency. Validated reliable broadcast Totality 2 gives eventual consistency.

For correct-announcement liveness, let p_j be correct and let $p_i \in D_3$. Open quorum consistency gives $R \in \mathcal{K}_i$ for every $R \in \mathcal{Q}_j$, so $\mathsf{AnnounceValid}_i(p_j, \mathcal{Q}_j)$ holds. Validated reliable broadcast Totality 1 therefore delivers p_j's announcement throughout D_3.

For certified depth, consider $R \in \widehat{\mathbb{Q}}_i[p_j]$ at a process $p_i \in D_3$. External validity gives a process $p_k \in D_d$ at which $\mathsf{AnnounceValid}_k(p_j, \widehat{\mathcal{Q}}_j)$ held. By depth, p_k has a quorum $Q_k \in \mathcal{Q}_k$ with $Q_k \subseteq D_{d-1}$. Since $R \in \mathcal{K}_k$, it intersects Q_k and therefore contains a process in D_{d-1}.

C.4 Validated Graded Gather

In the following proofs, a delivered message is a message delivered by validated reliable broadcast at a correct process.

Lemma 9 (Depth of a referenced certified quorum). *Let R be a certified quorum of p_j referenced by a message delivered at a correct process. Then R contains a process $p_k \in D_4$, and the announce entry of p_j is eventually defined in the eventually consistent quorum system of every process in D_3.*

Proof. External validity of the message referencing R gives a process $p_i \in D_5$ that passed its predicate and therefore had $\widehat{\mathbb{Q}}_i[p_j]$ defined with $R \in \widehat{\mathbb{Q}}_i[p_j]$. Certified depth from Lemma 1 implies that the referenced quorum contains a process in D_4. Eventual consistency from Lemma 1 propagates the entry throughout D_3.

Lemma 10 (Descent by one collection level). *Let $2 \leq r \leq L$ and consider the payloads T and T' of any two delivered level-r relay messages. There is a delivered level-$(r-1)$ relay payload T_k such that $T_k \subseteq T \cap T'$.*

Proof. Each level-r relay copies a delivered aggregation message. By Lemma 9, the supporting certified quorums of these aggregations contain processes $p_i, p_j \in D_4$. The first aggregation therefore references the level-$(r-1)$ collection message of p_i, and the second references the corresponding collection message of p_j.

The processes p_i and p_j are correct, so their collection messages use their own quorums $Q_i \in \mathcal{Q}_i$ and $Q_j \in \mathcal{Q}_j$. Open quorum consistency gives a correct process $p_k \in Q_i \cap Q_j$. Both collection messages reference the same level-$(r-1)$ relay message of p_k. Its payload T_k is included in both unions, so $T_k \subseteq T \cap T'$.

Lemma 11 (Common core). *If a level-L relay is delivered at a correct process, there are a process $p_i \in D_4$, a quorum $Q_i \in \mathcal{Q}_i$, and a level-two relay payload S_0 such that S_0 contains a tagged proposal value for every member of Q_i and is contained in every level-L relay payload delivered at a correct process. Moreover, S_0 contains a value proposed by a process in D_5.*

Proof. Consider all level-L relay messages delivered at correct processes in the execution. Validated reliable broadcast consistency fixes at most one such message per sender, so their number is between one and n. Pair their payloads and apply Lemma 10 to each pair, pairing any unpaired payload with itself. Each resulting payload is contained in the original payloads represented by its pair, and the number of payloads decreases from m to $\lceil m/2 \rceil$. Continue pairing, including self-pairing when only one payload remains, for $L - 2 = \lceil \log_2 N \rceil$ steps. Since $n \leq N$, at most one payload remains after these steps. This produces one level-two relay payload S_0 contained in every original level-L payload.

The aggregation copied by this level-two relay has a certified supporting quorum containing some process $p_i \in D_4$ by Lemma 9. It references the correct collection message of p_i at level 1. That collection uses a quorum $Q_i \in \mathcal{Q}_i$ and contains the tagged proposal value of every member of Q_i. These values are therefore all contained in S_0.

Let $U \subseteq D_5$ be a universal kernel. It intersects Q_i because p_i is correct. Choose $p_j \in U \cap Q_i$. The process p_j is correct, so its message at level 1 contains the value with which it invoked *gg-propose*. Thus S_0 contains a value proposed by a process in D_5.

Lemma 12 (Candidate containment). *For any two delivered candidate relays with payloads (A, B) and (A', B'), it holds that $A \subseteq B'$.*

Proof. The supporting certified quorums of the two candidate aggregations contain processes $p_i, p_j \in D_4$. The aggregations reference the correct candidate collection messages of p_i and p_j, in turn, their own supporting quorums intersect in a correct process p_k. Both candidate collections therefore reference the same level-L relay message of p_k. Writing T_k for its payload, the definitions give

$$A \subseteq A_i^{\mathrm{col}} \subseteq T_k \subseteq B_j^{\mathrm{col}} \subseteq B'.$$

Lemma 13 (Four-layer boundedness). *Let $\mathbf{S} = (S_1, S_2, S_3, S_4)$ and $\mathbf{S}' = (S_1', S_2', S_3', S_4')$ be outputs delivered by correct processes. Then $S_r \subseteq S_{r+1}'$ for every $r \in \{1, 2, 3\}$.*

Proof. The certified quorums referenced by the two output aggregations contain processes $p_i, p_j \in D_4$. Since the processes are honest, their output collection

messages use quorums that intersect in a correct process p_k. Both output collections therefore reference the same candidate relay of p_k with payload (A_k, B_k). The set definitions give

$$S_1 \subseteq S_{1,i}^{\mathrm{col}} \subseteq A_k \subseteq S_{2,j}^{\mathrm{col}} \subseteq S_2' \quad \text{and} \quad S_3 \subseteq S_{3,i}^{\mathrm{col}} \subseteq B_k \subseteq S_{4,j}^{\mathrm{col}} \subseteq S_4'.$$

Every candidate inner set used to form S_2 is contained in every candidate outer set used to form S_3' by Lemma 12. Taking the union of the former sets and the intersection of the latter sets gives $S_2 \subseteq S_3'$.

Lemma 14 (Validity and agreement). *The protocol satisfies validity, external validity, and agreement.*

Proof. External validity ensures that every reference in a message delivered at a correct process was itself delivered at a correct process that checked the corresponding structural predicate. Following the prescribed order of references therefore traces every tagged value in a delivered payload to a proposal at level 1 delivered at a correct process. If a proposal sender is correct, validated reliable broadcast validity implies that it previously invoked *gg-propose* with that value. External validity of the proposal message gives a process in D_5 at which V held. The singleton proposal format binds each tagged value to its validated reliable broadcast sender. Validated reliable broadcast consistency therefore prevents two delivered proposals from attributing different values to the same sender.

Lemma 15 (Progress at constant depth). *Assume that every process in D_3 invokes gg-propose and that every input x_j proposed by a process $p_j \in D_3$ eventually satisfies $\mathsf{V}_k(p_j, x_j)$ at every $p_k \in D_3$. Then every process in D_3 eventually delivers an output.*

Proof. Correct-announcement liveness from Lemma 1 makes the quorum system of every process in D_3 available throughout D_3. For every $p_j \in D_3$, the validity predicate for its proposal at level 1 eventually holds at every process in D_3 by hypothesis. Since p_j broadcasts this proposal, validated reliable broadcast Totality 1 ensures that every process in D_3 eventually delivers it.

Whenever a process in D_3 broadcasts a collection, aggregation, or relay message, it has already delivered every message it references. By validated reliable broadcast Totality 2, every process in D_3 eventually delivers each referenced message, even if its sender lies outside D_3 or is faulty. By validated reliable broadcast consistency, these processes deliver the same payload for each referenced message. By announce eventual consistency, every process in D_3 also eventually delivers the referenced announce entries. By correct-announcement liveness, each of these processes eventually delivers the sender's own announce entry. Thus the structural predicate for this message eventually holds at every process in D_3, and validated reliable broadcast Totality 1 ensures that each of them eventually delivers it.

By depth, each $p_i \in D_4$ has a quorum $Q_i \in \mathcal{Q}_i$ with $Q_i \subseteq D_3$. As shown above, every process in D_3 eventually delivers the proposals from all members

of Q_i. Hence every process in D_4 eventually broadcasts a collection at level 1, unless it has already broadcast one using another quorum. Each chosen collection is delivered throughout D_3 by the preceding argument. Every process in D_5 has a quorum contained in D_4, which becomes a certified quorum once its own announce entry is delivered. It therefore eventually broadcasts a level-two aggregation, unless it has already selected another certified quorum. Each chosen aggregation is delivered throughout D_3, so every process in D_3 broadcasts and delivers a level-two relay.

Inductively, level-r relays from every process in D_3 enable level-r collections at every process in D_4, level-$(r+1)$ aggregations at every process in D_5, and level-$(r+1)$ relays at every process in D_3. Every selected message is delivered throughout D_3 by the same propagation argument. After level L, this argument applies to the candidate collections, aggregations, and relays, and then to the output collections and aggregations. Thus every process in D_3 eventually delivers an output.

Theorem 3 (Validated graded gather). *Assume the open quorum consistency conditions of Definition 5, the announce protocol from Lemma 1 instantiated at validity depth five, universal kernel depth five, and a common upper bound $N \geq |\mathcal{P}|$. Then Algorithms 5 and 6 implements $gg^4_{\mathsf{V}}[5, 0, 3]$ using $L = \lceil \log_2 N \rceil + 2$ collection levels. Whenever a correct process delivers, the common core contains a tagged value for every member of a quorum of a process in D_4 and contains an input proposed by a process in D_5.*

Proof. If a correct process delivers an output, following the references in this output leads to at least one level-L relay delivered at a correct process. Lemma 11 gives a set S_0 contained in every such relay payload. All subsequent candidate inner sets and output sets S_1 are intersections of nonempty collections of these payloads, so they contain S_0. The tagged values for correct members of the quorum Q_i from Lemma 11 are their proposed inputs by Lemma 14. Boundedness follows from Lemma 13. Validity, external validity, and agreement follow from Lemma 14. Totality follows from Lemma 15 with $ld = 3$. Each correct process outputs only upon its first delivery of an output aggregation, giving integrity.

C.5 Verifiable Sharing

Lemma 16 (Availability of the completion condition). *Assume the open quorum consistency conditions and universal kernel depth at least five. If a dealer in D_3 starts sharing, then condition (2) eventually holds for some C, R, independently of whether Byzantine processes distribute keys.*

Proof. Choose a live quorum $C^* \in \mathcal{Q}_D$ consisting of processes of depth at least two. All its members are correct, so their announce entries eventually reach the dealer. For every $p_j \in C^*$, $Q \in \mathcal{Q}_j$, and $H \in \mathcal{Q}_D$, $Q \cap H$ contains a correct process by the second consistency clause of Definition 5, since both quorum owners have depth at least one. The dealer eventually requests that process's session key, and

its key broadcast eventually reaches the dealer by VRB Totality 1. There are finitely many indicated pairs of quorums. Consequently, the dealer's available-key set eventually contains a correct member of every such intersection. At that point, $Q \cap R$ intersects every $H \in \mathcal{Q}_D$ for every $p_j \in C^*$ and $Q \in \mathcal{Q}_j$. Thus C^* and the accumulated set R satisfy the condition.

Lemma 17 (Sharing liveness and totality). *Under the assumptions of Lemma 16, a dealer in D_3 that starts sharing eventually completes sharing at every process in D_3. If any process in D_3 completes a sharing, every process in D_3 eventually completes the same sharing.*

Proof. A dealer in D_3 eventually fixes a transcript by the preceding lemma. Every referenced announce entry and key broadcast was delivered at the depth-three dealer. Their totality properties therefore make these references available at every process in D_3, even when a referenced sender is Byzantine. Quorum consistency gives $C \in \mathcal{K}_i$ at every such correct verifier p_i. All remaining structural checks hold by construction, and completeness of the sharing argument eventually makes the local validity predicate hold. VRB Totality 1 completes the honest dealer's sharing everywhere in D_3. For any dealer, VRB Totality 2 propagates a sharing already completed at a process in D_3.

Lemma 18 (Commitment and reconstruction with adversarial keys). *Assume universal kernel depth $d \geq 5$ and VRB validity depth d. Except with negligible probability, a transcript completed by a correct process determines a unique secret, and every reconstruction delivered by a correct process equals that secret. If some process in D_3 completes sharing and every process in D_3 invokes reconstruction after local completion, every process in D_3 eventually reconstructs.*

Proof. Sharing consistency follows from VRB consistency. Fix any transcript completed by a correct process. By external validity, some verifier $p_i \in D_d$ accepted the sharing transcript. Choose a quorum $H \in \mathcal{Q}_i$ consisting of processes in D_{d-1}. Since $C \in \mathcal{K}_i$, there is a process $p_j \in C \cap H$. This correct process has a live quorum $Q \in \mathcal{Q}_j$ consisting of processes in D_{d-2}. Announce validity identifies this Q with a quorum in the transcript's advertised system for p_j. The set $A_{j,Q} = Q \cap R$ is nonempty by the structural checks and consists entirely of processes in D_3. Every key used for these recipients is honestly generated and correctly attributed by key-broadcast validity.

Sharing soundness gives a witness for the additive-sharing equations. Perfect correctness and unique decryption fix every share in this all-correct recipient set, and hence fix the common secret across every possible sharing witness. Now consider any recipient set from which a correct process reconstructs. Opening soundness gives a key-generation witness for every accepted opening in that set. Thus each of those keys is in the image of KeyGen, even if its owner is Byzantine. Perfect correctness and unique decryption make each accepted value equal to the corresponding plaintext in the sharing witness. Their sum therefore equals the committed secret. For a correct dealer, its actual sharing witness proves that the committed secret is its input.

For liveness, the depth-d validator accepted the local validity predicate only after delivering the referenced announce entries and key broadcasts. Their totality properties make these references available throughout D_3. Since a process in D_3 completed sharing, sharing totality makes the committed transcript available to every member of the all-correct set $A_{j,Q}$ and to every recipient in D_3. All members of that set belong to D_3, so they eventually invoke reconstruction under the stated premise. They know their stored key-generation randomness and can prove their openings to every correct recipient. Reliable delivery and completeness eventually give every process in D_3 accepted openings from all members of $A_{j,Q}$.

Theorem 4. *Assume the encryption and concurrent zero-knowledge argument primitives specified in Section 8.1. Assume also the announce and VRB constructions from Lemma 1 and Corollary 1. With universal kernel depth $d \geq 5$, Algorithm 7 satisfies all properties of Definition 13, including reconstruction liveness.*

Proof. Soundness applies to the interleaved sharing and opening arguments accepted by correct verifiers. Combine Lemmas 17 and 18.

C.6 Approximate Common Coin

For analysis, extend the local weight function by setting $g_i[D] = 0$ whenever $p_D \notin S_{3,i}$. Let $\mathcal{U}$ contain all dealers that occur in a set S_3 delivered by a correct process during the execution. Let n_{coin} count the dealers in $\mathcal{U}$ assigned positive weight by at least one correct process with frozen weights. In the polynomially bounded executions considered here, every identifier in $\mathcal{U}$ occurs in a delivered protocol message processed by PPT parties, so $n_{\mathsf{coin}} \leq |\mathcal{U}|$ is polynomial in λ. Consequently, $n_{\mathsf{coin}} \leq 2^{\lambda}$ for all sufficiently large λ.

Lemma 19 (Nearby weights). *For every $p_D \in \mathcal{U}$ and every two correct processes p_i, p_j that have frozen their weights at line 18, $|g_i[D] - g_j[D]| \leq 1$. Moreover, $g_i[D] = g_j[D] = 0$ whenever p_D lies outside both sets $S_{3,i}$ and $S_{3,j}$.*

Proof. If $p_D \in S_{3,i} \cap S_{3,j}$, the claim is graded agreement of the corresponding proxcensus instance. Suppose $p_D \in S_{3,i} \setminus S_{3,j}$. Boundedness from Lemma 13 implies that p_D belongs to no correct process's set S_2, since every such set is contained in $S_{3,j}$. Thus no correct process proposes 1 in this instance, and proxcensus validity forces every correct proxcensus output to be 0. The symmetric case is identical, and the final statement follows from the definition of the extended weight function.

Lemma 20 (Proximity). *If correct processes p_i and p_j deliver coin values c_i and c_j, then $\mathsf{dist}_M(c_i, c_j) \leq n_{\mathsf{coin}}$. The same bound holds between every correct output and the reference value c^* determined when the first correct process enables reconstruction. In particular, $\mathsf{dist}_M(c_i, c_j) \leq 2^{\lambda}$ for all sufficiently large λ.*

Proof. VSS commitment implies that both processes use the same representative $\bar{s}_D \in \{0, \ldots, M-1\}$ whenever both sums contain dealer p_D. Only dealers with positive weight at one of the two processes contribute to the difference of their sums. If $n_{\mathsf{coin}} = 0$, both sums are zero. For $n_{\mathsf{coin}} > 0$, the extended zero weights and Lemma 19 give

$$|X_i - X_j| \leq \sum_{p_D \in \mathcal{U}} \frac{|g_i[D] - g_j[D]|}{M} \bar{s}_D < n_{\mathsf{coin}}.$$

Nearest-integer rounding increases the distance by at most one, giving a distance strictly less than $n_{\mathsf{coin}} + 1$ and hence at most n_{coin}, since rounded values are integers. Reduction modulo M cannot increase circular distance. The calculation uses only the frozen weights and the committed secrets, so it also applies to the first process to enable reconstruction even if that process never learns all its secrets. The asymptotic bound follows from $n_{\mathsf{coin}} \leq 2^\lambda$.

Lemma 21 (Correct dealer with weight M). *If a correct process has frozen its weights, some dealer p_{D^*} of depth at least 3 has a VSS instance of weight M at every correct process with frozen weights.*

Proof. Let p_i and $A^* \in \mathcal{Q}_i$ be supplied by Theorem 3. Because p_i has depth at least 4, it has a live quorum $L_i \in \mathcal{Q}_i$ consisting of processes of depth at least 3. Quorum consistency gives a dealer $p_{D^*} \in A^* \cap L_i$. This dealer has depth at least 3, and it sampled its secret uniformly in $\mathbb{Z}_M$ after fixing its recipient description and before proposing to graded gather. Since A^* is contained in every correct set S_1, no correct process proposes 0 in the proxcensus instance of p_{D^*}. Proxcensus validity therefore gives weight M for this instance at every correct process with frozen weights.

Computing other committed contributions. Fix a coin invocation and a dealer p_D, and call its sharing the target sharing. The other dealers' sharings in this invocation are called non-target sharings. An execution algorithm stores every fresh key pair that it generates while emulating an uncorrupted holder in a table. The table is indexed by the invocation, dealer, and holder and stores the public key, secret key, and key-generation randomness. It keeps these entries if the holder is subsequently corrupted. Keys supplied by Byzantine holders need not have corresponding entries.

For a transcript Γ_j in a non-target sharing, consider the following deterministic computation. Enumerate its recipient sets in a fixed order and select the first set for which every referenced public key matches a table entry with the exact context of this sharing and holder. Decrypt the ciphertexts for this set and return the sum of their plaintexts in $\mathbb{Z}_M$. Return a distinguished failure symbol if no such set exists or if a required decryption is not a group element.

Lemma 22 (Recovery from generated recipient keys). *In a real execution, except with negligible probability, the computation above returns the committed secret of every non-target transcript completed by a correct process. The*

computation remains available if the reduction does not know one recipient's secret key for the target sharing.

Proof. By Lemma 18, every completed transcript has a nonempty recipient set consisting entirely of eventually correct processes. Every member of this set generated the referenced fresh key honestly. The table therefore contains a matching entry for every member. All these entries have the non-target dealer's context, so none uses the unknown secret key for the target sharing. The search consequently finds at least one set. Sharing soundness and perfect correctness imply that every set whose keys match the table decrypts to shares summing to the same committed secret, including if some of its holders were later corrupted.

A fixed stopping event. Fix a reference process p_i and one position in the explicit quorum list $\mathcal{Q}_D$. Let L be the quorum at that position, and abort if the position is absent. The quorum systems and authenticated identifiers are fixed by the initialization, which precedes generation of the fresh sharing keys. Quorum choices may depend on the authentication data associated with those identifiers.

Run the execution up to the event immediately before p_i first enables reconstruction in this invocation. Abort before a corruption of any process in $\{p_i, p_D\} \cup L$, before any member of L begins target-sharing reconstruction or decrypts a target ciphertext, or if a target recipient set misses L. Also abort if a process in $\{p_i, p_D\} \cup L$ completes a target transcript different from the dealer's generated transcript. For each target recipient set, select its first holder in L according to a fixed order, and abort if that holder's referenced key does not match the key generated for its exact target-sharing context. An execution that does not reach the stopping event within its polynomial computation bound contributes zero to the event being tested. At the event, require that $p_D \in S_{3,i}$ and $g_i[D] = M$. For every other positive-weight dealer, apply the computation of Lemma 22 to the exact transcript in p_i's local commitment state, and abort on failure. The guard at line 18 ensures that these transcripts are available.

Let Y consist of the public and adversarial view, the reference process's state in the protocol invoking the coin, its sets and weights, and these recovered non-target contributions. The experiment outputs Y if and only if it reaches the stopping event and all the preceding checks succeed. The protocol invoking the coin accesses it through its stated interface and does not read an uncorrupted dealer's fresh secret or sharing state. In particular, an expansion grade fixed before coin-start is included in Y.

Lemma 23 (Direct bound at a fixed event). *For any efficiently testable set $B(Y) \subseteq \mathbb{Z}_M$ with cardinality at most $b(\lambda)$, the experiment satisfies*

$$\Pr[\textit{experiment outputs } Y \ \wedge \ C_i \in B(Y)] \leq \frac{b(\lambda)}{M} + \mathsf{negl}(\lambda),$$

where aborted executions contribute zero and

$$C_i = \mathsf{Round}_M \left(\bar{s}_D + \sum_{p_j \in S_{3,i} \setminus \{p_D\} :\, g_i[j] > 0} \frac{g_i[j]}{M} \bar{s}_j \right).$$

Here $\bar{s}_D$ is the canonical representative of s_D, and each s_j is the output of the specified recovery computation, with canonical representative $\bar{s}_j$.

Proof. The proof first replaces the target dealer's sharing arguments by simulation and then replaces selected target ciphertexts by encryptions of zero. All other proof sessions are executed normally. Their honest prover inputs, randomness, and verifier states are generated by the execution algorithm and can be exposed on corruption.

For each target recipient set $A_{j,Q}$, select the first holder in $A_{j,Q} \cap L$ according to a fixed order. Generate the additive sharing by sampling all other shares uniformly and setting this selected share to the difference between s_D and their sum. This has exactly the honest sharing distribution. The entire target transcript Γ_D is formed before any of its sharing arguments begins.

At that point, view the continuation of the entire system as a single concurrent verifier for the target sharing arguments, all of which have the same fixed NP statement Γ_D. At the stopping event, the verifier can run the non-target recovery computations using its table of generated keys. Consequently, auxiliary-input concurrent zero knowledge replaces all target sharing arguments jointly while preserving the probability of the efficient final event test. An algorithm outside this simulated continuation keeps s_D and uses it with Y to test the event in Lemma 23. The auxiliary-input guarantee applies also to distinguishers with that value.

The continuation given to the black-box simulator omits the target secret, the plaintexts of the shares selected above, the dealer's private encryption randomness, and the portions of its sampling tape that determine these values. These values are not read by this continuation. The target dealer's corruption aborts, no member of L opens the target sharing, and all holders outside L have only independently sampled target shares. The target dealer's simulated sharing arguments do not require its witness. Only this algorithm stores the original secret, which is not given to the simulator or the continuation it rewinds. Future random choices use independent coins, and no state made available to the continuation contains the omitted sampling data.

Now replace the ciphertext of each selected share by an encryption of zero, one ciphertext at a time. For a particular replacement, the reduction embeds its IND-CPA public key as $\mathsf{pk}_{\sigma,D,k}$ for the selected holder p_k. It generates all other fresh key pairs normally. If the selected recipient is not determined until the dealer fixes the recipient sets in its sharing, it guesses the holder's process position before generating these fresh keys. On a mismatch it returns a fixed bit without using the challenge. Since there are only polynomially many positions, a nonnegligible hybrid gap would give a nonnegligible IND-CPA advantage.

The reduction never needs the missing secret key or its key-generation randomness. It aborts before p_k is corrupted or begins target-sharing reconstruction, so it never attempts a decryption with the missing key. Every other sharing uses a different context and fresh keys, whose honest-generation states are known. In particular, all non-target recovery computations remain executable. The key-broadcast protocol requires only the public key and makes no proof of knowledge

of the corresponding secret key. Any unknown encryption randomness of the challenge ciphertext occurs only in the omitted target dealer state and target sharing witness.

For each replacement, the reduction obtains its one ciphertext challenge while constructing Γ_D, before invoking the concurrent simulator. The simulator subsequently runs on an ordinary polynomial-time verifier with a fixed challenge ciphertext and never rewinds an external challenger. Zero knowledge is used only at the first replacement of genuine target proofs, where the statement is true. The same bounded simulator is an efficient computation on later statements even if those statements are false, so the later hybrid steps use only IND-CPA security.

In the final hybrid, all selected ciphertexts encrypt zero, while every other target share and ciphertext was generated independently of s_D. The dealer fixed C, R, the advertised quorum systems, and the referenced key broadcasts before sampling s_D. The simulator's continuation and its auxiliary state contain no remaining dependence on s_D. This includes non-target contributions: they are now computed from the simulated transcripts and the table entries for other dealers, both independent of s_D. Thus Y, every abort decision, and the decision whether to output Y are independent of the original uniform s_D.

Condition on an output Y of the experiment and put

$$Z(Y) = \sum_{p_j \in S_{3,i} \setminus \{p_D\}:\, g_i[j] > 0} \frac{g_i[j]}{M} \bar{s}_j.$$

Rounding commutes with integer addition, so the final test uses $s_D + \mathsf{Round}_M(Z(Y))$ modulo M. Its probability of membership in $B(Y)$ is at most $b(\lambda)/M$. Multiplying by the probability that the experiment outputs a value preserves this bound. The polynomially many encryption hybrids and the concurrent zero-knowledge replacement contribute a negligible total difference.

Theorem 8 (Direct bound with protocol-generated encryption keys). *Assume the open quorum consistency conditions, universal kernel depth at least five, and the cryptographic requirements of Section 8.1, including IND-CPA security for honestly generated encryption keys. Use Algorithm 8 with the sharing construction of Section 8, graded gather from Theorem 3, and proxcensus from Theorem 2. Let $q(\lambda)$ be a fixed polynomial upper bound on $|\mathcal{P}| \sum_{p_D \in \mathcal{P}} |\mathcal{Q}_D|$ over the experiment's possible initializations. Suppose each process fixes a value x_i in the protocol invoking the coin before starting the coin. Let E be the event that a correct process enables reconstruction, and let x_* and c^* be the fixed value and reference coin at the first such event. Then*

$$\Pr[E \ \wedge \ c^* \in B_\Delta(x_*)] \leq q(\lambda) \frac{4\Delta + 4}{M} + \mathsf{negl}(\lambda).$$

The bound is negligible for $M = 2^{2\lambda}$ and $\Delta = 2^\lambda$.

Proof. Exclude the negligible event of a false sharing or opening argument accepted by an eventually correct verifier in the real execution. By Lemma 21,

there is an eventually correct dealer $p_D \in D_3$ whose weight is M at the first correct reconstruction event at p_i. Choose a live quorum $L \in \mathcal{Q}_D$. Every target recipient set intersects L by the completion condition. The fixed experiment for these indices does not abort on corruption, a target opening by L, or a different target transcript before this event. Lemma 22 ensures that every other positive-weight contribution is recovered correctly. For every execution in which E holds and $c^* \in B_\Delta(x_*)$, there is a choice of reference process, dealer, and quorum-list position for which the fixed experiment outputs Y and $C_i \in B_\Delta(x_i)$. Apply Lemma 23 with $B(Y) = B_\Delta(x_i)$ and $b(\lambda) = 4\Delta + 4$, and take a union bound over at most $q(\lambda)$ choices.

Lemma 24 (Termination). *Suppose every process in D_3 invokes coin-start. Then every process of depth at least 3 eventually delivers a coin value.*

Proof. Every dealer in D_3 eventually completes its sharing phase and proposes shared to graded gather by VSS sharing liveness. Every proposal by a correct dealer also eventually satisfies $\mathsf{CoinValid}_k$ at every $p_k \in D_3$, including proposals by dealers of smaller depth. Indeed, the dealer's local sharing completion implies, by VRB external validity, that a depth-5 validator accepted its transcript Γ and delivered all the referenced announce entries. Announce eventual consistency propagates those entries to every process in D_3. The validator also delivered every referenced key broadcast, so VRB Totality 2 propagates those keys throughout D_3. The correct dealer sends the same transcript and carries out its sharing argument with every such process, and its commitment quorum belongs to every such process's kernel by quorum consistency. Thus $\mathsf{Valid}_k(\Gamma)$ eventually holds at every $p_k \in D_3$, so VRB Totality 1 completes sharing at all of them. By graded-gather totality, every process in D_3 eventually outputs four finite sets. Fix a dealer $p_D \in S_{3,i}$ for some process $p_i \in D_3$. Graded-gather external validity gives a depth-5 process that completed its VSS sharing phase, so sharing totality makes that completion available throughout D_3. If p_D belongs to $S_{1,j}$ for some correct process p_j, boundedness puts it in $S_{2,k}$ for every process $p_k \in D_3$, and all of them propose 1. Otherwise, p_D lies outside every correct set S_1, while boundedness puts it in every correct set S_4, so all processes in D_3 propose 0 upon obtaining their gather outputs. Thus in either case some bit is eventually proposed by every process in D_3; processes may also propose the other bit. The universal kernel of depth at least 5 is contained in D_3 and meets every correct process's quorum, so these proposals satisfy the liveness premise of Definition 8. Proxcensus liveness therefore defines all weights in S_3 at every process in D_3. The local sharing completions established above also satisfy the guard at line 18, so every such process eventually enables reconstruction. If a dealer has positive weight at one correct process, the argument in Lemma 19 shows that it lies in every correct set S_3, since otherwise proxcensus validity would force its weight to zero. Thus every process in D_3 invokes reconstruction for every dealer that contributes to a correct output, even if its own weight for that dealer is zero. By VSS reconstruction liveness, the process eventually reconstructs every secret to which it assigns positive weight, after which line 24 is enabled.

C.7 Binary Agreement by Expansion and Extraction

Lemma 25. *If every correct process proposes at least one bit, then some bit is proposed by a kernel of correct processes for every process in D_1. Consequently, the strong-liveness premise in Definition 15 holds for every $ld \geq 1$.*

Proof. Let S_b be the correct processes that propose b, so $S_0 \cup S_1 = D_0$; these sets may overlap. If neither bit has the claimed property, there are $p_i, p_j \in D_1$ and quorums $Q_i \in \mathcal{Q}_i$, $Q_j \in \mathcal{Q}_j$ with $Q_i \cap S_0 = \emptyset$ and $Q_j \cap S_1 = \emptyset$. Their intersection contains no correct process, since every correct process belongs to S_0 or S_1. This contradicts open quorum consistency for the owners $p_i, p_j \in D_1$. Thus some S_b is a kernel for every process in D_1, and in particular for every process in D_{ld}.

Lemma 26. *Suppose that every correct process p_j that decides has a grade $x_j \in \{0, \ldots, M\}$ and a coin $c_j \in \mathbb{Z}_M$ satisfying $|x_j - x_*| \leq 1$ and $\mathsf{dist}_M(c_j, c_*) \leq \Delta$. If the reference coin satisfies the direct bound from Definition 14, then the probability that two correct decisions differ is at most $q(\lambda)(4\Delta+4)/M + \mathrm{negl}(\lambda)$.*

Proof. Suppose $c_* \notin B_\Delta(x_*)$, with B_Δ as defined in (3). The first exclusion ensures that proximity to c_* does not cross the modular boundary, and hence $|\bar{c}_j - \bar{c}_*| \leq \Delta$ for every correct deciding process p_j. The second exclusion gives $|\bar{c}_* - x_*| > \Delta + 1$, including when x_* equals 0 or M. Consequently,

$$|(\bar{c}_j - x_j) - (\bar{c}_* - x_*)| \leq |\bar{c}_j - \bar{c}_*| + |x_j - x_*| \leq \Delta + 1.$$

All differences $\bar{c}_j - x_j$ therefore have the same strict sign, so all decisions agree. Consequently, disagreement requires $c_* \in B_\Delta(x_*)$ after reconstruction has been enabled. The direct bound from Definition 14 therefore bounds disagreement by

$$\Pr[E \wedge c_* \in B_\Delta(x_*)] \leq q(\lambda)\frac{4\Delta + 4}{M} + \mathrm{negl}(\lambda).$$

Theorem 6. *Assume that expansion uses $\mathrm{Prox}[vd, 0, ld]$, with $vd \geq 0$ and $ld \geq 1$. Assume also that the coin satisfies Definition 14 with safety depth 0 and liveness depth ld. Then Algorithm 9 implements $BA[vd, 0, ld]$ from Definition 15. The probability that any two correct decisions differ is at most $q(\lambda)(2^{2-\lambda} + 2^{2-2\lambda}) + \mathrm{negl}(\lambda)$, which is negligible in λ.*

Proof. For validity, suppose that a correct process decides b. The decision rule implies that its expansion grade differs from $(1-b)M$. By proxcensus validity, some process in D_{vd} previously prox-proposed b, and the expansion instance accepts a correct process's proposal only when it invokes *ba-propose*(b).

If any correct process decides, some correct process has enabled coin reconstruction. The first such process has already fixed x_*. Graded agreement bounds every other correct expansion output within one of x_*, including outputs obtained after reconstruction starts. Coin proximity to the reference value c_* and the direct probability bound give the hypotheses of Lemma 26. Substituting $M = 2^{2\lambda}$ and $\Delta = 2^\lambda$ gives the claimed bound, with any negligible subprotocol failure probabilities absorbed into the negligible term.

For strong liveness, let S_b be the correct processes that eventually propose b, and assume some $b \in \{0,1\}$ satisfies $S_b \in \mathcal{K}_i$ for every $p_i \in D_{ld}$. Every member of S_b is correct and forwards its proposal to the expansion instance, establishing the proxcensus liveness premise. Every process in D_{ld} therefore obtains a grade and invokes *coin-start*. Coin termination gives each of them a coin output and hence a decision. Liveness follows from strong liveness and Lemma 25. Finally, the *decided* flag ensures integrity.

Corollary 2 (Binary agreement with unknown participants). *Assume the authenticated communication model of Section 2, the open quorum consistency conditions of Definition 5, universal kernel depth $d \geq 5$, and the encryption and interactive argument primitives specified in Section 8.1. Encryption keys are generated during execution, with Byzantine processes permitted to supply arbitrary keys. For polynomially bounded executions, instantiate the common upper bound on the number of processes as $N = 2^\lambda$, both proxcensus constructions with $k = 2\lambda + 1$, and the VSS secret group as $\mathbb{Z}_{2^{2\lambda}}$. Then the announce, validated graded gather, VSS, proxcensus, and approximate common coin protocols of this paper, composed as in Algorithm 9, implement $BA[d-1, 0, 3]$ for all sufficiently large λ.*

Proof. The reliable-consensus construction provides $RC[d, 0, 3]$ by Theorem 1, and validated reliable broadcast from Corollary 1 implements announce by Lemma 1. Since $d \geq 5$, these instances also provide the validity depth five used by graded gather in Theorem 3. The same reliable consensus gives $Prox[d-1, 0, 3]$ by Theorem 2. The cryptographic primitives and announce give sharing consistency, sharing liveness, sharing totality, commitment, dealer correctness, and reconstruction liveness by Theorem 4. For the coin's gather predicate, a correct dealer proposes only after its own sharing completes; sharing totality makes this predicate eventually hold throughout D_3, as proved in Lemma 24. Thus Theorem 5 guarantees termination, proximity, and the direct probability bound with safety depth zero and liveness depth three. Finally, Theorem 6 gives the claimed BA properties and negligible disagreement probability. All instances have distinct tags. The direct bound covers concurrent adaptive execution, and a polynomial union over invocations preserves negligibility.

C.8 Agreement on a Core Set

Theorem 7 (Validated agreement on a core set). *Assume universal kernel depth $d \geq 5$, validated graded gather from Theorem 3, binary agreement $BA[d-1, 0, 3]$ from Corollary 2. Then Algorithm 10 implements $ACS_{\mathsf{V}}[5, 0, 3]$.*

Proof. Assume the ACS liveness premise with $ld = 3$. Every process $p_j \in D_3$ invokes *gg-propose*(m_j) at line 2, and $\mathsf{V}_k(p_j, m_j)$ eventually holds at every $p_k \in D_3$. Graded-gather totality therefore ensures that every process in D_3 eventually delivers an output. Fix a tagged input (p_j, m_j) occurring in $S_{3,i}$ for such an output. If it belongs to $S_{1,k}$ for some correct process p_k that delivers a graded-gather output, graded-gather boundedness places it in $S_{2,l}$ for every correct process

p_l that delivers a graded-gather output, so every process in D_3 proposes 1 to BA_j. Otherwise it lies outside every correct S_1, while graded-gather boundedness places it in every correct S_4, so every process in D_3 proposes 0 to BA_j. In either case, universal kernel depth and binary-agreement strong liveness make every process in D_3 decide in this instance. Thus every process in D_3 eventually obtains all decisions required for its set S_3 and outputs.

Suppose that BA_j decides 1 at a correct process. Binary-agreement validity gives a correct process that proposed 1, which occurs only when (p_j, m_j) belongs to its set S_2. Graded-gather boundedness therefore places (p_j, m_j) in S_3 at every correct process that delivers a graded-gather output. Binary-agreement agreement and graded-gather agreement now imply that all correct ACS outputs contain exactly the same tagged payloads. External validity and proposal validity follow from the corresponding graded-gather properties.

Let S_0 be the common core supplied by graded gather. It is contained in every correct process's set S_1 and therefore in every correct process's set S_2. Every correct process that participates proposes 1 and never proposes 0 in the corresponding binary-agreement instances. Binary-agreement validity excludes decision 0, so every member of S_0 belongs to the common ACS output. Core inclusion follows from the common-core guarantee in Theorem 3. After its first delivery, a process has *delivered* = `true`, so the condition $\neg$*delivered* at line 7 prevents a second delivery.