$\widetilde{\text{O}}$ptimal Adaptively Secure Hash-based Asynchronous Common Subset

Hanwen Feng, Zhenliang Lu, and Qiang Tang

* School of Computer Science,
The University of Sydney, Australia
{hanwen.feng, zhenliang.lu, qiang.tang}@sydney.edu.au

Abstract. Asynchronous multiparty computation (AMPC) requires an input agreement phase where all participants have a consistent view of the set of private inputs. While the input agreement problem can be precisely addressed by a Byzantine fault-tolerant consensus known as Asynchronous Common Subset (ACS), existing ACS constructions with potential post-quantum security have a large $\widetilde{\mathcal{O}}(n^3)$ communication complexity for a network of n nodes. This poses a bottleneck for AMPC in the same setting. In contrast, ACS has optimal constructions with quadratic communication complexity based on bilinear map assumptions.

In this paper, we bridge this gap by introducing a nearly optimal ACS, which solely relies on the blackbox use of collision-resistant hash functions. It exhibits $\widetilde{\mathcal{O}}(n^2)$ communication complexity, expected constant round complexity, and security against adaptive adversaries who can corrupt up to $n/3$ nodes and perform "after-fact-removal" attacks.

At the core of our new ACS is the first nearly optimal hash-based Multi-valued Validated Byzantine Agreement (MVBA). To reduce cubic communication while avoiding heavy cryptographic tools, we introduce a new design paradigm, with several new components. We define and analyze our MVBA and components within the UC-framework, facilitating their modular use in broader applications, particularly in AMPC.

1 Introduction

Input agreement challenge in asynchronous MPC. In secure multiparty computation (MPC), it is essential for all parties to reach an agreement on the set of "privately committed" inputs first [1], over which the subsequent computation should be carried out. For instance, in MPC based on (threshold) fully homomorphic encryption [41, 48], participants must agree on the same set of ciphertexts at the outset, while subsequent computation can be performed locally. Similar challenges exist in secret-sharing-based MPCs [5], where the parties need to decide on the set of secretly shared values. Under the synchronous network assumption, existing MPC protocols handle the input agreement issue by leveraging *broadcast channels*. On the other hand, since the broadcast channel is unavailable in the asynchronous network where messages may be arbitrarily delayed but eventually delivered, asynchronous MPC protocols [8, 27] (AMPC)

have to apply dedicated asynchronous Byzantine fault-tolerant (BFT) protocols to facilitate the process, which, however, may incur significant overhead and sometimes even dominate the overall communication cost of AMPC.

The core asynchronous BFT protocols employed in all existing AMPC protocols [8,23,27,42] include *reliable broadcast* (RBC) [14] and *agreement on a core set* (ACoreSet) [1, 8]. RBC can be seen as the asynchronous analog of the broadcast channel, but cannot guarantee termination in the face of a malicious sender. To patch the non-termination issue, the n participants of the AMPC, after disseminating their inputs (e.g., homomorphic ciphertexts, or shares of private inputs) via n RBC instances, will jointly execute an ACoreSet to decide which $n - f$ RBC instances have terminated. Here f is the maximal number of nodes that an adversary is allowed to corrupt, and we consider the optimal resilience of $f = \lceil \frac{n}{3} \rceil - 1$ throughout this paper. However, running n instances of RBC [1], and ACoreSet used in existing works will each incur $\mathcal{O}(n^3)$-bits communication cost, which is prohibitive when the AMPC is deployed on a moderate scale.

While much MPC research focuses on reducing communication, they are mostly on the parts relevant to the computation task, (while ignoring the cost of the input agreement phase first). However, the actual dominating complexity remains in the input agreement phase. For example, evaluating a circuit with M multiplication gates and D layers can only incur $\mathcal{O}(M \cdot n + D \cdot n^2)$ communication complexity [2], as shown in many AMPC protocols [24–27]. For computational tasks of interest, such as oblivious message permutation for anonymous communication [42], M is merely $\widetilde{\mathcal{O}}(n)$ and D is $O(\log^2 n)$, making the cubic communication cost resulting from input agreement a bottleneck.

Asynchronous common subset. The input agreement problem that arises in the literature of AMPC indeed has broader connections. A BFT consensus primitive, *Asynchronous Common Subset* (ACS), which allows a network of n nodes, each with input, to eventually agree on a set of $n - f$ input values, solves exactly the input agreement problem (and can replace the paradigm of using n RBC instances plus an ACoreSet altogether). Historically, Ben-Or et al. [9] directly used ACS to denote agreement on a core set. Recent asynchronous consensus studies [34, 38, 39, 44, 55] distilled the notion into its current form, and demonstrated it as a core tool for realizing a public distributed ledger in asynchronous networks. Indeed, several ACS constructions [44,55] directly follow similar approaches of using n RBC plus an ACoreSet, which still suffers from the high communication and round costs. However, better ACS constructions beyond this traditional path could, in turn, have implications on the classical input agreement problem, which has not been well explored in AMPC.

MVBA: the only known way towards optimal ACS. Multi-valued Validated Byzantine Agreement (MVBA), originally proposed by Cachin et al [16], allows n parties to jointly decide one common output, which will be guaranteed

[1] In this work, we focus on strong adaptive adversaries as per [2], and $\mathcal{O}(n^2)$-bit communication cost is necessary for an adaptively secure RBC and any other consensus.

[2] Here we do not count the communication cost incurred in the offline phase for generating the Beaver triples, which can be performed via pre-processing.

valid (according to a predefined *predicate*), when all honest parties input valid values. As hinted implicitly in [16], an MVBA protocol can easily imply an ACS protocol by letting each party multicast its input value, and then they jointly execute an MVBA, each using a *vector* of received values as input. Here, the *predicate* could simply be that the vector contains values from $n - f$ parties (e.g., endorsed by digital signatures.). MVBA was long considered of theoretical interest (since the original MVBA [16] itself has a cubic communication complexity), until recently, a sequence of progress re-establish it as a critical component for practical asynchronous consensus [36–39] and beyond [32, 33, 37]. Notably, exciting recent progress on MVBA itself also significantly reduced the communication complexity: Abraham et al. constructed a protocol called VABA that achieved quadratic communication [3], yet it still left the gap for ACS, as now the input (vector) size itself becomes $\mathcal{O}(n)$ when compiling MVBA to ACS, still resulting a $\mathcal{O}(n^3)$ communication for ACS. Finally, Lu et al [43] expanded the communication complexity into detailed terms, and presented an MVBA with optimal communication complexity of $\mathcal{O}(n\ell + n^2\lambda)$ (called Dumbo-MVBA, and an extension framework), where ℓ is the input size [3]. Incorporating Dumbo-MVBA into Cachin et al.'s framework finally yields the first ACS with quadratic communication complexity of $\mathcal{O}(n^2\ell + n^2\lambda)$; and it is also applicable to AMPC.

Striving for post-quantum or information-theoretic security. The complexity reductions in [3, 43] do not come for free. They heavily rely on cryptographic tools, particularly non-interactive threshold signature, (e.g., BLS from pairing-based assumptions) to generate succinct proofs to compress communication. This puts several restrictions on applying them to AMPC. (1) Besides the need for private setup, one particular constrain is (in)security against quantum attackers, as we do not have any candidate yet for post-quantum non-interactive threshold signatures; while the traditional approach (with high cubic communication) of using RBC and ACoreSet can be information-theoretically (IT) secure or using collision-resistant hash only. And other parts of AMPC, for example, the secret-sharing based framework [7,24,27], the other parts of the online phase besides input agreement, and the whole offline phase could also be information-theoretically secure. In principle, we may consider *interactive* threshold signatures with post-quantum security, however, this itself is a special form of AMPC that may require the input agreement phase to begin with, thus causing a circular problem. (2) Those algebraic operations can introduce substantial computational overhead. For instance, pairing operations are significantly more costly, being approximately 10^5 times slower than hash computations.

There are recently renewed interests in exploring ACS and MVBA in the setting solely using hash functions [34], and information-theoretic settings. Unfor-

[3] Due to the FLP impossibility, all asynchronous Byzantine agreement protocols are randomized. This also applies to MVBA, which can be easily shown to imply an asynchronous Byzantine agreement [16]. Throughout this paper, we assume a common coin, and omit its cost when computing complexity metrics; jumping ahead, in AMPC applications, a common coin can be easily implemented with the help of offline pre-processing, thus having no impact on the communication of online phase.

tunately, without the help of threshold signatures, all those constructions again suffer from large *cubic* communication complexity (and more, see Table. 2 for details). This poses a natural question:

Is it possible to design an optimal (and adaptively secure) ACS *without public-key cryptography?*

1.1 Our Contributions

We answer the above question affirmatively by presenting a near-optimal hash-based MVBA, which allows us to obtain a near-optimal hash-based ACS. We detail our contributions in the following.

Table 1. Comparison of the MVBA protocols

Protocol	Resilience	Communication	Message	Round	Crypto Tool
CKPS-MVBA [16]	$f < n/3$	$\mathcal{O}(\ell n^2 + \lambda n^2 + n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	threshold signature
VABA [3]	$f < n/3$	$\mathcal{O}(\ell n^2 + \lambda n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	threshold signature
sMVBA [38]	$f < n/3$	$\mathcal{O}(\ell n^2 + \lambda n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	threshold signature
Dumbo-MVBA [43]	$f < n/3$	$\mathcal{O}(\ell n + \lambda n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	threshold signature
DYX+22 [33]	$f < n/3$	$\mathcal{O}(\ell n^2 + \lambda n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(\log n)$	DCR, DDH and hash
Trivial hash-MVBA†	$f < n/3$	$\mathcal{O}(\ell n + \lambda n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	hash
Fin-MVBA-1 [34]	$f < n/3$	$\mathcal{O}(\ell n^2 + \lambda n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(1)$	hash
Fin-MVBA-2 [34]	$f < n/3$	$\mathcal{O}(\ell n^2 + n^3 \log n)$	$\mathcal{O}(n^3)$	$\mathcal{O}(1)$	none
Our hash-MVBA	$f < n/3$	$\mathcal{O}(\ell n + \lambda n^2 \log n + \kappa \lambda n^2)$	$\mathcal{O}(\kappa n^2)$	$\mathcal{O}(\log \kappa)$	hash
Our IT-MVBA	$f < n/3$	$\mathcal{O}(\ell \kappa n^2)$	$\mathcal{O}(\kappa n^2)$	$\mathcal{O}(\log \kappa)$	none

κ is the statistical security parameter, and λ is the computational security parameter. In practice, we usually use $\kappa = 40$ and $\lambda = 128$, which are constants independent of n. Following the standard practice in asynchronous consensus literature, we assume a common coin and consistently discount the cost of it for all protocols. † "Trivial hash-MVBA" denotes a variant of Dumbo-MVBA protocol where the threshold signature is instantiated with a concatenation of hash-based digital signatures.

Nearly optimal hash-based MVBA, ACS and better AMPC. We present the first nearly optimal MVBA protocols with optimal resilience of $f < n/3$, constant rounds, quadratic message complexity, and $\mathcal{O}(n\ell + \lambda n^2 \log n + \kappa n^2 \lambda)$ communication complexity. Here, κ is the statistical security parameter, and λ is the computational security parameter. In practice, κ, λ are usually small constants independent of the number of nodes n. These protocols solely make use of collision-resistant hash functions and achieve security against *strongly adaptive* attackers. The gap towards the optimal quadratic complexity[4] is at the small factor of $\mathsf{max}\{\log n, \kappa\}$. Our MVBA can easily be adapted to be IT secure, at the cost of increasing the communication to $\mathcal{O}(\ell \kappa n^2)$, which still outperforms all existing IT-secure MVBA and may be of independent interest. Please see Sect.6.3 for more discussions. We summarize MVBA constructions in Table 1.

We can then instantiate the general compilation to obtain a hash-based ACS, with optimal resilience of $f < n/3$, constant round complexity, $\mathcal{O}(\kappa n^2)$ message

[4] Since every node will receive $\mathcal{O}(\ell)$ bits, and $\mathcal{O}(n^2)$ communication is necessary for any adaptively secure consensus, a lower bound on MVBA is $\mathcal{O}(\ell n + n^2)$.

complexity, and $\mathcal{O}\big(n^2\ell + \lambda n^2 \log n + \kappa n^2 \lambda\big)$ communication complexity, by using hash-based signatures like [10]. In Table 2, we compare our ACS with existing optimally resilient ACS schemes, where the cost of common coin generation is consistently discounted for all schemes. [5]

Table 2. Comparison of ACS protocols

Protocol	Resilience	Communication	Message	Round	Crypto Tool
CKPS-ACS [16]	$f < n/3$	$\mathcal{O}(\ell n^2 + \lambda n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	threshold signature
Dumbo2 [39]	$f < n/3$	$\mathcal{O}(\ell n^2 + \lambda n^3 \log n)$	$\mathcal{O}(n^3)$	$\mathcal{O}(1)$	threshold signature
sDumbo [38]	$f < n/3$	$\mathcal{O}(\ell n^2 + \lambda n^3 \log n)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	threshold signature
via Dumbo-MVBA [43]	$f < n/3$	$\mathcal{O}(\ell n^2 + \lambda n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	threshold signature
HBBFT [44]	$f < n/3$	$\mathcal{O}(\ell n^2 + \lambda n^3 \log n)$	$\mathcal{O}(n^3)$	$\mathcal{O}(\log n)$	hash
{HBBFT [44] or PACE [55]} with RBC from [4]	$f < n/3$	$\mathcal{O}(\ell n^2 + n^3 \log n)$	$\mathcal{O}(n^3)$	$\mathcal{O}(\log n)$	none
Fin-ACS [34]	$f < n/3$	$\mathcal{O}(\ell n^2 + \lambda n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(1)$	hash
Fin-ACS [34] with RBC from [4]	$f < n/3$	$\mathcal{O}(\ell n^2 + n^3 \log n)$	$\mathcal{O}(n^3)$	$\mathcal{O}(1)$	none
DDL+24 [31]	$f < n/3$	$\mathcal{O}(\ell n^2 + \lambda n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(1)$	none
Our ACS	$f < n/3$	$\mathcal{O}\big(\ell n^2 + \lambda n^2 \log n + \kappa \lambda n^2\big)$	$\mathcal{O}(\kappa n^2)$	$\mathcal{O}(\log \kappa)$	hash

We then demonstrate how to use ACS to solve the input agreement problem in secret-sharing-based AMPC. Combining this with existing techniques on function evaluation [7,30], we obtain the first hash-based online-phase AMPC protocol with a quadratic communication of $\mathcal{O}\big(M \cdot n + \ell \cdot n^2\big) + \mathcal{O}\big(\lambda n^2 \log n + \kappa n^2\big)$, while the previous best one has to include an extra cubic term $\mathcal{O}\big(n^3\big)$ due to the input agreement. Here, M is the number of multiplicative gates in the circuit C, n is the number of nodes, and ℓ is the input size.

In the context of online/offline AMPC, the common coins required by our ACS can be easily implemented using the built-in pre-processing phase. Specifically, all participants can obtain sufficient Beaver triples and pre-shared random values during the offline pre-processing phase (for example, by using protocols such as those in [27]). Then, during the online phase, they can simply multicast their shares and reconstruct a random value that serves as a coin. This approach incurs only an $O(n^2)$ communication cost in the online phase and does not rely on algebraic operations.

A new framework, new components, and UC security. Our advancements in MVBA, to be detailed in Sect.1.2, stem from a novel framework significantly divergent from the ones in existing MVBA protocols, and multiple new tools we propose. As we briefly mentioned above, translating the design methodology of existing optimal MVBA in [43] directly into the hash-based setting would result in cubic communication complexity. We present the construction in a modular way,

[5] We note that a very recent work by Das et al. [31] (dubbed DDL+24) presented a hash-based ACS protocol (with communication complexity of $\mathcal{O}\big(\ell n^2 + \lambda n^3\big)$) and a hash-based coin protocol (with communication complexity of $\mathcal{O}\big(\lambda n^3\big)$).

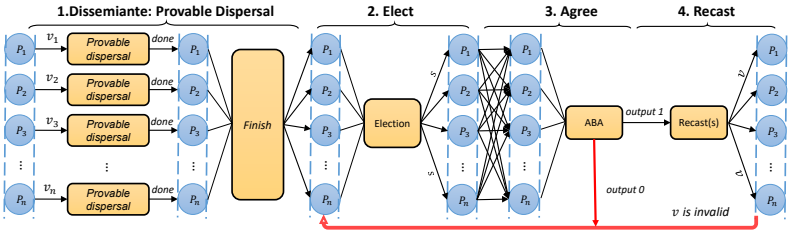

Fig. 1. The execution flow of Dumbo-MVBA [43].

and illustrate how to construct a nearly optimal MVBA using three novel components: *somewhat-good multi-dealer information dispersal* (SMID, see Sect.3), *synchronized multi-valued broadcast* (SMB, see Sect.4), and *asynchronous reliable consensus* (ARC, see Sect.5), along with established components like asynchronous binary agreement (ABA) [46] and common coins.

Notably, ARC was explored in a recent work [12] for different purposes; however, our new construction is a deterministic protocol with IT security against strong adaptive adversaries, while theirs is randomized and computationally secure and cannot withstand strongly adaptive adversaries. Additionally, SMB generalizes the *synchronized binary-value broadcast* in [46], a core component in the seminal ABA protocol [46].

Also, to facilitate their modular uses in future applications, particularly in AMPC, we also formulate all these concepts as ideal functionalities within the Universal Composability (UC) framework [20], in contrast with all existing treatments in the literature that are property based. Because of asynchronous communication and adaptive corruption, and inherent limitations of several components, describing those ideal functionalities precisely requires care. We demonstrate that all constructions presented in this paper UC-realize the corresponding functionalities, enabling the protocols to be flexibly composed in larger protocol designs. We believe those new components as well as the simulation-based formulations are of independent interests and could be more broadly useful.

1.2 Challenges and Technical Overview

Conventional design of MVBA. Recall the objective of MVBA, which is to reach consensus and terminate on a "valid" input (when all honest nodes contribute valid inputs). One intuitive approach is to randomly select a node $\mathcal{P}_i$ to provide its input, and let all nodes converge on $\mathcal{P}_i$'s input (e.g., letting $\mathcal{P}_i$ to reliably broadcast his value). This way, there is a constant probability that the selected value is valid, which may be amplified easily, say sample κ nodes to make sure at least one of them is honest and will respond. However, this approach will fail in the face of adaptive adversaries which can corrupt *all* κ (as usually κ is a small constant that could be significantly smaller than $n/3$) selected nodes, prevent them from providing valid inputs, or simply all mute.

To counter the adaptive corruption, existing MVBA protocols [3,16,34,38,43] let all nodes "provably disseminate" their inputs to the network before the elec-

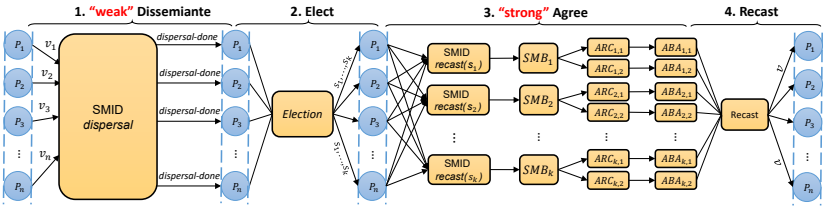

Fig. 2. The execution flow of our MVBA.

tion, such that (1) a disseminated input value from an honest node is always available (even if corrupted later after selection), as it can be consistently reconstructed by all others; (2) the "proof" of dissemination also needs to be delivered to all nodes, which will later inform the nodes whether to repeat or not after a random selection. Doing such procedures in a communication efficient way requires care. Particularly, the"optimal" dissemination of Dumbo-MVBA [43] is achieved delicately by *provable dispersal*, which makes careful uses of the threshold signature, vector commitment (VC) [21], and the erasure code [11]. Each node shall encode its input into n smaller fragments via an erasure code scheme, and use VC to commit all fragments into a short vc while generating a validity proof for each fragment. Then, each node sends its vc, along with a fragment and the validity proof, to each other node. When receiving a valid fragment from another sender, each node will respond with a partial signature on vc, so each honest node will be able to obtain a threshold signature on his vc as proof to show his dispersal is completed. See Figure 1 as a pictorial illustration.

Challenges of using hash alone to reduce communication. Note that threshold signature plays a vital role in Dumbo-MVBA, forming a *certificate* to show the availability of a dispersed value, so that nodes know how to proceed after selection. In the hash-based setting, one might consider using a concatenation of $n - f$ hash-based signatures instead of a threshold signature, to use the optimal ACS obtained via Dumbo-MVBA, however, it will lead to cubic communication.

Now without threshold signatures, we must deviate from the traditional approach to attain the desired communication complexity. Instead, we start with a "weak" dissemination (without a strong proof), but augment the latter phase to be "strong" to still enable nodes to gradually figure out how to proceed after selection, with only some minimal and unreliable hint. Specifically, as outlined in Figure 2, we begin with a "somewhat-good" information dispersal (SMID) as our dissemination phase. SMID replaces partial signatures with simple *echo* messages. However, it does not serve as proof that can be publicly verified, thus cannot even inform the nodes whether a selected node actually completed the dispersal, let alone ensuring there is only one consistent value if jointly reconstructed. The strengthening of the "Agree" part turns out to be highly non-trivial.

Further challenges of adaptive security with sub-cubic communication. Adaptive corruption makes it even more challenging. In the election phase, even if we select a "so-far-honest" node $\mathcal{P}_s$ that honestly sent fragments to $n - f$ nodes, once the adaptive adversary learns that $\mathcal{P}_s$ has been selected (when there could still

f honest nodes that have not received anything yet), it can immediately corrupt $\mathcal{P}_s$, and modify the remaining f fragments yet to be delivered. It follows that the remaining f honest nodes will receive fragments corresponding to different values. Now, during recast, it is possible that $f+1$ honest nodes recast and obtain original v_s, while another f honest nodes may recast different values. [6]

To enable nodes to converge to the same output value (for agreement & termination), we proceed in multiple steps:

We first introduce a new primitive called SMB (Synchronized Multi-valued Broadcast) and give an efficient construction. An SMB ensures that, if $n-2f$ honest nodes have the same input v, then all honest nodes can output a set with size at most 2. And, for any two honest nodes $\mathcal{P}_i$ and $\mathcal{P}_j$, if $\mathcal{P}_i$ outputs a set $val_i = \{v'\}$, then v' must be subset of $\mathcal{P}_j$'s output set val_j. Additionally, if $|val_i| = |val_j| = 2$, then $val_i = val_j$. The SMB acts as a robust *filtering* mechanism, limiting the input values for the next step (vc of an elected node) among all honest nodes to be at most two. For details about SMB, please see Sec.4.

Next, to select one of these values to recast the final output, we make use of and construct an asynchronous reliable consensus (ARC). This is like a consensus analog of reliable broadcast that ensures agreement, and a conditional termination, if all honest nodes have the same input value v, they terminate and output v. Considering there are at most two values after SMB, hence, we let all honest nodes participate in two ARC protocols.

Now note that, there is a possibility that one of the two values is not held by all honest nodes, introducing a further termination issue. To help with the further decision, we employ an ABA (Asynchronous Binary Agreement) protocol after each ARC to "vote". The input for $\mathcal{P}_i$ of each ABA instance is 1 if $\mathcal{P}_i$ already outputs in the corresponding ARC. When $\mathcal{P}_i$ outputs 1 in one ABA instance, he will assign 0 as inputs for all remaining ABA instances, if their corresponding ARC instances have not terminated. The discussion above assumes a "so-far-honest" node having completed its dispersal before the election phase and being elected. To ensure this, we adjust the election phase to choose κ random nodes, ensuring that at least one of them to be honest, where κ is a statistical security parameter (usually a small constant). Now since at least one node is "so-far-honest" thus for one ARC instance, sufficient $(n-2f)$ honest nodes will have the same vc as input, thus the corresponding ARC and ABA will terminate, which in turn helps other remaining instances to terminate.

2 Model, Goal and Preliminary

2.1 System Model

We consider an *asynchronous* network where nodes are pairwise connected via *authenticated* channels, and we assume the identities of all participating nodes

[6] We remark that dealing with adaptive corruption with cubic communication could be easy, as we can simply let each node *reliably broadcast* its full value (without erasure encoding) at the beginning.

are publicly known. We focus on optimal resilience, meaning that the total number of corrupted nodes is at most $f < n/3$, where the adversary $\mathcal{A}$ is *adaptive* and can corrupt nodes at any point during the protocol's execution. An adaptive adversary can retract any undelivered messages that were sent by a newly corrupted node. During execution, the nodes that have remained honest up to a given point are referred to as *so-far-honest* nodes, while those that remain honest until the end of execution are called *forever-honest* nodes. For simplicity, when we refer to honest nodes in this paper, we mean *forever-honest* nodes. Additionally, in this paper, we use the terms "node" and "party" interchangeably, both referring to a participant in the protocol.

To precisely describe the threat model and support the modular use of our protocols and components, we define all concepts and analyze the constructions presented in this paper within the UC framework [20]. Specifically, we adopt the formulation language from [28], which accurately captures the adversary's ability to delay messages in an asynchronous network. For a background introduction to UC security, we refer readers to Appendix.B.

Setup-wise, a standard PKI setup is needed for some protocols. The system also needs a common random string as the key of a collision-resistant hash function; However, the hash key can be generated by a common coin protocol.

2.2 Design Goals

In this paper, we focus on constructing a communication-efficient Asynchronous Common Subset (ACS) [9, 44] and Multi-valued Validated Byzantine Agreement (MVBA) [3, 16, 43], by solely using collision-resistant hash functions. In this section, we present the ideal functionalities for the two primitives, which capture their existing security properties and can directly serve as subroutines in designing larger protocols.

Asynchronous common subset. ACS enables n nodes each having an input value to eventually agree on the same set of $n - f$ values, where f is the maximal number of nodes an adversary can corrupt. We formulate ACS as an ideal functionality $\mathcal{F}_{\mathsf{ACS}}$ [7] in Fig. 3, and $\mathcal{F}_{\mathsf{ACS}}$ satisfies all the security *properties* listed below that are considered in ACS.

- **Agreement.** The outputs of any two honest nodes must be the same.
- **Validity.** The output set must contain at least $n - f$ values including the inputs of at least $n - 2f$ honest nodes.
- **Totality.** All honest nodes can eventually output, as long as there are $n - f$ honest nodes participating in the protocol with an input.

[7] We remark that in a concurrent work Shoup [52] formalized the ideal functionality of ACS, which, however, is for *index-ACS*, a special variant introduced in [31] where the inputs are restricted to be validated indexes in $[n]$. Moreover, the modelings in [52] directly allow the adversary to specify the "core set", which, as discussed in [28], may fail to mimic a real-world execution; In contrast, ours follows Cohen et al.'s approach [28] to avoid potential simulation failures.

Functionality $\mathcal{F}_{\mathsf{ACS}}$

$\mathcal{F}_{\mathsf{ACS}}$ proceeds as follows, running with $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ and the advesary. At the first activation, verify $(\mathtt{sid}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ for the session ID $\mathtt{sid}$ and the set of nodes. Initialize: $\mathtt{participated}_i = 0$, $v_i = \perp$ and $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{output}} = 1$ for all $i \in [n]$; $\mathsf{commonSet} = \emptyset$.

- Upon receiving $(\mathtt{delay}, \mathtt{sid}, \mathcal{P}_i, \mathtt{type}, D)$ from the adversary for any $i \in [n]$, if $\mathtt{type} \in \{\mathtt{input}, \mathtt{output}\}$ and $D \in \mathbb{Z}$ represented in unary notation, then update $\mathsf{D}_i^{\mathtt{type}} = \max\{1, \mathsf{D}_i^{\mathtt{type}} + D\}$, and provide $(\mathtt{delay\text{-}set}, \mathtt{sid})$ to the adversary.
- Upon receiving $(\mathtt{input}, \mathtt{sid}, v')$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and provide $(\mathtt{input}, \mathtt{sid}, v', \mathcal{P}_i)$ to the adversary.
- Upon receiving $(\mathtt{fetch}, \mathtt{sid})$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and the **Output Release Procedure**, and provide $(\mathtt{fetch}, \mathtt{sid}, \mathcal{P}_i)$ to the adversary.

Input Submission Procedure: If $\mathtt{participated}_i = 0$, and $(\mathtt{input}, \mathtt{sid}, v')$ has been provided by $\mathcal{P}_i$, do the following:

- Update $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{input}} - 1$;
- If $\mathsf{D}_i^{\mathtt{input}} = 0$, update $\mathtt{participated}_i = 1$, and record $v_i = v'$. If $|\mathsf{commonSet}| < n - f$, then update $\mathsf{commonSet} = \mathsf{commonSet} \cup \{v_i\}$.

Output Release Procedure: If $\sum_{j \in [n]} \mathtt{participated}_j \geq n - f$, do the following:

- Update $\mathsf{D}_i^{\mathtt{output}} = \mathsf{D}_i^{\mathtt{output}} - 1$;
- if $\mathsf{D}_i^{\mathtt{output}} = 0$, set $(\mathtt{output}, \mathtt{sid}, \mathsf{commonSet})$ to be sent to $\mathcal{P}_i$.

Fig. 3. The asynchronous common subset functionality $\mathcal{F}_{\mathsf{ACS}}$

$\mathcal{F}_{\mathsf{ACS}}$, as well as the other ideal functionalities defined in this paper, captures the adversary's capabilities of adaptively corrupting honest nodes based on their inputs and of retracting the inputs. Particularly, the input value of an honest node is leaked to the adversary, and then the adversary may corrupt the node based on the input value, and subsequently, provide a new input value on behalf of this node. Following [5] and also [27], we consider the *corruption-aware* functionalities, namely, the functionality knows which node has been corrupted by the adversary, and thus allows the simulator to submit a different input value on behalf of a corrupted node. Nonetheless, as honest nodes do not have private states in ACS, $\mathcal{F}_{\mathsf{ACS}}$ does not need to explicitly handle a corruption request from the adversary, except keeping track of the set of corrupted nodes.

Multi-valued Validated Byzantine Agreement. MVBA is defined w.r.t. a binary predicate Predicate. When each honest node $\mathcal{P}_i$ provides a valid input v_i, i.e., $\mathsf{Predicate}(v_i) = 1$, MVBA ensures all honest nodes output a same valid value. Formally, an MVBA protocol satisfies the following properties with all but negligible probability:

- **Termination.** If every honest node p_i inputs an externally valid value v_i, then every honest node outputs a value.
- **External-Validity.** If a honest node outputs a value v, then $Q(v) = 1$.
- **Agreement.** Any two distinct honest nodes always output the same value.

Functionality $\mathcal{F}_{\mathsf{MVBA}}$

$\mathcal{F}_{\mathsf{MVBA}}$ proceeds as follows, running with $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ and the adversary. At the first activation, verify $(\mathtt{sid}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ for the session ID $\mathtt{sid}$ and the set of nodes. Initialize: $\mathtt{participated}_i = 0$, $v_i = \perp$ and $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{output}} = 1$ for all $i \in [n]$; $y = a = \perp$.

- Upon receiving $(\mathtt{delay}, \mathtt{sid}, \mathcal{P}_i, \mathtt{type}, D)$ from the adversary for any $i \in [n]$, if $\mathtt{type} \in \{\mathtt{input}, \mathtt{output}\}$ and $D \in \mathbb{Z}$ represented in unary notation, then update $\mathsf{D}_i^{\mathtt{type}} = \max\{1, \mathsf{D}_i^{\mathtt{type}} + D\}$, and provide $(\mathtt{delay\text{-}set}, \mathtt{sid})$ to the adversary.
- Upon receiving $(\mathtt{input}, \mathtt{sid}, v')$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and provide $(\mathtt{input}, \mathtt{sid}, v', \mathcal{P}_i)$ to the adversary.
- Upon receiving $(\mathtt{decide\text{-}output}, \mathtt{sid}, v')$ from the adversary, if $\mathsf{Predicate}(v') = 1$, then record $a = v'$; otherwise, ignore this message.
- Upon receiving $(\mathtt{fetch}, \mathtt{sid})$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and the **Output Release Procedure**, and provide $(\mathtt{fetch}, \mathtt{sid}, \mathcal{P}_i)$ to the adversary.

Input Submission Procedure: If $\mathtt{participated}_i = 0$, and $(\mathtt{input}, \mathtt{sid}, v')$ has been provided by $\mathcal{P}_i$, do the following:

- Update $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{input}} - 1$;
- If $\mathsf{D}_i^{\mathtt{input}} = 0$, update $\mathtt{participated}_i = 1$, and record $v_i = v'$.

Output Release Procedure: If there exists a subset $\mathbb{I} \subset [n]$ such that $|\mathbb{I}| \geq n - f$ and $\mathsf{Predicate}(v_i) = 1$ for all $i \in \mathbb{I}$, do the following:

- Update $\mathsf{D}_i^{\mathtt{output}} = \mathsf{D}_i^{\mathtt{output}} - 1$;
- if $\mathsf{D}_i^{\mathtt{output}} = 0$, then do the following. If $y = \perp$: if $a = \perp$, sample $j \leftarrow\!\!\$\ \mathbb{I}$, and set $y = v_j$; if $a \neq \perp$, set $y = a$; provide $(\mathtt{output}, \mathtt{sid}, y)$ to the adversary. Set $(\mathtt{output}, \mathtt{sid}, y)$ to be sent to $\mathcal{P}_i$.

Fig. 4. The asynchronous multi-valued validated byzantine agreement functionality $\mathcal{F}_{\mathsf{MVBA}}$ with predicate function $\mathsf{Predicate}$

We formulate MVBA as an ideal functionality $\mathcal{F}_{\mathsf{MVBA}}$ in Fig. 4. Similarly, this functionality captures various security *properties* of MVBA used in the literature. Particularly, the *agreement* can be implied since $\mathcal{F}_{\mathsf{MVBA}}$ returns the same value to all nodes. Since the output must satisfy the predicate in the ideal functionality, this inherently ensures that external validity is also enforced. Moreover, as there are $n - f$ valid inputs, the functionality will provide an output to all, implying the *termination* property. On the other hand, since $\mathcal{F}_{\mathsf{MVBA}}$ allows the adversary to specify a valid output, we cannot guarantee the *quality* considered in [3], which requires that the output is the input of an honest node with constant probability. Nonetheless, it suffices for many applications, including ACS.

2.3 Preliminary

Our protocols utilize several cryptographic primitives and protocols. Details and the relevant notations used throughout the paper can be found in Appendix C.

Asynchronous binary agreement ($\mathcal{F}_{\mathsf{ABA}}$). In $\mathcal{F}_{\mathsf{ABA}}$ among n nodes, if the honest nodes input a single bit, either 0 or 1, then all honest nodes will output

a common bit $b \in \{0,1\}$, where b corresponds to the input of some honest node. The ideal functionality $\mathcal{F}_{\mathsf{ABA}}$ is described in Fig. 10.

Erasure code. A (k,n)-erasure code [11] comprises two deterministic algorithms: $\mathsf{EC.Encode}$ and $\mathsf{EC.Decode}$. The $\mathsf{EC.Encode}$ takes a value $\mathbf{m}$ as input and outputs n fragments $\mathbf{c} = (\mathsf{c}_1, \ldots, \mathsf{c}_n)$. Any k elements in $\mathbf{c}$ can reconstruct the $\mathbf{m}$ using the $\mathsf{EC.Decode}$.

Vector commitment (VC). A VC scheme [21] consists of a tuple of algorithms: ($\mathsf{VC.Setup}, \mathsf{VCom}, \mathsf{Open}, \mathsf{VerifyOpen}$). Throughout this paper, we focus on the Merkle-tree-based *deterministic* VC scheme, whose security is solely based on the collision resistance of the underlying hash function. Particularly, the commitment size of vc is $\mathcal{O}(\lambda)$ bits, and the witness size π is $\mathcal{O}(\lambda \log n)$.

3 Somewhat-good Multi-dealer Information Dispersal

In this section, we introduce our first component: Somewhat-good Multi-dealer Information Dispersal (SMID). Similar to other information dispersal protocols [18, 43], in SMID, it also contains two phases: *dispersal* and *recast*. In the dispersal phase, each dealer encodes its input into multiple fragments and subsequently transmits these fragments across the network. This allows a receiver to collect sufficient fragments from the network and reconstruct the input. Our SMID is designed to replace the concurrent n instances of asynchronous verifiable information dispersal (AVID) [18], where there are f malicious senders. However, while AVID makes significant efforts to ensure the "availability" of any dispersed data, ensuring that any honest node can always retrieve the same data value, our SMID is merely a "best-effort dispersal". In our approach, an honest sender disseminates the fragments across the network, allowing at least $f+1$ honest nodes to retrieve the same value, rather than ensuring that all honest nodes can do so. Weakening the security allows us to design a more communication-efficient protocol than n concurrent AVID instances, without relying on heavy cryptographic tools. Formally, the SMID has the following properties:

- **Termination.** Every honest node can complete the dispersal phase of the SMID instance if every honest node P_i inputs a value v_i.
- **Validity.** Let $\{P_i\}_{i \in \mathbb{H}}$ be the set of initially honest nodes, and each P_i has an input value v_i. If one honest node has completed the dispersal phase, then there exists a subset $\mathbb{I} \subset \mathbb{H}$ s.t. $|\mathbb{I}| \geq f+1$. During the recast phase, if all honest nodes with index $i \in \mathbb{I}$ as input, then at least $n-2f$ honest nodes can reconstruct the initial input value v_i other f P_i.

3.1 Ideal Functionality of SMID $\mathcal{F}_{\mathsf{SMID}}$

In this section, we first formulate SMID as an ideal functionality $\mathcal{F}_{\mathsf{SMID}}$, capturing all security properties SMID can provide. Then, we present Algorithm 1 which indeed *UC-realizes* $\mathcal{F}_{\mathsf{SMID}}$.

The classical AVID ensures that all honest nodes can retrieve the same value, as long as an honest node has finished the dispersal phase, regardless of whether

Functionality $\mathcal{F}_{\mathsf{SMID}}$

$\mathcal{F}_{\mathsf{SMID}}$ proceeds as follows, running with $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ and the adversary. At the first activation, verify $(\mathtt{sid}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ for the session ID $\mathtt{sid}$ and the set of nodes. Initialize: $\mathtt{dispersed}_i = 0$, $v_i = \perp$, $\mathtt{C\text{-}counter}_i = 0$, $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{output}} = 1$, $\mathsf{done}_i = 0$, $a_{i,j} = (0, \perp)$, $\mathsf{requested}_{i,j} = 0$, and $\mathsf{D}_{i,j}^{\mathtt{input}} = \mathsf{D}_{i,j}^{\mathtt{output}} = 1$ for all $i, j \in [n]$ and $i \neq j$. Let $\mathbb{H}$ be the set of "so-far-honest" nodes, and $\mathbb{C}$ be the set of corrupted nodes.

- Upon receiving $(\mathtt{delay}, \mathtt{sid}, \mathcal{P}_i, \mathtt{type}, D)$ (or $(\mathtt{delay}, \mathtt{sid}, \mathcal{P}_i, \mathcal{P}_j, \mathtt{type}, D)$) from the adversary for $i, j \in [n]$, if $\mathtt{type} \in \{\mathtt{input}, \mathtt{output}\}$ and $D \in \mathbb{Z}$ represented in unary notation, then update $\mathsf{D}_i^{\mathtt{type}} = \max\{1, \mathsf{D}_i^{\mathtt{type}} + D\}$ (or $\mathsf{D}_{i,j}^{\mathtt{type}} = \max\{1, \mathsf{D}_{i,j}^{\mathtt{type}} + D\}$), and provide $(\mathtt{delay\text{-}set}, \mathtt{sid})$ to the adversary.
- Upon receiving $(\mathtt{disperse}, \mathtt{sid}, v' \neq \perp)$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and provide $(\mathtt{disperse}, \mathtt{sid}, v', \mathcal{P}_i)$ to the adversary.
- Upon receiving $(\mathtt{configure\text{-}done}, \mathtt{sid}, \mathcal{P}_i)$ from the adversary, update $\mathsf{done}_i = 1$ if $\sum_{j \in \mathbb{H}} \mathtt{dispersed}_j + |\mathbb{C}| \geq n - f$.
- Upon receiving $(\mathtt{configure\text{-}output}, \mathtt{sid}, \mathcal{P}_i, \mathcal{P}_j, v')$ from the adversary, do the following: If $(\mathtt{recast\text{-}output}, \mathtt{sid}, i, v)$ has been delivered to $\mathcal{P}_j$, or $\mathcal{P}_i$ is not corrupted, ignore this message. If $v_i = \perp$, then record $a_{i,j} = (1, v')$. If $v_i \neq \perp$, and $\mathtt{C\text{-}counter}_i < f$, then record $a_{i,j} = (1, v')$, and update $\mathtt{C\text{-}counter}_i = \mathtt{C\text{-}counter}_i + 1$. Otherwise, ignore this message. Here v' can be $\perp$.
- Upon receiving $(\mathtt{fetch}, \mathtt{status}, \mathtt{sid})$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure-I** and the **Output Release Procedure-I**, and provide $(\mathtt{fetch}, \mathtt{status}, \mathtt{sid}, \mathcal{P}_i)$ to the adversary.
- Upon receiving $(\mathtt{recast}, \mathtt{sid}, j)$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure-II**.
- Upon receiving $(\mathtt{fetch}, \mathtt{recast\text{-}output}, \mathtt{sid}, j)$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure-II** and the **Output Release Procedure-II**, and provide $(\mathtt{fetch}, \mathtt{recast\text{-}output}, \mathtt{sid}, j)$ to the adversary.

Input Submission Procedure-I: If $\mathtt{dispersed}_i = 0$, and $(\mathtt{disperse}, \mathtt{sid}, v')$ has been provided by $\mathcal{P}_i$, do the following: (1) Update $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{input}} - 1$; (2) If $\mathsf{D}_i^{\mathtt{input}} = 0$, update $\mathtt{dispersed}_i = 1$, and record $v_i = v'$.
Output Release Procedure-I: If $\sum_{j \in [n]} \mathtt{dispersed}_j \geq n - f$, update $\mathsf{done}_i = 1$; if $\mathsf{done}_i = 1$, do the following: (1) Update $\mathsf{D}_i^{\mathtt{output}} = \mathsf{D}_i^{\mathtt{output}} - 1$; (2) If $\mathsf{D}_i^{\mathtt{output}} = 0$, set $(\mathtt{disperse\text{-}done}, \mathtt{sid})$ to be sent to $\mathcal{P}_i$.
Input Submission Procedure-II: If $\mathsf{done}_j = 1$, and $(\mathtt{recast}, \mathtt{sid}, j)$ has been provided by $\mathcal{P}_i$, do the following: (1) Update $\mathsf{D}_{i,j}^{\mathtt{input}} = \mathsf{D}_{i,j}^{\mathtt{input}} - 1$; (2) If $\mathsf{D}_{i,j}^{\mathtt{input}} = 0$, update $\mathsf{requested}_{i,j} = 1$.
Output Release Procedure-II: If $a_{j,i} = (1, v') \wedge v' \neq \perp$, or $\sum_{i \in \mathbb{H}} \mathsf{requested}_{i,j} \geq n - f$, do the following: (1) Update $\mathsf{D}_{i,j}^{\mathtt{output}} = \mathsf{D}_{i,j}^{\mathtt{output}} - 1$; (2) if $\mathsf{D}_{i,j}^{\mathtt{output}} = 0$, do the following: If $a_{j,i} = (1, v')$, then set $(\mathtt{recast\text{-}output}, j, v')$ to be sent to $\mathcal{P}_i$. Otherwise, if $v_j \neq \perp$, then set $(\mathtt{recast\text{-}output}, j, v_j)$ to be sent to $\mathcal{P}_i$.

Fig. 5. The somewhat-good multi-dealer information dispersal functionality $\mathcal{F}_{\mathsf{SMID}}$

the dealer was honest or not. Its security guarantee is in the flavor of reliable broadcast [14], and for similar reasons, dispersing ℓ-bit value via AVID will incur at least $\mathcal{O}(\ell n + n^2)$ bit cost [4]. Considering concurrent n instances of AVID

where each node acts as a dealer, the communication cost is cubic. To eliminate the cubic communication, we relinquish all security guarantees for the values dispersed by malicious dealers.

With this weakening, we may define the ideal functionality of SMID as follows: it stores every value provided by each node, and whenever an honest node requests the value stored by another honest node, the functionality returns that value. However, the adversary can specify what will be returned when someone attempts to retrieve a value "stored" by a corrupted node. The asynchronous nature and adaptive corruption introduce subtleties that require further weakening of the functionality to make it easy to implement.

One significant subtlety is that the adversary can essentially alter the value retrieved by an honest node, even when the value was dispersed by another honest node. To see this, consider the best-effort dispersal, where an honest node $\mathcal{P}_i$ sends fragments to all other nodes. $\mathcal{P}_i$ will consider the fragments have been delivered to honest nodes if $\mathcal{P}_i$ receives responses from $n - f$ distinct nodes. However, among the responses, there might be only $n - 2f$ of them from honest nodes, while up to f honest but "unfortunate" nodes have not received the corresponding fragments. When the network decides to retrieve the value dispersed by $\mathcal{P}_i$, an adaptive adversary can corrupt $\mathcal{P}_i$ and send different fragments to those f "unfortunate" nodes, causing them to recast a different value.

Considering this subtlety and many others, we formulate the ideal functionality $\mathcal{F}_{\mathsf{SMID}}$ in Fig. 5. Specifically, we allow the adversary to alter a retrieved value for up to f honest nodes (as captured by the `configure-output` instructions). We also permit the adversary to send a `configure-done` instruction, which represents the adversary's capability of allowing some honest nodes to finish the dispersal phase earlier than others. Nonetheless, this ideal functionality guarantees the properties of SMID.

3.2 Details of the SMID protocol

We now present SMID. The detailed procedure for SMID can be found in Algorithm 1. The only cryptographic tool in the construction is a vector commitment scheme that helps the network identify the correct fragments sent by an honest sender. The technique is standard and has been employed in many previous designs [43, 49]. The details of SMID-dispersal and SMID-recast are as follows:

1. *Fragment phase* (lines 1-5). When an honest node $\mathcal{P}_i$ receives a valid input value v, it initially computes the codewords of value v using EC.Encode. Subsequently, it computes the corresponding vector commitment and position proofs. $\mathcal{P}_i$ then disperses its input value v through FRAGMENT messages, which include a vector commitment, a codeword, and a position proof.
2. *Ok phase* (lines 5-9). For any honest node $\mathcal{P}_i$, upon receiving a valid FRAGMENT message from $\mathcal{P}_j$, it records the content of the message as $fragment[j]$ and subsequently sends an OK back to $\mathcal{P}_j$.
3. *Completed phase* (lines 10-13). For any honest node $\mathcal{P}_i$, upon receiving $n - f$ OK messages from distinct nodes, it multicasts a COMPLETED message.

Algorithm 1 The Π_{SMID} protocol with identify id for $\mathcal{P}_i$

Initialize: $fragment[j] \leftarrow \perp$ for $j \in [n]$

1: **upon** receiving input v_i s.t. $\mathsf{Predicate}(v_i) = true$ **do** ▷ SMID-dispersal(id, v_i)
2: Let $\boldsymbol{c} := [c_1, c_2, \ldots, c_n] \leftarrow \mathsf{EC.Encode}(v_i, n, f+1)$
3: Let $\mathsf{vc} \leftarrow \mathsf{VCom}(\boldsymbol{c})$
4: **for** $j \in [n]$, $\pi_j \leftarrow \mathsf{Open}(\mathsf{vc}, c_j, j)$
5: **send** $(\textsc{Fragment}, \mathsf{vc}, c_j, \pi_j)$ to $\mathcal{P}_j$ for every $j = 1, \ldots, n$
6: **upon** receiving $(\textsc{Fragment}, \mathsf{vc}, c_i, \pi_i)$ from $\mathcal{P}_j$ for the first time **do**
7: **if** $\mathsf{VerifyOpen}(\mathsf{vc}, c_i, i, \pi_i) = 1$ **then**
8: $fragment[j] \leftarrow (\mathsf{vc}, c_i, \pi_i)$
9: **send** Ok to $\mathcal{P}_j$
10: **upon** receiving Ok from $\mathcal{P}_j$ for the first time **do**
11: $S_{ok} \leftarrow \mathsf{S}_{ok} \cup \{j\}$
12: **if** $|S_{ok}| = n - f$ **then**
13: **multicast** Completed to all
14: **upon** receiving Completed from $\mathcal{P}_j$ for the first time **do**
15: $S_{comp} \leftarrow S_{comp} \cup \{j\}$
16: **if** $|S_{comp}| = n - f$ **then**
17: **return** $(\texttt{disperse-done}, \mathsf{id})$
18: **upon** receiving input s **do** ▷ SMID-recast(id, s)
19: **if** $fragment[s] := (\mathsf{vc}, c_i, \pi_i)$ **then**
20: **multicast** $(\textsc{Recast}, s, \mathsf{vc}, c_i, \pi_i)$ to all
21: **upon** receiving $(\textsc{Recast}, s, \mathsf{vc}', c_j, \pi_j)$ from $\mathcal{P}_j$ for the first time **do**
22: **if** $\mathsf{vc} = \mathsf{vc}'$ and $\mathsf{VerifyOpen}(\mathsf{vc}, c_j, j, \pi_j) = 1$ **then**
23: $Recast_s \leftarrow Recast_s \cup \{(j, c_j)\}$
24: **upon** $|Recast_s| = f + 1$ **do**
25: $v_s \leftarrow \mathsf{EC.Decode}(Recast_s, n, f+1)$
26: **return** $(\texttt{recast-output}, s, v_s)$

4. *Output phase* (lines 14-17). For any honest node $\mathcal{P}_i$, upon receiving $n - f$ Completed messages from distinct nodes, it returns $(\texttt{disperse-done}, \mathsf{id})$.
5. *recast phase* (lines 18-26). For any honest node $\mathcal{P}_i$, if it decides to recast $\mathcal{P}_s$'s input, it first checks whether $fragment[s] \neq \perp$. If yes, it multicasts it via a Recast message and then waits for $f + 1$ valid Recast messages from distinct nodes. Afterward, it decodes these received codewords and outputs the result. If $fragment[s] = \perp$, then the honest node doesn't produce any output when invoking $\mathsf{SMID\text{-}recast}(\mathsf{id}, s)$.

In the following theorem, we establish that Π_{smid} can UC-realizes $\mathcal{F}_{\mathsf{SMID}}$, while a formal proof is deferred to Appendix D.

Theorem 1. *Assuming the underlying hash function is collision resistant, the protocol Π_{SMID} in Algorithm 1 UC-realizes $\mathcal{F}_{\mathsf{SMID}}$, in the presence of a computationally bounded and adaptive adversary who may corrupt up to $f < n/3$ parties.*

Complexity analysis. In SMID-dispersal, each honest node can send at most $\mathcal{O}(n)$ Fragment messages, $\mathcal{O}(n)$ Ok messages, and $\mathcal{O}(n)$ Completed mes-

sages. Here, the size of the input value v is ℓ, and the size of a FRAGMENT is $\mathcal{O}(\ell/n + \log n\lambda)$, while both OK and COMPLETED messages have a size of $\mathcal{O}(1)$. Consequently, the message complexity is $\mathcal{O}(n^2)$, and the communication complexity is $\mathcal{O}(n\ell + n^2 \log n\lambda)$. In $\mathsf{SMID\text{-}recast}$, for any single input index s, each honest node sends at most $\mathcal{O}(n)$ RECAST messages, where the size of a RECAST is $\mathcal{O}(\ell/n + \log n\lambda)$. As a result, the message complexity is $\mathcal{O}(n^2)$, and the communication complexity is $\mathcal{O}(n\ell + n^2 \log n\lambda)$.

Information-theoretic instantiation. In Algorithm 1, we leverage a vector commitment such that the construction Π_{smid} is computationally secure. If each node simply multicasts its input value, instead of doing information dispersal, the vector commitment scheme is no longer needed. The resulting protocol will be information-theoretically secure, with the communication complexity of $\mathcal{O}(n^2\ell)$, which may be good enough for small input sizes.

4 Synchronized Multi-valued Broadcast

In this section, we introduce our main new ingredient of MVBA: the Synchronized Multi-valued Broadcast (SMB), which is a generalization of Synchronized Binary-value Broadcast (SBV), a core component of a classic ABA protocol [46].

4.1 Overview of SMB

Intuition and a property-based definition. Assume that n nodes have dispersed their inputs through $\mathcal{F}_{\mathsf{SMID}}$. When the network is trying to recast a value, it could result in $f+1$ honest nodes having the correct value while other f honest nodes have arbitrary values provided by the adversary; thus, there could be up to $f + 1$ different values held by honest nodes. Looking ahead, in our MVBA construction, we hope the network will agree on one valid value. SMB is the tool we utilize to "winnow" the values, such that only a few values are left after SMB. At the same time, we hope that, after SMB, the nodes have been "synchronized" about the input values, *i.e.*, if an honest node terminates on a single value val, then this value must appear in any other honest node's output.

Specifically, in SMB with n nodes, each node $\mathcal{P}_i$ participates and may be provided with some input v_i (which could initially be empty for some node when he joins an SMB instance), and finally, each $\mathcal{P}_i$ shall output a set of values val_i. We require the following properties:

- **Justification.** If $\mathcal{P}_i$ is an honest node, then for every $v \in val_i$, v is the input of some honest node.
- **Termination.** If at least $n - 2f$ honest nodes have the same input value v, then every honest node $\mathcal{P}_i$ will terminate and output a set val_i.
- **Obligation.** If at least $n - 2f$ honest nodes have the same input value v, then the set val_i returned by an honest node $\mathcal{P}_i$ is not empty, and $|val_i| \leq 2$.
- **Validity.** If at least $n-2f$ honest nodes have the same input value v, and $\mathcal{P}_i$ and $\mathcal{P}_j$ are honest nodes with $|val_i| \leq |val_j|$, then $val_i \subset val_j$. Additionally, if $|val_i| = 2$, then $v \in val_i$.

Construction Overview. Our construction can be viewed as a generalization of the SBV [46], which exhibits the following functionality: each node inputs a binary value and outputs a set of binary values. The main challenge of SMB lies in the *multi-valued* setting, where there could be many different input values, whereas SBV is designed to handle only two possible inputs. To address this, we introduce a "filter" mechanism to reduce the number of values, which works as follows: every node multicasts its input and will echo the values sent by at least $f+1$ nodes; then, only the values echoed by at least $n-f$ nodes will be processed after the filter. In the case that $f+1$ of all honest nodes share the same input value, it is easy to argue that there are at most two values after filtering.

Assume v_1 and v_2 are the values left. Now, different honest nodes may have $\{v_1\}$, $\{v_2\}$, or $\{v_1, v_2\}$, while their outputs are supposed to be "inclusive". In other words, we should avoid the case where an honest node outputs $\{v_1\}$ while another honest node outputs $\{v_2\}$. We adopt the techniques from SBV to guarantee the inclusion property, which works as follows: First, each node will maintain a local set of candidate values. Then, each node will endorse exactly one candidate value through an AUX message. Finally, an honest node will output a subset of candidate values endorsed by $n-f$ AUX messages. If an honest node outputs a single value v_1, there must be $n-f$ AUX messages carrying v_1. Consequently, every other node must include v_1 in its output, ensuring its inclusion.

Comparison with MV-broadcast [47]. Another generalization to SBV, known as multi-valued validated all-to-all broadcast (MV-broadcast), was introduced in [47]. Different from our SMB, MV-broadcast ensures all the above properties without the condition that at least $n-2f$ honest nodes have the same input value. Therefore, MV-broadcast is stronger than SMB. However, the communication complexity of MV-broadcast can be as high as $\mathcal{O}(n^3\ell)$, assuming the size of the input value is ℓ, while we can have an SMB with quadratic communication.

4.2 Ideal functionality of SMB $\mathcal{F}_{\mathsf{SMB}}$

We formulate SMB as an ideal functionality in Fig. 6. Particularly, since each output value v nominated by the adversary must fulfill the requirement that $|\mathbb{J}_v| + |\mathbb{C}| \geq n-2f$, which means that at least $n-3f$ honest nodes input v, thus implying the *justification*. *Termination* is implied by the fact that the `terminate` will be updated to be 1 whenever there are $n-2f$ honest inputs. *Obligation* is from the fact that the adversary is only allowed to nominate the third output when there are no $f+1$ honest nodes having the same inputs. *Validity* directly follows how the functionality returns the output to each node.

4.3 Details of the SMB protocol

According to the definition of SMB, except for the Justification property, all other properties are satisfied under the condition that at least $n-2f$ honest nodes have the same input value. Although it is a weaker version of MV-broadcast, it is sufficient for us to build our MVBA in the Section 6. In this section, we provide a construction for SMB with IT-security, and the detailed procedure for SMB can be found in Algorithm 2. Here is a comprehensive description:

Functionality $\mathcal{F}_{\mathsf{SMB}}$

$\mathcal{F}_{\mathsf{SMB}}$ proceeds as follows, running with $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ and the adversary. At the first activation, verify $(\mathtt{sid}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ for the session ID $\mathtt{sid}$ and the set of nodes. Initialize: $\mathtt{participated}_i = 0$, $v_i = \perp$, $\mathsf{subset}_i = \perp$ and $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{output}} = 1$ for all $i \in [n]$; $\mathsf{terminate} = 0$, $\mathsf{singleInx} = \perp$, and $\hat{v}_1 = \hat{v}_2 = \hat{v}_3 = \perp$. Let $\mathbb{H}$ be the set of "so-far-honest" nodes, and $\mathbb{C}$ be the set of corrupted nodes.

- Upon receiving $(\mathtt{delay}, \mathtt{sid}, \mathcal{P}_i, \mathtt{type}, D)$ from the adversary for any $i \in [n]$, if $\mathtt{type} \in \{\mathtt{input}, \mathtt{output}\}$ and $D \in \mathbb{Z}$ represented in unary notation, then update $\mathsf{D}_i^{\mathtt{type}} = \max\{1, \mathsf{D}_i^{\mathtt{type}} + D\}$, and provide $(\mathtt{delay\text{-}set}, \mathtt{sid})$ to the adversary.
- Upon receiving $(\mathtt{input}, \mathtt{sid}, v')$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and provide $(\mathtt{input}, \mathtt{sid}, v', \mathcal{P}_i)$ to the adversary.
- Upon receiving $(\mathtt{terminate}, \mathtt{sid}, v' \neq \perp)$ from the adversary, if $\mathsf{terminate} = 0$, update $\mathsf{terminate} = 1$; Use $\mathbb{J}_v$ to denote the set $\{j \in \mathbb{H}, v_j = v\}$ for any $v \neq \perp$.
  - if $\hat{v}_1 = \perp$, and $|\mathbb{J}_{v'}| + |\mathbb{C}| \geq n - 2f$, update $\hat{v}_1 = v'$;
  - if $\hat{v}_2 = \perp$, $v' \neq \hat{v}_1$, and there exists c_1, c_2, such that $c_1 + c_2 = |\mathbb{C}|$, $|\mathbb{J}_{\hat{v}_1}| + c_1 \geq n - 2f$, and $|\mathbb{J}_{v'}| + c_2 \geq n - 2f$, update $\hat{v}_2 = v'$;
  - if $v' \neq \hat{v}_1 \neq \hat{v}_2$, $\sum_{j \in \mathbb{H}} \mathtt{participated}_j \geq n - f$, and there is no v^* s.t. $v_j = v^*$ for at least $n - 2f$ different $j \in \mathbb{H}$, then then update $\hat{v}_3 = v'$.
- Upon receiving $(\mathtt{decide\text{-}output}, \mathtt{sid}, \mathcal{P}_i, \mathsf{subset} \subset \{1, 2, 3\})$ from the adversary, if $\mathsf{terminate} = 1$, and $\mathcal{P}_i$ has not received its output, update $\mathsf{subset}_i = \mathsf{subset}$.
- Upon receiving $(\mathtt{fetch}, \mathtt{sid})$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and the **Output Release Procedure**, and provide $(\mathtt{fetch}, \mathtt{sid}, \mathcal{P}_i)$ to the adversary.

Input Submission Procedure: If $\mathtt{participated}_i = 0$, and $(\mathtt{input}, \mathtt{sid}, v')$ was the latest input provided by $\mathcal{P}_i$, do the following:

- Update $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{input}} - 1$;
- If $\mathsf{D}_i^{\mathtt{input}} = 0$, update $\mathtt{participated}_i = 1$, and record $v_i = v'$.

Output Release Procedure: If (1) $\mathsf{terminate} = 1$, or (2) there exists a subset $\mathbb{J} \subset \mathbb{H}$ such that $|\mathbb{J}| \geq n - 2f$, and $\exists\ v = v_j \neq \perp$ for all $j \in \mathbb{J}$, do the following:

- If (2) and $\hat{v}_1 = \perp$, update $\hat{v}_1 = v$ and $\mathsf{terminate} = 1$. Provide $(\mathtt{leakage}, \mathtt{sid}, \hat{v}_1)$ to the adversary.
- If $\hat{v}_1 \neq \perp$, do the following:
  - Update $\mathsf{D}_i^{\mathtt{output}} = \mathsf{D}_i^{\mathtt{output}} - 1$;
  - if $\mathsf{D}_i^{\mathtt{output}} = 0$, do the following: if $\mathsf{subset}_i = \{k^*\}$ and $\mathsf{singleInx} = \perp$, update $\mathsf{singleInx} = k^*$. If $\mathsf{subset}_i \neq \perp$, $\hat{v}_j \neq \perp, \forall\ j \in \mathsf{subset}_i$, and $\mathsf{singleIdx} \in \mathsf{subset}_i$, set $(\mathtt{output}, \mathtt{sid}, \{\hat{v}_j\}_{j \in \mathsf{subset}_i})$ to be sent to $\mathcal{P}_i$; otherwise, update $\mathsf{singleInx} = 1$ if it is $\perp$, and set $(\mathtt{output}, \mathtt{sid}, \hat{v}_{\mathsf{singleInx}})$ to be sent to $\mathcal{P}_i$.

Fig. 6. The synchronized multi-valued broadcast functionality $\mathcal{F}_{\mathsf{SMB}}$

1. *Filter phase* (lines 1-2). All honest nodes multicast their input via Filter messages.

2. *FilterEcho phase* (lines 3-4). For any honest node $\mathcal{P}_i$, if it receives $n - 2f$ Filter messages carrying the same value v' from distinct nodes, then it will multicast the value v' via a FilterEcho message.

Algorithm 2 The SMB protocol Π_{SMB} with identifier id for $\mathcal{P}_i$

$values_i \leftarrow \{\}$, and $auxs_i \leftarrow \{\}$
1: **upon** receiving input v_i **do**
2: **multicast** (FILTER, id, v_i) to all
3: **upon** receiving (FILTER, id, v') from $n-2f$ distinct nodes **do**
4: **multicast** (FILTERECHO, id, v') to all
5: **upon** receiving (FILTERECHO, id, v) from $n-f$ distinct nodes **do**
6: **multicast** (VAL, id, v) to all
7: **upon** receiving (VAL, id, v) from $n-2f$ distinct nodes **do**
8: **if** (VAL, id, v) not yet sent **then**
9: **multicast** (VAL, id, v) to all
10: **upon** receiving (VAL, id, v) from $n-f$ distinct nodes and $v \notin values_i$ **do**
11: $values_i \leftarrow values_i \cup \{v\}$
12: **wait** until $values_i \neq \emptyset$
13: **multicast** (AUX, id, ω), where $\omega \in values_i$
14: **upon** receiving (AUX, id, ω) from $\mathcal{P}_j$ for the first time **do**
15: **if** weight_ω has not been initialized **then**
16: Let $\mathsf{weight}_\omega \leftarrow 0$
17: $auxs_i \leftarrow auxs_i \cup \{\omega\}$, and $\mathsf{weight}_\omega = \mathsf{weight}_\omega + 1$
18: **wait** until $\sum_{\omega \in val_i} \mathsf{weight}_\omega \geq n-f$, for $val_i = values_i \cap auxs_i$
19: **return** val_i

3. *Val phase* (lines 5-9). For any honest node $\mathcal{P}_i$, if it receives $n-f$ FILTERECHO messages carrying the same value v from distinct nodes, then it will multicast the value v via a VAL message. Besides, upon receiving $n-2f$ VAL messages carrying the same value v from distinct nodes and not having multicast (VAL, id, v), it will multicast a VAL message along with the value v.
4. *Aux phase* (lines 10-13). For any honest node $\mathcal{P}_i$, if it receives $n-f$ VAL messages carrying the same value v from distinct nodes, then it will add the value v to the $value_i$ set. If the $value_i$ set is not empty, it will multicast (AUX, id, ω) to all, where $\omega \in values_i$.
5. *Output phase* (lines 14-19). For any honest node $\mathcal{P}_i$, it counts the number weight_w of AUX messages carrying the value w, and records all received w in the set aux_i. $\mathcal{P}_i$ outputs the intersection val_i of aux_i and $values_i$, only when it has received at least $n-f$ AUX messages that carry values already in $values_i$, i.e., $\sum_{w \in val_i} \mathsf{weight}_w >= n-f$.

In our protocol, it is crucial to emphasize that each honest node $\mathcal{P}_i$ can multicast at most two FILTERECHO messages. Furthermore, if at least $n-2f$ honest nodes multicast the same value via FILTER, then every honest node $\mathcal{P}_i$ is allowed to multicast at most two VAL messages, resulting in at most two distinct VAL messages being sent among all honest nodes. Additionally, the set of $value_i$ can be updated even if $\mathcal{P}_i$ multicasts a AUX message. Regarding security, we establish the following result, with the full proof provided in Appendix D.

Theorem 2. *The protocol Π_{SMB} in Algorithm 2 perfectly UC-realizes $\mathcal{F}_{\mathsf{SMB}}$, in the presence of any adaptive corruptions up to $f < n/3$ nodes.*

Complexity analysis. In Algorithm 2, each honest node has the capability to multicast Filter, FilterEcho, Val, and Aux messages, each of which has a size of ℓ corresponding to the input value v of size ℓ. In the worst case, each honest node multicasts one Filter message, two different FilterEcho messages, at most three different Val messages, and one Aux message. Each party processes these messages from other parties accordingly. It is evident that each node multicasts each type of message only $\mathcal{O}(1)$ times in the worst case, resulting in a message complexity of $\mathcal{O}(n^2)$ and a communication complexity of $\mathcal{O}(n^2\ell)$.

5 Asynchronous Reliable Consensus

In this section, we introduce our last component of MVBA: Asynchronous Reliable Consensus (ARC). ARC was first studied and constructed by Blum et al. [12] for different purposes. In contrast, our protocol is information-theoretically secure and deterministic, designed to resist an adaptive adversary, whereas the ARC in [12] is computationally secure, randomized, and cannot withstand the adaptive adversary considered in this work. Our construction is similar to the one-sided voting introduced in [53]; however, it only considers unitary inputs [8]. We defer the details of ARC construction to Appendix F.

Formally, in the ARC protocol, each node takes a value as input and collaboratively decides on a common value. The protocol is designed to satisfy the following properties, except with negligible probability:

- **Totality.** If an honest node outputs v, then all honest nodes output v'.
- **Agreement.** If any two honest nodes output v and v' respectively, then $v = v'$.
- **Validity.** If all honest nodes with the same input value v, then all honest nodes output v.
- **Justification.** If an honest node outputs v, then v is the input of at least $n - 2f$ honest node.

5.1 Ideal Functionality of ARC $\mathcal{F}_{\mathsf{ARC}}$

We formulate the ideal functionality $\mathcal{F}_{\mathsf{ARC}}$ in Fig. 7. Note that $\mathcal{F}_{\mathsf{ARC}}$ captures all properties of ARC. Particularly, due to how the functionality returns output, it ensures that as long as an honest node outputs v, all other honest nodes eventually output v, implying *totality and agreement*. If all honest nodes have the same input value, they will output that value, thereby satisfying *validity*. Additionally, the protocol guarantees that the output value must be the input of at least $n-2f$ honest nodes, fulfilling the requirement of *justification*. Regarding security, we establish the following theorem, whose full proof is deferred to Appendix F.

Theorem 3. *The protocol Π_{ARC} in Algorithm 5 perfectly UC-realizes $\mathcal{F}_{\mathsf{ARC}}$, in the presence of adaptive corruptions of up to $f < n/3$ nodes.*

[8] A concurrent work [31] also introduces an ARC protocol that achieves the same goal as our work, while we also provide the ARC ideal functionality.

Functionality $\mathcal{F}_{\mathsf{ARC}}$

$\mathcal{F}_{\mathsf{ARC}}$ proceeds as follows, running with $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ and the adversary. At the first activation, verify $(\mathtt{sid}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ for the session ID $\mathtt{sid}$ and the set of nodes. Initialize: $\mathtt{participated}_i = 0$, $v_i = \perp$ and $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{output}} = 1$ for all $i \in [n]$; $y = \perp$.

- Upon receiving $(\mathtt{delay}, \mathtt{sid}, \mathcal{P}_i, \mathtt{type}, D)$ from the adversary for any $i \in [n]$, if $\mathtt{type} \in \{\mathtt{input}, \mathtt{output}\}$ and $D \in \mathbb{Z}$ represented in unary notation, then update $\mathsf{D}_i^{\mathtt{type}} = \max\{1, \mathsf{D}_i^{\mathtt{type}} + D\}$, and provide $(\mathtt{delay\text{-}set}, \mathtt{sid})$ to the adversary.
- Upon receiving $(\mathtt{input}, \mathtt{sid}, v')$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and provide $(\mathtt{input}, \mathtt{sid}, v', \mathcal{P}_i)$ to the adversary.
- Upon receiving $(\mathtt{fetch}, \mathtt{sid})$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and the **Output Release Procedure**, and provide $(\mathtt{fetch}, \mathtt{sid}, \mathcal{P}_i)$ to the adversary.

Input Submission Procedure: If $\mathtt{participated}_i = 0$, and $(\mathtt{input}, \mathtt{sid}, v')$ has been provided by $\mathcal{P}_i$, do the following:

- Update $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{input}} - 1$;
- If $\mathsf{D}_i^{\mathtt{input}} = 0$, update $\mathtt{participated}_i = 1$, and record $v_i = v'$.

Output Release Procedure: If $\exists\ \mathbb{I} \subset [n]$ such that $|\mathbb{I}| \geq n - f$ and $\exists\ v = v_i \neq \perp$ for all $i \in \mathbb{I}$, do the following:

- If $y = \perp$, set $y = v$, provide $(\mathtt{output}, \mathtt{sid}, y)$ to the adversary;
- Update $\mathsf{D}_i^{\mathtt{output}} = \mathsf{D}_i^{\mathtt{output}} - 1$;
- if $\mathsf{D}_i^{\mathtt{output}} = 0$, set $(\mathtt{output}, \mathtt{sid}, y)$ to be sent to $\mathcal{P}_i$.

Fig. 7. The aynchronous reliable consensus functionality $\mathcal{F}_{\mathsf{ARC}}$

Complexities of ARC protocol. In Algorithm 5, each honest node can multicast DIFFUSION and ECHO messages, each having a size of ℓ corresponding to the input value v of size ℓ. As a result, the message complexity is $\mathcal{O}(n^2)$, and the communication complexity is $\mathcal{O}(n^2\ell)$.

6 Near-optimal Hash Based MVBA

In this section, we present a protocol for achieving MVBA with adaptive security, leveraging only hash functions. Our approach attains subcubic communication complexity of $\mathcal{O}(n\ell + n^2 \log n\lambda + \kappa n^2\lambda)$, and optimal fault tolerance with $n \geq 3f + 1$, where ℓ represents the size of the input values, κ is a statistical security parameter, λ denotes the size of a hash value, and n refers to the number of nodes. Specifically, if the input size ℓ exceeds $\mathcal{O}((\kappa + \log n)n\lambda)$, our MVBA protocol achieves $\mathcal{O}(n\ell)$ optimal communication complexity.

6.1 Overview of our MVBA protocol

As shown in Fig.2 in the introduction, our construction revolves around the properties of the Somewhat-good Multi-dealer Information Dispersal, particularly focusing on its validity. The fundamental ideas of our MVBA protocol are

Algorithm 3 The MVBA protocol Π_{MVBA} for $\mathcal{P}_i$

Initialize $\mathsf{value}_z = \mathsf{vc}_z = \mathsf{vc}_{z,a} = \perp$, $\pi_{z,i} = c_{z,i} = \perp$, for $z \in [\kappa]$, $a \in \{1,2\}$, and $i \in [n]$; $\mathsf{terminate} = 0$ and $\mathsf{Fragments} = \emptyset$; keep sending $(\texttt{fetch}, \star)$ message to each ideal functionality until receiving the desired output.

upon receiving input v_i s.t. $\mathsf{Predicate}(v_i) = true$ **do**
  send $(\texttt{disperse}, \texttt{sid}, v_i)$ to $\mathcal{F}_{\mathsf{SMID}}$ ▷ dispersal
upon receiving $(\texttt{disperse-done}, \texttt{sid})$ from $\mathcal{F}_{\mathsf{SMID}}$ **do**
  send $(\texttt{request}, \texttt{sid})$ to $\mathcal{F}_{\mathsf{coin}}^{V}$ ▷ election
upon receiving $(\texttt{output}, \texttt{sid}, s_1, \ldots, s_\kappa) \in [n]^\kappa$ from $\mathcal{F}_{\mathsf{coin}}^{V}$ **do**
  send $(\texttt{recast}, \texttt{sid}, s_z)$ to $\mathcal{F}_{\mathsf{SMID}}$, for each $z \in [\kappa]$ ▷ recast
upon receiving $(\texttt{recast-output}, s_z, v_{s_z})$ from $\mathcal{F}_{\mathsf{SMID}}$ for any $z \in [\kappa]$ **do**
  if $\mathsf{Predicate}(v_{s_z}) = 1$ **then**
    Let $\boldsymbol{c}_z := [c_{z,1}, c_{z,2}, \ldots, c_{z,n}] \leftarrow \mathsf{EC.Encode}(v_{s_z}, n, f+1)$
    record $\mathsf{vc}_z := \mathsf{VCom}(\boldsymbol{c}_z)$ and $\mathsf{value}_z = v_{s_z}$
    for $j \in [n]$, **record** $\pi_{z,j} \leftarrow \mathsf{Open}(\mathsf{vc}_z, c_j, j)$
    send $(\texttt{input}, \texttt{sid}|z, \mathsf{vc}_z)$ to $\mathcal{F}_{\mathsf{SMB}}$ ▷ SMB
upon receiving $(\texttt{output}, \texttt{sid}|z, \mathsf{vset})$ from $\mathcal{F}_{\mathsf{SMB}}$ for any $z \in [\kappa]$ **do**
  if Parse $\mathsf{vset} = \{\mathsf{vc}', \mathsf{vc}''\}$ or $\mathsf{vset} = \{\mathsf{vc}\}$ **then** ▷ vc equals to vc' or vc''
    if $|\mathsf{vset}| = 1$ **then** Let $\mathsf{vc}' \leftarrow \mathsf{vc}$ and $vc'' \leftarrow vc$
    send $(\texttt{input}, \texttt{sid}|z|1, \mathsf{vc}')$ to $\mathcal{F}_{\mathsf{ARC}}$ ▷ ARC
    send $(\texttt{input}, \texttt{sid}|z|2, \mathsf{vc}'')$ to $\mathcal{F}_{\mathsf{ARC}}$
upon receiving $(\texttt{output}, \texttt{sid}|z|a, \mathsf{vc})$ from $\mathcal{F}_{\mathsf{ARC}}$ for any $z \in [\kappa]$ and $a \in \{1,2\}$ **do**
  record $\mathsf{vc}_{z,a} = \mathsf{vc}$
  if have not sent input message $(\texttt{input}, \texttt{sid}|z|a, \star)$ to $\mathcal{F}_{\mathsf{ABA}}$ **then**
    send $(\texttt{input}, \texttt{sid}|z|a, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$ ▷ ABA
upon receiving $(\texttt{output}, \texttt{sid}|z|a, 1)$ from $\mathcal{F}_{\mathsf{ABA}}$ for any $z \in [\kappa]$ and $a \in \{1,2\}$ **do**
  for $z' \in [\kappa]$ and $a' \in \{1,2\}$ **do**
    if have not sent input message $(\texttt{input}, \texttt{sid}|z'|a', \star)$ to $\mathcal{F}_{\mathsf{ABA}}$ **then**
      send $(\texttt{input}, \texttt{sid}|z'|a', 0)$ to $\mathcal{F}_{\mathsf{ABA}}$
wait receiving $(\texttt{output}, \texttt{sid}|z|a, \star)$ from $\mathcal{F}_{\mathsf{ABA}}$ for all $z \in [\kappa]$ and $a \in \{1,2\}$
  Let (z^*, a^*) be the smallest pair such that $\mathcal{F}_{\mathsf{ABA}}$ outputs $(\texttt{output}, \texttt{sid}|z^*|a^*, 1)$
  if $\mathsf{vc}_{z^*,a^*} = \perp$ **then**
    wait receiving $(\texttt{output}, \texttt{sid}|z^*|a^*, \mathsf{vc})$ from $\mathcal{F}_{\mathsf{ARC}}$
      record $\mathsf{vc}_{z^*,a^*} = \mathsf{vc}$
  if $\mathsf{vc}_{z^*} = \mathsf{vc}_{z^*,a^*}$ **then**
    send $(\texttt{fragment}, \texttt{sid}, \mathsf{vc}_{z^*}, c_{z^*,j}, \pi_{z^*,j})$ to $\mathcal{P}_j$ for every $j \in [n]$
  upon receiving $(\texttt{fragment}, \texttt{sid}, \mathsf{vc}', c'_i, \pi'_i)$ such that $\mathsf{vc}_{z^*,a^*} = \mathsf{vc}'$ **do**
    if $\mathsf{VerifyOpen}(\mathsf{vc}', c'_i, i, \pi'_i) = 1$ and have not send $\texttt{forward}$ message **then**
      send $(\texttt{forward}, \texttt{sid}, \mathsf{vc}', c'_i, \pi'_i)$ to all parties; Update $\mathsf{terminate} = 1$
  upon receiving $(\texttt{forward}, \texttt{sid}, \mathsf{vc}', c'_j, \pi'_j)$ from $\mathcal{P}_j$ for the first time **do**
    if $\mathsf{vc}' = \mathsf{vc}_{z^*,a^*}$ and $\mathsf{VerifyOpen}(\mathsf{vc}', c'_j, j, \pi'_j) = 1$ **then**
      Update $\mathsf{Fragments} \leftarrow \mathsf{Fragments} \cup \{(j, c'_j)\}$
      upon $|\mathsf{Fragments}| = f + 1$ **do**
        Update $\mathsf{value}_{z^*} \leftarrow \mathsf{EC.Decode}(\mathsf{Fragments}, n, f+1)$ ▷ recovery
  wait until $\mathsf{value}_{z^*} \neq \perp$ and $\mathsf{terminate} = 1$
    return $(\texttt{output}, \texttt{sid}, \mathsf{value}_{z^*})$ ▷ output

as follows: all honest nodes participate in the SMID-dispersal instance to disperse their own input values, and they wait until the SMID-dispersal has been successfully completed. During this time, the entire network ensures that there exists a set I with at least $n/3$ indexes, and if all honest nodes with index $i(\in I)$ as input, then at least $n-2f$ honest nodes can reconstruct the same value that satisfies the Predicate function. Afterward, a common coin protocol coin is invoked to randomly elect κ nodes $\{\mathcal{P}_{s_1}, \ldots, \mathcal{P}_{s_\kappa}\}$ such that one of the nodes belongs to the set I. Subsequently, all honest nodes participate in the κ SMID-recast subprotocols. For an honest node $\mathcal{P}_i$, if the corresponding SMID-recast subprotocol returns a value, then $\mathcal{P}_i$ inputs the value into the SMB_k. If the SMB_k returns a valid set vset (i.e., $|\mathsf{vset}| \leq 2$), then the element of vset is input to the corresponding ARC protocol. All honest nodes await the output of 2κ ARC instances, and if any ARC protocol produces an output, then 1 is input into the corresponding ABA. For an honest node $\mathcal{P}_i$, if any of the ABA instances outputs 1, then $\mathcal{P}_i$ inputs 0 into the remaining ABA instances that do not have an input. The termination condition for the entire protocol is the termination of the 2κ ABA instances.

6.2 Details of the MVBA protocol

The flow of our MVBA is outlined in Fig. 2. We now provide a detailed description. Specifically, an MVBA instance with identifier id proceeds as follows:

1. *SMID-dispersal phase* (lines 1-2). All honest nodes disperse their input values v_i to the entire network by sending $(\texttt{disperse}, \texttt{sid}, v_i)$ to $\mathcal{F}_{\mathsf{SMID}}$.
2. *Election phase* (lines 3-4). Upon $\mathcal{P}_i$ receiving $(\texttt{disperse-done}, \texttt{sid})$ from $\mathcal{F}_{\mathsf{SMID}}$, indicating that at least $n-2f$ honest nodes have completed their dispersal, $\mathcal{P}_i$ will multicast $(\texttt{request}, \texttt{sid})$ to $\mathcal{F}_{\mathsf{coin}}^V$ to select κ distinct nodes.
3. *SMID-recast phase* (lines 5-6). If $\mathcal{F}_{\mathsf{coin}}^V$ returns κ values $\{s_1, \ldots, s_\kappa\}$, then all honest nodes send $(\texttt{recast}, \texttt{sid}, s_z)$ to $\mathcal{F}_{\mathsf{SMID}}$ for every $z \in [\kappa]$ to attempt to recast the inputs of these nodes $\{\mathcal{P}_{s_1}, \ldots, \mathcal{P}_{s_\kappa}\}$.
4. *SMB phase* (lines 7-12). If honest node $\mathcal{P}_i$ receives $(\texttt{recast-output}, s_z, v_{s_z})$ from $\mathcal{F}_{\mathsf{SMID}}$ for any $z \in [\kappa]$, and the value v_{s_z} satisfies the predicate Predicate, then $\mathcal{P}_i$ computes the corresponding vector commitment vc_z and position proofs $\{\pi_{z,j}\}_{j\in[n]}$. $\mathcal{P}_i$ subsequently sends $(\texttt{input}, \texttt{sid}|z, \mathsf{vc}_z)$ to $\mathcal{F}_{\mathsf{SMB}}$.
5. *ARC phase* (lines 13-17). Suppose $\mathcal{F}_{\mathsf{SMB}}$ returns $(\texttt{output}, \texttt{sid}|z, \mathsf{vset})$ to honest node $\mathcal{P}_i$. If the size of the set vset is no more than 2, then $\mathcal{P}_i$ considers the elements of vset as the input for $\mathcal{F}_{\mathsf{ARC}}$.
6. *ABA phase* (lines 18-25). If $\mathcal{F}_{\mathsf{ARC}}$ returns $(\texttt{output}, \texttt{sid}|z|a, \mathsf{vc})$ to $\mathcal{P}_i$, then he sends $(\texttt{input}, \texttt{sid}|z|a, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$ if he hasn't done so already. Furthermore, if $\mathcal{P}_i$ receives $(\texttt{output}, \texttt{sid}|z|a, 1)$ from $\mathcal{F}_{\mathsf{ABA}}$, then for any $z' \in [\kappa]$ and $a' \in \{1, 2\}$, $\mathcal{P}_i$ sends $(\texttt{input}, \texttt{sid}|z'|a', \star)$ to $\mathcal{F}_{\mathsf{ABA}}$ if he has not sent it yet.
7. *Output phase* (lines 26-42). Wait until $\mathcal{F}_{\mathsf{ABA}}$ returns 2κ values, then, an honest node $\mathcal{P}_i$ outputs the value corresponding to the vc in the output of $\mathcal{F}_{\mathsf{ARC}}$ with identifier $\texttt{sid}|z^*|a^*$, where (z^*, a^*) is the smallest pair among all $\mathcal{F}_{\mathsf{ABA}}$ outputs $(\texttt{output}, \texttt{sid}|z^*|a^*, 1)$. It's possible that some honest nodes have not received the corresponding value. To help these nodes receive it,

two additional types of messages are introduced: `fragment` and `forward`. Specifically, if some honest nodes have the corresponding value, they send a `fragment` to every node. If a node receives a valid `fragment` message for the first time, it sends a `forward` message to all nodes. Then, if some honest nodes have not received the value, they will wait for $f + 1$ valid `forward` messages from distinct nodes. After decoding, they output the decoded value.

Regarding security, we present the following theorem.

Theorem 4. *Assuming the underlying hash function is collision resistant, the protocol Π_{MVBA} in Algorithm 3 UC-realizes $\mathcal{F}_{\mathsf{MVBA}}$ in the $(\mathcal{F}_{\mathsf{coin}}^{[n]^\kappa}, \mathcal{F}_{\mathsf{ABA}}, \mathcal{F}_{\mathsf{SMID}}, \mathcal{F}_{\mathsf{ARC}}, \mathcal{F}_{\mathsf{SMB}})$-hybrid model, in the presence of a computationally bounded and adaptive Byzantine adversary who may corrupt up to $f < n/3$ nodes.*

Proof (sketch). For any adversary $\mathcal{A}$, a simulator $\mathcal{S}$ can be constructed as follows: it runs a copy of $\mathcal{A}$, simulates all subroutines including $\mathcal{F}_{\mathsf{coin}}^{[n]^\kappa}, \mathcal{F}_{\mathsf{ABA}}, \mathcal{F}_{\mathsf{SMID}}, \mathcal{F}_{\mathsf{ARC}}$, and $\mathcal{F}_{\mathsf{SMB}}$, and honestly plays the roles of all honest nodes after learning their input values from $\mathcal{F}_{\mathsf{MVBA}}$. In general, $\mathcal{S}$ forwards the messages between $\mathcal{A}$ and the environment, and adjusts the delay counters based on the simulation execution influenced by $\mathcal{A}$. $\mathcal{S}$ runs the copies of $\mathcal{F}_{\mathsf{ABA}}, \mathcal{F}_{\mathsf{SMID}}, \mathcal{F}_{\mathsf{ARC}}$, and $\mathcal{F}_{\mathsf{SMB}}$ honestly. For $\mathcal{F}_{\mathsf{coin}}^{[n]^\kappa}$, $\mathcal{S}$ runs a modified version where a node P_s, which has finished dispersal in $\mathcal{F}_{\mathsf{SMID}}$, is included in the output.

It is straightforward to argue that, from the perspective of $\mathcal{A}$, the simulated execution by $\mathcal{S}$ is indistinguishable from a real execution. It remains to show that the output of the simulated execution matches the output of the ideal functionality $\mathcal{F}_{\mathsf{MVBA}}$. Note that $\mathcal{F}_{\mathsf{coin}}^{[n]^\kappa}$ only produces output when at least $f + 1$ nodes have queried, which implies that at least one honest node has received $(\texttt{disperse-done}, \texttt{sid})$ from $\mathcal{F}_{\mathsf{SMID}}$. According to the specification of $\mathcal{F}_{\mathsf{SMID}}$, before $\mathcal{F}_{\mathsf{coin}}^{[n]^\kappa}$ returns, there exists a subset $\mathbb{I}$ such that $|\mathbb{I}| \geq f + 1$. For each $i \in \mathbb{I}$, at least $n - 2f$ honest nodes can reconstruct the initial input value v_i of P_i.

Moreover, in $\mathcal{F}_{\mathsf{coin}}^{[n]^\kappa}$, an index $z \in \mathbb{I}$ is included in the coin output. This implies that at least $f + 1$ honest nodes will have the same input value v_z for the same SMB instance with identifier $\texttt{sid}|z$. According to the description of $\mathcal{F}_{\mathsf{SMB}}$, there will be two values, v'_1 and v'_2, such that every honest node P_i will receive a set $val_i \subseteq \{v'_1, v'_2\}$ from $\mathcal{F}_{\mathsf{SMB}}$ with identifier $\texttt{sid}|z$. Moreover, if $|val_j| \leq |val_i|$, then $val_j \subseteq val_i$. Hence, one of the following cases must occur: (1) all honest nodes input v'_1 into the instance with identifier $(\texttt{sid}|z|1)$, or (2) all honest nodes input v'_2 into $\mathcal{F}_{\mathsf{ARC}}$ with identifier $(\texttt{sid}|z|2)$. As a result, all honest nodes will receive the same output from one session of $\mathcal{F}_{\mathsf{ARC}}$, which implies that all honest nodes will input 1 into the corresponding session of $\mathcal{F}_{\mathsf{ABA}}$. For any honest node P_i, once a session of $\mathcal{F}_{\mathsf{ABA}}$ returns 1, it will input 0 into all other $\mathcal{F}_{\mathsf{ABA}}$ sessions if it has not yet provided inputs. Consequently, all $\mathcal{F}_{\mathsf{ABA}}$ sessions can terminate.

According to $\mathcal{F}_{\mathsf{ABA}}$, all nodes will receive the same outputs. For any $\mathcal{F}_{\mathsf{ABA}}$ session, if it outputs $(\texttt{output}, \texttt{sid}|z|a, 1)$, then, according to the protocol description, at least one honest node has received a valid $\mathsf{vc}_{z,a}$ from the corresponding $\mathcal{F}_{\mathsf{ARC}}$ session. Based on the descriptions of $\mathcal{F}_{\mathsf{ARC}}$ and $\mathcal{F}_{\mathsf{SMV}}$, at least one node

holds the corresponding value v. Due to the collision resistance of the hash function, all honest nodes will eventually output the same value. For brevity, the formal proof is deferred to Appendix.G. □

Complexities of MVBA protocol. In Algorithm 3, illustrated in Figure 2, all honest nodes engage in one SMID-dispersal, one election protocol $\mathcal{F}_{\mathsf{coin}}^{V}$, and subsequently, κ instances of SMID-recast, κ instances of SMB, 2κ instances of ARC, and 2κ instances of ABA. The input size of SMID-dispersal is ℓ, while the input size of SMB and ARC is λ. Throughout this paper, we assume the use of an ideal common coin, and thus, we omit its associated cost. The cost performance of a single instance/phase is as follows: (1) both SMID-dispersal and SMID-recast have a message complexity of $\mathcal{O}(n^2)$ and a communication complexity of $\mathcal{O}(n\ell + n^2 \log n\lambda)$; (2) both SMB and ARC exhibit the same cost performance, with a message complexity of $\mathcal{O}(n^2)$ and communication complexity of $\mathcal{O}(n^2\lambda)$; (3) each ABA instance has a message complexity of $\mathcal{O}(n^2)$ and communication complexity of $\mathcal{O}(n^2)$; (4) in the output phase, each honest node sends n `fragment` and n `forward` messages, so the total number of exchanged messages is $\mathcal{O}(n^2)$, and the bit communication cost is $\mathcal{O}(n\ell + n^2 \log n\lambda)$.

As a result, the overall message complexity of MVBA is $\mathcal{O}(\kappa n^2)$, and the communication complexity is $\mathcal{O}(n\ell + n^2 \log n\lambda + \kappa n^2\lambda)$.

6.3 MVBA with information-theoretic security

As discussed in Section 3, if each node multicasts its input value without executing the information dispersal, the vector commitment scheme becomes unnecessary. Then, the SMID protocol would be information-theoretically secure with a communication complexity of $\mathcal{O}(n^2\ell)$. Consequently, by making minor modifications to Algorithm 3 as follows: employing the information-theoretically-secure SMID protocol and skipping the *SMID-recast* phase after the *election phase*, all honest nodes directly participate in the *SMB phase* with the received actual data from SMID. The output of SMB is then taken as the input of ARC in the *ARC phase*. In the *output phase*, there is no need to send `fragment` and `forward` messages; the output of ARC serves as the final output. Based on these modifications, we have the following theorem.

Theorem 5. *Following the above described modifications in Section 3, the adjusted protocol* Π_{MVBA} *UC-realizes* $\mathcal{F}_{\mathsf{MVBA}}$ *in the* $(\mathcal{F}_{\mathsf{coin}}^{[n]^\kappa}, \mathcal{F}_{\mathsf{ABA}}, \mathcal{F}_{\mathsf{IT-SMID}}, \mathcal{F}_{\mathsf{ARC}}, \mathcal{F}_{\mathsf{SMB}})$*-hybrid model. This holds in the presence of an unbounded and adaptive Byzantine adversary with the ability to corrupt up to* $f < n/3$ *nodes.*

7 Optimal Hash-based ACS and Applications to Asynchronous MPC

In this section, we present a direct application of our MVBA to ACS, and then show how we can improve the state-of-art asynchronous MPC using our ACS.

7.1 Optimal Asynchronous Common Subset from MVBA

Our ACS essentially follows Cachin et al.'s framework [16], which builds an ACS protocol from a MVBA protocol using digital signatures (and thus assuming a bare PKI). In the framework, each node firstly multicasts its input value along with its signature for the value, then after collecting $n - f$ signed values from distinct nodes, invokes an MVBA instance with a vector of received signed values, to agree on one vector. It is easy to argue the elegant construction satisfies all security properties of ACS. At a high level, the `termination` and `agreement` of MVBA guarantee the `totality` and `agreement` of ACS, while the external validity of MVBA plus the unforgeability of the underlying signature ensures the validity of ACS. Due to the space limit, we demonstrate the formal construction in the $(\mathcal{F}_{\mathsf{MVBA}}, \mathcal{F}_{\mathsf{PKI}})$-hybrid model in Algorithm 6 in Appendix H, and prove it UC-realizes $\mathcal{F}_{\mathsf{ACS}}$. The full proof is deferred to Appendix H.

Theorem 6. *Assuming the underlying signature scheme satisfies the existential unforgeability, the protocol Π_{ACS} UC-realizes $\mathcal{F}_{\mathsf{ACS}}$ in the $(\mathcal{F}_{\mathsf{MVBA}}, \mathcal{F}_{\mathsf{PKI}})$-hybrid model, against any computationally bounded adaptive Byzantine adversary who may corrupt up to $f < n/3$ nodes.*

Instantiation and complexity analysis. Using Π_{MVBA} in Algorithm 3 (along with our Π_{SMID}, Π_{ARC} and Π_{SMB} protocols, and the ABA protocol from [46]), we then obtain an input protocol in the $(\mathcal{F}_{\mathsf{PKI}}, \mathcal{F}_{\mathsf{coin}})$-hybrid model, while assuming a digital signature scheme and a collision-resistant hash function. As there are hash-based digital signature schemes (for example, [10]), the only computational assumption will be the existence of collision-resistant hash.

Regarding the communication cost, the multicast phase incurs $\mathcal{O}\big((\ell + \lambda)n^2\big)$ bits communication for n nodes each with ℓ-bit input. The input of MVBA is $n - f$ values along with the corresponding signatures, whose length is $\mathcal{O}((\ell + \lambda)n)$. Therefore, the overall communication complexity is $\mathcal{O}\big(\ell n^2 + \lambda n^2 \log n + \kappa n^2 \lambda\big)$.

7.2 Applications to Asynchronous MPC

Next, we show how our ACS could solve the input agreement problem in AMPC.

Overview of offline-online AMPC. Almost all existing AMPC protocols [6,9,13,19,22,24–26,42] follow the offline-online paradigm. As the name suggests, the offline phase is independent of the function and the data to be evaluated and thus can be executed before the actual computation. The correlated randomness generated in the offline phase will be utilized in the subsequent online phase. The online phase will be invoked when the data and the computation task are known, which consists of an **input sub-phase** and a **function evaluation sub-phase**. In the input sub-phase, Each node $\mathcal{P}_i$ on input a private value x_i distributes secret shares of x_i to other nodes. After the execution of this phase, honest nodes should have consistent secret shares of the same set of $n - f$ input values. Then, in the function evaluation phase, the nodes jointly evaluate a function f on the shared inputs $(x_1, \ldots, x_n)$ and finally obtain a set of consistent secret shares on $f(x_1, \ldots, x_n)$.

Algorithm 4 The Π_{Input} protocol

Preprocessing: For each $i \in [n]$, uniformaly sample r_i from the input space, and generate $(r_{i,1}, \ldots, r_{i,n}) \leftarrow \mathsf{SS}(r_i)$. Send $(r_i, (r_{j,i})_{j\in[n]})$ to each $\mathcal{P}_i$.

1: **upon** receiving input x_i **do** ▷ For any $\mathcal{P}_i$
2: Compute $\bar{x}_i = r_i + x_i$
3: **send** $(\texttt{input}, \texttt{sid}, \bar{x}_i)$ to $\mathcal{F}_{\mathsf{ACS}}$
4: **wait** receiving $(\texttt{output}, \texttt{sid}, \mathsf{commonSet})$ from $\mathcal{F}_{\mathsf{ACS}}$
5: Parse $\mathsf{commonSet} = \{(j, \bar{x}_j)\}_{j\in\mathbb{J}}$
6: Compute $x_{j,i} = \bar{x}_j - r_{j,i}$, for each $j \in \mathbb{J}$
7: **return** $(\texttt{output}, \texttt{sid}, \mathbb{J}, \{x_{j,i}\}_{j\in\mathbb{J}})$.

In general, as the offline phase can be executed prior to the actual computation, it can accept an expensive offline protocol. On the other hand, the online phase, including both the input phase and the function evaluation phase, is always expected to be efficient. Moreover, there is a lot of work pushing down the communication complexity of the functional evaluation phase. Notably, in the information-theoretical setting, we know solutions to evaluate a function f (modeled as a circuit C) at the cost of $\mathcal{O}(M \cdot n + D \cdot n^2)$ [24–27], where M is the number of multiplicative gates in the circuit C, n is the number of nodes, and D is the depth of the circuit. Existing AMPC protocols apply n instances of reliable broadcast [15] or asynchronous completed secret sharing [27] to disseminate the private inputs, causing $\Omega(n^3)$ communication cost already; An agreement on core set protocol is further employed to decide which $n - f$ broadcast instances have terminated, incurring another $\mathcal{O}(n^3)$ communication cost. In summary, the current input agreement employed in existing AMPC constitutes a bottleneck in terms of communication complexity, particularly when the number of multiplicative gates to be evaluated is not very large, e.g., $\mathcal{O}(n^2)$.

The input functionality $\mathcal{F}_{\mathsf{Input}}$ and its realization from ACS. To formally study the input phase, we present the ideal functionality $\mathcal{F}_{\mathsf{Input}}$ of the input phase in Fig. 11 in Appendix I. $\mathcal{F}_{\mathsf{Input}}$ is defined w.r.t. any secret sharing scheme SS. Moreover, different from other ideal functionalties of the consensus primitives considered in this paper, it captures the private input, *i.e.*, when a node sends an input to the ideal functionality, only the input length is leaked to the simulator. On the other hand, it explicitly considers adaptive corruption, as the private input should be leaked to the adversary when the node gets corrupt.

We focus on the Shamir sharing, as it is employed in almost all AMPC protocols with a liveness guarantee. Particularly, we leverage Choudhury et al.'s [23] trick of using a preshared random value r to mask the real input x, such that each node can obtain a secret share of x by combining the share of r and the masked $\bar{x}$. The masked value $\bar{x}$ will not leak any information about x, and thus we can employ $\mathcal{F}_{\mathsf{ACS}}$ and let all honest nodes agree on a set of masked values. Remark that the input protocol can be easily generalized to AMPC protocols not based on Shamir's secret sharing. For example, in the threshold fully homomorphic encryption based MPC, we can use $\mathcal{F}_{\mathsf{ACS}}$ to let all nodes agree on a set of ciphertexts such that further computation can be carried out on them.

We present an input phase protocol in the $\mathcal{F}_{\mathsf{ACS}}$-hybrid model in Algorithm 4, and the full proof is deferred to Appendix I.

Theorem 7. *The protocol Π_{input} UC-realizes $\mathcal{F}_{\mathsf{Input}}$ in the $\mathcal{F}_{\mathsf{ACS}}$-hybrid model, against any adaptive Byzantine adversary who corrupts up to $n/3$ nodes.*

Instantiation and complexity analysis. Using our Π_{ACS} in Algorithm 6 to instantiate Π_{input} gives us an input protocol in the $(\mathcal{F}_{\mathsf{MVBA}}, \mathsf{PKI})$-hybrid model, which further relies a digital signature scheme from collision-resistant hash functions. Since we already assume the correlated randomness, $\mathcal{F}_{\mathsf{coin}}$ comes "for free". I.e., the nodes can simply reconstruct a pre-shared random value as the coin.

Combining the existing function evaluation protocols with our input protocol, we have an online-phase AMPC protocol in the $\mathcal{F}_{\mathsf{PKI}}$-hybrid model against adaptive Byzantine adversaries which corrupts up to $n/3$ nodes, with the communication complexity of $\mathcal{O}\big(Mn + Dn^2 + \lambda n^2 \log n + \kappa n^2\big)$, where M is the number of multiplicative gates in the circuit C, n is the number of nodes, and D is the depth of the circuit.

References

1. Abraham, I., Asharov, G., Patra, A., Stern, G.: Perfectly secure asynchronous agreement on a core set in constant expected time. IACR Cryptol. ePrint Arch. p. 1130 (2023)
2. Abraham, I., Chan, T.H., Dolev, D., Nayak, K., Pass, R., Ren, L., Shi, E.: Communication complexity of byzantine agreement, revisited. In: PODC. pp. 317–326. ACM (2019)
3. Abraham, I., Malkhi, D., Spiegelman, A.: Asymptotically optimal validated asynchronous byzantine agreement. In: Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing. pp. 337–346 (2019)
4. Alhaddad, N., Das, S., Duan, S., Ren, L., Varia, M., Xiang, Z., Zhang, H.: Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In: Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing. pp. 399–417 (2022)
5. Asharov, G., Lindell, Y.: A full proof of the BGW protocol for perfectly secure multiparty computation. J. Cryptol. **30**(1), 58–151 (2017)
6. Backes, M., Bendun, F., Choudhury, A., Kate, A.: Asynchronous mpc with a strict honest majority using non-equivocation. In: Proceedings of the 2014 ACM symposium on Principles of distributed computing. pp. 10–19 (2014)
7. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: CRYPTO. Lecture Notes in Computer Science, vol. 576, pp. 420–432. Springer (1991)
8. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. In: Proceedings of the twenty-fifth annual ACM symposium on Theory of computing. pp. 52–61 (1993)
9. Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience (extended abstract). In: PODC. pp. 183–192. ACM (1994)
10. Bernstein, D.J., Hopwood, D., Hülsing, A., Lange, T., Niederhagen, R., Papachristodoulou, L., Schneider, M., Schwabe, P., Wilcox-O'Hearn, Z.: SPHINCS: practical stateless hash-based signatures. In: EUROCRYPT (1). Lecture Notes in Computer Science, vol. 9056, pp. 368–397. Springer (2015)

11. Blahut, R.E.: Theory and practice of error control codes. Addison-Wesley (1983)
12. Blum, E., Katz, J., Liu-Zhang, C.D., Loss, J.: Asynchronous byzantine agreement with subquadratic communication. In: Theory of Cryptography: 18th International Conference, TCC 2020, Durham, NC, USA, November 16–19, 2020, Proceedings, Part I 18. pp. 353–380. Springer (2020)
13. Blum, E., Liu-Zhang, C.D., Loss, J.: Always have a backup plan: fully secure synchronous mpc with asynchronous fallback. In: Annual International Cryptology Conference. pp. 707–731. Springer (2020)
14. Bracha, G.: An asynchronous [(n-1)/3]-resilient consensus protocol. In: Proceedings of the third annual ACM symposium on Principles of distributed computing. pp. 154–162. ACM (1984)
15. Bracha, G.: Asynchronous byzantine agreement protocols. Information and Computation **75**(2), 130–143 (1987)
16. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Annual International Cryptology Conference. pp. 524–541. Springer (2001)
17. Cachin, C., Mićić, J., Steinhauer, N., Zanolini, L.: Quick order fairness. In: International Conference on Financial Cryptography and Data Security. pp. 316–333. Springer (2022)
18. Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: 24th IEEE Symposium on Reliable Distributed Systems (SRDS'05). pp. 191–201. IEEE (2005)
19. Canetti, R.: Studies in secure multiparty computation and applications. Scientific Council of The Weizmann Institute of Science (1996)
20. Canetti, R.: Universally composable security: A new paradigm for cryptographic protocols. In: FOCS. pp. 136–145. IEEE Computer Society (2001)
21. Catalano, D., Fiore, D.: Vector commitments and their applications. In: Public Key Cryptography. Lecture Notes in Computer Science, vol. 7778, pp. 55–72. Springer (2013)
22. Chopard, A., Hirt, M., Liu-Zhang, C.D.: On communication-efficient asynchronous mpc with adaptive security. In: Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II 19. pp. 35–65. Springer (2021)
23. Choudhury, A., Orsini, E., Patra, A., Smart, N.P.: Linear overhead optimally-resilient robust MPC using preprocessing. In: SCN. Lecture Notes in Computer Science, vol. 9841, pp. 147–168. Springer (2016)
24. Choudhury, A., Pappu, N.: Perfectly-secure asynchronous mpc for general adversaries. In: Progress in Cryptology–INDOCRYPT 2020: 21st International Conference on Cryptology in India, Bangalore, India, December 13–16, 2020, Proceedings 21. pp. 786–809. Springer (2020)
25. Choudhury, A., Patra, A.: Optimally resilient asynchronous mpc with linear communication complexity. In: Proceedings of the 16th International Conference on Distributed Computing and Networking. pp. 1–10 (2015)
26. Choudhury, A., Patra, A.: An efficient framework for unconditionally secure multiparty computation. IEEE Transactions on Information Theory **63**(1), 428–468 (2016)
27. Choudhury, A., Patra, A.: On the communication efficiency of statistically secure asynchronous MPC with optimal resilience. J. Cryptol. **36**(2), 13 (2023)
28. Cohen, R., Forghani, P., Garay, J.A., Patel, R., Zikas, V.: Concurrent asynchronous byzantine agreement in expected-constant rounds, revisited. In: TCC (4). Lecture Notes in Computer Science, vol. 14372, pp. 422–451. Springer (2023)

29. Coretti, S., Garay, J.A., Hirt, M., Zikas, V.: Constant-round asynchronous multiparty computation based on one-way functions. In: ASIACRYPT (2). Lecture Notes in Computer Science, vol. 10032, pp. 998–1021 (2016)
30. Damgård, I., Nielsen, J.B.: Scalable and unconditionally secure multiparty computation. In: CRYPTO. Lecture Notes in Computer Science, vol. 4622, pp. 572–590. Springer (2007)
31. Das, S., Duan, S., Liu, S., Momose, A., Ren, L., Shoup, V.: Asynchronous consensus without trusted setup or public-key cryptography. IACR Cryptology ePrint Archive (2024), https://eprint.iacr.org/2024/677
32. Das, S., Xiang, Z., Kokoris-Kogias, L., Ren, L.: Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 5359–5376 (2023)
33. Das, S., Yurek, T., Xiang, Z., Miller, A., Kokoris-Kogias, L., Ren, L.: Practical asynchronous distributed key generation. In: 2022 IEEE Symposium on Security and Privacy (SP). pp. 2518–2534. IEEE (2022)
34. Duan, S., Wang, X., Zhang, H.: Fin: Practical signature-free asynchronous common subset in constant time. In: Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security. pp. 815–829 (2023)
35. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Tech. rep., Massachusetts Inst of Tech Cambridge lab for Computer Science (1982)
36. Gao, Y., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 1187–1201 (2022)
37. Gao, Y., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Efficient asynchronous byzantine agreement without private setups. In: ICDCS. pp. 246–257. IEEE (2022)
38. Guo, B., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Speeding dumbo: Pushing asynchronous bft closer to practice. In: The Network and Distributed System Security Symposium (NDSS) (2022)
39. Guo, B., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo: Faster asynchronous bft protocols. In: Proc. ACM CCS 2020. ACM (2020)
40. Hu, B., Zhang, Z., Chen, H., Zhou, Y., Jiang, H., Liu, J.: DyCAPS: Asynchronous proactive secret sharing for dynamic committees. IACR Cryptology ePrint Archive (2022), https://eprint.iacr.org/2022/1169
41. López-Alt, A., Tromer, E., Vaikuntanathan, V.: On-the-fly multiparty computation on the cloud via multikey fully homomorphic encryption. In: STOC. pp. 1219–1234. ACM (2012)
42. Lu, D., Yurek, T., Kulshreshtha, S., Govind, R., Kate, A., Miller, A.: Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In: Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security. pp. 887–903 (2019)
43. Lu, Y., Lu, Z., Tang, Q., Wang, G.: Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In: Proceedings of the 39th symposium on principles of distributed computing. pp. 129–138 (2020)
44. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of bft protocols. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 31–42. ACM (2016)
45. Mostefaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous byzantine consensus with $t < n/3$ and $\mathcal{O}(n^2)$ messages. In: Proceedings of the 2014 ACM symposium on Principles of distributed computing. pp. 2–9. ACM (2014)

46. Mostéfaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous binary byzantine consensus with t < n/3, o(n2) messages, and O(1) expected time. J. ACM **62**(4), 31:1–31:21 (2015)
47. Mostéfaoui, A., Raynal, M.: Signature-free asynchronous byzantine systems: from multivalued to binary consensus with t¡ n/3 t¡ n/3, o (n^ 2) o (n 2) messages, and constant time. Acta Informatica **54**, 501–520 (2017)
48. Myers, S., Sergi, M., et al.: Threshold fully homomorphic encryption and secure computation. Cryptology ePrint Archive (2011)
49. Nayak, K., Ren, L., Shi, E., Vaidya, N.H., Xiang, Z.: Improved extension protocols for byzantine broadcast and agreement. In: DISC. LIPIcs, vol. 179, pp. 28:1–28:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2020)
50. Patra, A., Choudhury, A., Pandu Rangan, C.: Efficient asynchronous verifiable secret sharing and multiparty computation. Journal of Cryptology **28**, 49–109 (2015)
51. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. Journal of the ACM (JACM) **27**(2), 228–234 (1980)
52. Shoup, V.: A theoretical take on a practical consensus protocol. IACR Cryptol. ePrint Arch. p. 696 (2024)
53. Shoup, V., Smart, N.P.: Lightweight asynchronous verifiable secret sharing with optimal resilience. Journal of Cryptology **37**(3), 27 (2024)
54. Yurek, T., Xiang, Z., Xia, Y., Miller, A.: Long live the honey badger: Robust asynchronous DPSS and its applications. In: USENIX Security. pp. 5413–5430 (2023)
55. Zhang, H., Duan, S.: Pace: Fully parallelizable bft from reproposable byzantine agreement. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 3151–3164 (2022)

A Related Work

Byzantine Agreement (BA) [51] is a fundamental problem in Byzantine Fault-Tolerant (BFT) distributed computing. The primary objective of BA is to enable all nodes in a distributed system to reach consensus on their initial input values, even when there is a malicious adversary with control over up to f nodes. This problem has been extensively studied by researchers, leading to investigations in various network models and adversary assumptions. One of the important variants of the BA problem is the Asynchronous Common Subset (ACS) [9]. It plays a crucial role in solving asynchronous Multi-Party Computation (MPC) problems. All existing asynchronous MPC protocols [6, 9, 13, 19, 22, 24–27, 42] utilize ACS to achieve consensus on which private inputs can be used as input for a computation circuit [50].

The earliest ACS protocol relied on n concurrent ABA sub-protocols, resulting in high communication complexity and sub-linear time complexity. Despite the high cost, HBBFT [44] demonstrated certain advantages compared to ACS constructions that reduce to MVBA [16]. The most recent work, Dumbo [39], introduced an improved approach to applying MVBA for constructing ACS, achieving better performance both theoretically and practically. Due to the advantageous features of MVBA, it has become a versatile and widely applicable primitive in various cryptographic protocols, serving as an essential underlying

component, including State Machine Replication (SMR) or Atomic Broadcast (ABC) [17, 36, 38, 44], Distributed Key Generation [32, 33, 37], and Dynamic-committee Proactive Secret Sharing (DPSS) [40, 54].

As a powerful underlying tool, the performance of MVBA significantly influences the efficiency of applications relying on it. Initially proposed in [16] with a communication complexity of $\mathcal{O}(n^2\ell + n^2\lambda + n^3)$, substantial progress has been made to enhance its performance. Initially, Abraham et al. [3] removed a $\mathcal{O}(n^3)$-term, achieving quadratic communication complexity for the first time at $\mathcal{O}(\ell n^2 + \lambda n^2)$. Later, Lu et al. [43] further reduced the $\mathcal{O}(\ell n^2)$-term to $\mathcal{O}(\ell n)$ for optimal communication complexity when $\ell \geq \lambda n$. The MVBA work in [43] marked a milestone for achieving the first optimal ACS with quadratic communication complexity, leveraging the approach presented in [16]. This enabled the realization of ACS with $\mathcal{O}(n^2)$ communication complexity, benefiting from the improved performance of MVBA. Additionally, Guo et al. [38] presented an MVBA with significantly fewer rounds. However, all these MVBA efforts rely on threshold signatures, which are vulnerable to quantum attacks and necessitate a trusted setup, posing challenges in various applications, such as Distributed Key Generation (DKG).

In the hash-based setting, recent MVBA protocols like Fin-MVBA [34] have been introduced. However, they exhibit cubic communication complexity. Therefore, achieving sub-cubic communication complexity for hash-based MVBA protocols remains an open challenge.

B UC Model

Simulation-based security. We analyze our protocols using the standard simulation-based security, which aligns with the design of most cryptographic protocols. This approach allows us to comprehensively capture all security goals of a primitive by comparing it with an "ideal" version of some desirable functionality. It eliminates the need to enumerate a list of potentially overlapping properties and is particularly useful when arguing the security of a protocol that utilizes other protocols as sub-routines. The simulation-based security is formulated with a *real world* and an *ideal world*. In the real world, the protocol is executed by nodes that exchange messages among themselves according to the protocol specification, while an adversary can interfere with the protocol execution within certain rules. In the ideal world, all nodes only communicate with a trusted third party, referred to as an ideal functionality, which assists them in obtaining the desired output based on their inputs. Informally, a protocol is considered secure if whatever an adversary can do in the real world can also be achieved in the ideal world.

In the following, we provide a brief overview of the formal description of simulation-based security, which takes into account both adaptive adversaries and asynchronous networks within the Universal Composability (UC) framework.

The real world. An n-party protocol Π consists of an n-tuple of probabilistic polynomial time (PPT) interactive Turing machines (ITMs), representing the

parties $\mathcal{P}_1, \ldots, \mathcal{P}_n$, respectively. Additionally, there are two ITMs representing the adversary $\mathcal{A}$ and the environment $\mathcal{Z}$. An execution of the protocol Π consists of a series of activations of these ITMs. It starts with the environment $\mathcal{Z}$ providing inputs to and collecting outputs from the nodes and the adversary. Upon receiving inputs or other messages, a node is activated and can perform local computations, write on its output tape, or send messages to other nodes.

Network Model. We consider an *asynchronous* network where nodes are pairwise connected with *authenticated* channels. To model the worst-case scenario, we let $\mathcal{A}$ be responsible for delivering messages between honest nodes. The adversary cannot omit, change, or inject these messages. However, the adversary can reorder the messages and arbitrarily delay them, although it cannot delay them indefinitely. These requirements are formalized using the *eventual-delivery secure message-transmission* (ED-SMT) ideal functionality in [29]. Throughout this paper, whenever we say a node $\mathcal{P}_i$ sends a message to another node $\mathcal{P}_j$, we implicitly mean that $\mathcal{P}_i$ and $\mathcal{P}_j$ are invoking the ED-SMT ideal functionality involving the adversary $\mathcal{A}$. When we say a node $\mathcal{P}_i$ *multicasts* a message, we mean $\mathcal{P}_i$ sends the message to all nodes in the network.

Corruption Model. The adversary $\mathcal{A}$ is *adaptive*, which can corrupt the nodes at any time during the protocol execution. Once a node $\mathcal{P}_i$ is corrupted, $\mathcal{P}_i$ sends its entire local state to $\mathcal{A}$, and in all future activations follows the instructions from $\mathcal{A}$. Throughout this paper, we focus on optimal resilience, which means the total number of corrupted nodes is at most $f < n/3$. At any time of the execution, the nodes that remain honest so far are referred to as *so-far-honest* nodes, and the nodes that remain honest till the end of the execution are referred to as *forever-honest* nodes. Particularly, recall that the messages are delivered by $\mathcal{A}$. If $\mathcal{A}$ corrupts a node that just sent a message, $\mathcal{A}$ can choose not to deliver this message if it has not been delivered before corruption. Such an ability of $\mathcal{A}$ is also known as *after-the-fact removal* [2], which is also captured by the ED-SMT functionality. During the protocol execution, the environment $\mathcal{Z}$ can arbitrarily communicate with the adversary $\mathcal{A}$ for an arbitrary number of times. The execution is complete when all forever-honest nodes obtain their respective outputs; the outputs are then returned to $\mathcal{Z}$. We use $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}}$ to denote the distribution ensemble corresponding to the binary output of $\mathcal{Z}$ at the end of an execution of Π with the adversary $\mathcal{A}$ (in the real world).

The ideal world. A computation in the ideal world consists of n *dummy* nodes and an ideal functionality $\mathcal{F}$ modeled as an ITM. There is also an adversary (or called a simulator) $\mathcal{S}$, which, interacting with $\mathcal{F}$ in a restricted and clearly defined way, is supposed to mimic an adversary $\mathcal{A}$ in the real world. In the beginning, the environment $\mathcal{Z}$ provides the initial inputs to the dummy nodes and $\mathcal{S}$. The dummy nodes provide inputs to $\mathcal{F}$ and wait to collect outputs. $\mathcal{Z}$ can communicate with $\mathcal{S}$ arbitrarily. At the end of execution, $\mathcal{Z}$ collects the outputs of dummy nodes and returns a binary value. There is no restriction on how an environment provides input/output requests. However, by the protocol description, an honest node will ignore other types of messages and only accept one input. In the ideal world, the ideal functionality in general only accepts one input message

from a dummy party unless it has been corrupted. We use $\mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$ to denote the distribution ensemble with respect to the binary output of $\mathcal{Z}$.

Modeling Delayed Input And Output. An ideal functionality should explicitly capture how an adversary might interfere with the protocol execution and thus exclude all unspecified interference. Due to the asynchronous nature of the network, an adversary inherently possesses the ability to arbitrarily delay messages. Conversely, an asynchronous protocol (if it satisfies termination) typically needs to progress upon being provided with $n-f$ "valid-looking" messages since waiting for the last f messages can result in indefinite waiting. Translating these facts into the ideal world, an ideal functionality should expect at most $n-f$ inputs or participants (referred to as the "core set") and proceed based on them. The "core set" is somehow selected by the adversary from those who have been activated by the environment. To capture this, Cohen et al.'s [28] approach allows the adversary to specify a delay counter for each input procedure. A node will keep pinging the ideal functionality to fetch the output across its activations until it receives an output from the ideal functionality. The delay counter is decremented every time the node pings the ideal functionality; once the counter becomes 0, the corresponding input will be included in the "core set". Similarly, the output from the ideal functionality to a node may be delayed by the adversary, and we use the delay counter to capture it as well. It's important to note that the delay counter has to be encoded in unary, ensuring that the delay must be bound by the adversary's computational resources. We follow Cohen et al.'s approach and include the delay counter for both the input process and the output process in the descriptions of all asynchronous ideal functionalities in this paper. For further discussion, refer to [28].

Modeling Adaptive Corruption in The Ideal World. In the ideal world, we allow the simulator $\mathcal{S}$ to send corruption messages of the form $(\texttt{corruption}, \texttt{sid}, \mathcal{P}_i)$ to the ideal functionality $\mathcal{F}$, indicating that the node $\mathcal{P}_i$ is to be corrupted. $\mathcal{S}$ maintains a set $\mathbb{C}$ that keeps track of all corrupted nodes. When receiving such a message, $\mathcal{F}$ first checks whether the number of corrupted nodes has reached the threshold f; if not, it updates $\mathbb{C} \leftarrow \mathbb{C} \cup \{\mathcal{P}_i\}$ and allows $\mathcal{S}$ to send future messages on behalf of $\mathcal{P}_i$. In this paper, since most functionalities do not capture the secrecy of inputs, these ideal functionalities do not need to provide the internal states of a corrupted party to the simulator. In this case, for the sake of notational simplicity, we omit the corruption message from the simulator to the ideal functionality in the description of each ideal functionality.

UC security. With the real world and the ideal world, we can define the UC security of a protocol. In particular, we say a protocol Π *UC-realizes* an ideal functionality $\mathcal{F}$ if, for any PPT adversary $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that, for any PPT environment $\mathcal{Z}$, it holds that $\mathsf{EXEC}_{\Pi,\mathcal{A},\mathcal{Z}} \approx \mathsf{IDEAL}_{\mathcal{F},\mathcal{S},\mathcal{Z}}$.

Hybrid Model. Let $\mathcal{G}$ be an ideal functionality. In the $\mathcal{G}$-hybrid model, we can design a protocol Π_F in which the nodes have access to $\mathcal{G}$. The UC framework guarantees that all security properties of Π_F can be preserved if we replace $\mathcal{G}$ with any protocol that UC-realizes $\mathcal{G}$. The hybrid model captures the essence of modular protocol design. Formally, we have the following results from [20].

Lemma 1 ([20]). *Suppose Π_F is a protocol that UC-realizes an ideal functionality $\mathcal{F}$ in the $\mathcal{G}$-hybrid model, and Π_G is a protocol that UC-realizes $\mathcal{G}$. If Π_F^G is a protocol obtained by replacing every call to $\mathcal{G}$ in Π_F with an execution of Π_G, then Π_F^G UC-realizes $\mathcal{F}$ without access to $\mathcal{G}$.*

Asynchronous consensus in $\mathcal{F}_{\mathsf{coin}}$-hybrid model. As emphasized by the well-known FLP impossibility result [35], randomness is necessary for asynchronous Byzantine agreement. In the computational setting, an efficient MVBA protocol can easily imply an asynchronous Byzantine agreement [16], so it is also subjected to the impossibility. This approach distills coin generation as an independent problem and focuses on the consensus part. Most asynchronous consensus protocols are typically described within the $\mathcal{F}_{\mathsf{coin}}$-hybrid model, where participants have access to the common coin ideal functionality $\mathcal{F}_{\mathsf{coin}}$. This ideal functionality provides a uniformly random value to all nodes upon receiving a sufficient number of requests. We define $\mathcal{F}_{\mathsf{coin}}^V$ in Fig. 9 (in Appendix C), parameterized by V, the domain of random values, which is adapted from [27, Fig. 3 $\mathcal{F}_{\mathsf{rand}}$]. In the rest of the paper, we follow the standard approach and present our consensus protocols in $\mathcal{F}_{\mathsf{coin}}$-hybrid model.

C Preliminary

Notations. We express our protocols through a series of numbered steps. During the execution of such a protocol, a node is expected to iteratively follow these steps in a sequential order, executing each instruction. Certain instructions may have specific preconditions, and if these conditions are not met, the node will skip the corresponding steps. When we say "**Upon**{Condition}{Instruction}", we mean the instruction should be executed every time the condition is triggered. When we say "**Wait**{Condition}{Instruction}", the instruction is supposed to be only executed once no matter how many times the condition may be triggered. In a protocol Π a message with the format $(\textsc{MsgType}, \mathtt{sid}, \dots)$ indicates that the message is associated with an instance of the protocol Π identified by $\mathtt{sid}$. Any message exchanged between two parties follows the format $(\textsc{MsgType}, \mathtt{sid}, \dots)$, where $\mathtt{sid}$ denotes the identifier associated with the protocol instance, and MsgType indicates the type of message. We denote the computational security parameter and the statistical security parameter by λ and κ, respectively. We say a function F defined over positive integers is negligible in λ, denoted by $|F(\lambda)| < \mathsf{negl}(\lambda)$, if for any polynomial function P, there exists a positive integer N, such that for any $\lambda > N$, it holds that $|F(\lambda)| < \frac{1}{P(\lambda)}$. We say a probability $\mathtt{p}$ is overwhelming, if $\mathtt{p} > 1 - \mathsf{negl}(\lambda)$.

C.1 Ideal Functionalities for Standard Primitives

Public Key Infrastructure ($\mathcal{F}_{\mathsf{PKI}}$). We introduce the ideal functionality $\mathcal{F}_{\mathsf{PKI}}^V$ (see Fig. 8) to provide support for digital signatures, as utilized in Section 7.

Functionality $\mathcal{F}_{\mathsf{PKI}}$

$\mathcal{F}_{\mathsf{PKI}}$ proceeds as follows, running with $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ and $\mathcal{S}$. At the first activation, verify $(\mathtt{sid}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ for the session ID $\mathtt{sid}$ and the set of node. Initialize: $vk_i = \perp$ for all $i \in [n]$.

- Upon receiving $(\mathtt{register}, \mathtt{sid}, vk)$ from $\mathcal{P}_i$ (or $\mathcal{S}$ on behalf of a corrupted node), record $vk_j = vk$.
- Upon receiving $(\mathtt{retrive}, \mathtt{sid}, j)$ from $\mathcal{P}_i$ (or $\mathcal{S}$ on behalf of a corrupted node), if $vk_j \neq \perp$, return $(\mathtt{retrived}, \mathtt{sid}, j, vk_j)$.

Fig. 8. The PKI functionality $\mathcal{F}_{\mathsf{PKI}}$

Functionality $\mathcal{F}_{\mathsf{coin}}$

$\mathcal{F}_{\mathsf{coin}}$ proceeds as follows, running with $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ and the adversary. At the first activation, verify $(\mathtt{sid}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ for the session ID $\mathtt{sid}$ and the set of nodes. Initialize: $\mathtt{participated}_i = 0$ and $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{output}} = 1$ for all $i \in [n]$; $r = \perp$.

- Upon receiving $(\mathtt{delay}, \mathtt{sid}, \mathcal{P}_i, \mathtt{type}, D)$ from the adversary for any $i \in [n]$, if $\mathtt{type} \in \{\mathtt{input}, \mathtt{output}\}$ and $D \in \mathbb{Z}$ represented in unary notation, then update $\mathsf{D}_i^{\mathtt{type}} = \max\{1, \mathsf{D}_i^{\mathtt{type}} + D\}$, and provide $(\mathtt{delay\text{-}set}, \mathtt{sid})$ to the adversary.
- Upon receiving $(\mathtt{request}, \mathtt{sid})$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and provide $(\mathtt{request}, \mathtt{sid}, \mathcal{P}_i)$ to the adversary.
- Upon receiving $(\mathtt{fetch}, \mathtt{sid})$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and the **Output Release Procedure**, and provide $(\mathtt{fetch}, \mathtt{sid}, \mathcal{P}_i)$ to the adversary.

Input Submission Procedure: If $\mathtt{participated} = 0$, and $(\mathtt{request}, \mathtt{sid})$ has been provided by $\mathcal{P}_i$, do the following:

- Update $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{input}} - 1$;
- If $\mathsf{D}_i^{\mathtt{input}} = 0$, update $\mathtt{participated}_i = 1$.

Output Release Procedure: If $\sum_{j \in [n]} \mathtt{participated}_j \geq f + 1$, do the following:

- Update $\mathsf{D}_i^{\mathtt{output}} = \mathsf{D}_i^{\mathtt{output}} - 1$;
- if $\mathsf{D}_i^{\mathtt{output}} = 0$, then do the following. If $r = \perp$, $r \leftarrow\!\!\$\ V$, provide $(\mathtt{output}, \mathtt{sid}, r)$ to the adversary. Set $(\mathtt{output}, \mathtt{sid}, r)$ to be sent to $\mathcal{P}_i$.

Fig. 9. The common coin functionality $\mathcal{F}_{\mathsf{coin}}^V$ for the randomness domain of V

Asynchronous binary agreement ($\mathcal{F}_{\mathsf{ABA}}$). Asynchronous Binary Agreement (ABA) [45] stands as the fundamental primitive in asynchronous consensus. On rough terms, it enables a collection of participants, each equipped with a binary input, to eventually converge on a binary output. It ensures that the output is $b \in \{0, 1\}$ if all honest nodes input b. The ideal functionality $\mathcal{F}_{\mathsf{ABA}}$ is outlined in Fig. 10, which is adapted from [28, Fig 4] by focusing on the case of binary inputs. It is easy to argue that the ABA protocols from [46] can UC-realize $\mathcal{F}_{\mathsf{ABA}}$ in $\mathcal{F}_{\mathsf{coin}}$-hybrid model, with communication cost of $\mathcal{O}(n^2)$ bits and the expected round complexity of $\mathcal{O}(1)$.

Functionality $\mathcal{F}_{\mathsf{ABA}}$

$\mathcal{F}_{\mathsf{ABA}}$ proceeds as follows, running with $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ and the adversary. At the first activation, verify $(\mathtt{sid}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ for the session ID $\mathtt{sid}$ and the set of node. Initialize: $\mathtt{participated}_i = 0$, $b_i = \perp$ and $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{output}} = 1$ for all $i \in [n]$; $y = \perp$.

- Upon receiving $(\mathtt{delay}, \mathtt{sid}, \mathcal{P}_i, \mathtt{type}, D)$ from the adversary for any $i \in [n]$, if $\mathtt{type} \in \{\mathtt{input}, \mathtt{output}\}$ and $D \in \mathbb{Z}$ represented in unary notation, then update $\mathsf{D}_i^{\mathtt{type}} = \max\{1, \mathsf{D}_i^{\mathtt{type}} + D\}$, and provide $(\mathtt{delay\text{-}set}, \mathtt{sid})$ to the adversary.
- Upon receiving $(\mathtt{input}, \mathtt{sid}, b')$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and provide $(\mathtt{input}, \mathtt{sid}, b', \mathcal{P}_i)$ to the adversary.
- Upon receiving $(\mathtt{fetch}, \mathtt{sid})$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and the **Output Release Procedure**, and provide $(\mathtt{fetch}, \mathtt{sid}, \mathcal{P}_i)$ to the adversary.

Input Submission Procedure: If $\mathtt{participated}_i = 0$, and $(\mathtt{input}, \mathtt{sid}, b')$ has been provided by $\mathcal{P}_i$, do the following:

- Update $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{input}} - 1$;
- If $\mathsf{D}_i^{\mathtt{input}} = 0$, update $\mathtt{participated}_i = 1$, and record $b_i = b'$.

Output Release Procedure: If $\sum_{j \in [n]} \mathtt{participated}_j \geq n - f$, do the following:

- Update $\mathsf{D}_i^{\mathtt{output}} = \mathsf{D}_i^{\mathtt{output}} - 1$;
- if $\mathsf{D}_i^{\mathtt{output}} = 0$, then do the following. If $y = \perp$, set $y = b$ for a binary value b such that $b = b_i$ for at least $n - 2f$ input values b_i, and provide $(\mathtt{output}, \mathtt{sid}, y)$ to the adversary. Set $(\mathtt{output}, \mathtt{sid}, y)$ to be sent to $\mathcal{P}_i$.

Fig. 10. The aynchronous binary agreement functionality $\mathcal{F}_{\mathsf{ABA}}$

C.2 Other Primitives

Erasure code. A (k, n)-erasure code [11] comprises two deterministic algorithms, denoted as $\mathsf{EC.Encode}$ and $\mathsf{EC.Decode}$. The $\mathsf{EC.Encode}$ algorithm takes a data value $\mathbf{m}$ with k data fragments $(m_1, \cdots, m_k)$ as input and outputs n coded fragments $\mathbf{c} = (\mathsf{c}_1, \ldots, \mathsf{c}_n)$. Any k elements in the code vector $\mathbf{c}$ can reconstruct the original data $\mathbf{m}$ using the $\mathsf{EC.Decode}$ algorithm. Formally, a (k, n)-erasure code involves two deterministic algorithms:

1. $\mathsf{EC.Encode}(\mathbf{m}, n, k) \to \mathbf{c}$: Given a data $\mathbf{m} \in \mathcal{B}^k$ as input, this deterministic encoding algorithm outputs a vector $\mathbf{c} := \{c_1, \cdots, c_n\}$.
2. $\mathsf{EC.Decode}(\{(i, c_i)\}_{i \in S}, n, k) \to \mathbf{m}$: Given a set $\{(i, c_i)\}_{i \in S}$ where $S \subset [n]$ and $|S| = k$ as input, this deterministic decoding algorithm outputs $\mathbf{m}$.

Correctness. Assuming $\mathcal{B}$ is the field of each fragment, for any $\mathbf{m} \in \mathcal{B}^k$ and any $\mathbb{I} \subset [n]$ with $|\mathbb{I}| = k$, then

$$\Pr[\mathsf{EC.Decode}(\{(i, \mathsf{c}_i)\}_{i \in \mathbb{I}}) = \mathbf{m} \mid \mathbf{c} := (\mathsf{c}_1, \cdots, \mathsf{c}_n) \leftarrow \mathsf{EC.Encode}(\mathbf{m})] = 1.$$

Throughout the paper, our focus is on optimally-resilient scenarios. Therefore, we consider a $(f + 1, n)$-erasure code where $n \geq 3f + 1$.

Vector commitment (VC). A VC scheme [21] consists of a tuple of algorithms: $(\mathsf{VC.Setup}, \mathsf{VCom}, \mathsf{Open}, \mathsf{VerifyOpen})$, where $\mathsf{VC.Setup}$ produces a public parameter p, VCom produces a vector commitment by committing to any n-sized vectors, Open is used to generate a proof for its specified position value, and $\mathsf{VerifyOpen}$ is used to verify that the given value indeed is the specified position value corresponding to the vector commitment. Formally, a VC scheme can be abstracted as a tuple comprising the following algorithms:

1. $\mathsf{VC.Setup}(\lambda, n, \mathcal{M}) \to \mathsf{p}$. Given the security parameter λ, the size n of the input vector, and the message space $\mathcal{M}$ of each vector element, it outputs the public parameter p, which is an implicit input to all other algorithms.
2. $\mathsf{VCom}(\mathbf{m}) \to \mathsf{vc}$. Taking a vector $\mathbf{m} = (m_1, ..., m_n)$ as input, this algorithm produces a commitment vc.
3. $\mathsf{Open}(\mathsf{vc}, m_i, i) \to \pi_i$. Given m_i, position i, and commitment vc, this algorithm generates an opening string π_i serving as proof that m_i is the i-th committed element.
4. $\mathsf{VerifyOpen}(\mathsf{vc}, m_i, i, \pi_i) \to 0/1$. Given m_i, i, commitment vc, and opening proof π_i, it outputs 1 if $\mathsf{Open}(\mathsf{vc}, m_i, i) = \pi_i$, otherwise, it outputs 0.

Correctness. VC is correct, if for all $\mathbf{m} \in \mathcal{M}^n$ and $i \in [n]$,

$$\Pr[\mathsf{VerifyOpen}(\mathsf{vc}, m_i, i, \mathsf{Open}(\mathsf{vc}, m_i, i)) = 1 \mid \mathsf{vc} \leftarrow \mathsf{VCom}(\mathbf{m})] = 1.$$

Position binding. VC is position binding, if for any PPT adversary $\mathcal{A}$,

$$\Pr\left[\begin{array}{c}(\mathsf{vc}, i, m, \pi, m', \pi') \leftarrow \mathcal{A}(1^\lambda) : \mathsf{VerifyOpen}(\mathsf{vc}, m, i, \pi) = 1 \\ \wedge \ m \neq m' \ \wedge \ \mathsf{VerifyOpen}(\mathsf{vc}, m', i, \pi') = 1\end{array}\right] \leq \mathsf{negl}(\lambda).$$

D Security proof of SMID

Theorem 8. *Assuming the underlying hash function is collision resistant, the protocol Π_{SMID} in Algorithm 1 UC-realizes $\mathcal{F}_{\mathsf{SMID}}$, in the presence of a computationally bounded and adaptive Byzantine adversary who may corrupt up to $f < n/3$ parties.*

Proof. Let $\mathcal{A}$ be an adversary in the real world. We construct a simulator $\mathcal{S}$ in the ideal world, such that no environment $\mathcal{Z}$ can distinguish whether it is interacting with the protocol Π_{SMID} and $\mathcal{A}$, or with $\mathcal{F}_{\mathsf{SMID}}$ and $\mathcal{S}$.

Here are some "general principles" $\mathcal{S}$ will follow. $\mathcal{S}$ runs a copy of $\mathcal{A}$, and plays the role of all "so-far-honest" parties (and hybrids if any) in a simulated execution of the protocol. All inputs from $\mathcal{Z}$ to $\mathcal{A}$ are forwarded to $\mathcal{A}$, and all outputs from $\mathcal{A}$ to $\mathcal{Z}$ are forwarded to $\mathcal{Z}$. Additionally, whenever $\mathcal{A}$ corrupts a party in the simulation, $\mathcal{S}$ corrupts the same party in the ideal world by interacting with $\mathcal{F}_{\mathsf{SMID}}$, and sends $\mathcal{A}$ the party's state and thereafter follows $\mathcal{A}$'s instructions for that party. Moreover, $\mathcal{S}$ adjust delay counters for both input submission and output release procedures based on the simulated execution influenced by the

adversary. Typically, the simulator delays all inputs to the ideal functionality until the first so-far-honest party outputs in the simulated execution. Whenever the so-far-honest party outputs, (1) the simulator, if it has not, delivers enough inputs to the ideal functionality so that it determines the correct output value for the party. Then, (2) the output delay for the corresponding party is set to zero. This ensures that the order of outputs seen by the environment in the ideal world is indistinguishable from that in the real world.

The simulated execution begins when the ideal functionality $\mathcal{F}_{\mathsf{SMID}}$ provides the messages $(\texttt{disperse}, \texttt{sid}, \cdot, \cdot)$ to $\mathcal{S}$. These messages leak the input values of "so-far-honest" parties to the simulator. Based on these values, $\mathcal{S}$ simulates the execution for $\mathcal{A}$ and interacts with $\mathcal{F}_{\mathsf{SMID}}$ as follows, while following the general principles introduced above.

Simulating a"so-far-honest" party $\mathcal{P}_i$. Upon receiving the input of $\mathcal{P}_i$ from $\mathcal{F}_{\mathsf{SMID}}$, run the code of $\mathcal{P}_i$ with the input (as specified in Algorithm 5), until $\mathcal{P}_i$ returns the output or becomes corrupted by $\mathcal{A}$.

Further interaction with $\mathcal{F}_{\mathsf{SMID}}$. When an honest party $\mathcal{P}_i$ in the simulated execution multicasts Completed to all (i.e., executing the code in line 13), manage the delay such that $\texttt{dispersed}_i = 1$ in the ideal world. When an honest node $\mathcal{P}_i$ returns $(\texttt{disperse-done}, \mathsf{id})$, send $(\texttt{configure-done}, \texttt{sid}, \mathcal{P}_i)$ to $\mathcal{F}_{\mathsf{SMID}}$. When an honest party $\mathcal{P}_i$ returns $(\texttt{recast-output}, s, v')$, then he sends message $(\texttt{configure-output}, \texttt{sid}, \mathcal{P}_s, \mathcal{P}_i, v')$ to $\mathcal{F}_{\mathsf{SMID}}$.

Note that at the point of $\mathcal{A}$'s view, the simulated execution is perfectly indistinguishable from a real execution. Since $\mathcal{S}$ just forwards the communication between $\mathcal{A}$ and $\mathcal{Z}$, it remains to show that the outputs from $\mathcal{F}_{\mathsf{SMID}}$ to honest $\mathcal{P}_i$'s are identical to those in the simulated execution (and thus distinguishable with those in a real execution).

To demonstrate this, we establish the following facts: (1) Lemma 2: When an honest node $\mathcal{P}_i$ returns $(\texttt{disperse-done}, \mathsf{id})$, $\mathcal{S}$ can manage to set $\mathsf{done}_i = 1$ in the ideal world. Moreover, when at least $n-f$ honest parties have participated in the dispersal phase, all honest parties eventually return $(\texttt{disperse-done}, \mathsf{id})$. (2) Lemma 3: For each "so-far-honest" party $\mathcal{P}_j$ who has multicasted the Completed message, at most f honest nodes may receive a different value other than $\mathcal{P}_j$'s input; Moreover, when at least $n-f$ honest parties have participated the recast phase for $\mathcal{P}_j$'s value and $v_j \neq \perp$, at least $n-2f$ honest parties could receive v_j. The first guarantees the honest parties' outputs in the dispersal phase of the simulated execution are identical to the outputs from $\mathcal{F}_{\mathsf{SMID}}$. The second one ensures that $\mathcal{S}$ can always configure the output of an honest party according to its output in the simulated execution, thus the honest parties' outputs in the recast phase of the simulated execution are identical to the outputs from $\mathcal{F}_{\mathsf{SMID}}$. □

Lemma 2. *When an honest node $\mathcal{P}_i$ returns $(\texttt{disperse-done}, \mathsf{id})$, $\mathcal{S}$ can manage to set $\mathsf{done}_i = 1$ in the ideal world. Moreover, when at least $n-f$ honest parties have participated in the dispersal phase, all honest parties eventually return $(\texttt{disperse-done}, \mathsf{id})$.*

Proof. Since an honest party $\mathcal{P}_i$ outputs $(\texttt{disperse-done}, \mathsf{id})$ only after receiving $n-f$ Completed messages. If an honest party $\mathcal{P}_j$ has multicasted the

COMPLETED message, the simulator $\mathcal{S}$ must have updated $\mathsf{dispersed}_j = 1$ in $\mathcal{F}_{\mathsf{SMID}}$. Therefore, when $\mathcal{S}$ sends $(\texttt{configure-done}, \texttt{sid}, \mathcal{P}_i)$ to $\mathcal{F}_{\mathsf{SMID}}$, $\mathsf{done}_i = 1$ will be updated accordingly.

Since every honest node $\mathcal{P}_i$ has an input value v_i, following the protocol, $\mathcal{P}_i$ multicasts FRAGMENT messages to all. Upon receiving a valid FRAGMENT message from $\mathcal{P}_i$, any honest node $\mathcal{P}_j$ sends an OK back to $\mathcal{P}_i$. With at most f malicious nodes, $\mathcal{P}_i$ can receive $n - f$ OK messages from distinct nodes. Afterward, $\mathcal{P}_i$ multicasts a COMPLETED message to all. Again, since the number of malicious nodes is at most f, so at least $n - f$ honest nodes will multicast COMPLETED messages to all. Consequently, all honest nodes eventually receive at least $n-f$ COMPLETED messages from distinct nodes, leading to the completion of the dispersal phase for all honest nodes in an SMID instance. □

Lemma 3. *Given a collision-resistant hash function and an adversary with computationally bounded power, the following holds: If a "so-far-honest" party $\mathcal{P}_j$ inputs v_j and multicast the* COMPLETED *message, then at most f honest nodes might receive fragments that do not correspond to the value v_i. Furthermore, when at least $n - f$ honest parties input j partake in the recast phase, at least $n - 2f$ honest parties could recast v_j.*

Proof. Following the procedure outlined in Algorithm 1, when an honest party multicasts a COMPLETED message, it implies that at least $n - 2f$ honest nodes received valid FRAGMENT messages from $\mathcal{P}_i$, and these FRAGMENT messages contain the same vector commitment (vc). Assuming the underlying hash function is collision-resistant, the reconstructed message is identical to the original input message v_j. Since $\mathcal{A}$ cannot change the received value of those $n-2f$ honest nodes, at most f honest nodes may receive a different value other than $\mathcal{P}_j$'s initial input v_j due to the adaptive adversary $\mathcal{A}$. Moreover, if an honest node received $fragment[j]$ in the dispersal phase, then it will only output a message that is reconstructed from $f + 1$ valid fragments, and these fragments have the same vc as $fragment[j]$. Hence, during the recast phase, all honest nodes will multicast their $fragment[j]$ if $fragment[j] \neq \perp$. Since at least $n - 2f$ honest nodes received a valid fragment message from $\mathcal{P}_j$, allowing them to receive at least $n-2f > f+1$ valid fragment messages that share the same vc. This enables these honest nodes to reconstruct the same v_i. □

E Security proof of SMB

Theorem 9. *The protocol Π_{SMB} in Algorithm 2 perfectly UC-realizes $\mathcal{F}_{\mathsf{SMB}}$, in the presence of any adaptive Byzantine adversary who corrupts up to $f < n/3$ parties.*

Proof. Let $\mathcal{A}$ be an adversary in the real world. We construct a simulator $\mathcal{S}$ in the ideal world, such that no environment $\mathcal{Z}$ can distinguish whether it is interacting with the protocol Π_{SMB} and $\mathcal{A}$, or with $\mathcal{F}_{\mathsf{SMB}}$ and $\mathcal{S}$.

$\mathcal{S}$ follows the "general principles" outlined in the proof of Theorem 1. The simulated execution begins when the ideal functionality $\mathcal{F}_{\mathsf{SMB}}$ provides the messages $(\mathtt{input}, \mathtt{sid}, \cdot, \cdot)$ to $\mathcal{S}$. These messages leak the input values of "so-far-honest" parties to the simulator. Based on these values, $\mathcal{S}$ simulates the execution for $\mathcal{A}$ and interacts with $\mathcal{F}_{\mathsf{SMB}}$ as follows, while following the general principles introduced above.

Simulating a"so-far-honest" party $\mathcal{P}_i$. Upon receiving the input of $\mathcal{P}_i$ from $\mathcal{F}_{\mathsf{SMB}}$, run the code of $\mathcal{P}_i$ with the input (as specified in Algorithm 2), until $\mathcal{P}_i$ returns the output or becomes corrupted by $\mathcal{A}$.

Futher Interaction with $\mathcal{F}_{\mathsf{SMB}}$. Initialize $\hat{v}_1 = \hat{v}_2 = \hat{v}_3 = \perp$.

- Upon receiving $(\mathtt{leakage}, \mathtt{sid}, v)$ from $\mathcal{F}_{\mathsf{SMB}}$, update $\hat{v}_1 = v$.
- Keep tracking of all $values_i$ (defined in Line 10-11 in Algorithm 2) of honest parties. Whenever a new value v' appears in any of these $values_i$'s, send $(\mathtt{terminate}, \mathtt{sid}, v')$ to $\mathcal{F}_{\mathsf{SMB}}$, and record $\hat{v}_k = v'$ for the smallest k s.t. $\hat{v}_k = \perp$. Before sending a terminate message, make sure enough inputs whose value is v' have been recorded by $\mathcal{F}_{\mathsf{SMB}}$. Before sending the third message, make sure the inputs from all "so-far-honest" parties have been recorded by $\mathcal{F}_{\mathsf{SMB}}$.
- Upon $\mathcal{P}_i$ returns val_i, it will send message $(\mathtt{decide\text{-}output}, \mathtt{sid}, \mathcal{P}_i, \mathsf{subset}_i)$ to $\mathcal{F}_{\mathsf{SMB}}$ such that $\{\hat{v}_k\}_{k \in \mathsf{subset}_i} = val_i$.

Note that at the point of $\mathcal{A}$'s view, the simulated execution is perfectly indistinguishable from a real execution. Since $\mathcal{S}$ just forwards the communication between $\mathcal{A}$ and $\mathcal{Z}$, it remains to show that the outputs from $\mathcal{F}_{\mathsf{SMB}}$ to honest $\mathcal{P}_i$'s are identical to those in the simulated execution (and thus distinguishable with those in a real execution).

To demonstrate this, we establish the following results. (1) Lemma 4: There are at most three different values among all $values_i$ of honest parties. (2) Lemma 5: All values sent by $\mathcal{S}$ via the termination messages can be recorded by $\mathcal{F}_{\mathsf{SMB}}$. (3) Lemma 6: If an honest party outputs, all honest parties can eventually output; All outputs are subsets of $\{\hat{v}_1, \hat{v}_2, \hat{v}_3\}$. (4) Lemma 7: If there exists an honest party that outputs a single value v^*, then v^* is included in all other honest parties' outputs. (5) Lemma 8: If at least $n - 2f$ honest parties have the same input value v, the protocol always terminates.

The first two results ensure that the list $(\hat{v}_1, \hat{v}_2, \hat{v}_3)$ maintained by $\mathcal{S}$ is identical to that maintained by $\mathcal{F}_{\mathsf{SMB}}$. Results (3) and (4) ensure that the output of an honest party by $\mathcal{F}_{\mathsf{SMB}}$ is always identical to its output in the simulated execution, once $\mathsf{terminate}$ is set to be 1. Result (5) ensures the protocol must terminate, if $n - 2f$ honest parties have provided the inputs of the same value to $\mathcal{F}_{\mathsf{SMB}}$. Combined them, we can conclude that outputs from $\mathcal{F}_{\mathsf{SMB}}$ to honest $\mathcal{P}_i$'s are always identical to those in the simulated execution, and thus complete the proof. □

Lemma 4. *In the worst case, there are at most three different values in all $values_i$ among honest parties.*

Proof. If a value v is in $values_i$ of an honest party, according to the code, this honest party received $n-f$ $(\textsc{Val}, \mathsf{id}, v)$ messages from distinct parties. This implies that at least $n-2f$ honest parties multicast $(\textsc{Val}, \mathsf{id}, v)$ to all honest parties. It follows that at least $n-3f$ honest parties have received $(\textsc{FilterEcho}, \mathsf{id}, v)$ from at least $n-f$ distinct parties, among which at least $n-2f$ messages are from honest parties. Note that each party can send at most two FilterEcho messages, and thus there are at most $2(n-f)$ FilterEcho messages from honest parties. Therefore, the number of values that can be recorded in $values_i$ is at most $\frac{2(n-f)}{n-2f}$. Recall that we consider $n \geq 3f+1$, so the number will be at most 3. □

Lemma 5. *When $\mathcal{S}$ sends $(\texttt{terminate}, \texttt{sid}, v')$ to $\mathcal{F}_{\mathsf{SMB}}$, $\mathcal{F}_{\mathsf{SMB}}$ will record it as $\hat{v}_k$ for some $k \in [3]$.*

Proof. Recall that $\mathcal{S}$ sends at most three termination messages. For the first value v'_1 sent by $\mathcal{S}$, since it is recorded in $values_i$ by some honest party $\mathcal{P}_i$, there are at least $n-3f$ honest parties have received $(\textsc{FilterEcho}, \mathsf{id}, v'_1)$ from at least $n-f$ distinct parties. According to the code, at least $n-2f$ honest parties have received $(\textsc{Filter}, \mathsf{id}, v'_1)$ from $n-2f$ distinct parties, which include both honest parties whose input is v'_1 and corrupted parties. Therefore, when $\mathcal{S}$ sends the first termination message $(\texttt{terminate}, \texttt{sid}, v'_1)$ to $\mathcal{F}_{\mathsf{SMB}}$, it satisfies $|\mathbb{J}_{v'_1}| + |\mathbb{C}| \geq n-2f$ and thus could be recorded as $\hat{v}_1$.

Regarding the second value v'_2 sent by $\mathcal{S}$, for the same reason, at least $n-3f$ honest parties have received $(\textsc{FilterEcho}, \mathsf{id}, v'_2)$ from at least $n-2f$ honest distinct parties. Considering $n \geq 3f+1$, there is at least one honest party who has sent both $(\textsc{FilterEcho}, \mathsf{id}, v'_1)$ and $(\textsc{FilterEcho}, \mathsf{id}, v'_2)$, which follows that this party has received $f+1$ $(\textsc{Filter}, \mathsf{id}, v'_1)$ messages and $f+1$ $(\textsc{Filter}, \mathsf{id}, v'_2)$ messages from distinct parties, including honest parties whose input value is v'_1 or v'_2 and corrupted parties. Therefore, when $\mathcal{S}$ sends the second termination message $(\texttt{terminate}, \texttt{sid}, v'_2)$ to $\mathcal{F}_{\mathsf{SMB}}$, it satisfies $|\mathbb{J}_{v'_1}| + c_1 \geq n-2f$, $|\mathbb{J}_{v'_2}| + c_2 \geq n-2f$, and $c_1 + c_2 = |\mathbb{C}|$. Therefore, it could be recorded as $\hat{v}_2$.

Finally, if at least $n-2f$ honest parties have the same input value v, then every honest party can receive $n-2f$ $(\textsc{Filter}, \mathsf{id}, v)$ messages from distinct parties, resulting in every honest party multicasting $(\textsc{FilterEcho}, \mathsf{id}, v)$ to all. Due to $n \geq 3f+1$, according to the code, every honest party can multicast at most two different FilterEcho messages. Therefore, in this case, there is no third value in $values_i$ of any honest party $\mathcal{P}_i$. Thus, if $\mathcal{S}$ found the third value v'_3 and sent it to $\mathcal{F}_{\mathsf{SMB}}$, at least $n-f$ honest parties will not have $n-2f$ inputs of the same value. In this case, v'_3 can be recorded as $\hat{v}_3$. □

Lemma 6. *If an honest party output, all honest parties can eventually output; all outputs are subsets of $\{\hat{v}_1, \hat{v}_2, \hat{v}_3\}$.*

Proof. An honest party $\mathcal{P}_i$'s output is a subset of $values_i$. As we shown in Lemma 4, it must be a subset of $\{\hat{v}_1, \hat{v}_2, \hat{v}_3\}$.

We then argue that every value v in $values_i$ will eventually appear in another honest party $\mathcal{P}_j$'s $values_j$, if $\mathcal{P}_j$ has not terminated. Particularly, according to

the code, $\mathcal{P}_i$ received $n - f$ $(\text{VAL}, \mathsf{id}, v)$ messages from distinct parties. This implies that at least $n - 2f$ honest parties multicast $(\text{VAL}, \mathsf{id}, v)$ to all honest parties. Subsequently, all honest parties will multicast $(\text{VAL}, \mathsf{id}, v)$, following the procedure outlined in Algorithm 2. It also indicates that all honest parties can receive $(\text{VAL}, \mathsf{id}, v)$ from $n - f$ honest parties. Consequently, v will be added to the set *values*.

Since every honest party will multicast an AUX message containing a value in its $values_i$, all other honest parties will be able to receive it. Therefore, every honest party could eventually receive at least $n - f$ AUX messages from honest parties. All values in those AUX messages will eventually be included in $values_j$ for any honest $\mathcal{P}_j$, and thus every honest $\mathcal{P}_j$ can eventually terminate. □

Lemma 7. *If there exists an honest party that outputs a single value v^*, then v^* is included in all other honest parties' outputs.*

Proof. According to the algorithm description, every honest party will output a set of values that are conveyed by at least $n - f$ AUX messages. Assume that an honest $\mathcal{P}_i$ outputs accoriding the set AUX_i of $n - f$ AUX messages, while $\mathcal{P}_j$ outputs according to AUX_j. It is easy to see that AUX_i and AUX_j have at least $n-3f$ common messages, which means their outputs val_i and val_j have at least common value. Thus, whenever one honest party outputs a single value, it is in all other honest parties' outputs. □

Lemma 8. *If at least $n - 2f$ honest parties have the same input value v, the protocol always terminates.*

Proof. If at least $n - 2f$ honest parties have the same input value v, then every honest party can receive $n - 2f$ $(\text{FILTER}, \mathsf{id}, v)$ messages from distinct parties, resulting in every honest party multicasting $(\text{FILTERECHO}, \mathsf{id}, v)$ to all. Due to $n \geq 3f + 1$, according to the code, every honest party can multicast at most two different FILTERECHO messages. Since every honest party multicasts $(\text{FILTERECHO}, \mathsf{id}, v)$, every honest party can receive $n - f$ $(\text{FILTERECHO}, \mathsf{id}, v)$ messages from distinct parties; hence, every honest party multicast a $(\text{VAL}, \mathsf{id}, v)$ message to all, resulting in all honest parties eventually adding v to the set *values*. However, it is possible that some honest parties multicast $(\text{VAL}, \mathsf{id}, v')$ messages, where $v \neq v'$. Consequently, it is also possible that $v' \in values_i$. Therefore, all honest parties will multicast a AUX message. Now consider two cases:

- **Case 1:** If some honest parties multicast $(\text{AUX}, \mathsf{id}, v)$ messages, while another set of honest parties multicast $(\text{AUX}, \mathsf{id}, v')$ messages, following the procedure outlined in Algorithm 2, then for any honest party $\mathcal{P}_i$, the set $values_i$ will eventually contain both v and v'.
- **Case 2:** If all honest parties can multicast $(\text{AUX}, \mathsf{id}, \bar{v})$ messages, where $\bar{v} = v$ or v', it is clear that all honest parties will have $values_i = \{\bar{v}\}$.

In either case, every honest party $\mathcal{P}_i$ will meet the condition in line 14 of Algorithm 2. Therefore, every honest party $\mathcal{P}_i$ will terminate and output a set val_i. □

Algorithm 5 The ARC protocol Π_{ARC} with identifier id for $\mathcal{P}_i$

1: **upon** receiving input v_i **do**
2: **multicast** (DIFFUSION, id, v_i) to all
3: **upon** receiving (DIFFUSION, id, v) from node P_j for the first time **do**
4: $D_v \leftarrow D_v \cup \{j\}$
5: **if** $|D_v| = n - f$ and ECHO has not been sent yet **then**
6: **multicast** (ECHO, id, v)
7: **upon** receiving (ECHO, id, v) from node P_j for the first time **do**
8: $EH_v \leftarrow EH_v \cup \{j\}$
9: **if** $|EH_v| = f + 1$ and ECHO has not been sent yet **then**
10: **multicast** (ECHO, id, v)
11: **if** $|EH_v| = n - f$ **then**
12: **return** v

We establish the following corollary for highlighting that the number of possible outputs is only 2 in "good case", which justifies why two ARC instances after each SMB instance is enough. The corollary is implied by Lemma 4 and Lemma 5.

Corollary 1. *If at least $n-2f$ honest parties have the same input value v, then there are at most two different values in all values$_i$ among honest parties.*

F Details and Security Proof of ARC

Following the definition of ARC, if the initial state of an ARC protocol does not satisfy the validity condition, there is no assurance regarding the termination of honest nodes. Consequently, our ARC protocol is deterministic, eliminating the need for reliance on randomness. In contrast to MBA, which, due to the asynchronous network, has to depend on randomness to overcome the FLP impossibility [35]. We present a construction for ARC with IT-security, and the detailed procedure for ARC can be found in Algorithm 5. Below is a detailed description of the process of the ARC protocol:

1. *Diffusion phase* (lines 1-2). All honest nodes are multicast their input v_i via a DIFFUSION message.
2. *Echo phase* (lines 3-10). For any honest node $\mathcal{P}_i$, if it receives $n-f$ DIFFUSION messages carrying the same value v from distinct nodes, then it will multicast the value v via an ECHO message. If it receives $f+1$ ECHO messages carrying the same value v from distinct nodes and has not multicast an ECHO message, then it will multicast an ECHO message along with the value v.
3. *output phase* (lines 11-12). For any honest node $\mathcal{P}_i$, if it receives $n-f$ ECHO messages carrying the same value v from distinct nodes, then it outputs v.

Theorem 10. *The protocol Π_{ARC} in Algorithm 5 perfectly UC-realizes $\mathcal{F}_{\mathsf{ARC}}$, in the presence of any adaptive Byzantine adversary who corrupts up to $f < n/3$ parties.*

Proof. Let $\mathcal{A}$ be an adversary in the real world. We construct a simulator $\mathcal{S}$ in the ideal world, such that no environment $\mathcal{Z}$ can distinguish whether it is interacting with the protocol Π_{ARC} and $\mathcal{A}$, or with $\mathcal{F}_{\mathsf{ARC}}$ and $\mathcal{S}$.

$\mathcal{S}$ follows the "general principles" outlined in the proof of Theorem 1. The simulated execution begins when the ideal functionality $\mathcal{F}_{\mathsf{ARC}}$ provides the messages $(\mathtt{input}, \mathtt{sid}, \cdot, \cdot)$ to $\mathcal{S}$. These messages leak the input values of "so-far-honest" parties to the simulator. Based on these values, $\mathcal{S}$ simulates the execution for $\mathcal{A}$ and interacts with $\mathcal{F}_{\mathsf{ARC}}$ as follows, while following the general principles introduced above.

Simulating a"so-far-honest" party $\mathcal{P}_i$. Upon receiving the input of $\mathcal{P}_i$ from $\mathcal{F}_{\mathsf{ARC}}$, run the code of $\mathcal{P}_i$ with the input (as specified in Algorithm 5), until $\mathcal{P}_i$ returns the output or becomes corrupted by $\mathcal{A}$.

Further interaction with $\mathcal{F}_{\mathsf{ARC}}$: When an honest party outputs a value v, send $(\mathtt{input}, \mathtt{sid}, v)$ on behalf of all corrupted parties, and make sure that all honest parties have participated, i.e., $\mathtt{participated}_i = 1$ for all honest $\mathcal{P}_i$.

Note that at the point of $\mathcal{A}$'s view, the simulated execution is perfectly indistinguishable from a real execution. Since $\mathcal{S}$ just forwards the communication between $\mathcal{A}$ and $\mathcal{Z}$, it remains to show that the outputs from $\mathcal{F}_{\mathsf{ARC}}$ to honest $\mathcal{P}_i$'s are identical to those in the simulated execution (and thus distinguishable with those in a real execution).

To demonstrate this, we establish the following facts: (1)Lemma 9: When an honest party outputs v, $\mathcal{S}$ can make sure at least $n - f$ parties (including corrupted and honest ones) have participated with v in the ideal functionality. (2) Lemma 10: When an honest party outputs v, all other honest parties eventually output the same v. (3) Lemma 11: If there are $n - f$ "forever-honest" parties with the input v, all honest parties will output v. The first two ensure whenever an honest party outputs in a real execution, the dummy parties output the same in the ideal world. The last one ensures that whenever the parties should output in the ideal world, they also output in the real execution. □

Lemma 9. *When an honest party outputs v, $\mathcal{S}$ can make sure at least $n - f$ parties (including corrupted and honest ones) have participated with v in the ideal functionality.*

Proof. If an honest node $\mathcal{P}_i$ outputs v, according to the Algorithm 5, then $\mathcal{P}_i$ received $n - f$ $(\textsc{Echo}, \mathrm{id}, v)$ messages from distinct nodes. Thus, at least one honest node received $n - f$ identical $(\textsc{Diffusion}, \mathrm{id}, v)$ messages from distinct nodes. The senders of those Diffusion messages include both honest parties with v as inputs or corrupted parties. Therefore, $\mathcal{S}$ will be able to make at least $n - f$ parties participate the ideal functionality with input v. □

Lemma 10. *When an honest party outputs v, all other honest parties eventually output the same v.*

Proof. First, we show that when one honest node $\mathcal{P}_i$ multicasts $(\textsc{Echo}, \mathrm{id}, v)$ and another honest node $\mathcal{P}_j$ multicasts $(\textsc{Echo}, \mathrm{id}, v')$, then $v = v'$. If an honest node $\mathcal{P}_i$ multicasts $(\textsc{Echo}, \mathrm{id}, v)$, according to the code, $\mathcal{P}_i$ received $(\textsc{Diffusion}, \mathrm{id}, v)$

from $n-f$ distinct nodes. If another honest node $\mathcal{P}_j$ multicasts $(\text{ECHO}, \text{id}, v')$, it implies that $\mathcal{P}_j$ received $(\text{DIFFUSION}, \text{id}, v')$ from $n-f$ distinct nodes. Since there are at most f malicious nodes, at least $n-f \geq 2f+1$ honest nodes multicast $(\text{ECHO}, \text{id}, v)$ and $(\text{ECHO}, \text{id}, v')$. If $v \neq v'$, based on the assumption $n \geq 3f+1$, it implies that one honest node multicasts two different DIFFUSION messages, leading to a contradiction. Therefore, $v = v'$.

Then, when an honest node $\mathcal{P}_i$ outputs v, according to the code, $\mathcal{P}_i$ received $n-f$ identical $(\text{ECHO}, \text{id}, v)$ messages from distinct nodes. Due to the presence of at most f malicious nodes, at least $f+1$ honest nodes have multicast $(\text{ECHO}, \text{id}, v)$ messages to all. As a result, all honest nodes will receive at least $f+1$ $(\text{ECHO}, \text{id}, v)$ messages from distinct nodes, leading to all honest nodes multicasting an ECHO message to all. Given the previous analysis, since all ECHO messages carry the same values, so all honest nodes can receive $n-f$ ECHO messages carrying the same value v. Consequently, all honest nodes output v.

Lemma 11. *If there are $n-f$ "forever-honest" parties with the input v, all honest parties will output v.*

Proof. If $n-f$ "forever-honest" parties have the same input value v, by the code, every honest node will multicast $(\text{ECHO}, \text{id}, v)$ messages to all nodes. Hence, every honest node will receive at least $n-f$ $(\text{ECHO}, \text{id}, v)$ messages, resulting in all honest nodes outputting v. □

G Security proof of MVBA

Theorem 11. *Assuming the underlying hash function is collision resistant, the protocol Π_{MVBA} in Algorithm 3 UC-realizes $\mathcal{F}_{\mathsf{MVBA}}$ in the $(\mathcal{F}_{\mathsf{coin}}^{[n]^\kappa}, \mathcal{F}_{\mathsf{ABA}}, \mathcal{F}_{\mathsf{SMID}}, \mathcal{F}_{\mathsf{ARC}}, \mathcal{F}_{\mathsf{SMB}})$-hybrid model, in the presence of a computationally bounded and adaptive Byzantine adversary who may corrupt up to $f < n/3$ nodes.*

Proof. Let $\mathcal{A}$ be an adversary in the real world. We construct a simulator $\mathcal{S}$ in the ideal world, such that no PPT environment $\mathcal{Z}$ can distinguish whether it is interacting with the protocol Π_{MVBA} and $\mathcal{A}$, or with the ideal functionality $\mathcal{F}_{\mathsf{MVBA}}$ and $\mathcal{S}$.

$\mathcal{S}$ follows the "general principles" outlined in the proof of Theorem 1. The simulated execution begins when the ideal functionality $\mathcal{F}_{\mathsf{MVBA}}$ provides the messages $(\texttt{input}, \texttt{sid}, \cdot, \cdot)$ to $\mathcal{S}$. These messages leak the input values of "so-far-honest" nodes to the simulator. Based on these values, $\mathcal{S}$ simulates the execution for $\mathcal{A}$ and interacts with $\mathcal{F}_{\mathsf{MVBA}}$ as follows, while following the general principles introduced above.

Simulating a "so-far-honest" node $\mathcal{P}_i$. Upon receiving $(\texttt{input}, \texttt{sid}, v', \mathcal{P}_i)$ from $\mathcal{F}_{\mathsf{MVBA}}$, run the code of $\mathcal{P}_i$ with the input v' (as specified in Algorithm 3), until $\mathcal{P}_i$ returns the output or becomes corrupted by $\mathcal{A}$.

Simulating $\mathcal{F}_{\mathsf{SMID}}$. Run the hybrid $\mathcal{F}_{\mathsf{SMID}}$ as an ITM by follwoing the code in Fig.5. Moroever, upon receiving $(\texttt{disperse}, \texttt{sid}, v')$ from the adversary $\mathcal{A}$ on behalf of a corrupted node, send $(\texttt{input}, \texttt{sid}, v')$ to $\mathcal{F}_{\mathsf{MVBA}}$ on behalf of the

corrupted node. Additionally, keep track of the set of nodes $\mathbb{I}_{\mathsf{SMID}}$ who have dispersed valid values, *i.e.*, for $i \in \mathbb{I}_{\mathsf{SMID}}$, $\mathsf{Predicate}(v_i) = 1$ and $\mathtt{dispersed}_i = 1$.
Simulating $\mathcal{F}_{\mathsf{coin}}^{V}$. Run a modified copy of the hybrid $\mathcal{F}_{\mathsf{coin}}^{V}$ as an ITM. The modified copy almost follows the specification in Fig.9, except that the output $(s_1, \ldots, s_\kappa)$ is uniformly sampled from $V = [n]^\kappa$ under the condition that at least one $s_z \in \mathbb{I}_{\mathsf{SMID}}$.
Simulating other hybrids. Run $\mathcal{F}_{\mathsf{ABA}}, \mathcal{F}_{\mathsf{ARC}}$ and $\mathcal{F}_{\mathsf{SMB}}$ as ITMs by following the theirs codes in Fig.10, 7, and 6, respectively.
Further interaction with $\mathcal{F}_{\mathsf{MVBA}}$. Upon an honest node returns $(\mathtt{output}, \mathtt{sid}, v)$ for the first time, send $(\mathtt{set\text{-}output}, \mathtt{sid}, v)$ to $\mathcal{F}_{\mathsf{MVBA}}$. Make sure no nodes have received their output message in the ideal world before the `set-output` message is sent.

Given the above simulator $\mathcal{S}$, in Lemma 12, we prove that at the point of $\mathcal{A}$'s view, the simulated execution is statistically indistinguishable from a real execution. Denoting the output distribution of all honest nodes in the real execution (resp. the simulated execution) by $\mathsf{Dist}_{\mathsf{real}}$ (resp. $\mathsf{Dist}_{\mathsf{sim}}$), and the view of adversary in the real execution (resp. the simulated execution) by $\mathsf{Aview}_{\mathsf{real}}$ (resp. $\mathsf{Aview}_{\mathsf{sim}}$). It holds the following equation:

$$(\mathsf{Dist}_{\mathsf{real}}, \mathsf{AView}_{\mathsf{real}}) \overset{s}{\approx} (\mathsf{Dist}_{\mathsf{sim}}, \mathsf{AView}_{\mathsf{sim}}). \tag{1}$$

In Lemma 13, we prove that all honest nodes eventually terminate in the simulated execution, and that all honest nodes will output the same valid value in the simulated execution, such that the output of $\mathcal{F}_{\mathsf{MVBA}}$ is identical to that of the simulated execution. Denoting the output distribution of dummy nodes in the ideal world by $\mathsf{Dist}_{\mathsf{ideal}}$, it holds the following equation:

$$(\mathsf{Dist}_{\mathsf{ideal}}, \mathsf{AView}_{\mathsf{sim}}) \overset{c}{\approx} (\mathsf{Dist}_{\mathsf{sim}}, \mathsf{AView}_{\mathsf{sim}}). \tag{2}$$

Note that the simulator $\mathcal{S}$ always forwards the message between $\mathcal{A}$ and the environment $\mathcal{Z}$. Without loss of generality, for any polynomial-time environment $\mathcal{Z}$, we consider there is a PPT algorithm $\mathsf{ZOutput}$, such that the binary output of $\mathcal{Z}$ is the output of $\mathsf{ZOutput}$ on inputs the output of honest nodes and the adversary's view. Namely, $\mathsf{EXEC}_{\Pi_{\mathsf{MVBA}}, \mathcal{A}, \mathcal{Z}} = \mathsf{ZOutput}(\mathsf{Dist}_{\mathsf{real}}, \mathsf{AView}_{\mathsf{real}})$, and $\mathsf{IDEAL}_{\mathcal{F}, \mathcal{S}, \mathcal{Z}} = \mathsf{ZOutput}(\mathsf{Dist}_{\mathsf{ideal}}, \mathsf{AView}_{\mathsf{sim}})$. Putting Eq.1 and 2 together, we have the following results:

$$\mathsf{EXEC}_{\Pi_{\mathsf{MVBA}}, \mathcal{A}, \mathcal{Z}} \overset{c}{\approx} \mathsf{IDEAL}_{\mathcal{F}, \mathcal{S}, \mathcal{Z}}, \tag{3}$$

which completes the proof of the theorem. □

Lemma 12. *For any $\mathcal{A}$, in the simulated execution provided by $\mathcal{S}$, the view of $\mathcal{A}$, denoted by $\mathsf{AView}_{\mathsf{sim}}$, and the output distribution of honest nodes, denoted by $\mathsf{Dist}_{\mathsf{sim}}$, are statistically indistinguishable with the view and the distribution in a real execution. Namely, $(\mathsf{Dist}_{\mathsf{real}}, \mathsf{AView}_{\mathsf{real}}) \overset{s}{\approx} (\mathsf{Dist}_{\mathsf{sim}}, \mathsf{AView}_{\mathsf{sim}})$.*

Proof. Note that at the point of $\mathcal{A}$'s view, the only difference between the simulated execution and a real execution is how $\mathcal{F}_{\mathsf{coin}}^{V}$ returns the common randomness $r \in [n]^{\kappa}$. In a real execution of Π_{MVBA} in $\mathcal{F}_{\mathsf{coin}}^{V}$-hybrid model, r is from the uniform distribution over $V = [n]^{\kappa}$, which we denote by U_V. In the simulated execution, r is from the uniform distribution over $\widehat{\mathbb{B}_{\mathsf{SMID}}} = V \setminus \mathbb{B}_{\mathsf{SMID}}$, denoted by $U_{\widehat{\mathbb{B}_{\mathsf{SMID}}}}$, where $\mathbb{B}_{\mathsf{SMID}}$ is the subset of V such that any $\vec{b}$ does not contain any index in $\mathbb{I}_{\mathsf{SMID}}$ (which is the set of nodes who have successfully dispersed valid values in $\mathcal{F}_{\mathsf{SMID}}$). Note that the size of $\mathbb{B}_{\mathsf{SMID}}$ is bounded by $(2f)^{\kappa}$. The statistical distance between U_V and $U_{\widehat{\mathbb{B}_{\mathsf{SMID}}}}$ is therefore bounded by

$$\Delta_{U_V, U_{\widehat{\mathbb{B}_{\mathsf{SMID}}}}} = \frac{1}{2}\left(\left(\frac{1}{|\widehat{\mathbb{B}_{\mathsf{SMID}}}|} - \frac{1}{|V|}\right) \cdot |\widehat{\mathbb{B}_{\mathsf{SMID}}}| + \frac{1}{|V|} \cdot |\mathbb{B}_{\mathsf{SMID}}|\right) < \left(\frac{f}{n}\right)^{\kappa},$$

which is negligible in the security parameter κ. Note that the statistical distance between $(\mathsf{Dist}_{\mathsf{real}}, \mathsf{AView}_{\mathsf{real}})$ and $(\mathsf{Dist}_{\mathsf{sim}}, \mathsf{AView}_{\mathsf{sim}})$ is bounded by $\Delta_{U_V, U_{\widehat{\mathbb{B}_{\mathsf{SMID}}}}}$, which is therefore negligible. □

Lemma 13. *For any polynomial-time adversary $\mathcal{A}$, all honest nodes in the simulated execution provided by $\mathcal{S}$ will terminate eventually and output the same valid value, with an overwhelming probability, under the condition that all honest nodes provide valid inputs.*

Proof. First, by the description of $\mathcal{F}_{\mathsf{SMID}}$, all "so-far-honest" nodes can eventually receive $(\texttt{disperse-done}, \texttt{sid})$ from $\mathcal{F}_{\mathsf{SMID}}$, as long as there are $n - f$ nodes having sent $(\texttt{disperse}, \texttt{sid}, \cdot)$ to $\mathcal{F}_{\mathsf{SMID}}$. By the protocol description in Algorithm 3 (line 3-4), all "so-far-honest" nodes should send $(\texttt{request}, \texttt{sid})$ to $\mathcal{F}_{\mathsf{coin}}^{V}$. By the description of $\mathcal{F}_{\mathsf{coin}}^{V}$ and $\mathcal{S}$, all "so-far-honest" nodes will receive a vector of κ indexes $(s_1, \ldots, s_\kappa)$, among which there is at least one $s_{\hat{z}}$ such that $v_{s_{\hat{z}}}$ has been dispersed in $\mathcal{F}_{\mathsf{SMID}}$ and $\mathsf{Predicate}(v_{s_{\hat{z}}}) = 1$. We assume the number of "so-far-honest" nodes is H, which is always not less than $n - f$. By the description of $\mathcal{F}_{\mathsf{SMID}}$, at least $H - f$ "so-far-honest" nodes can receive $(\texttt{recast-output}, \texttt{sid}, s_{\hat{z}}, v_{s_{\hat{z}}})$ from $\mathcal{F}_{\mathsf{SMID}}$.

We now focus on the sub-sessions with the prefix of $\texttt{sid}|\hat{z}$, and argue how these sub-sessions can guarantee termination of the execution. Following line 7-12 in Algorithm 3, these $H - f$ "so-far-honest" nodes encode $v_{s_{\hat{z}}}$ via erasure code, apply the hash-based vector commitment to the code blocks, and send $(\texttt{input}, \texttt{sid}|\hat{z}, \mathsf{vc}_{\hat{z}})$ to $\mathcal{F}_{\mathsf{SMB}}$, where $\mathsf{vc}_{\hat{z}}$ is the vector commitment value of the code blocks. For other "so-far-honest" nodes who did not receive $v_{s_{\hat{z}}}$ but received other valid values, they follow the same steps. By the description of $\mathcal{F}_{\mathsf{SMB}}$, since these $H - f$ nodes can provide the same input to $\mathcal{F}_{\mathsf{SMB}}$, all "so-far-honest" nodes will receive either $\{\mathsf{vc}\}$ or $\{\mathsf{vc}', \mathsf{vc}''\}$, where vc equals to either vc' or vc'', and both vc' and vc'' have been provided as inputs to $\mathcal{F}_{\mathsf{SMV}}$ by at least $n - 2f$ nodes, which means at least $n - 3f$ "forever-honest" nodes have the entire messages of the outputted vector commitments.

Next, by the code, all "so-far-honest" nodes will send $(\texttt{input}, \texttt{sid}|\hat{z}|1, \mathsf{vc}')$ and $(\texttt{input}, \texttt{sid}|\hat{z}|2, \mathsf{vc}'')$ to $\mathcal{F}_{\mathsf{ARC}}$. Note that at least for one $\hat{a} \in \{1, 2\}$, all

"so-far-honest" nodes send the same vc to $\mathcal{F}_{\mathsf{ARC}}$. By the description of $\mathcal{F}_{\mathsf{ARC}}$, all "so-far-honest" nodes will eventually receive the same $(\mathtt{output}, \mathtt{sid}|\hat{z}|\hat{a}, \mathsf{vc})$ from $\mathcal{F}_{\mathsf{ARC}}$.

Next, we argue that all "so-far-honest" nodes will receive $(\mathtt{output}, \mathtt{sid}|z|a, \dot{)}$ from $\mathcal{F}_{\mathsf{ABA}}$ for all $z \in [\kappa]$ and $a \in \{1, 2\}$. By the description of Algorithm 3 (line 25), all "so-far-honest" nodes will provide inputs with all the session IDs to $\mathcal{F}_{\mathsf{ABA}}$, under the condition that $\mathcal{F}_{\mathsf{ABA}}$ already outputs $(\mathtt{output}, \mathtt{sid}|z|a, 1)$ for some z and a. Meanwhile, according to line 21, all "so-far-honest" will send $(\mathtt{input}, \mathtt{sid}|\hat{z}|\hat{a}, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$, if $\mathcal{F}_{\mathsf{ABA}}$ has not returned 1. Putting them together, all "so-far-honest" nodes eventually provide inputs with all the session IDs to $\mathcal{F}_{\mathsf{ABA}}$, and thus will be able to receive $(\mathtt{output}, \mathtt{sid}|z|a, \dot{)}$ from $\mathcal{F}_{\mathsf{ABA}}$ for all $z \in [\kappa]$ and $a \in \{1, 2\}$. In other words, all "so-far-honest" nodes can execute the code block starting from line 26 in Algorithm 3.

Next, we show that for the smallest (z^*, a^*) such that $(\mathtt{output}, \mathtt{sid}|z^*|a^*, 1)$ is returned by $\mathcal{F}_{\mathsf{ABA}}$, all "so-far-honest" nodes will eventually receive the same vector commitment vc_{z^*,a^*}. Note that at least $n-2f$ nodes have sent $(\mathtt{input}, \mathtt{sid}|z^*|a^*, 1)$ to $\mathcal{F}_{\mathsf{ABA}}$, such that at least $n-3f$ "forever-honest" nodes sent this message. According to line 21 in Algorithm 3, at least $n-3f$ "forever-honest" nodes received $(\mathtt{output}, \mathtt{sid}|z^*|a^*, \mathsf{vc}_{z^*,a^*})$ from $\mathcal{F}_{\mathsf{ARC}}$. By the description of $\mathcal{F}_{\mathsf{ARC}}$, all "so-far-honest" nodes will eventually receive the same output $(\mathtt{output}, \mathtt{sid}|z^*|a^*, \mathsf{vc}_{z^*,a^*})$.

Then, we show at least $n-3f$ "forever-honest" nodes have the entire message v^* whose vector commitment value is vc_{z^*,a^*}. By the description of $\mathcal{F}_{\mathsf{ARC}}$, at least $n-f$ nodes have sent $(\mathtt{output}, \mathtt{sid}|z^*|a^*, \mathsf{vc}_{z^*,a^*})$ to $\mathcal{F}_{\mathsf{ARC}}$, which means at least $n-2f$ "forever-honest" nodes have sent this message. According to line 16-17, at least $n-2f$ "forever-honest" nodes have received $(\mathtt{output}, \mathtt{sid}|z^*, \mathsf{vset})$ from $\mathcal{F}_{\mathsf{SMB}}$, such that $\mathsf{vc}_{z^*,a^*} \in \mathsf{vset}$. By the description, the $\mathcal{F}_{\mathsf{SMB}}$, vc_{z^*,a^*} must be provided by $n-2f-|\mathcal{C}|$ "so-far-honest" nodes, such that at least $n-3f$ "forever-honest" nodes provided it to $\mathcal{F}_{\mathsf{SMB}}$. According to line 8-12, at least $n-3f$ "forever-honest" nodes have the entire valid message v^* for the commitment vc_{z^*,a^*}.

Finally, we show all "so-far-honest" nodes eventually terminate and output the same valid value. Since there exist at least $n-3f$ "forever-honest" nodes who have a valid value v^* whose commitment is vc_{z^*,a^*}, every "so-far-honest" node $\mathcal{P}_j$ can eventually receive the $(\mathtt{fragment}, \mathtt{sid}, \mathsf{vc}_{z^*,a^*}, c_j, \pi_j)$ which contains a valid fragment, and thus multicasts this fragment to all (line 33-35). Finally, all "so-far-honest" nodes can receive $f+1$ valid fragments and reconstruct a message. By the security of the vector commitment, all honest nodes reconstruct the same message, which is identical to the one held by those $n-3f$ "forever-honest" nodes and thus is valid. We therefore completed the proof. □

H The Implementation and Security Proof of ACS

Theorem 12. *Assuming the underlying signature scheme satisfies the existential unforgeability, the protocol Π_{ACS} UC-realizes $\mathcal{F}_{\mathsf{ACS}}$ in the $(\mathcal{F}_{\mathsf{MVBA}}, \mathcal{F}_{\mathsf{PKI}})$-*

Algorithm 6 ACS protocol (for each party $\mathcal{P}_i$), adapted from Fig 3 in [16]

Let $\Sigma = \{\mathsf{KeyGen}, \mathsf{Sign}, \mathsf{Vrfy}\}$ be the underlying signature scheme.
Initialize: $\mathbb{S} \leftarrow \emptyset$, $\mathsf{VK} \leftarrow \emptyset$, $\mathsf{registered} = 0$. Define $\mathsf{Predicate}$ of $\mathcal{F}_{\mathsf{MVBA}}$ as follows: $\mathsf{Predicate}(\{(j, v_j, \sigma_j)\}_{j\in\mathbb{J}}) = 1$, if and only if: (1) $\mathbb{J} \subset [n]$; (2) $|\mathbb{J}| = n - f$; and (3) $\Sigma.\mathsf{Vrfy}(vk_j, v_j, \sigma_j) = 1$ for all $j \in \mathbb{J}$.

1: **if** $\mathsf{registered} = 0$ **then** ▷ PKI-Setup
2: $(vk_i, sk_i) \leftarrow \mathsf{KeyGen}(1^\lambda)$
3: **send** $(\mathtt{register}, \mathtt{sid}, vk_i)$ to $\mathcal{F}_{\mathsf{PKI}}$, update $\mathsf{registered} = 1$
4: **send** $(\mathtt{retrive}, \mathtt{sid}, j)$ to $\mathcal{F}_{\mathsf{PKI}}$ for all $j \in [n]$
5: **upon** receiving $(\mathtt{retrieved}, \mathtt{sid}, j, vk_j)$ from $\mathcal{F}_{\mathsf{PKI}}$ **do**
6: Update $\mathsf{VK} = \mathsf{VK} \cup \{(j, vk_j)\}$
7: **upon** receiving input v_i **do**
8: Sign v_i: $\sigma_i \leftarrow \Sigma.\mathsf{Sign}(vk_i, sk_i, v_i)$
9: **multicast** $(\textsc{Diffusion}, (v_i, \sigma_i))$ to all
10: **upon** receive $(\textsc{Diffusion}, (v_j, \sigma_j))$ message from $\mathcal{P}_j$ for the first time **do**
11: **if** $\Sigma.\mathsf{Vrfy}(vk_j, v_j, \sigma_j) = 1$ and $|\mathbb{S}| < n - f$ **then**
12: Update $\mathbb{S} = \mathbb{S} \cup \{(j, v_j, \sigma_j)\}$
13: **wait** until $|\mathbb{S}| = n - f$
14: **send** $(\mathtt{input}, \mathtt{sid}, \mathbb{S})$ to $\mathcal{F}_{\mathsf{MVBA}}$
15: **wait** receiving $(\mathtt{output}, \mathtt{sid}, \mathbb{S}')$ from $\mathcal{F}_{\mathsf{MVBA}}$
16: Parse $\mathbb{S}' = \{(j, v_j, \sigma_j)\}_{j\in\mathbb{J}}$, and let $\mathsf{commonSet} = \{(j, v_j)\}_{j\in\mathbb{J}}$.
17: **return** $\mathsf{commonSet}$.

hybrid model, against any computationally bounded adaptive Byzantine adversary who may corrupt up to $f < n/3$ nodes.

Proof. Let $\mathcal{A}$ be an adversary in the real world. We construct a simulator $\mathcal{S}$ in the ideal world, scuh that no polynomial-time environment $\mathcal{Z}$ can distinguish whether it is interacting with the protocol Π_{ACS} and $\mathcal{A}$, or with $\mathcal{F}_{\mathsf{ACS}}$ and $\mathcal{S}$.

$\mathcal{S}$ follows the "general principles" outlined in the proof of Theorem 1. In the simulated execution, $\mathcal{S}$ honestly executes the codes of honest nodes, $\mathcal{F}_{\mathsf{PKI}}$ and $\mathcal{F}_{\mathsf{MVBA}}$. Moreover, it interacts with $\mathcal{F}_{\mathsf{ACS}}$ as follows: When the hybrid $\mathcal{F}_{\mathsf{MVBA}}$ returns $(\mathtt{output}, \mathtt{sid}, \mathbb{S} = \{(j, v_j, \sigma_j)\}_{j\in\mathbb{J}})$ to an honest node, $\mathcal{S}$ identifies the subset $\mathbb{J}_H = \mathbb{J} \cap \mathbb{H}$, where $\mathbb{H}$ is the set of all "so-far-honest" nodes. Then, $\mathcal{S}$ manages the delay to make sure all $\mathcal{P}_j$ for $j \in \mathbb{J}_H$ have participated in $\mathcal{F}_{\mathsf{ACS}}$, i.e., $\mathtt{participated}_j = 1$, while sending $(\mathtt{input}, \mathtt{sid}, v_k)$ on behalf of all corrupted $\mathcal{P}_k$ for $k \in \mathbb{J} \setminus \mathbb{J}_H$ without delays. All other inputs from $\mathcal{P}_j$ for $j \notin \mathbb{J}$ are delayed.

As $\mathcal{S}$ honestly executes the codes of honest nodes and hybrids, at the point of $\mathcal{A}$'s view, the simulated execution is identical to a real execution. Thus, it remains to show if the outputs of honest nodes from $\mathcal{F}_{\mathsf{ACS}}$ are identical to those in the simulated execution.

By the description, with $n - f$ nodes following the protocol, all honest nodes can eventually receive $n - f$ signed values and thus send the values to $\mathcal{F}_{\mathsf{MVBA}}$. By the description of $\mathcal{F}_{\mathsf{MVBA}}$, all honest nodes will receive the same output from $\mathcal{F}_{\mathsf{MVBA}}$ which satisfies the predicate, *i.e.*, it contains $n - f$ value/signature pairs created by distinct nodes. Ensured by the existential unforgeability of the

Functionality $\mathcal{F}_{\mathsf{Input}}$

$\mathcal{F}_{\mathsf{Input}}$ proceeds as follows, running with $(\mathcal{P}_1, \ldots, \mathcal{P}_n)$ and the adversary. At the first activation, verify $(\mathtt{sid}, \{\mathcal{P}_1, \ldots, \mathcal{P}_n\})$ for the session ID $\mathtt{sid}$ and the set of nodes. Initialize: $\mathtt{participated}_i = 0$, $x_i = \perp$ and $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{output}} = 1$ for all $i \in [n]$; $x_{i,j} = \perp$, for all $i, j \in [n]$; $\mathsf{participants} = \emptyset$. Let $\mathbb{C}$ be the set of corrupted nodes.

- Upon receiving $(\mathtt{delay}, \mathtt{sid}, \mathcal{P}_i, \mathtt{type}, D)$ from the adversary for any $i \in [n]$, if $\mathtt{type} \in \{\mathtt{input}, \mathtt{output}\}$, and $D \in \mathbb{Z}$ represented in unary notation, then update $\mathsf{D}_i^{\mathtt{type}} = \max\{1, \mathsf{D}_i^{\mathtt{type}} + D\}$, and provide $(\mathtt{delay\text{-}set}, \mathtt{sid})$ to the adversary.
- Upon receiving $(\mathtt{corrupt}, \mathtt{sid}, \mathcal{P}_i)$ from the adversary, if $|\mathbb{C}| < f$, then $\mathbb{C} = \mathbb{C} \cup \{i\}$, and return $(\mathtt{corrupted}, \mathtt{sid}, \mathcal{P}_i, x)$ to the adversary, where x is the input from $\mathcal{P}_i$.
- Upon receiving $(\mathtt{input}, \mathtt{sid}, v')$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and provide $(\mathtt{input}, \mathtt{sid}, |v'|, \mathcal{P}_i)$ to the adversary.
- Upon receiving $(\mathtt{fetch}, \mathtt{sid})$ from $\mathcal{P}_i$ (or the adversary on behalf of a corrupted node), run the **Input Submission Procedure** and the **Output Release Procedure**, and provide $(\mathtt{fetch}, \mathtt{sid}, \mathcal{P}_i)$ to the adversary.

Input Submission Procedure: If $\mathtt{participated}_i = 0$, and $(\mathtt{input}, \mathtt{sid}, v')$ has been provided by $\mathcal{P}_i$, do the following:

- Update $\mathsf{D}_i^{\mathtt{input}} = \mathsf{D}_i^{\mathtt{input}} - 1$;
- If $\mathsf{D}_i^{\mathtt{input}} = 0$, update $\mathtt{participated}_i = 1$; If $|\mathsf{participants}| < n - f$, update $\mathsf{participants} \leftarrow \mathsf{participants} \cup \{i\}$; Record $x_i = v'$, and update $(x_{i,1}, \ldots, x_{i,n}) \leftarrow \mathsf{SS}(x_i)$.

Output Release Procedure: If $|\mathsf{participants}| = n - f$, do the following:

- Update $\mathsf{D}_i^{\mathtt{output}} = \mathsf{D}_i^{\mathtt{output}} - 1$;
- if $\mathsf{D}_i^{\mathtt{output}} = 0$, set $(\mathtt{output}, \mathtt{sid}, \mathsf{participants}, \{(x_{j,i})\}_{j \in \mathsf{participants}})$ to be sent to $\mathcal{P}_i$.

Fig. 11. The input functionality $\mathcal{F}_{\mathsf{Input}}$ w.r.t a secret sharing scheme SS

underlying signature scheme, among the $n - f$ values, each signed by an honest node must be the input value of the node. According to how $\mathcal{S}$ specifies the inputs of corrupted nodes to $\mathcal{F}_{\mathsf{ACS}}$, it follows that an honest node's output in the simulated execution is identical to the output from $\mathcal{F}_{\mathsf{ACS}}$. □

I The functionality and security Proof of Π_{input}

Theorem 13. *The protocol Π_{input} UC-realizes $\mathcal{F}_{\mathsf{Input}}$ in the $\mathcal{F}_{\mathsf{ACS}}$-hybrid model, against any adaptive Byzantine adversary who corrupts up to $n/3$ nodes.*

Proof. For any $\mathcal{A}$, we build a simulator $\mathcal{S}$, such that any environment $\mathcal{Z}$ cannot decide whether it is interacting with Π_{input} and $\mathcal{A}$, or with $\mathcal{F}_{\mathsf{Input}}$.

The simulator $\mathcal{S}$ follows the "general principles" outlined in the proof of Theorem 1. In the simulated execution, $\mathcal{S}$ honestly executes the code of $\mathcal{F}_{\mathsf{ACS}}$. In the following, we detail how $\mathcal{S}$ simulates the preprocessing phase and a "so-far-honest" node, handles adaptive corruptions, and interacts with $\mathcal{F}_{\mathsf{Input}}$.

Simulating the preprocessing phase. For every corrupted node $\mathcal{P}_i$, uniformly sample $\{r_i, \{r_{j,i}\}_{j \in [n]}\}$ and sent them to $\mathcal{P}_i$. Send $\{r_{i,j}\}_{i \in \mathbb{C}}$ to each honest node $\mathcal{P}_j$, where $\mathbb{C}$ is the set of corrupted nodes.

Simulating a "so-far-honest" node $\mathcal{P}_i$. Upon receiving $(\texttt{input}, \texttt{sid}, |v'|, \mathcal{P}_i)$ from $\mathcal{F}_{\mathsf{MVBA}}$, uniformly samples a value $\bar{x}$ from the domain of v', and send $(\texttt{input}, \texttt{sid}, \bar{x})$ to $\mathcal{F}_{\mathsf{ACS}}$ on behalf of $\mathcal{P}_i$.

Handling adaptive corruption. When an honest node $\mathcal{P}_i$ becomes corrupted, send $(\texttt{corrupt}, \texttt{sid}, \mathcal{P}_i)$ to $\mathcal{F}_{\mathsf{Input}}$ and obtain $(\texttt{corrupted}, \texttt{sid}, \mathcal{P}_i, x)$. Then, if $(\texttt{input}, \texttt{sid}, x)$ has been sent to $\mathcal{F}_{\mathsf{ACS}}$ on behalf of $\mathcal{P}_i$, let $r_i \leftarrow \bar{x} - x$, and generate the secret shares of r_i for honest nodes $\{r_{i,j}\}_{j\in\mathbb{H}}$ such that the shares are consistent with $\{r_{i,j}\}_{j\in\mathbb{C}}$, where $\mathbb{H}$ and $\mathbb{C}$ are the set of honest nodes and the set of corrupted nodes, respectively. Moreover, uniformly sample $\{r_{j,i}\}_{j\in\mathbb{H}}$. Finally, return $(x, r_i, \{r_{j,i}\}_{j\in[n]})$ to the adversary.

Interaction with $\mathcal{F}_{\mathsf{Input}}$. After receiving message $(\texttt{output}, \texttt{sid}, \mathsf{commonSet})$ from $\mathcal{F}_{\mathsf{ACS}}$, parse $\mathsf{commonSet} = \{j, \bar{x}_j\}_{j\in\mathbb{J}}$. For $\mathbb{J}_1 = \mathbb{J}\cap\mathbb{H}$ and $j \in \mathbb{J}_1$, manage the delay such that $\texttt{participated}_j = 1$ in $\mathcal{F}_{\mathsf{Input}}$. For $\mathbb{J}_2 = \mathbb{C} \cap \mathbb{J}$, compute $x_j = \bar{x}_j - r_j$ for all $j \in \mathbb{J}_2$, and send $(\texttt{input}, \texttt{sid}, x_j)$ to $\mathcal{F}_{\mathsf{Input}}$ on behalf of $\mathcal{P}_j$ without any delays.

It is easy to verify the simulated execution is perfectly indistinguishable from a real execution, and the output of a dummy node from $\mathcal{F}_{\mathsf{Input}}$ is also identical to its output in a real execution. □