Dumbo-MVBA: Optimal Multi-value Validated Asynchronous Byzantine Agreement, Revisited *

Yuan Lu[1], Zhenliang Lu[2], Qiang Tang[2], and Guiling Wang[3]

[1] Institute of Software Chinese Academy of Sciences
[2] The University of Sydney
[3] New Jersey Institute of Technology
luyuan@iscas.ac.cn,{zhenliang.lu,qiang.tang}@sydney.edu.au,gwang@njit.edu

Abstract. Multi-value validated asynchronous Byzantine agreement (MVBA), proposed in the seminal work of Cachin et al. (CRYPTO '01), is fundamental for critical fault-tolerant services such as atomic broadcast in the asynchronous network. It was left as an open problem to asymptotically reduce its $\mathcal{O}(Ln^2 + \lambda n^2 + n^3)$ communication cost in the authenticated setting with setup assumptions (where n is the number of parties, L is the input length, and λ is the security parameter). Recently, Abraham et al. (PODC '19) removed the n^3 term to partially answer the question when the input size L is small. However, in other typical cases, e.g., building atomic broadcast through MVBA, the input length of MVBA $L \geq \lambda n$, and thus the communication is dominated by the Ln^2 term and the problem raised by Cachin et al. remains open. We fill the gap and answer the remaining part of this open problem. In particular, we present two MVBA protocols with expected $\mathcal{O}(Ln+\lambda n^2)$ communication cost, which is asymptotically optimal when $L \geq \lambda n$. Our MVBA protocols also maintain other benefits including optimal resilience to tolerate up to $n/3$ adaptive Byzantine corruptions, asymptotically optimal $\mathcal{O}(1)$ expected rounds and $\mathcal{O}(n^2)$ expected messages.

At the core of our design, we propose the notion of asynchronous provable dispersal broadcast (APDB), in which each input can be split and dispersed to every party and later recovered in an efficient way using only $\mathcal{O}(n)$ messages. In particular, the dispersal phase of APDB can be implemented with only $\mathcal{O}(n)$ messages. Leveraging APDB and binary asynchronous Byzantine agreement, we design our first optimal MVBA protocol, Dumbo-MVBA, showing the feasibility of completely answering the long-standing problem of achieving expected $\mathcal{O}(Ln+\lambda n^2)$-bit and constant-round MVBA protocol. We further present another MVBA extension protocol Dumbo-MVBA*, realizing a general self-bootstrap framework that can compile any MVBA protocol with quality property (here quality ensures a constant probability of deciding some honest party's input as MVBA output) into another communication-efficient MVBA implementation.

Finally, we demonstrate an enticing application of our MVBA protocols to construct expected constant-round asynchronous common subset (ACS) with only $\mathcal{O}(\ell n^2+\lambda n^2)$ communication complexity in expect, where ℓ is the input length of ACS represented in bits. The resulting ACS has asymptotically optimal communication cost when $\ell \geq \lambda$ and might have a broad array of applications like asynchronous atomic broadcasts or asynchronous multi-party computation.

1 Introduction

Byzantine agreement (BA) was proposed by Lamport, Pease and Shostak in their seminal papers [64,52] and considered the scenario that a few spacecraft controllers input some readings from a sensor and try to decide a common output, despite some of them are faulty [73]. The original specification of BA [64] allows input to be multi-valued, for example, the sensor's reading in the domain of $\{0,1\}^L$. This general case is also known as multi-valued BA [72], which generalizes the particular case of binary BA where input is restricted to either 0 or 1 [52,60].

* An abridged version [57] of the paper appeared at ACM PODC 2020. This full version adds a new instantiation of *asymptotically optimal* asynchronous common subset protocol (Sec. 7) using Dumbo-MVBA as key component. Part of the work was done while Yuan Lu, Zhenliang Lu and Qiang Tang were affiliated with New Jersey Institute of Technology.

Recently, the renewed attention to multi-valued BA is gathered in the *asynchronous* setting [4,59,33,43], due to an unprecedented demand of deploying asynchronous atomic broadcast (ABC) [24] that is usually instantiated by sequentially executing multi-valued asynchronous BA instances with some fine-tuned validity [29,21].

The seminal work of Cachin et al. in 2001 [21] proposes *external validity* for multi-valued BA and defines validated asynchronous BA, from which a simple construction of ABC can be achieved. In this multi-valued validated asynchronous BA (MVBA), each party takes a value as input and decides *one* of the proposed values as output, as long as the decided output satisfies the external validity condition. Later, MVBA was used as a core building block to implement a broad array of fault-tolerant protocols beyond ABC [50,67,20,74,31].

Cachin et al. also presented the first MVBA construction in [21]. Their construction is secure against computationally-bounded adversaries in the authenticated setting with the random oracle and setup assumptions (e.g., PKI and established threshold cryptosystems). The solution tolerates maximal Byzantine corruptions up to $f < n/3$ and attains expected $\mathcal{O}(1)$ running time and $\mathcal{O}(n^2)$ messages, but it incurs expected $\mathcal{O}(Ln^2 + \lambda n^2 + n^3)$ bits, which is large. Here, n is the number of parties, L represents the bit-length of MVBA input, and λ is the security parameter that captures the bit-length of digital signatures. As such, Cachin et al. raised the open problem of reducing the communication of MVBA protocols (and thus improve their ABC construction) [21], which can be rephrased as:

How to asymptotically improve the communication cost of MVBA *protocols by an* $\mathcal{O}(n)$ *factor?*

After nearly twenty years, in a recent breakthrough of Abraham et al. [4], the n^3 term in the communication complexity was removed, and they achieved optimal $\mathcal{O}(n^2)$ word communication, conditioned on each system word can encapsulate a constant number of input values and some small-size strings such as digital signatures. Their result can be directly translated to bit communication as a partial answer to the above question if the input length L is small (e.g., comparable to cryptographic security parameter λ).

Nevertheless, both of the above MVBA constructions contain an Ln^2 term in their communication complexities, which was reported in [21] as a major obstacle-ridden factor in a few typical use-cases where the input length L is not that small. For instance, Cachin et al. [21] noticed their ABC construction requires the underlying MVBA's input length L to be at least $\mathcal{O}(\lambda n)$, as each MVBA input is a set of $(n-f)$ values digitally signed by $(n-f)$ distinct parties. In this case, the Ln^2 term becomes the dominating factor as it is cubic in n. For this reason, it was even considered in [59,33] that existing MVBA protocols are sub-optimal for constructing ABC due to their large communication cost. It follows that, despite the recent breakthrough of [4], the long-standing question from [21] *remains open* for the moderately large input size $L \geq \mathcal{O}(\lambda n)$.

1.1 Our contributions

We answer the remaining part of the open question for large inputs and present the first MVBA protocols with expected $\mathcal{O}(Ln + \lambda n^2)$ communicated bits. More precisely, we showcase:

Theorem 1. *There exist protocols in the authenticated setting with setup assumptions and random oracle, such that it solves the* MVBA *problem [4,21] among n parties against an adaptive adversary controlling up to* $f \leq \lfloor \frac{n-1}{3} \rfloor$ *parties, with expected* $\mathcal{O}(Ln + \lambda n^2)$ *communicated bits and expected constant running time, where* L *is the input length and* λ *is a cryptographic security parameter.*

Our MVBA protocols not only improve the communication complexity of earlier results [21,4] as illustrated in Table 1, but also are *optimal* in the asynchronous setting regarding following performance metrics[1]:

[1] For communication cost, Abraham et al. [4] define a word to contain a constant number of signatures and input values (which implicitly assumes relatively small input value size that is comparable to cryptographic security parameter λ), such that word complexity is considered as their communication metric. Our MVBA protocols also achieve optimal $\mathcal{O}(n^2)$ word communication like [4], because (i) they attain $\mathcal{O}(n^2)$ message complexity and (ii) each message only contains $\mathcal{O}(1)$ words.

Table 1. Comparing the asymptotic performance of MVBA protocols among n parties with L-bit input and λ-bit security parameter.

MVBA Protocols	Comm. (Bits)	Word†	Round	Msg.
Cachin et al. [21]‡	$\mathcal{O}(Ln^2 + \lambda n^2 + n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$
Abraham et al. [4]	$\mathcal{O}(Ln^2 + \lambda n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$
Guo et al. [42]	$\mathcal{O}(Ln^2 + \lambda n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$
Duan et al. [34]	$\mathcal{O}(Ln^2 + \lambda n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(1)$	$\mathcal{O}(n^3)$
Our Dumbo-MVBA	$\mathcal{O}(Ln + \lambda n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$
Our Dumbo-MVBA⋆	$\mathcal{O}(Ln + \lambda n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$

† [4] defines a word to be $\mathcal{O}(\lambda)$ bits, so each word can contain a constant number of elements from a cryptographically-secure group/field. This implies every word can carry a digital signature or hash digest. Furthermore, [4] implicitly assumes that a word can encapsulate at least an MVBA input, so the counting of word complexity is considered in a setting where MVBA input length $L \leq \mathcal{O}(\lambda)$.
‡ [21] realizes that their construction can be generalized against adaptive adversary, when given threshold cryptosystems with adaptive security.

1. Their executions incur $\mathcal{O}(Ln + \lambda n^2)$ bits on average, which coincides with the *optimal communication* $\mathcal{O}(Ln)$ when $L \geq \mathcal{O}(\lambda n)$. This optimality can be seen trivially, since each honest party has to receive the L-bit output, indicating a minimum communication of $\Omega(Ln)$ bits.
2. Same to [4], they can tolerate *adaptive adversaries* controlling up to $\lfloor \frac{n-1}{3} \rfloor$ Byzantine parties, which achieves the *optimal resilience* in the asynchronous network according to the upper bound of resilience shown in literature [71,19].
3. As same as [21,4], they can terminate in expected constant asynchronous rounds with overwhelming probability, which is essentially *asymptotically optimal* for asynchronous BA [36,12].
4. Like [21,4], they attain *asymptotically optimal* $\mathcal{O}(n^2)$ messages, which meets the lower bound of the messages of optimally-resilient asynchronous BA against strong adaptive adversary with "after-the-fact-removal" [4,1].

Table 2. Comparing the asymptotic performance of ACS protocols among n parties with ℓ-bit input and λ-bit security parameter.

ACS Protocols ‡	Round	Comm.(Bits)	Msg.	Word†
HoneyBadger-ACS [59] + [30,5]	$\mathcal{O}(\log n)$	$\mathcal{O}(\ell n^2 + \lambda n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$
PACE-ACS [75] + [30,5]	$\mathcal{O}(\log n)$	$\mathcal{O}(\ell n^2 + \lambda n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$
Dumbo-ACS [43] + [30,5]	$\mathcal{O}(1)$	$\mathcal{O}(\ell n^2 + \lambda n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$
FIN-ACS [34] + [30,5]	$\mathcal{O}(1)$	$\mathcal{O}(\ell n^2 + \lambda n^3)$	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$
CKPS-ACS [21] + CKPS-MVBA [21]	$\mathcal{O}(1)$	$\mathcal{O}(\ell n^3 + \lambda n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
CKPS-ACS [21] + ASM-MVBA [4]	$\mathcal{O}(1)$	$\mathcal{O}(\ell n^3 + \lambda n^3)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^3)$
Our ACS instantiation ([21]+Dumbo-MVBA)	$\mathcal{O}(1)$	$\mathcal{O}(\ell n^2 + \lambda n^2)$	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$

† Similar to [4], we let a word to be $\mathcal{O}(\lambda)$ bits, and while counting the word complexity, we assume the bit length of ACS input $\ell \leq \mathcal{O}(\lambda)$.
‡ For fair comparison, we assume that HoneyBadgerBFT, PACE, Dumbo and FIN instantiate their reliable broadcast building blocks by the state-of-the-art results [30,5].

Furthermore, we can adopt our MVBA protocols to immediately instantiate the first asynchronous common subset (ACS) protocol with expected constant round and $\mathcal{O}(\ell n^2 + \lambda n^2)$ communication, as illustrated in Table 2. Here ℓ represents the bit-length of each ACS input, and ACS is special variant of asynchronous Byzantine agreement, the validity of which allows every party to

take a value as input and decide an output that is a common subset of all parties' inputs. Moreover, the output common subset shall cover a sufficient number of inputs proposed by distinct parties (usually $n - f$ different parties). More precisely, we demonstrate:

Corollary 1. *There exist a protocol in the authenticated setting with setup assumptions and random oracle, such that it realizes* ACS *[14,59] among n parties against an adaptive adversary controlling up to $f \leq \lfloor \frac{n-1}{3} \rfloor$ parties, with expected $\mathcal{O}(\ell n^2 + \lambda n^2)$ communicated bits and expected constant running time, where ℓ is the input length and λ is a cryptographic security parameter.*

Clearly, our ACS instantiation meets the $\lfloor \frac{n-1}{3} \rfloor$ resilience upper bound of BA protocols in asynchrony [19]. Its $\mathcal{O}(n^2)$ message complexity is asymptotically optimal under strong adaptive adversaries [1], and its expected $\mathcal{O}(1)$ round complexity is asymptotically optimal regarding asynchronous BA problem. More importantly, our ACS instantiation is also communication-optimal for input size $\ell \geq \mathcal{O}(\lambda)$: it costs $\mathcal{O}(\lambda n^2 + \lambda n^2)$ bits on average, matching the *lower bound* $\Omega(\ell n^2)$ of communication complexity when $\ell \geq \mathcal{O}(\lambda)$. The optimality of communication can be seen from the validity definition of asynchronous common subset, where each honest party has to receive $(n - f)$ distinct parties' inputs to form its output set, indicating a minimal communication cost of $\Omega(\ell n(n - f)) = \Omega(\ell n^2)$.

Noticeably, ACS is a critical building block widely used in many asynchronous Byzantine fault tolerant (BFT) systems, inferring that our MVBA and resulting ACS protocols might be beneficial to a broader array of applications such as asynchronous atomic broadcast (i.e., distributed ledger consensus) [21,43,39,59,42], asynchronous multi-party computation [14,56,27], and asynchronous proactive threshold cryptosystem [20,74].

1.2 Challenges and techniques

Let us begin with a very brief tour to revisit a couple of existing MVBA constructions [21,4]. In CKPS-MVBA [21], each party $\mathcal{P}_i$ firstly broadcasts its input value v_i to all others using a broadcast protocol (e.g., Reiter's consistent broadcast protocol [69] using quorum certificate made from threshold signature to attest that $n - 2f$ honest parties have received the common value). Once sufficient values are received from $n - f$ different parties' broadcasts, each party informs everyone else which $n - f$ values it has received, which can be realized by broadcasting a vector of n encoded bits (where each j-th bit represents whether the corresponding j-th broadcast is received or not). Thus, every party $\mathcal{P}_i$ can locally form an $\mathcal{O}(n^2)$ size "matrix", in which the j-th entry tracks that $\mathcal{P}_j$ has received which $n - f$ broadcasts. Then, after the $\mathcal{O}(n^2)$ size "matrix" has at least $n - f$ entries, the parties enter a random leader election protocol, uniformly selecting a candidate party $\mathcal{P}_l$, such that an asynchronous binary agreement (ABA) is further run to vote on whether to decide the input of selected candidate party v_l as output, depending on if enough parties have already received v_l. ABA protocols for random selected candidates will be repeated until 1 is returned. Another recent study AMS-MVBA [4], instead, expands the conventional design idea of ABA and directly constructs MVBA without black-box invocations to ABA in the following way: first, multiple rounds of broadcasts are executed by every party to form quorum certificates called *commit* proofs on their input values. Then, a random party $\mathcal{P}_l$ is elected. If any party already receives a *commit* proof for v_l, it decides to output v_l; and other undecided parties use v_l as input to enter a repetition of the whole procedure. We can see that [4] can get rid of the $\mathcal{O}(n^3)$ communication as the phase where each party receives the $\mathcal{O}(n^2)$ size matrix is removed.

Note that in the first phase of both CKPS-MVBA nad AMS-MVBA protocols [21,4], every party broadcasts its own input to all parties, which already results in Ln^2 communicated bits and corresponds to the major efficiency challenge to overcome in this paper.

Using the dispersal-then-recast approach to overcome redundant input broadcasts. Our key observation is that an MVBA protocol only outputs a single party's input, it is thus unnecessary for *every* party to send its input to all parties. Following the observation, we design Dumbo-MVBA, a novel reduction from MVBA to ABA by using a *dispersal-then-recast* methodology to reduce communication. Instead of letting each party directly send its input to everyone, we let everyone to disperse the coded *fragments* of its input across the network. Later, after the dispersal

phase has been completed, the parties could (randomly) choose a dispersed value to collectively recover it. Thanks to the external predicate, all parties can locally check the validity of the recovered value, such that they can consistently decide to output the value, or to repeat random election of another dispersed value to recover.

Remaining efficiency challenge of implementing dispersal-then-recast methodology. However, challenges remain due to our multiple efficiency requirements. For example, the number of messages to disperse a value shall be at most linear, otherwise n dispersals would cost more than quadratic messages and make MVBA not optimal anymore. The requirement rules out a few related candidates such as asynchronous verifiable information dispersal (AVID) [23,44] that needs $\mathcal{O}(n^2)$ messages to disperse a value. In addition, the protocol must terminate in expected constant time, that means at most a constant number of dispersed values will be recovered on average.

More efficient MVBA protocol from $\mathcal{O}(n)$-message asynchronous provable dispersal. We set forth a new notion of asynchronous provable dispersal broadcast (APDB) to overcome the efficiency issue of existing AVID protocols. The primitive weakens the agreement of AVID when the sender is corrupted, but it can be implemented more efficiently with only $\mathcal{O}(n)$ messages. To complement the somewhat weaker agreement property of APDB, we also introduce two succinct "proofs" in APDB as hinted by the nice work of Abraham et al. [4]. During the dispersal of APDB, two proofs *lock* and *done* could be produced: (i) when any honest party delivers a *lock* proof, enough parties have delivered the coded fragments of the dispersed value, and thus the value can be collectively recovered by all honest parties, and (ii) the *done* proof attests that enough parties deliver *lock*, so all honest parties can activate ABA with input 1 and then decide 1 to jointly recover the dispersed value. To take the most advantage of APDB, we leverage the design in [4] to let the parties exchange their *done* proofs to collectively quit all dispersals, and then borrow the idea in [21] to randomly elect a party and vote via ABA to decide whether to output the elected party's input value (if the value turns to be valid after being recovered). Intuitively, this idea reduces the communication cost, because (i) each fragment has only $\mathcal{O}(L/n)$ bits, so n dispersals of L-bit input incur only $\mathcal{O}(Ln)$ bits, (ii) the parties can reconstruct a valid value after expected constant number of ABA and recoveries. See detailed discussions in Section 4.

Another generic MVBA extension protocol catering for large input. Finally, we present another MVBA extension protocol Dumbo-MVBA$\star$, which is a general self-bootstrap framework to reduce the communication of any existing MVBA for sufficiently large input. After applying our APDB protocol, we can use small input (i.e., the "proofs" of APDB) to invoke the underlying MVBA to pick the dispersed value to recast, thus avoiding communication blow-up due to the broadcast of large input. Moreover, Dumbo-MVBA$\star$ is centered around the advanced building block of MVBA instead of the basic module of binary agreement, allowing it can better utilize MVBA to remove the rounds generating the *done* proof in APDB, which further results in a much simpler modular design.

1.3 Additional related work

Besides the closely related studies that have been discussed, we thoroughly review a few pertinent topics that are also related to this paper in order to help unfamiliar readers better understand the context.

Validity conditions. The asynchronous BA problem [19,14,25] was studied in diverse flavors, depending on validity conditions.

Strong validity [37,62] requires that if an honest party outputs v, then v is input of some honest party. This is arguably the strongest notion of validity for multi-valued BA. The sequential execution of BA instances with strong validity gives us an ABC protocol, even in the asynchronous setting. Unfortunately, implementing strong validity is not easy. In [37], the authors even proved some disappointing bounds of strong validity in the asynchronous setting, which include: (i) the maximal number of corruptions is up to $f < n/(2^\ell + 1)$, and (ii) the optimal running time is $\mathcal{O}(2^\ell)$ asynchronous rounds, where ℓ is the input size in bit.

Weak validity [51,32], only requires that if all honest parties input v, then every honest party outputs v. This is one of most widely adopted validity notions for multi-valued BA. However, it

states nothing about output when the honest parties have different inputs. Weak validity is strictly weaker than strong validity [37,62], except that they coincide in binary BA [22,55,60]. Abraham et al. [4] argued: it is not clear how to achieve a simple reduction from ABC to asynchronous multi-valued BA with weak validity; in particular, the sequential execution of multi-valued BA instances with weak validity fails in the asynchronous setting, because non-default output is needed for the liveness [21] or censorship resilience [59].

External validity was proposed by Cachin et al. [21] to circumvent the limits of above validity notions, and it requires the decided output of honest parties to satisfy a globally known predicate. This delicately tuned notion brings a few definitional advantages: (i) compared to strong validity, it is easier to be instantiated, (ii) in contrast with weak validity, ABC is simply reducible to it. For example, Cachin et al. [21] showcased a simple reduction from ABC to MVBA (with using a notion called asynchronous common subset, i.e., ACS, as a bridge). This succinct construction sequentially executes the ACS instances, each of which allows every party to propose an input value and then solicits $n-f$ input values (from distinct parties) to output. The work also instantiates ACS due to a reduction to MVBA by centering around a specific external validity condition, namely, input/output must be a set containing $2f+1$ valid message-signature pairs generated by distinct parties, where each signed message is an ABC input. Although their reduction is simple, the communication cost (per delivered bit) in their ABC was cubic (and is still amortizedly quadratic even if using the recent technique of batching in [59]), mainly because the communication cost of the underlying MVBA module contains a quadratic term factored by the MVBA's input length.

Cryptography vs. information theory. We focus on the cryptographic setting with trusted setup like many earlier studies [65,71,21,22,60,59,4] in order to realize more efficient constructions of asynchronous BA protocols. Nevertheless, there are a few nice theoretic studies emphasizing on designing information-theoretically (IT) secure asynchronous BA protocols in the more stringent information-theoretic setting without trusted setup assumption.

In the most challenging IT setting without even secure private channel, Ben-Or [12] and Bracha [19] give ABA constructions with exponential round complexity. In the same setting, Kapron et al. [46] give an ABA construction that attains $\tilde{\mathcal{O}}(n^2)$ bits and sub-exponential rounds against static full information adversaries, King and Saia [48] present a full information-theoretic ABA protocol with polynomial rounds against adaptive corruption, but their design can only tolerate $f < n/500$ Byzantine faults, and recently, Huang et al. [45] improve earlier results by giving an information-theoretic polynomial-round ABA protocol with $n/4$ resilience against adaptive full information adversaries. Nevertheless, there is a fundamental gap between the full information model (without setup assumption or private channels) and our computationally-bounded cryptographic model with setup assumptions, as pointed out in [10,6,7]. In particular, it is impossible to realize expected $o(n)$-round randomized asynchronous BA protocols against an adaptive full information adversary [6,7]. Like a few earlier studies [65,71,21,22,60,59,4], we use pre-shared common coins (implemented via unique threshold signature in the random oracle model) to overcome this lower bound and obtain the asymptotically optimal $\mathcal{O}(1)$ expected rounds.

Setup-free IT ABAs are also studied with an additional assumption of secure private channel [2,9,35]. In the setting, [35] can realize an ABA protocol with expected $\mathcal{O}(1)$ rounds but only tolerates $f < n/4$ corruption, [2] later improves the resilience to optimal $n/3$ but has an expected round complexity of $\mathcal{O}(n^2)$, and Bangalore et al. [9] present an optimal-resilient ABA that can terminate in expected $\mathcal{O}(n)$ rounds. However, none of them can achieve complexities matching the state-of-the-art results in the cryptographic setting.

Strong vs. Weak adaptive adversaries. We consider a strong form of adaptive adversary with "after-the-fact-removal": the adversary can adaptively corrupt some node and then remove this node's undelivered messages. It is known that BA protocols need at least $\Omega(f^2)$ communication cost in the presence of such strong adaptive adversary [1], indicating $\Omega(n^2)$ communication lower bound when $f < n/3$. Blum et al. [16] and Cohen et al. [28] recently consider a less stringent setting of weak adaptive adversaries that cannot perform "after-the-fact-removal" attacks, and they present a couple of near-optimal resilient asynchronous BA protocols (achieving weak validity) with $o(n^2)$ communication against such weak adaptive adversaries, respectively. Noticeably, those

$o(n^2)$-communication results do not violate the known communication lower bound $\Omega(n^2)$ under optimal resilience against strong adaptive adversaries.

Extension protocols of BAs. In the asynchronous setting, there exist a few nice extension protocols that can invoke Byzantine agreement with using short input to accommodate large multi-valued input [61,63,38]. To the best of our knowledge, all the existing extension protocols focus on weak validity, and their techniques cannot be directly borrowed to handle external validity. The challenge in our extension MVBA protocol Dumbo-MVBA$\star$ stems from that the externally valid output can come from one corrupt party. More specifically, in the extension protocols for weak validity, it has to decides an output only when all honest parties share the same input value, which can be ensured through a simple dispersal-recast technique [61] since sufficient honest parties already share the same input; while in our case of MVBA for external validity, when the decided output is from a malicious party, there are no sufficient honest parties share the same value as input to assist the recovery, so we have to ensure all input values are indeed correctly dispersed and become recoverable (which is realized via the recastability in our APDB design) by enforcing each input dispersal to attach an extra short proof attesting its successful dissemination.

Efficiency problems of existing ACS protocols. There are mainly two paradigms to construct ACS protocols: one is initiated by Cachin et al. [21] to reduce ACS to MVBA. The other method, initiated by Ben-Or et al. [14] and recently improved by HoneyBadgerBFT (HBBFT) [59] and PACE [75], builds ACS using n asynchronous binary agreements (ABAs) directly. During the past years, the former approach (i.e., building ACS from MVBA) was considered as sub-optimal to instantiate ABC in literature [59,33], because of the large communication complexity of existing MVBA protocols. In a very recent work, Dumbo BFT [43], proposes a novel reduction from ACS to MVBA, which results in an ACS protocol that attains only constant expected round; in contrast, [59,33] have a round complexity of $\mathcal{O}(\log n)$ with having the same communication and message complexities. Moreover, the improvement of Dumbo BFT is achieved despite invoking existing MVBA protocols (e.g., the first MVBA proposed by Cachin et al. two decades ago) with seemingly large communication complexity.

Nevertheless, all existing ACS protocols still suffer from cubic communication complexity despite the recent progress. In this paper, we directly reduce the communication complexity of MVBA protocols for an $\mathcal{O}(n)$ factor. This allows us more efficiently instantiate the ACS construction due to Cachin et al. [21], resulting in the first adaptively secure ACS protocol with only quadratic communication overhead. The idea lets each party sign its ACS input and multicast the signed value, then every party waits for receiving $n - f$ signed values from different parties, and uses the vector of $n - f$ signed values to invoke a communication-optimal MVBA protocol, and finally the honest parties can decide the $n - f$ values carried by MVBA output as their ACS output. For sake of completeness, Section 7 will present how to use our communication-optimal MVBA protocols to efficiently instantiate ACS with expected constant round and $\mathcal{O}(\ell n^2 + \lambda n^2)$ communication (i.e., optimal communication if ACS input length $\ell \geq \lambda$).

2 Problem Formulation

2.1 System model: asynchronous authenticated setting

We use the standard authenticated asynchronous message-passing model [4,21] to capture the system consisting of n parties and a computationally-bounded adversary with setup assumptions.

Established identities and trusted setup. There are n designated parties denoted by $\{\mathcal{P}_i\}_{i\in[n]}$, where $[n]$ is short for $\{1, \ldots, n\}$ through the paper. Moreover, we consider trusted setup for threshold signature and common reference string for all necessary cryptographic primitives. Taking threshold signature as example, each party shall have obtained its own secret key share and all public keys before the start of protocol execution. For presentation simplicity, we consider the trusted setup for granted, while in practice it can be done via a trusted dealer or distributed key generation protocols [53,17,47].

Adaptive Byzantine corruption. The adversary $\mathcal{A}$ can adaptively corrupt any party at any time during the course of protocol execution, until $\mathcal{A}$ already controls f parties (e.g., $3f + 1 = n$).

If a party $\mathcal{P}_i$ was not corrupted by $\mathcal{A}$ at some stage of the protocol, it followed the protocol and kept all internal states secret against $\mathcal{A}$, and we say it is *so-far-uncorrupted*. Once a party $\mathcal{P}_i$ is corrupted by $\mathcal{A}$, it leaks all internal states to $\mathcal{A}$ and remains fully controlled by $\mathcal{A}$ to arbitrarily misbehave. By convention, the corrupted party is also called *Byzantine fault*. If and only if a party is not corrupted through the entire execution, we say it is *honest*.

Computing model. Following standard cryptographic practices [21,22], we let the n parties and the adversary $\mathcal{A}$ to be probabilistic polynomial-time interactive Turing machines (ITMs). A party $\mathcal{P}_i$ is an ITM defined by the protocol: it is activated upon receiving an incoming message to carry out some computations, update its states, possibly generate some outgoing messages, and wait for the next activation. $\mathcal{A}$ is a probabilistic ITM bounded by polynomial time (in the number of message bits generated by honest parties). Moreover, we explicitly require the message bits generated by honest parties to be probabilistic uniformly bounded by a polynomial in the security parameter λ, which was formulated as *efficiency* in [21,4] to rule out infinite protocol executions and thus restrict the run time of the adversary through the entire protocol. Same to [21] and [4], all system parameters (e.g., n) are bounded by polynomials in λ.

Fully-meshed reliable asynchronous point-to-point network. Any two parties in the system are connected via an asynchronous *reliable authenticated* point-to-point channel. When a party $\mathcal{P}_i$ attempts to send a message to another party $\mathcal{P}_j$, the adversary $\mathcal{A}$ is firstly notified about the message; then, $\mathcal{A}$ fully determines when $\mathcal{P}_j$ receives the message, but cannot drop or modify this message if both $\mathcal{P}_i$ and $\mathcal{P}_j$ are honest. The network model also allows the adaptive adversary $\mathcal{A}$ to perform "after-the-fact removal", that is, when $\mathcal{A}$ is notified about some messages sent from a *so-far-uncorrupted* party $\mathcal{P}_i$, it can delay these messages until it corrupts $\mathcal{P}_i$ to drop them.

2.2 Design goal: multi-valued validated asynchronous BA

We review hereunder the definition of (multi-valued) validated asynchronous Byzantine agreement (MVBA) due to [21,4].

Definition 1. *In an* MVBA *protocol with an external* Predicate : $\{0,1\}^\ell \to \{true, false\}$*, the parties take values satisfying* Predicate *as inputs and aim to output a common value satisfying* Predicate*. The* MVBA *protocol guarantees the following properties, except with negligible probability, for any identification* id*, in the asynchronous authenticated model:*

- ***Termination.*** *If every honest party $\mathcal{P}_i$ is activated on identification* id*, with taking as input a value v_i s.t.* Predicate$(v_i) = true$*, then every honest party outputs a value v for* id*.*
- ***External-Validity.*** *If an honest party outputs a value v for* id*, then* Predicate$(v) = true$*.*
- ***Agreement.*** *If any two honest parties output v and v' for* id *respectively, then $v = v'$.*
- ***Quality.*** *If an honest party outputs v for* id*, the probability that v was proposed by the adversary is at most 1/2.*

We make the following remarks about the above definition:

1. *Input length.* We focus on the general case that the input length ℓ can be a function in n. We emphasize that it captures many realistic scenarios. One remarkable example is to build ABC around MVBA as in [21] where the length of each MVBA input is at least $\mathcal{O}(\lambda n)$.
2. *External-validity* is a fine-grained validity requirement of BA. In particular, it requires the common output of the honest parties to satisfy a pre-specified global predicate function.
3. *Quality* was proposed by Abraham et al. in [4], which not only rules out trivial solutions w.r.t. some trivial predicates (e.g., output a known valid value) but also captures "fairness" to prevent the adversary from fully controlling the output.

2.3 Quantitative performance metrics

We consider the following standard quantitative metrics to characterize the performance aspects of asynchronous Byzantine fault tolerant protocols:

- **Resilience**. An MVBA protocol is said f-resilient, if it can tolerate an (adaptive) adversary that corrupts up to f parties. If $3f+1=n$, the MVBA protocol is said to be optimally-resilient [19]. Through the paper, we focus on the optimally-resilient MVBA against adaptive adversary.
- **Message complexity**. The message complexity measures the expected number of overall messages generated by honest parties during the protocol execution. For MVBA protocols with optimal resilience against adaptive adversary, the lower bound of message complexity is expected $\Omega(n^2)$ [4,1].
- **Communication complexity**. We consider the standard notion of communication complexity to characterize the expected number of bits sent among the honest parties during the protocol. For the optimally-resilient MVBA against adaptive adversary, the lower bound of communication complexity is $\Omega(\ell n + n^2)$, where the ℓn term represents a trivial lower bound that all honest parties have to deliver an externally valid value of ℓ bits in length [59,4,1], and the n^2 terms is a reflection of the lower bound of message complexity.
- **Round complexity**. We follow the standard approach due to Canetti and Rabin [25] to measure the running time of protocols by asynchronous rounds. Essentially, this measurement counts the number of messaging "rounds", when the protocol is embedded into a lock-step timing model. For asynchronous BA, the expected $\mathcal{O}(1)$ round complexity is optimal [36,12].

The above communication, message and round complexities shall be *probabilistically uniformly bounded* (see Section 3.2) independent to the adversary [21]. The complexity notions bring about the advantage that is closed under modular composition. We thus can design and analyze protocols in a modular way.

3 Preliminaries and Notations

3.1 Preliminary primitives

Erasure code scheme. A (k, n)-erasure code scheme [15] consists of a tuple of two deterministic algorithms Enc and Dec. The Enc algorithm maps any vector $\mathbf{v} = (v_1, \cdots, v_k)$ of k data fragments (called message word) into an vector $\mathbf{m} = (m_1, \cdots, m_n)$ of n coded fragments (called code word), such that any k elements in the code word $\mathbf{m}$ is enough to reconstruct the message word $\mathbf{v}$ due to the Dec algorithm. More formally, a (k, n)-erasure code scheme has a tuple of two deterministic algorithms:

1. $\mathsf{Enc}(\mathbf{v}) \rightarrow \mathbf{m}$. On input a vector $\mathbf{v} \in \mathcal{B}^k$ (message word), this *deterministic* encode algorithm outputs a vector $\mathbf{m} \in \mathcal{B}^n$ (code word). Note that $\mathbf{v}$ contains k data fragments and $\mathbf{m}$ contains n coded fragments, and $\mathcal{B}$ denotes the field of each fragment.
2. $\mathsf{Dec}(\{(i, m_i)\}_{i\in S}) \rightarrow \mathbf{v}$. On input a set $\{(i, m_i)\}_{i\in S}$ where $m_i \in \mathcal{B}$, and $S \subset [n]$ and $|S| = k$, this *deterministic* decode algorithm outputs the message word $\mathbf{v} \in \mathcal{B}^k$.

We require (k, n)-erasure code scheme is *maximum distance separable*, namely, the original data fragments $\mathbf{v}$ can be recovered from any k-size subset of the coded fragments $\mathbf{m}$, which can be formally defined as:

- **Correctness of erasure code**. For any $\mathbf{v} \in \mathcal{B}^k$ and any $S \subset [n]$ that $|S| = k$, $\Pr[\mathsf{Dec}(\{(i, m_i)\}_{i\in S}) = \mathbf{v} \mid \mathbf{m} := (m_1, \cdots, m_n) \leftarrow \mathsf{Enc}(\mathbf{v})] = 1$. If a vector $\mathbf{m} \in \mathcal{B}^n$ is indeed the code word of some message word $\mathbf{v} \in \mathcal{B}^k$, we say the $\mathbf{m}$ is well-formed; otherwise, we say the $\mathbf{m}$ is ill-formed.

Instantiation. Through the paper, we consider a $(f + 1, n)$-erasure code scheme where $3f + 1 = n$. Besides, we emphasize the erasure code scheme would implicitly choose a proper field $\mathcal{B}$ according to the actual length of each element in $\mathbf{v}$, such that the encoding causes only constant blow-up in size, namely, the bits of $\mathbf{m}$ are larger than the bits of $\mathbf{v}$ by at most a constant factor. There are a few well-known instantiations of such primitive like Rabin's [66], Reed-Solomon [68] and numerous their variants.

Position-binding vector commitment (VC). For an established position-binding n-vector commitment (VC), there is a tuple of algorithms (VCom, Open, VerifyOpen). On input a vector $\mathbf{m}$ of

any n elements, the algorithm VCom produces a commitment vc for the vector $\mathbf{m}$. On input $\mathbf{m}$ and vc, the Open algorithm can reveal the element m_i committed in vc at the i-th position while producing a short proof π_i, which later can be verified by $\mathsf{VerifyOpen}$.

Formally, a position-binding VC scheme (without hiding) is abstracted as:

1. $\mathsf{VC.Setup}(\lambda, n, \mathcal{M}) \to pp$. Given security parameter λ, the size n of the input vector, and the message space $\mathcal{M}$ of each vector element, it outputs public parameters pp, which are implicit inputs to all the following algorithms. We explicitly require $\mathcal{M} = \{0,1\}^*$, such that one VC scheme can commit any n-sized vectors.
2. $\mathsf{VCom}(\mathbf{m}) \to (vc; aux)$. On input a vector $\mathbf{m} = (m_1, ..., m_n)$, it outputs a commitment string vc and an auxiliary advice string aux. We might omit aux for presentation simplicity. Note we do not require the hiding property, and then let VCom to be a deterministic algorithm.
3. $\mathsf{Open}(vc, m_i, i; aux) \to \pi_i$. On input $m_i \in \mathcal{M}$, $i \in [n]$, the commitment vc and advice aux, it produces an opening string π to prove that m_i is the i-th committed element. We might omit aux for presentation simplicity.
4. $\mathsf{VerifyOpen}(vc, m_i, i, \pi_i) \to 0/1$. On input $m_i \in \mathcal{M}$ and $i \in [n]$, the commitment vc, and an opening proof π, the algorithm outputs 0 (accept) or 1 (reject).

An already established VC scheme shall satisfy **correctness** and **position binding**:

- **Correctness of VC**. An established VC scheme with public parameter pp is correct, if for all $\mathbf{m} \in \mathcal{M}^n$ and $i \in [n]$, $\Pr[\mathsf{VC.VerifyOpen}(vc, m_i, i, \mathsf{VC.Open}(vc, m_i, i, aux)) = 1 \mid (vc, aux) \leftarrow \mathsf{VC.VCom}(\mathbf{m})] = 1$.
- **Position binding**. An established VC scheme with public parameter pp is said position binding, if for any P.P.T. adversary $\mathcal{A}$, $\Pr[\mathsf{VC.VerifyOpen}(vc, m, i, \pi) = \mathsf{VC.VerifyOpen}(vc, m', i, \pi') = 1 \wedge m \neq m' \mid (vc, i, m, m', \pi, \pi') \leftarrow \mathcal{A}(pp)] < negl(\lambda)$, where $negl(\lambda)$ is a negligible function in λ.

Instantiation. There are a few simple solutions [58,26,54] to achieve the above VC notion. An example is hash Merkle tree [58] where the commitment vc is $\mathcal{O}(\lambda)$-bit and the openness π is $\mathcal{O}(\lambda \log n)$-bit. Moreover, when given computational Diffie-Hellman assumption and collision-resistant hash function, there is a position-binding vector commitment scheme [26], s.t., all algorithms (except setup) are deterministic, and both commitment and openness are $\mathcal{O}(\lambda)$ bits.

Relying on computational Diffie-Hellman (CDH) assumption and collision-resistant hash function, there is a construction due to Catalano et al. [26] that realizes the position-binding vector commitment as follows:

- $\mathsf{VC.Setup}(\lambda, n) \to pp$. Let $\mathcal{H}$ be a collision-resistant hash function $\mathcal{H} : \{0,1\}^* \to \mathbb{Z}_p$. Given λ, generate $\mathbb{G}$ and $\mathbb{G}_T$ that are two bilinear groups of prime order p with a bilinear map $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$; also choose g that is a random generator of $\mathbb{G}$. Randomly choose $(z_1, \cdots, z_n)$ from $\mathbb{Z}_p$, and for each $i \in [n]$, compute $h_i \leftarrow g^{z_i}$. For each $i, j \in [n]$ and $i \neq j$, compute $h_{i,j} = g^{z_i z_j}$. Set $pp = (g, \{h_i\}_{i\in[n]}, \{h_{i,j}\}_{i,j\in[n], i\neq j})$.
- $\mathsf{VCom}(\mathbf{m}) \to (vc; aux)$. Compute $vc = \prod_{i=1}^{n} h_i^{\mathcal{H}(m_i)}$ and $aux = (\mathcal{H}(m_1), \cdots, \mathcal{H}(m_n))$. The output aux might be omitted in the paper for simplicity.
- $\mathsf{Open}(vc, m_i, i; aux) \to \pi_i$. Compute $\pi_i = \prod_{j=1, j\neq i}^{n} h_{i,j}^{\mathcal{H}(m_i)} = (\prod_{j=1, j\neq i}^{n} h_j^{\mathcal{H}(m_i)})^{z_i}$. We might omit the input aux for presentation simplicity.
- $\mathsf{VerifyOpen}(vc, m_i, i, \pi_i) \to 0/1$. It checks whether $e(vc/h_i^{\mathcal{H}(m_i)}, h_i) = e(\pi_i, g)$ or not.

Regarding the *size* of the commitment and the openness proof, it is clear that they contain only a single group element in $\mathbb{G}$, which corresponds to only $\mathcal{O}(\lambda)$ bits and is independent to the length of each committed element. In addition, all algorithms run in polynomial time, and they (except the setup) are *deterministic*.

Non-interactive threshold signature (TSIG). Given an established (t, n)-threshold signature, each party $\mathcal{P}_i$ has a private function denoted by $\mathsf{SignShare}_{(t)}(sk_i, \cdot)$ to produce its "partial" signature, and there are also three public functions $\mathsf{VerifyShare}_{(t)}$, $\mathsf{Combine}_{(t)}$ and $\mathsf{VerifyThld}_{(t)}$, which can respectively validate the "partial" signature, combine "partial" signatures into a "full" signature, and validate the "full" signature. Note the subscript (t) denotes the threshold t through the

paper. Formally, a non-interactive (t, n)-threshold signature scheme TSIG is a tuple of hereunder algorithms/protocols among n parties $\{\mathcal{P}_i\}_{i\in[n]}$:

1. $\mathsf{TSIG.Setup}(\lambda, t, n) \rightarrow (mpk, \mathbf{pk}, \mathbf{sk})$. Given t, n and security parameter λ, generate a special public key mpk, a vector of public keys $\mathbf{pk} = (pk_1, \cdots, pk_n)$, and a vector of secret keys $\mathbf{sk} = (sk_1, \cdots, sk_n)$ where $\mathcal{P}_i$ gets sk_i only.
2. $\mathsf{SignShare}_{(t)}(sk_i, m) \rightarrow \sigma_i$. On input a message m and a secret key share sk_i, this deterministic algorithm outputs a "partial" signature share σ_i. Note that the subscript (t) denotes the threshold t.
3. $\mathsf{VerifyShare}_{(t)}(m, (i, \rho_i)) \rightarrow 0/1$. Given a message m, a "partial" signature ρ_i and an index i (along with implicit input mpk and $\mathbf{pk}$), this deterministic algorithm outputs 1 (accept) or 0 (reject).
4. $\mathsf{Combine}_{(t)}(m, \{(i, \rho_i)\}_{i\in S}) \rightarrow \sigma/\perp$. Given a message m and t indexed partial-signatures $\{(i, \rho_i)\}_{i\in S}$ (along with implicit input mpk and $\mathbf{pk}$), this algorithm outputs a "full" signature σ for message m (or $\perp$).
5. $\mathsf{VerifyThld}_{(t)}(m, \sigma) \rightarrow 0/1$. Given a message m and an "aggregated" full signature σ (along with the implicit input mpk), this algorithms outputs 1 (accept) or 0 (reject).

We require an established TSIG scheme to satisfy **correctness**, **robustness** and **unforgeability**:

- **Correctness of TSIG**. The correctness property requires that: (i) for $\forall$ m and $i \in [n]$, $\Pr[\mathsf{VerifyShare}_{(t)}(m, (i, \rho_i)) = 1 \mid \rho_i \leftarrow \mathsf{SignShare}_{(t)}(sk_i, m)] = 1$; (ii) for $\forall$ m and $S \subset [n]$ that $|S| = t$, $\Pr[\mathsf{VerifyThld}_{(t)}(m, \sigma) = 1 \mid \forall i \in S, \rho_i \leftarrow \mathsf{SignShare}_{(t)}(sk_i, m) \wedge \sigma \leftarrow \mathsf{Combine}_{(t)}(m, \{(i, \rho_i)\}_{i\in S})] = 1$.
- **Robustness**. No P.P.T. adversary can produce t valid "partial" signature shares, s.t. running $\mathsf{Combine}$ over these "partial" signatures does *not* produce a valid "full" signature, except with negligible probability in λ. Intuitively, robustness ensures any t valid "partial" signatures for a message must induce a valid "full" signature [70].
- **Unforgeability** (against adaptive adversary). The unforgeability can be defined by a threshold and adaptive version Existential UnForgeability under Chosen Message Attack game [53]. Intuitively, the unforgeability ensures that no P.P.T. adversary $\mathcal{A}$ that adaptively corrupts f parties ($f < t$) can produce a valid "full" signature except with negligible probability in λ, unless $\mathcal{A}$ receives some "partial" signatures produced by $t - f$ parties that are honest.

Instantiation. The above adaptively secure non-interactive threshold signature can be realize due to the construction in [53] assuming the hardness of symmetric external Diffie-Hellman (SXDH) in the random oracle model. Recently, Bacho and Loss [8] demonstrates that the threshold variant [17] of Boneh-Lynn-Shacham (BLS) signature [18] can also be adaptively secure under the assumption of algebraic group model (AGM) and random oracle (RO). In all above instantiations, each "partial" signature ρ and every "full" signature σ have a length of $\mathcal{O}(\lambda)$ bits. Noticeably, the mentioned non-interactive threshold signature schemes [53,17] are also unique, namely, given any message m and the corresponding partial signatures of all honest parties, it is still computationally infeasible for adversaries to generate two distinct "full" signatures that are both valid. The uniqueness property of threshold signature could be needed while implementing a threshold common coin, which is a critical building block used by us to overcome FLP impossibility and will soon be explained in the following paragraphs.

Note. We might also consider the notion of standard digital signature scheme, in which each party $\mathcal{P}_i$ has a private function denoted by $\mathsf{Sign}(sk_i, \cdot)$ (parameterized by a secret signing key sk_i and denoted by $\mathsf{Sign}_i(\cdot)$ for short) to generate a signature σ on the given message m, and there is also one public function $\mathsf{Verify}(pk_i, \cdot, \cdot)$ (parameterized by $\mathcal{P}_i$'s public verification key sk_i and denoted by $\mathsf{Verify}_i(\cdot, \cdot)$ for short) to validate whether a given signature σ is indeed signed by $\mathcal{P}_i$ regarding a given message m.

Threshold common coin (Coin). A (t, n)-Coin is a protocol among n parties, through which any t honest parties can mint a common coin r uniformly sampled over $\{0, 1\}^\kappa$. The adversary corrupting up to f parties (where $f < t$) cannot predicate coin r, unless $t - f$ honest parties invoke the protocol. Formally, an established (t, n)-Coin scheme satisfies the following properties in the

asynchronous authenticated setting (as described in Section 2), except with negligible probability in λ:

- **Termination.** Once t honest parties activate Coin associated to ID (denoted by Coin[ID]), each honest party that activates Coin[ID] will output a common value r.
- **Agreement.** If two honest parties output r and r' in Coin[ID] respectively, then $r = r'$.
- **Unpredictability and Unbiasedness.** A P.P.T. adversary $\mathcal{A}$ who can adaptively corrupt up to f parties (e.g. $f < t$) cannot predicate the output of Coin[ID] better than guessing over $\{0,1\}^\kappa$, unless $t - f$ honest parties activate Coin[ID]. This also implies the unbiasedness of r, namely, no P.P.T. adversary can bias the distribution of r.

Instantiation. (t, n)-Coin can be realized from non-interactive unique (t, n)-threshold signature in the random oracle model, as illustrated in the seminal work of Cachin, Kursawe and Shoup [22]. Moreover, it is immediately to generalize the Coin protocol in [22] against adaptive adversary [4,55], if being given non-interactive unique threshold signature with adaptive security [53]. In this paper, common coin is adaptively secure and can be instantiated through the construction in [55,22], which incurs $\mathcal{O}(\lambda n^2)$ bits, $\mathcal{O}(n^2)$ messages and constant $\mathcal{O}(1)$ rounds.

Random identity election. In our context, a random identity Election protocol is a $(2f + 1, n)$-Coin protocol that returns a common value over $\{1, \cdots, n\}$. Through the paper, this particular Coin is under the descriptive alias Election, a conventional terminology due to Ben-Or and El-Yaniv [13].

Asynchronous binary agreement (ABA). In an asynchronous binary agreement (ABA) protocol among n parties, the honest parties input a single bit, and aim to output a common bit $b \in \{0, 1\}$ which shall be input of at least one honest party. Formally, an ABA protocol satisfies the properties in the asynchronous authenticated setting (c.f. Section 2), except with negligible probability:

- **Termination**. If all honest parties receive binary inputs in $\{0, 1\}$ to activate ABA associated to ID (denoted by ABA[ID]), each honest party outputs a bit for ID.
- **Agreement**. If any two honest parties output b and b' for ID receptively, then $b = b'$.
- **Validity**. If any honest party outputs a bit $b \in \{0, 1\}$ for ID, then at least one honest party inputs b for ID.

Instantiation. Given adaptively secure non-interactive threshold signature in [53], the Coin scheme in [22] can immediately be generalized to defend against adaptive adversary [55], which further generalizes many ABA constructions [60,21] to be adaptively secure [55]. In particular, we adopt the ABA secure against adaptive adversary controlling up to $\lfloor \frac{n-1}{3} \rfloor$ parties in [55], which attains expected $\mathcal{O}(1)$ running time, asymptomatic $\mathcal{O}(n^2)$ messages and $\mathcal{O}(\lambda n^2)$ bits, where λ is the cryptographic security parameter.

3.2 Other notations and definition

Notations. We use $\langle x, y \rangle$ to denote a string concatenating two strings x and y. Any message between two parties is of the form $(\text{MsgType}, \mathsf{ID}, \dots)$, where ID represents the identifier that tags the protocol instance and MsgType specifies the message type. Moreover, $\Pi[\mathsf{ID}]$ refers to an instance of the protocol Π with identifier ID, and $y \leftarrow \Pi[\mathsf{ID}](x)$ means to invoke $\Pi[\mathsf{ID}]$ on input x and obtain y as output.

Probabilistically uniformly bounded statistic. Here we showcase the definition of uniformly-bounded statistic, which is a conventional notion [21,4] to rigorously describe the performance metrics of fault-tolerant protocols under the influence of arbitrary adversary, such as the messages sent by honest parties.

Definition 2. ***Uniformly Bounded Statistic***. *Let X be a random variable representing a protocol statistic (for example, the number of messages generated by the honest parties during the protocol execution). We say that X is probabilistically uniformly bounded, if there exists a fixed polynomial $T(\lambda)$ and a fixed negligible function $\delta(k)$, such that for any adversary $\mathcal{A}$ (of the protocol), there exists a negligible function $\epsilon(\lambda)$ for all $\lambda \geq 0$ and $k \geq 0$,*

$$\Pr[X \geq kT(\lambda)] \leq \delta(k) + \epsilon(\lambda).$$

A probabilistically uniformly bounded statistic of a protocol performance metric X cannot exceed the uniform bound except with negligible probability, independent of the adversary. More precisely, this means, there exists a constant c, s.t. for any adversary $\mathcal{A}$, the expected value of X must be bounded by $cT(k) + \epsilon'(\lambda)$, where $\epsilon'(\lambda)$ is a negligible function.

4 APDB: Asynchronous Provable Dispersal Broadcast

The dominating $O(Ln^2)$ term in the communication complexity of existing MVBA protocols [21,4] is because every party broadcasts its own input *all* other parties. This turns out to be unnecessary, as in the MVBA protocol, only one single party's input is decided as output. To remedy the needless communication overhead in MVBA, we introduce a new *dispersal-then-recast* methodology, through which each party $\mathcal{P}_i$ only has to spread the coded *fragments* of its input v_i to every other party instead of its entire input.

This section introduces the core building block, namely, the asynchronous provable dispersal broadcast (APDB), to instantiate the *dispersal-then-recast* idea. The notion is carefully tailored to be efficiently implementable. Especially, in contrast to related AVID protocols [23,44], APDB can disperse a value at a cost of linear messages instead of $\mathcal{O}(n^2)$, as a reflection of following trade-offs:

- The APDB notion weakens AVID, so upon that a party outputs a coded fragment in the dispersal instance of APDB, there is no guarantee that other parties will output the consistent fragments. Thus, it could be not enough to recover the dispersed value by only $f + 1$ honest parties, as these parties might receive (probably inconsistent) fragments.
- To compensate the above weakenings, we let the sender to spread the coded fragments of its input along with a succinct vector commitment of all these fragments, and then produce two succinct "proofs" *lock* and *done*. The "proofs" facilitate: (i) the *lock* proof ensures that $2f + 1$ parties receive some fragments that are committed in the same vector commitment, so the honest parties can either recover the same value, or output $\perp$ (that means the committed fragments are inconsistent); (ii) the *done* proof ensures that $2f + 1$ parties deliver valid *lock*s, thus allowing the parties to reach a common decision, e.g., via a (biased) binary BA [21], to all agree to jointly recover the dispersed value, which makes the value deemed to be recoverable.

In this way, the overall communication of dispersing a value can be brought down to minimum as the size of each fragment is only $\mathcal{O}(L/n)$ where L is the bit length of input v. Moreover, this well-tuned notion can be easily implemented in light of [4] and costs only linear messages. These efficiencies are needed to achieve the optimal communication and message complexities for MVBA.

Defining asynchronous provable dispersal broadcast. Formally, the syntax and properties of an APDB protocol are defined as follows.

Definition 3. *An* APDB *protocol with a designated sender $\mathcal{P}_s$ is equipped with a pair of predicates* (ValidateLock, ValidateDone) *and consists of a provable dispersal subprotocol (*PD*) and a recast subprotocol (*RC*), the syntax of which can be described as follows:*

- **PD subprotocol**. *In the* PD *subprotocol (with identifier* ID*) among n parties, a designated sender $\mathcal{P}_s$ inputs a value $v \in \{0,1\}^L$, and aims to split v into n encoded fragments and disperses each fragment to the corresponding party. During the* PD *subprotocol with identifier* ID*, each party is allowed to invoke an abandon*(ID) *function. After* PD *terminates, each party shall output two strings store and lock, and the sender shall output an additional string done.*
  Note that the lock and done strings are said to be valid for the identifier ID*, if and only if* ValidateLock(ID, *lock*) $= 1$ *and* ValidateDone(ID, *done*) $= 1$*, respectively.*
- **RC subprotocol**. *In the* RC *subprotocol (with identifier* ID*), all honest parties take the output of the* PD *subprotocol (with the same* ID*) as input, and aim to output the value v that was dispersed in the* RC *subprotocol. Once* RC *is completed, the parties output a common value in $\{0,1\}^L \cup \perp$.*

An APDB *protocol (*PD, RC*) with identifier* ID *satisfies the following properties in the asynchronous authenticated setting (as described in Section 2), except with negligible probability:*

- **Termination**. *If the sender $\mathcal{P}_s$ is honest and all honest parties activate* PD[ID] *without invoking abandon*(ID)*, then each honest party would output store and valid lock for* ID*; additionally, the sender $\mathcal{P}_s$ outputs valid done for* ID.
- **Recast-ability**. *If all honest parties invoke* RC[ID] *with inputting the output of* PD[ID] *and at least one honest party inputs a valid lock, then: (i) all honest parties recover a common value; (ii) if the sender dispersed v in* PD[ID] *and has not been corrupted before at least one party delivers valid lock, then all honest parties recover v in* RC[ID].
  Intuitively, the recast-ability *captures that the valid lock is a "proof" attesting that the input value dispersed via* PD[ID] *can be consistently recovered by* all *parties through collectively running the corresponding* RC[ID] *instance.*
- **Provability**. *If the sender of* PD[ID] *produces valid done, then at least $f+1$ honest parties output valid lock.*
  Intuitively, the provability *indicates that done is a "completeness proof" attesting that at least $f+1$ honest parties output valid locks, such that the parties can exchange locks and then vote via* ABA *to reach an agreement that the dispersed value is deemed recoverable.*
- **Abandon-ability**. *If every party (and the adversary) cannot produce valid lock for* ID *and $f+1$ honest parties invoke abandon*(ID)*, no party would deliver valid lock for* ID.

4.1 Overview of APDB

For the PD subprotocol with identifier ID, it has a simple structure of four one-to-all or all-to-one rounds: sender $\xrightarrow{\text{STORE}}$ parties $\xrightarrow{\text{STORED}}$ sender $\xrightarrow{\text{LOCK}}$ parties $\xrightarrow{\text{LOCKED}}$ sender. Through a STORE message, every party $\mathcal{P}_i$ receives $store := \langle vc, m_i, i, \pi_i \rangle$, where m_i is an encoded fragment of the sender's input, vc is a (deterministic) commitment of the vector of all fragments, and π_i attests m_i's inclusion in vc at the i-th position; then, through STORED messages, the parties would give the sender "partial" signatures for the string $\langle \text{STORED}, \mathsf{ID}, vc \rangle$; next, the sender combines $2f+1$ valid "partial" signatures, and sends every party the combined "full" signature σ_1 for the string $\langle \text{STORED}, \mathsf{ID}, vc \rangle$ via LOCKED messages, so each party can deliver $lock := \langle vc, \sigma_1 \rangle$; finally, each party sends a "partial" signature for the string $\langle \text{LOCKED}, \mathsf{ID}, vc \rangle$, such that the sender can again combine the "partial" signatures to produce a valid "full" signature σ_2 for the string $\langle \text{LOCKED}, \mathsf{ID}, vc \rangle$, which allows the sender to deliver $done := \langle vc, \sigma_2 \rangle$.

For the RC subprotocol, it has only one-round structure, as each party only has to take some output of PD subprotocol as input (i.e., *lock* and *store*), and multicasts these inputs to all parties. As long as an honest party inputs a valid *lock*, there are at least $f+1$ honest parties deliver valid *store*s that are bound to the vector commitment vc included in *lock*, so all parties can eventually reconstruct the dispersed value that was committed in the commitment vc.

Algorithm 1 Validation func of APDB protocol, with identifier ID

function ValidateStore($i', store$): ▷ ValidateStore verifies *store* commits a fragment m_i in vc at i-th position
1: parse *store* as $\langle vc, i, m_i, \pi_i \rangle$
2: return VerifyOpen$(vc, m_i, i, \pi_i) \wedge i = i'$

function ValidateLock(ID, *lock*): ▷ ValidateLock validates *lock* contains a commitment vc signed by $2f+1$ parties
3: parse *lock* as $\langle vc, \sigma_1 \rangle$
4: return VerifyThld$_{(2f+1)}(\langle \text{STORED}, \mathsf{ID}, vc \rangle, \sigma_1)$

function ValidateDone(ID, *done*): ▷ ValidateDone validates *done* to attest $2f+1$ parties receive valid *lock*
5: parse *done* as $\langle vc, \sigma_2 \rangle$
6: return VerifyThld$_{(2f+1)}(\langle \text{LOCKED}, \mathsf{ID}, vc \rangle, \sigma_2)$

Algorithm 2 PD subprotocol, with identifier ID and sender $\mathcal{P}_s$

let $S_1 \leftarrow \{\ \}$, $S_2 \leftarrow \{\ \}$, $stop \leftarrow 0$

/* Protocol for the sender $\mathcal{P}_s$ */

1: **upon** receiving an input value v **do**
2: $m \leftarrow \mathsf{Enc}(v)$, where v is parsed as a $f+1$ vector and m is a n vector
3: $vc \leftarrow \mathsf{VCom}(m)$
4: **for** each $j \in [n]$ **do**
5: $\pi_j \leftarrow \mathsf{Open}(vc, m_j, j)$
6: **let** $store := \langle vc, m_j, j, \pi_j \rangle$ and **send** $(\textsc{Store}, \mathsf{ID}, store)$ to $\mathcal{P}_j$ ▷ multicast *store*
7: **wait** until $|S_1| = 2f+1$
8: $\sigma_1 \leftarrow \mathsf{Combine}_{(2f+1)}(\langle \textsc{Stored}, \mathsf{ID}, vc \rangle, S_1)$
9: **let** $lock := \langle vc, \sigma_1 \rangle$ and **multicast** $(\textsc{Lock}, \mathsf{ID}, lock)$ to all parties ▷ multicast *lock* proof
10: **wait** until $|S_2| = 2f+1$
11: $\sigma_2 \leftarrow \mathsf{Combine}_{(2f+1)}(\langle \textsc{Locked}, \mathsf{ID}, vc \rangle, S_2)$
12: **let** $done := \langle vc, \sigma_2 \rangle$ and **deliver** $done$ ▷ produce *done* proof

13: **upon** receiving $(\textsc{Stored}, \mathsf{ID}, \rho_{1,j})$ from $\mathcal{P}_j$ for the first time **do**
14: **if** $\mathsf{VerifyShare}_{(2f+1)}(\langle \textsc{Stored}, \mathsf{ID}, vc \rangle, (j, \rho_{1,j})) = 1$ and $stop = 0$ **then**
15: $S_1 \leftarrow S_1 \cup (j, \rho_{1,j})$

16: **upon** receiving $(\textsc{Locked}, \mathsf{ID}, \rho_{2,j})$ from $\mathcal{P}_j$ for the first time **do**
17: **if** $\mathsf{VerifyShare}_{(2f+1)}(\langle \textsc{Locked}, \mathsf{ID}, vc \rangle, (j, \rho_{2,j})) = 1$ and $stop = 0$ **then**
18: $S_2 \leftarrow S_2 \cup (j, \rho_{2,j})$

/* Protocol for each party $\mathcal{P}_i$ */

19: **upon** receiving $(\textsc{Store}, \mathsf{ID}, store)$ from sender $\mathcal{P}_s$ for the first time **do**
20: **if** $\mathsf{ValidateStore}(i, store) = 1$ and $stop = 0$ **then**
21: **deliver** $store$ and parse it as $\langle vc, i, m_i, \pi_i \rangle$ ▷ receive *store*
22: $\rho_{1,i} \leftarrow \mathsf{SignShare}_{(2f+1)}(sk_i, \langle \textsc{Stored}, \mathsf{ID}, vc \rangle)$
23: **send** $(\textsc{Stored}, \mathsf{ID}, \rho_{1,i})$ to $\mathcal{P}_s$

24: **upon** receiving $(\textsc{Lock}, \mathsf{ID}, lock)$ from sender $\mathcal{P}_s$ for the first time **do**
25: **if** $\mathsf{ValidateLock}(\mathsf{ID}, lock) = 1$ and $stop = 0$ **then**
26: **deliver** $lock$ and parse it as $\langle vc, \sigma_1 \rangle$ ▷ receive *lock*
27: $\rho_{2,i} \leftarrow \mathsf{SignShare}_{(2f+1)}(sk_i, \langle \textsc{Locked}, \mathsf{ID}, vc \rangle)$
28: **send** $(\textsc{Locked}, \mathsf{ID}, \rho_{2,i})$ to $\mathcal{P}_s$

---

procedure *abandon*(ID):
29: $stop \leftarrow 1$

4.2 Construction of APDB

As illustrated in Algorithm 1, the APDB protocol is designed with a few functions named ValidateStore, ValidateLock and ValidateDone to validate *done*, *lock* and *store*, respectively. ValidateStore checks whether a *store* message sent from the party $\mathcal{P}_i$ includes a fragment m_i that is committed in a vector commitment vc at the i-th position, ValidateLock validates *lock* to verify that $2f+1$ parties (i.e., at least $f+1$ honest parties) receive the fragments that are correctly committed in the same vector commitment vc, and ValidateDone validates *done* to verify that $2f+1$ parties (i.e., at least $f+1$ honest parties) have delivered valid *lock*s (that contain the same vc).

PD *subprotocol*. The details of the PD subprotocol are shown in Algorithm 2. In brief, a PD instance with session identifier ID (i.e., PD[ID]) allows a designated sender $\mathcal{P}_s$ to disperse a value v as follows:

1. *Store-then-Stored* (line 1-6, 13-15, 19-23). When the sender $\mathcal{P}_s$ receives an input value v to disperse, it encodes v to generate a vector of coded fragments $m = (m_1, \ldots, m_n)$ by an $(f+1, n)$-erasure code; then, $\mathcal{P}_s$ commits m in a vector commitment vc. Then $\mathcal{P}_s$ sends *store* including the commitment vc, the i-th coded fragment m_i and the commitment opening π_i to each party $\mathcal{P}_i$ by STORE messages. Upon receiving (STORE, ID, *store*) from the sender, $\mathcal{P}_i$ verifies whether *store* is valid. If that is the case, $\mathcal{P}_i$ delivers *store* and sends a $(2f+1, n)$-partial signature $\rho_{1,i}$ for $\langle$STORED, ID, $vc\rangle$ back to the sender through a STORED message.
2. *Lock-then-Locked* (line 7-9, 16-18, 24-28). Upon receiving $2f+1$ valid STORED messages from distinct parties, the sender $\mathcal{P}_s$ produces a full signature σ_1 for the string $\langle$STORED, ID, $vc\rangle$. Then, $\mathcal{P}_s$ sends *lock* including vc and σ_1 to all parties through LOCK messages. Upon receiving LOCK message, $\mathcal{P}_i$ verifies whether σ_1 is deemed as a valid full signature. If that is the case, $\mathcal{P}_i$ delivers $lock = \langle vc, \sigma_1\rangle$, and sends a $(2f+1, n)$-partial signature $\rho_{2,i}$ for the string $\langle$LOCKED, ID, $vc\rangle$ back to the sender through a LOCKED message.
3. *Done* (line 10-12). Once the sender $\mathcal{P}_s$ receives $2f+1$ valid LOCKED messages from distinct parties, it produces a full signature σ_2 for $\langle$LOCKED, ID, $vc\rangle$. Then $\mathcal{P}_s$ outputs the completeness proof $done = \langle vc, \sigma_2\rangle$ and terminates the dispersal.
4. *Abandon* (line 29). A party can invoke *abandon*(ID) to explicitly stop its participation in this dispersal instance with identification ID. In particular, if $f+1$ honest parties invoke *abandon*(ID), the adversary can no longer corrupt the sender of PD[ID] to disperse anything across the network.

RC *subprotocol*. The construction of the RC subprotocol is shown in Algorithm 3. The input of RC subprotocol consists of *lock* and *store*, which were probably delivered during the PD subprotocol. In brief, the execution of a RC instance with identification ID is as:

1. *Recast* (line 1-5). If the party $\mathcal{P}_i$ inputs *lock* and/or *store*, it multicasts them to all parties.
2. *Deliver* (line 6-18). If the party $\mathcal{P}_i$ receives a valid *lock* message, it waits for $f+1$ valid *store*s bound to this *lock*, such that $\mathcal{P}_i$ can reconstruct a value v (or a special symbol $\perp$).

4.3 Analyses of APDB

Here we present the detailed proofs along with the complexity analyses for APDB protocol.

Security intuition. The tuple of protocols in Algorithms 2 and 3 realize APDB among n parties against the adaptive adversary controlling up to $f \leq \lfloor \frac{n-1}{3} \rfloor$ parties, given (i) $(f+1, n)$-erasure code, (ii) deterministic n-vector commitment with the position-binding property, and (iii) established $(2f+1, n)$-threshold signature with adaptive security.

The high-level intuition is: (i) if any honest party outputs valid *lock*, then at least $f+1$ honest parties receives the code fragments committed in the same vector commitment, and the position-binding property ensures that the honest parties can collectively recover a common value (or the common $\perp$) from these committed fragments; (ii) whenever any party can produce a valid *done*, it attests that $2f+1$ (namely, at least $f+1$ honest) parties have indeed received valid *lock*s.

Security proof of APDB. Now we prove that Algorithms 2 and 3 would satisfy the properties of APDB as defined in Definition 3, with all but negligible probability in λ.

Algorithm 3 RC subprotocol with identifier ID, for each party $\mathcal{P}_i$

let $C \leftarrow [\,]$ ▷ a dictionary structure $C[vc]$ that stores the set of coded fragments committed in vc

1: **upon** receiving input $(store, lock)$ **do** ▷ multicast $lock$ and/or $store$
2: **if** $lock \neq \emptyset$ **then**
3: **multicast** (RcLock, ID, $lock$) to all
4: **if** $store \neq \emptyset$ **then**
5: **multicast** (RcStore, ID, $store$) to all

6: **upon** receiving (RcLock, ID, $lock$) **do**
7: **if** ValidateLock(ID, $lock$) $= 1$ **then** ▷ assert: only one valid $lock$ for each ID (c.f. Lemma 3)
8: **multicast** (RcLock, ID, $lock$) to all, if was not sent before
9: parse $lock$ as $\langle vc, \sigma_1 \rangle$
10: **wait** until $|C[vc]| = f + 1$
11: $v \leftarrow \mathsf{Dec}(C[vc])$ ▷ interpolate to decode the erasure code
12: **if** $\mathsf{VCom}(\mathsf{Enc}(v)) = vc$ **then return** v ▷ deliver the dispersed value (c.f. Corollary 1)
13: **else return** $\perp$

14: **upon** receiving (RcStore, ID, $store$) from $\mathcal{P}_j$ for the first time **do**
15: **if** ValidateStore$(j, store) = 1$ **then**
16: parse $store$ as $\langle vc, m_j, j, \pi_j \rangle$
17: $C[vc] \leftarrow C[vc] \cup (j, m_j)$ ▷ record fragments committed to each vc
18: **else discard** the invalid message

Proposition 1 *Considering a n-vector commitment scheme* $(\mathsf{VCom}, \mathsf{Open}, \mathsf{VerifyOpen})$ *and a (k, n)-erasure coding scheme* $(\mathsf{Enc}, \mathsf{Dec})$*, we have the following security assurances:*

- *Correctness. For any $S \subset [n]$ that $|S| = k$, $\Pr[\mathsf{Dec}(\{(i, m_i)\}_{j \in S}) = v \wedge \mathsf{VCom}(\mathsf{Enc}(v)) = vc \mid (m_1, \cdots, m_n) \leftarrow \mathsf{Enc}(v) \wedge vc \leftarrow \mathsf{VCom}((m_1, \cdots, m_n))] = 1$. The property reveals: if a vector of the coded fragments of a value v are committed to a vector commitment vc, then any k-subset of the fragments can recover the original value v, whose encoded fragments can be used to produce the same commitment vc.*
- *Decode binding. For any $S_1 \subset [n]$ and $S_2 \subset [n]$ that $|S_1| = |S_2| = k$ and $S_1 \neq S_2$, despite any P.P.T. adversary that generates vc and $\{(i, m_i, \pi_i)\}_{i \in [n]}$ that $\forall i \in S_1$ $\mathsf{VerifyOpen}(vc, m_i, i, \pi_i) = 1$ and $\forall j \in S_2$ $\mathsf{VerifyOpen}(vc, m_j, j, \pi_j) = 1$, there is an overwhelming probability that: (i) $\mathsf{VCom}(\mathsf{Enc}(\mathsf{Dec}(\{(i, m_i)\}_{i \in S_1}))) = \mathsf{VCom}(\mathsf{Enc}(\mathsf{Dec}(\{(j, m_j)\}_{j \in S_2}))) = vc \wedge \mathsf{Dec}(\{(i, m_i)\}_{i \in S_1}) = \mathsf{Dec}(\{(j, m_j)\}_{j \in S_2})$, or (ii) $\mathsf{VCom}(\mathsf{Enc}(\mathsf{Dec}(\{(i, m_i)\}_{i \in S_1}))) \neq vc \wedge \mathsf{VCom}(\mathsf{Enc}(\mathsf{Dec}(\{(j, m_j)\}_{j \in S_2}))) \neq vc$. The property indicates whenever the (probably malicious generated) coded fragments are committed to vc, any two k-subsets of these committed fragments must: either consistently recover a common value that can re-produce a vector commitment same to vc, or respectively decode some values that re-produce some commitments that are different from vc.*

Proof. This statement of correctness is trivial, because (i) the correctness of erasure coding ensures the decoding of coded fragments must recover the original message, and (ii) erasure coding and vector commitment are deterministically computed such that running vector commitment on the vector of same message's coded fragments twice would return the same commitment.

To prove the statement of decode binding, we can prove that two following bad events violating the statement have negligible probability:

- $BadEvent_1$: $\mathsf{VCom}(\mathsf{Enc}(\mathsf{Dec}(\{(i, m_i)\}_{i \in S_1}))) = vc \wedge \mathsf{VCom}(\mathsf{Enc}(\mathsf{Dec}(\{(j, m_j)\}_{j \in S_2}))) \neq vc$, when $\forall i \in S_1$ $\mathsf{VerifyOpen}(vc, m_i, i, \pi_i) = 1$ and $\forall j \in S_2$ $\mathsf{VerifyOpen}(vc, m_j, j, \pi_j) = 1$, for $(S_1, S_2, vc, \{\pi_i\}_{i \in [n]}, \{m_i\}_{i \in [n]}) \leftarrow \mathcal{A}(1^\lambda, pp)$ where $S_1 \subset [n]$ and $S_2 \subset [n]$ and $|S_1| = |S_2| = k$.
- $BadEvent_2$: $\mathsf{VCom}(\mathsf{Enc}(\mathsf{Dec}(\{(i, m_i)\}_{i \in S_1}))) = \mathsf{VCom}(\mathsf{Enc}(\mathsf{Dec}(\{(j, m_j)\}_{j \in S_2}))) = vc \wedge \mathsf{Dec}(\{(i, m_i)\}_{i \in S_1}) \neq \mathsf{Dec}(\{(j, m_j)\}_{j \in S_2})$, when $\forall i \in S_1$ $\mathsf{VerifyOpen}(vc, m_i, i, \pi_i) = 1$ and $\forall j \in S_2$ $\mathsf{VerifyOpen}(vc, m_j, j, \pi_j) = 1$, for $(S_1, S_2, vc, \{\pi_i\}_{i \in [n]}, \{m_i\}_{i \in [n]}) \leftarrow \mathcal{A}(1^\lambda, pp)$ where $S_1 \subset [n]$ and $S_2 \subset [n]$ and $|S_1| = |S_2| = k$.

We now show if $\Pr[BadEvent_1]$ is non-negligible, the adversary can violate the position binding property of vector commitment. Assuming $BadEvent_1$ occurs, it is noticed $\mathsf{Dec}(\{(i,m_i)\}_{i\in S_1}) \neq \mathsf{Dec}(\{(i,m_i)\}_{i\in S_2})$, so $\{(i,m_i)\}_{i\in[n]}$ are ill-formed coding fragments, and there exists some $i \in S_1$ s.t. $\mathsf{Enc}(\mathsf{Dec}(\{(i,m_i)\}_{i\in S_2}))[i] \neq m_i$ and also exists some $i \in S_2$ s.t. $\mathsf{Enc}(\mathsf{Dec}(\{(i,m_i)\}_{i\in S_1}))[i] \neq m_i$ (otherwise, $\mathsf{Dec}(\{(i,m_i)\}_{i\in S_2}) = \mathsf{Dec}(\{(i,m_i)\}_{i\in S_1})$). That means, the adversary can open vector commitment vc at some position $i \in S_2$ to two different message fragments, both of which can have valid opening proofs (where one proof is π_i for m_i, and the other proof is for $\mathsf{Enc}(\mathsf{Dec}(\{(i,m_i)\}_{i\in S_1}))[i] \neq m_i$ and can be computed due to the correctness of vector commitment). Therefore, $BadEvent_1$ violates the property of position binding. Thus $\Pr[BadEvent_1]$ is negligible as position binding holds with overwhelming probability.

For $BadEvent_2$, it can be similarly argued that the bad event violates the position binding property of vector commitment. Since $\mathsf{Dec}(\{(i,m_i)\}_{i\in S_1}) \neq \mathsf{Dec}(\{(i,m_i)\}_{i\in S_2})$, there must exist some $i \in S_1$ s.t. $\mathsf{Enc}(\mathsf{Dec}(\{(i,m_i)\}_{i\in S_2}))[i] \neq m_i$ and also must exist some $i \in S_2$ s.t. $\mathsf{Enc}(\mathsf{Dec}(\{(i,m_i)\}_{i\in S_1}))[i] \neq m_i$. That means, the adversary can open vector commitment vc at some position $i \in S_2$ (and also some position $i' \in S_1$) to two different message fragments. Thus $\Pr[BadEvent_2]$ is also negligible, as position binding has only negligible probability to be violated.

Lemma 1. Termination. *If the sender $\mathcal{P}_s$ is honest and all honest parties activate $\mathsf{PD}[\mathsf{ID}]$ without invoking abandon(ID), then each honest party would output store and valid lock for ID s.t. $\mathsf{ValidateLock}(\mathsf{ID}, lock) = 1$; additionally, the sender $\mathcal{P}_s$ outputs valid done for ID.*

Proof. In case all honest parties activate $\mathsf{PD}[\mathsf{ID}]$ without abandoning, all honest parties will follow the PD protocol specified by the pseudocode shown in Algorithm 2. In addition, since the sender $\mathcal{P}_s$ is honest, it also follows the protocol.

Due to the PD protocol, the honest sender firstly sends $(\textsc{Store}, \mathsf{ID}, store)$ to $\mathcal{P}_i$, where the $\textsc{Store}$ message satisfies $\mathsf{ValidateStore}(i, store) = 1$; after receiving the valid $\textsc{Store}$ message, all honest parties will send $\textsc{Stored}$ messages back to the sender. When $\mathcal{P}_s$ receiving $2f+1$ (the number of honest parties at least is $2f+1$) valid $\textsc{Stored}$ messages, $\mathcal{P}_s$ will combine these messages to generate valid signature σ_1 by $\mathsf{Combine}_{(2f+1)}$ algorithm and then obtain a valid $lock$. After that, $\mathcal{P}_s$ will send $(\textsc{Lock}, \mathsf{ID}, lock)$ to all, where the $\textsc{Lock}$ message satisfies $\mathsf{ValidateLock}(\mathsf{ID}, lock) = 1$.

Next, after receiving the valid $\textsc{Lock}$ message, all honest parties will send $\textsc{Locked}$ message back to the sender. When receiving $2f+1$ valid $\textsc{Locked}$ message, the sender can generate valid signature σ_2 by $\mathsf{Combine}_{(2f+1)}$ algorithm. Hence, the honest sender $\mathcal{P}_s$ can outputs a completeness proof $done = \langle vc, \sigma_2 \rangle$, s.t. $\mathsf{ValidateDone}(\mathsf{ID}, done) = 1$.

Lemma 2. Provability. *If the sender of $\mathsf{PD}[\mathsf{ID}]$ can produce done s.t. $\mathsf{ValidateDone}(\mathsf{ID}, done) = 1$, then at least $f+1$ honest parties output lock s.t. $\mathsf{ValidateLock}(\mathsf{ID}, lock) = 1$.*

Proof. $\mathsf{ValidateDone}(\mathsf{ID}, done) = 1$ is equivalent to that $\mathsf{VerifyThld}_{(2f+1)}(\langle \textsc{Locked}, \mathsf{ID}, vc \rangle, \sigma_2) = 1$ due to Algorithm 1, which means that without overwhelming probability, the sender does receive at least $2f+1$ $\textsc{Locked}$ message from distinct $\mathcal{P}_j$ to generate σ_2 (otherwise the threshold signature can be forged). With all but negligible probability, at least $f+1$ honest parties send $\textsc{Locked}$ messages to the sender with attaching their "partial" signatures for $\langle \textsc{Locked}, \mathsf{ID}, vc \rangle$. From Algorithm 2, we know the honest parties sends their "partial" signatures for $\langle \textsc{Locked}, \mathsf{ID}, vc \rangle$ to the sender, iff they deliver valid $lock$ which satisfies $\mathsf{ValidateLock}(\mathsf{ID}, lock) = 1$. So this lemma holds with overwhelming probability, otherwise it would break the unforgeability of the underlying threshold signature.

Lemma 3. *If two parties $\mathcal{P}_i$ and $\mathcal{P}_j$ deliver lock and $lock'$ in $\mathsf{RC}[\mathsf{ID}]$ and $\mathsf{ValidateLock}(\mathsf{ID}, lock) = 1 \wedge \mathsf{ValidateLock}(\mathsf{ID}, lock') = 1$, then $lock = lock'$.*

Proof. When $\mathsf{ValidateLock}(\mathsf{ID}, lock) = 1$, we can assert that $\mathsf{VerifyThld}_{(2f+1)}(\langle \textsc{Stored}, \mathsf{ID}, vc \rangle, \sigma_1) = 1$ due to Algorithm 1. According to the Algorithm 2, σ_1 was generated by combining $2f+1$ distinct parties' partial signatures for $\langle \textsc{Stored}, \mathsf{ID}, vc \rangle$. Therefore, there have at least $f+1$ honest parties produced a share signature for $(\textsc{Stored}, \mathsf{ID}, vc)$.

However, if $\mathsf{ValidateLock}(\mathsf{ID}, lock') = 1$, $\mathsf{VerifyThld}_{(2f+1)}(\langle \textsc{Stored}, \mathsf{ID}, vc' \rangle, \sigma_1') = 1$ also holds, which means that there are at least $f+1$ honest parties that also produce a share signature for $(\textsc{Stored}, \mathsf{ID}, vc')$. Hence, at least one honest party produce two different $\textsc{Stored}$ message if

$lock \neq lock'$. However, every honest parties compute the share signature for STORED message at most once for a given identifier ID. Hence, we have $lock = lock'$ with overwhelming probability; otherwise, the unforgeability of the underlying unique threshold signature would be broken.

Lemma 4. Recast-ability. *If all honest parties invoke* RC[ID] *with inputting the output of* PD[ID] *and at least one honest party inputs a valid lock, then: (i) all honest parties recover a common value* $\in \{0,1\}^L \cup \perp$*; (ii) if the sender dispersed* v *in* PD[ID] *and has not been corrupted before at least one party delivers valid lock, then all honest parties recover* v *in* RC[ID].

Proof. To prove the conclusion (i) of the Lemma, we would prove the following two statements: first, all honest parties can output a value; second, the output of any two honest parties would be same.

Part 1: Since at least one honest party delivers *lock* satisfying $\mathsf{ValidateLock}(\mathsf{ID}, lock) = 1$, according to the Algorithm 3, it will multicast $(\text{RCLOCK}, \mathsf{ID}, lock)$ to all. Hence, all honest parties can receive the valid *lock*.

Note whenever a valid $lock := \langle vc, \sigma_1 \rangle$ can be produced, there is a valid threshold signature σ_1 was generated by combining $2f+1$ distinct parties' partial signatures for $(\text{STORE}, \mathsf{ID}, vc)$, due to Algorithm 2. Also notice that an honest party partially signs $(\text{STORE}, \mathsf{ID}, vc)$, iff it delivers valid *store* that is bound to the commitment vc. So there are at least $f+1$ honest parties deliver the valid *store* that are committed to the same commitment string vc.

Thus there have at least $f+1$ honest parties $\mathcal{P}_i$ will multicast valid $(\text{RCSTORE}, \mathsf{ID}, store)$ message to all, due to Algorithm 3. For each honest party, they can eventually receive a valid valid RCLOCK message and $f+1$ valid RCSTORE messages, that are corresponding to the same vc. So all parties will always attempt the decode the received fragments carried by RCSTORE messages to eventually recover some value.

Part 2: From Lemma 3, all honest parties would receive valid RCLOCK messages with the same $lock := \langle vc, \sigma_1 \rangle$. Therefore, each honest party can receive $f+1$ valid RCSTORE messages, which contain $f+1$ fragments that are committed to the same vector commitment vc. Due to Proposition 1, either every honest party $\mathcal{P}_i$ have $\mathsf{VCom}(\mathsf{Enc}(\mathsf{Dec}(C[vc]))) = vc$ or every honest party $\mathcal{P}_i$ have $\mathsf{VCom}(\mathsf{Enc}(\mathsf{Dec}(C[vc]))) \neq vc$. Therefore, either all honest parties return a common value $\mathsf{Dec}(C[vc])$ in $\{0,1\}^\ell$, or they return a special symbol $\perp$.

The conclusion (i) of the Lemma holds immediately by following Part 1 and Part 2.

For the conclusion (ii) of the Lemma, if the sender $\mathcal{P}_s$ has not been corrupted (so-far-uncorrupted) before at least one party delivers valid *lock* and passed the value v into PD[ID] as input, the sender would at least follow the protocol to send STORE messages for dispersing v. Moreover, when the so-far-uncorrupted $\mathcal{P}_s$ delivers valid *lock*, at least $f+1$ honest parties already receive the STORE messages for dispersing v, so the adversary can no longer corrupts $\mathcal{P}_s$ to disperse a value v' different from v, as it cannot produce valid *lock* or valid *done* for v'. From the proving of conclusion (i), we know all parties would recover the value $\mathsf{Dec}(C[vc])$, which must be v due to the correctness of erasure code and vector commitment.

Lemma 5. Abandon-ability. *If every party (and the adversary) cannot produce valid lock for* ID *and* $f+1$ *honest parties invoke abandon*(ID)*, no party would deliver valid lock for* ID.

Proof. From Algorithm 2, we know it needs $2f+1$ valid STORED messages to produce a valid $lock := \langle vc, \sigma_1 \rangle$. Since any parties (including the adversary) has not yet produced a valid *lock* and $f+1$ honest parties invoke *abandon*(ID), there are at most $2f$ parties are participating in the PD[ID] instance. So there are at most $2f$ valid STORED messages, which are computationally infeasible for any party to produce a valid *lock*; otherwise, the unforgeability of underlying unique threshold signature would not hold.

Theorem 2. *The tuple of protocols described by Algorithms 2 and 3 solves asynchronous provable dispersal broadcast (*APDB*) among* n *parties against an adaptive adversary controlling* $f < n/3$ *parties, given (i)* $(f+1, n)$*-erasure code, (ii)* n*-vector commitment scheme, and (iii) established non-interactive* $(2f+1, n)$*-threshold signature with adaptive security.*

Proof. Lemma 1, 2, 4 and 5 complete the proof.

Complexity analysis of APDB. Through this paper, we consider L is the input length and λ is cryptographic security parameter (the length of signature, vector commitment, and openness proof for commitment), then:

- **PD complexities:** According to the process of Algorithm 2, the PD subprotocol has 4 one-to-all (or all-to-one) rounds. Hence, the total number of messages sent by honest parties is at most $4n$, which attains $\mathcal{O}(n)$ messages complexity and $\mathcal{O}(1)$ running time. Besides, the maximal size of messages is $\mathcal{O}(L/n + \lambda)$, so the communication complexity of PD is $\mathcal{O}(L + n\lambda)$.
- **RC complexities:** According to the process of Algorithm 3, the message exchanges appear in two places. First, all parties multicast the RcLock messages to all, so the first parts' messages complexity is $\mathcal{O}(n^2)$; second, all parties multicast the RcStore messages to all, thus the second parts incurring $\mathcal{O}(n^2)$ messages. Hence, the RC incurs $\mathcal{O}(n^2)$ messages complexity and constant running time. Besides, each RcLock message is sized to $\mathcal{O}(\lambda)$-bit, and the size of each RcStore message is $\mathcal{O}(L/n+\lambda)$-bit, so the communication complexity of RC is $\mathcal{O}(n^2\lambda)+\mathcal{O}(nL + n^2\lambda)$=$\mathcal{O}(nL + n^2\lambda)$ bits.

5 Dumbo-MVBA: An Asymptotically Optimal MVBA Protocol

We now apply our *dispersal-then-recast* methodology to design the optimal MVBA protocol Dumbo-MVBA, using APDB and ABA. It is secure against adaptively corrupted $\lfloor\frac{n-1}{3}\rfloor$ parties and exchanges expected $\mathcal{O}(Ln + \lambda n^2)$ bits, which is asymptotically better than all previous results [4,21] and optimal for sufficiently large input when $L \geq \lambda n$. Also it attains asymptotically optimal $\mathcal{O}(1)$ round and $\mathcal{O}(n^2)$ message complexities in expect.

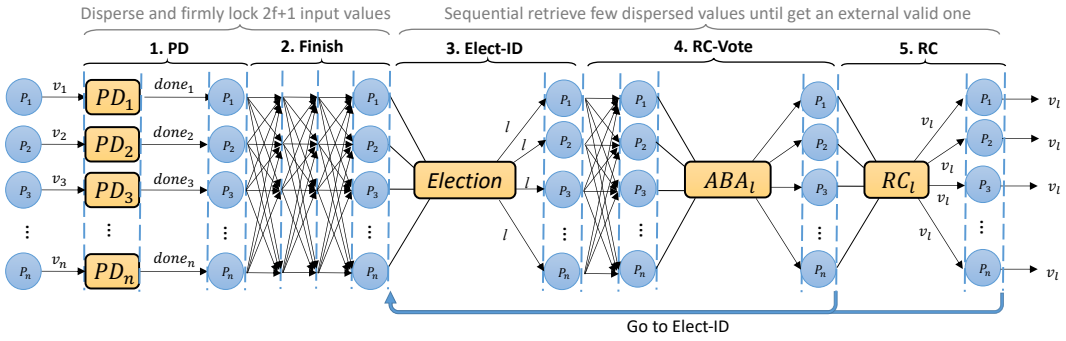

Fig. 1. The execution flow of Dumbo-MVBA.

5.1 Overview of Dumbo-MVBA

As illustrated in Figure 1, the basic ideas of our Dumbo-MVBA protocol are: (i) the parties disperse their own input values through n concurrent PD instances, until they consistently realize that enough *done*s proofs for the PD instances (i.e., $2n/3$) have been produced, so they can make sure that enough honest input values (i.e., $n/3$) have been firmly locked across the network; (ii) eventually, the parties can exchange *done*s proofs to explicitly stop all PD instances; (iii) then, the parties can invoke a common coin protocol Election to randomly elect a PD instance; (iv) later, the parties exchange their *lock* proofs of the elected PD instance and then leverage ABA to vote on whether to invoke the corresponding RC instance to recast the elected dispersal; (v) when ABA returns 1, all parties would activate the RC instance and might probably recast a common value that is externally valid; otherwise (i.e., either ABA returns 0 or RC recasts invalid value), they repeat Election, until an externally valid value is elected and collectively reconstructed.

5.2 Construction of Dumbo-MVBA

Our Dumbo-MVBA protocol invokes the following modules: (i) asynchronous provable dispersal broadcast APDB := (PD, RC); (ii) asynchronous binary agreement ABA against adaptive adversary; (iii) $(f+1, n)$ threshold signature with adaptive security; and (iv) adaptively secure $(2f+1, n)$-Coin scheme (in the alias Election) that returns random numbers over $[n]$.

Each instance of the underlying modules can be tagged by a unique extended identifier ID. These explicit IDs extend id and are used to distinguish multiple activated instances of every underlying module. For instance, $(\mathsf{PD}[\mathsf{ID}], \mathsf{RC}[\mathsf{ID}])$ represents a pair of (PD, RC) instance with identifier ID, where $\mathsf{ID} := \langle \mathsf{id}, i \rangle$ extends the identification id to represent a specific APDB instance with a designated sender $\mathcal{P}_i$. Similarly, ABA[ID] represents an ABA instance with identifier ID, where $\mathsf{ID} := \langle \mathsf{id}, k \rangle$ and $k \in \{1, 2, \dots\}$.

Protocol execution. Hereunder we are ready to present the detailed protocol description (as illustrated in Algorithm 4). Specifically, an Dumbo-MVBA instance with identifier id proceeds as:

1. *Dispersal phase* (line 1-2, 13-18). The n parties activate n concurrent instances of the provable dispersal PD subprotocol. Each party $\mathcal{P}_i$ is the designated sender of a particular PD instance $\mathsf{PD}[\langle \mathsf{id}, i \rangle]$, through which $\mathcal{P}_i$ can disperse the coded fragments of its input v_i across the network.
2. *Finish phase* (line 3, 19-35). This has a three-round structure to allow all parties consistently quit PD instances. It begins when a sender produces the *done* proof for its PD instance and multicasts *done* to all parties through a Done message, and finishes when all parties receive a Finish message attesting that at least $2f + 1$ PD instances has been "done". In addition, once receiving valid Finish, a party invokes *abandon*() to explicitly quit from all PD instances.
3. *Elect-ID phase* (line 5). Then all parties invoke the coin scheme Election, such that they obtain a common pseudo-random number l over $[n]$. The common coin l represents the identifier of a pair of $(\mathsf{PD}[\langle \mathsf{id}, l \rangle], \mathsf{RC}[\langle \mathsf{id}, l \rangle])$ instances.
4. *Recast-vote phase* (line 6-9, 36-39). Upon obtaining the coin l, the parties attempt to agree on whether to invoke the $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ instance or not. This phase has to cope with a major limit of RC subprotocol, that the $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ instance requires all parties to invoke it to reconstruct a common value. To this end, the *recast-vote* phase is made of a two-step structure. First, each party multicasts its locally recorded $lock[l]$ through RcBallotPrepare message, if the $\mathsf{PD}[\langle \mathsf{id}, l \rangle]$ instance actually delivers $lock[l]$; otherwise, it multicasts $\perp$ through RcBallotPrepare message. Then, each party waits for up to $2f + 1$ RcBallotPrepare from distinct parties, if it sees valid $lock[l]$ in these messages, it immediately activates $\mathsf{ABA}[\langle \mathsf{id}, l \rangle]$ with input 1, otherwise, it invokes $\mathsf{ABA}[\langle \mathsf{id}, l \rangle]$ with input 0. The above design follows the idea of biased validated binary agreement presented by Cachin et al. in [21], and $\mathsf{ABA}[\langle \mathsf{id}, l \rangle]$ must return 1 to each party, when $f + 1$ honest parties enter the phase with valid $lock[l]$.
5. *Recast phase* (line 10-12). When $\mathsf{ABA}[\langle \mathsf{id}, l \rangle]$ returns 1, all honest parties would enter this phase and there is always at least one honest party has delivered the valid *lock* regarding $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$. As such, the parties can always invoke the corresponding $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ instance to reconstruct a common value v_l. In case the recast value v_l does not satisfy the external predicate, the parties can consistently go back to *elect-ID phase*, which is trivial because all parties have the same external predicate; otherwise, they output v_l.

5.3 Analyses of Dumbo-MVBA

Here we present the detailed proofs along with the complexity analyses for our Dumbo-MVBA construction.

Security intuition. The Dumbo-MVBA protocol described by Algorithm 4 solves asynchronous validate byzantine agreement among n parties against adaptive adversary controlling $f \leq \lfloor \frac{n-1}{3} \rfloor$ parties, given (i) adaptively secure f-resilient APDB protocol, (ii) adaptively secure f-resilient ABA protocol, (iii) adaptively secure $(f + 1, n)$-Coin protocol (in the random oracle model), and (iv) adaptively secure $(f + 1, n)$ threshold signatures. We highlight here the key intuitions as follows:

- Termination and safety of *finish phase*. If any honest party leaves the finish phase and enters the elect-ID phase, then: (i) all honest parties will leave the finish phase, and (ii) at least $2f+1$ parties have produced *done* proofs for their dispersals.

Algorithm 4 Dumbo-MVBA protocol with identifier id and external Predicate, for each party $\mathcal{P}_i$

let $provens \leftarrow 0$, $RDY \leftarrow \{\ \}$
for each $j \in [n]$ **do**
  let $store[j] \leftarrow \emptyset$, $lock[j] \leftarrow \emptyset$, $rc\text{-}ballot[j] \leftarrow 0$
  initialize a provable dispersal instance $\mathsf{PD}[\langle \mathsf{id}, j \rangle]$

upon receiving input v_i s.t. $\mathsf{Predicate}(v_i) = true$ **do**
  pass v_i into $\mathsf{PD}[\langle \mathsf{id}, i \rangle]$ as input ▷ dispersal phase
  wait for receiving any valid FINISH message ▷ finish phase
  for each $k \in \{1, 2, 3, \dots\}$ **do**
    $l \leftarrow \mathsf{Election}[\langle \mathsf{id}, k \rangle]$ ▷ elect-id phase
    if $lock[l] \neq \emptyset$ **then multicast** (RCBALLOTPREPARE, id, l, $lock$) ▷ recast-vote phase
    else multicast (RCBALLOTPREPARE, id, l, $\perp$)
    wait for $rc\text{-}ballot[l] = 1$ or receiving $2f{+}1$ (RCBALLOTPREPARE, id, l, $\cdot$) messages from distinct parties
      $b \leftarrow \mathsf{ABA}[\langle \mathsf{id}, l \rangle](rc\text{-}ballot[l])$
      if $b = 1$ **then** ▷ recast phase
        $v_l \leftarrow \mathsf{RC}[\langle \mathsf{id}, l \rangle](store[l], lock[l])$
        if $\mathsf{Predicate}(v_l) = true$ **then** output v_l

upon $\mathsf{PD}[\langle \mathsf{id}, j \rangle]$ delivers $store$ **do** ▷ record $store[j]$ for each $\mathsf{PD}[\langle \mathsf{id}, j \rangle]$
  $store[j] \leftarrow store$

upon $\mathsf{PD}[\langle \mathsf{id}, j \rangle]$ delivers $lock$ **do** ▷ record $lock[j]$ for each $\mathsf{PD}[\langle \mathsf{id}, j \rangle]$
  $lock[j] \leftarrow lock$

upon $\mathsf{PD}[\langle \mathsf{id}, i \rangle]$ delivers $done$ **do** ▷ multicast completeness proof $done$ for $\mathsf{PD}[\langle \mathsf{id}, i \rangle]$
  multicast (DONE, id, $done$)

upon receiving (DONE, id, $done$) from party $\mathcal{P}_j$ for the first time **do**
  if $\mathsf{ValidateDone}(\langle \mathsf{id}, j \rangle, done) = 1$ **then**
    $provens \leftarrow provens + 1$
    if $provens = n - f$ **then** ▷ one honest READY $\Rightarrow n - f$ DONE $\Rightarrow f + 1$ honest DONE
      $\rho_{rdy,i} \leftarrow \mathsf{SignShare}_{(f+1)}(sk_i, \langle \text{READY}, \mathsf{id} \rangle)$
      multicast (READY, id, $\rho_{rdy,i}$)

upon receiving (READY, id, $\rho_{rdy,j}$) from party $\mathcal{P}_j$ for the first time **do**
  if $\mathsf{VerifyShare}_{(f+1)}(\langle \text{READY}, \mathsf{id} \rangle, (j, \rho_{rdy,j})) = 1$ **then**
    $RDY \leftarrow RDY \cup (j, \rho_{rdy,j})$
    if $|RDY| = f + 1$ **then** ▷ $f + 1$ READY $\Rightarrow$ one honest READY
      $\sigma_{rdy} \leftarrow \mathsf{Combine}_{(f+1)}(\langle \text{READY}, \mathsf{id} \rangle, RDY)$
      multicast (FINISH, id, σ_{rdy}) to all, if was not sent before

upon receiving (FINISH, id, σ_{rdy}) from party $\mathcal{P}_j$ for the first time **do**
  if $\mathsf{VerifyThld}_{(f+1)}(\langle \text{READY}, \mathsf{id} \rangle, \sigma_{rdy}) = 1$ **then** ▷ valid FINISH $\Rightarrow f + 1$ READY
    $abandon(\langle \mathsf{id}, j \rangle)$ for each $j \in [n]$
    multicast (FINISH, id, σ_{rdy}) to all, if was not sent before
  else discard this invalid message

upon receiving (RCBALLOTPREPARE, id, l, $lock$) from $\mathcal{P}_j$ **do**
  if $\mathsf{ValidateLock}(\langle \mathsf{id}, l \rangle, lock) = 1$ **then**
    $lock[l] \leftarrow lock$
    $rc\text{-}ballot[l] \leftarrow 1$ ▷ $rc\text{-}ballot[l] = 1 \Rightarrow lock[l]$ is valid $\Rightarrow \mathsf{PD}[\langle \mathsf{id}, j \rangle]$ is recoverable

- Termination and safety of *elect-ID phase*. Since the threshold of Election is $2f+1$, $\mathcal{A}$ cannot learn which dispersals are elected to recover before $f+1$ honest parties explicitly abandon all dispersals, which prevents the adaptive adversary from "tampering" the values dispersed by uncorrupted parties. Moreover, Election terminates in constant time.
- Termination and safety of *recast-vote* and *recast*. The honest parties would consistently obtain either 0 or 1 from recast-vote. If recast-vote returns 1, all parties invoke a RC instance to recast the elected dispersal, which will recast a common value to all parties. Those cost expected constant time.
- Quality of *recast-vote* and *recast*. The probability that *recast-vote* returns 1 is at least 2/3. Moreover, conditioned on *recast-vote* returns 1, the probability that the *recast* phase returns an externally valid value is at least 1/2.

Security proof of Dumbo-MVBA. Now we prove our Algorithm 4 satisfies all properties of MVBA with all but negligible probability.

Lemma 6. *Suppose a party $\mathcal{P}_l$ multicasts* (Done, id, *done*)*, where* ValidateDone($\langle \mathsf{id}, l\rangle$, *done*) *=1. If all honest parties participate in the* ABA[$\langle \mathsf{id}, l\rangle$] *instance, then the* ABA[$\langle \mathsf{id}, l\rangle$] *returns 1 to all.*

Proof. If a party $\mathcal{P}_l$ did multicast a valid (Done, id, *done*), we know at least $f+1$ honest parties delivers valid *lock*[l] s.t. ValidateLock(ID, *lock*[l]) $= 1$, due to the *Provability* properties of APDB. Then according to the pseudocode of Algorithm 4, at least $f+1$ honest parties will multicast valid (RcBallotPrepare, id, l, *lock*) to all. In this case, since all honest parties need to wait for $2f+1$ RcBallotPrepare messages from distinct parties, then all honest parties must see a valid (RcBallotPrepare, id, l, *lock*) message. Therefore, all honest parties would input 1 to ABA[$\langle \mathsf{id}, l\rangle$] instance. From the *validity* properties of the ABA protocol, we know that the ABA[$\langle \mathsf{id}, l\rangle$] returns 1 to all.

Lemma 7. *Suppose all honest parties participate the* ABA[$\langle \mathsf{id}, l\rangle$] *instance and* ABA[$\langle \mathsf{id}, l\rangle$] *return 1 to all. If all honest parties invoke* RC[$\langle \mathsf{id}, l\rangle$]*, then the* RC[$\langle \mathsf{id}, l\rangle$] *will return a same value to all honest parties. Besides, if $\mathcal{P}_l$ (sender) is an honest party, then the* RC[$\langle \mathsf{id}, l\rangle$] *will return an externally validated value.*

Proof. Since the ABA[$\langle \mathsf{id}, l\rangle$] returns 1, we know at least one honest party inputs 1 to ABA, due to the *validity* properties of ABA. It also means that at least one honest party $\mathcal{P}_i$ receives a message (RcBallotPrepare, id, l, *lock*) which satisfies ValidateLock($\langle \mathsf{id}, l\rangle$, *lock*) $= 1$. According to the *Recast-ability* properties of APDB, all honest parties will terminate in RC[$\langle \mathsf{id}, l\rangle$], and recover a common value, conditioned on all honest parties invoke RC[$\langle \mathsf{id}, l\rangle$].

In addition, if $\mathcal{P}_l$ is an honest party, $\mathcal{P}_l$ always inputs an externally valid value to PD[$\langle \mathsf{id}, l\rangle$], due to the *Recast-ability* properties of APDB, the RC[$\langle \mathsf{id}, l\rangle$] will return the exactly same valid value to all parties.

Lemma 8. *If an honest party invokes* Election[$\langle \mathsf{id}, k\rangle$]*, then at least $2f+1$ distinct* PD *instances have completed, and all honest parties also invoked* Election[$\langle \mathsf{id}, k\rangle$].

Proof. Suppose an honest party $\mathcal{P}_i$ invokes Election[$\langle \mathsf{id}, k\rangle$], then it means that $\mathcal{P}_i$ receives a valid Finish message. It also means that at least $f+1$ parties multicast valid Ready message, it implies that at least one honest party received $2f+1$ valid Done messages from distinct parties. Since each Done message can verify the PD instance is indeed completed, at least $2f+1$ distinct PD instances have been completed.

In addition, for $k = 1$, before an honest party invokes Election[$\langle \mathsf{id}, 1\rangle$], it must multicast the valid Finish message, if it was not sent before. For the other honest parties, they will also invoke the Election[$\langle \mathsf{id}, 1\rangle$], upon receiving a valid Finish message. For $k > 1$, without loss of generality, suppose an honest party $\mathcal{P}_i$ halts after invoking Election[$\langle \mathsf{id}, k\rangle$], and another honest party $\mathcal{P}_j$ halts after invoking Election[$\langle \mathsf{id}, k'\rangle$], where $k' > k$. However, according the *agreement* of ABA, all honest parties will output the same bit 0 (not recast) or 1 (to recast); in addition, according to the *recast-ability* properties of APDB, all honest parties will recover the same value if ABA returns 1. So, if

$\mathcal{P}_i$ halts after invoking $\mathsf{Election}[\langle \mathsf{id}, k\rangle]$, $\mathcal{P}_j$ shall also halt after invoking $\mathsf{Election}[\langle \mathsf{id}, k\rangle]$. Hence, the honest party $\mathcal{P}_j$ would not enter $\mathsf{Election}[\langle \mathsf{id}, k'\rangle]$, when another honest party $\mathcal{P}_i$ would not invoke $\mathsf{Election}[\langle \mathsf{id}, k'\rangle]$.

Lemma 9. Termination. *If every honest party $\mathcal{P}_i$ activates the protocol on identification* id *with proposing an input value v_i such that* $\mathsf{Predicate}(v_i) = true$*, then every honest party outputs a value v for* id *in constant time.*

Proof. According to Algorithm 4, the Dumbo-MVBA protocol first executes n concurrent PD instances. Since all honest parties start with externally valid values and all massages sent among honest parties have been delivered, from the *termination* of APDB, if no honest party abandons the PD, any honest parties can know at least $n - f$ PD instances have completed; if any honest party abandons the PD instances, it means that this party has seen a valid FINISH messages, which attests at least $n - f$ PD instances have completed.

When an honest party learns at least $n - f$ PD instances have completed, it will invoke $\mathsf{Election}[\langle \mathsf{id}, k\rangle]$ to elect a random number l. From Lemma 8, we know all other honest parties also will invoke $\mathsf{Election}[\langle \mathsf{id}, k\rangle]$. In addition, all honest parties will input a value to $\mathsf{ABA}[\langle \mathsf{id}, l\rangle]$, from the *termination* and *agreement* properties of ABA, the $\mathsf{ABA}[\langle \mathsf{id}, l\rangle]$ will return a same value to all. Next, let us consider three following cases:

- **Case 1:** $\mathsf{ABA}[\langle \mathsf{id}, l\rangle]$ returns 1 to all. According to the *recast-ability* properties of APDB, the $\mathsf{RC}[\langle \mathsf{id}, l\rangle]$ instance will terminate and recover a same value to all. The recast value can be valid and satisfy the global Predicate, then this value will be decided as output by all parties.
- **Case 2:** $\mathsf{ABA}[\langle \mathsf{id}, l\rangle]$ returns 1 to all. Due to the *recast-ability* of APDB, the $\mathsf{RC}[\langle \mathsf{id}, l\rangle]$ instance will terminate and recover a same value to all. The value can be invalid due to the global external Predicate, the honest parties will repeat Election, until Case 1 occasionally happens.
- **Case 3:** If $\mathsf{ABA}[\langle \mathsf{id}, l\rangle]$ returns 0 to all, then the honest parties will repeat Election, until Case 1 occasionally happens.

Now, we prove that the protocol terminates, after sequentially repeating ABA (and RC). Recall all honest parties start with dispersing externally valid values, so after $\mathsf{Election}[\langle \mathsf{id}, k\rangle]$ returns l for every $k \geq 1$, the probability that $\mathcal{P}_l$ is honest and completes $\mathsf{PD}[\langle \mathsf{id}, l\rangle]$ is at least $p = 1/3$. Due to the *unbiasedness* of Election, the coin L returned by Election is uniform over $[n]$.

As such, let the event E_k represent that the protocol does not terminate when $\mathsf{Election}[\langle \mathsf{id}, k\rangle]$ has been invoked, so the probability of the event E_k, $\Pr[E_k] \leq (1-p)^k$. It is clear to see $\Pr[E_k] \leq (1-p)^k \to 0$ when $k \to \infty$, so the protocol eventually halts. Moreover, let K to be the random variable that the protocol just terminates when $k = K$, so $\mathbb{E}[K] \leq \sum_{K=1}^{\infty} K(1-p)^{K-1}p = 1/p = 3$, indicating the protocol terminates in expected constant time.

Lemma 10. External-Validity. *If an honest party outputs a value v for* id*, then* $\mathsf{Predicate}(v) = true$.

Proof. According to Algorithm 4, when an honest party outputs a value v, there is always $\mathsf{Predicate}(v) = true$. Therefore, the external-validity trivially holds.

Lemma 11. Agreement. *If any two honest parties output v and v' for* id *respectively, then $v = v'$.*

Proof. From lemma 8, we know if an honest party invokes $\mathsf{Election}[\langle \mathsf{id}, k\rangle]$, then all honest parties also invoke $\mathsf{Election}[\langle \mathsf{id}, k\rangle]$. From the *agreement* properties of Election, all honest parties get the same coin l. Hence, all honest parties will participate in the same $\mathsf{ABA}[\langle \mathsf{id}, l\rangle]$ instance. Besides, due to the *agreement* of ABA, all honest parties will get a same bit b. Hence, upon $\mathsf{ABA}[\langle \mathsf{id}, l\rangle] = 1$, then all honest parties will participate in the same $\mathsf{RC}[\langle \mathsf{id}, l\rangle]$ instance. According to the *recast-ability* property of APDB, all honest parties must output the same value.

Lemma 12. Quality. *If an honest party outputs v for* id*, the probability that v was proposed by the adversary is at most 1/2.*

Proof. Due to Lemma 8, as long as an honest party activates Election, at least $2f + 1$ distinct PD instances have completed, which means these PD instances' senders can produce valid completeness *done* proofs. Moreover, if any honest party invokes Election$[\langle \mathsf{id}, k\rangle]$, all honest parties will eventually invoke Election$[\langle \mathsf{id}, k\rangle]$ as well. Suppose Election$[\langle \mathsf{id}, k\rangle]$ returns l, then all honest parties will participate in the ABA$[\langle \mathsf{id}, l\rangle]$ instance. If the sender $\mathcal{P}_l$ has completed the PD protocol, due to Lemma 6, the ABA$[\langle \mathsf{id}, l\rangle]$ will return 1 to all.

Then, if ABA$[\langle \mathsf{id}, l\rangle]$ returns 0, all parties will go to the next iteration to enter Election$[\langle \mathsf{id}, k+1\rangle]$; otherwise, ABA$[\langle \mathsf{id}, l\rangle]$ returns 1, all honest parties will participate in the RC$[\langle \mathsf{id}, l\rangle]$ instance, and the RC$[\langle \mathsf{id}, l\rangle]$ instance will return a common value to all parties, due to Lemma 7.

Let $\mathbb{P}_a$ to denote the set of the parties that are already corrupted by the adversary, when the adversary can tell the output of Election with non-negligible probability. Due to the *unpredictability* property of Election, upon the adversary can realize the output of Election, at least $f + 1$ honest parties have already activated Election and therefore have abandoned all PD instances. This further implies that, once the adversary realizes the output of Election, the adversary can no longer disperse adversarial values by adaptively corrupting any so-far-uncorrupted senders outside $\mathbb{P}_a$.

Moreover, when the adversary is able to predicate the output of Election, at least $2f + 1$ PD instances have been completed, out of which at most $|\mathbb{P}_a|$ instances are dispersed by the adversary. Therefore, we consider the worst case that: (i) only $f + 1$ honest parties have completed their PD instances, and (ii) $|\mathbb{P}_a| = f$ and these f PD instances sent by the adversary have completed. In addition, due to the *unbiasedness* property of Election, the adversary cannot bias the distribution of the output of Election. So Election$[\langle \mathsf{id}, k\rangle]$ returns a coin l that is uniformly sampled over $[n]$, which yields the next three cases for any $k \in \{1, 2, \dots\}$:

- **Case 1:** If the sender $\mathcal{P}_l$ has not completed the PD instance yet, and the ABA$[\langle \mathsf{id}, l\rangle]$ returns 0, then repeats Election, the probability of this case at most is $1/3$; in such the case, the protocol would go to Election to repeat;
- **Case 2:** If the sender $\mathcal{P}_l$ has completed the PD protocol and the sender' input was determined by the adversary (which might or might not be valid regarding the global predicate), the probability of this case at most is $1/3$;
- **Case 3:** If the sender $\mathcal{P}_l$ has completed the PD protocol and the sender' input was not determined by the adversary, the probability of this case at least is $1/3$;

Hence, the probability of deciding an output value v proposed by the adversary is at most $\sum_{k=1}^{\infty}(1/3)^k = 1/2$.

Theorem 3. *In random oracle model, the protocol described by Algorithm 4 (*Dumbo-MVBA*) realizes asynchronous validate byzantine agreement among n parties against adaptive adversary controlling $f < n/3$ parties, given (i) f-resilient* APDB *protocol against adaptive adversary, (ii) f-resilient* ABA *protocol against adaptive adversary, and (iii) adaptively secure non-interactive $(2f+1, n)$ and $(f + 1, n)$ threshold signatures.*

Proof. Lemma 9, 10, 11 and 12 complete the proof.

Complexity analysis of Dumbo-MVBA. The Dumbo-MVBA achieves: (i) asymptotically optimal round and message complexities, and (ii) asymptotically optimal communicated bits $\mathcal{O}(Ln + \lambda n^2)$ for any input $L \geq \lambda n$.

According to the pseudocode of algorithm 4, the breakdown of its cost can be briefly summarized in the next five phases: (i) the dispersal phase that consists of the n concurrent PD instances; (ii) the finish phase which is made of three all-to-all multicasts of Done, Ready and Finish messages; (iii) the elect-ID phase where is an invocation of Election; (iv) the recast-vote phase that has one all-to-all multicast of RcBallotPrepare messages and an invocation of ABA instance; (v) the recast phase where is to executes an RC instance.

Due to the complexity analysis of APDB in section 4.3, we know the PD's message complexity is $\mathcal{O}(n)$ and its communication complexity is $\mathcal{O}(L + n\lambda)$; the RC's message complexity is $\mathcal{O}(n^2)$ and its communication complexity is $\mathcal{O}(nL + n^2\lambda)$. So the complexities of Dumbo-MVBA protocol can be summarized as:

- **Round complexity:** The protocol terminates in expected constant running time due to Lemma 9.
- **Message complexity:** In the dispersal phase, there are n PD instances, each of which incurs $\mathcal{O}(n)$ messages. In the finish phase, there are three all-to-all multicasts, which costs $\mathcal{O}(n^2)$ messages. In the elect phase, there is one common coin, that incurs $\mathcal{O}(n^2)$ messages. In the rc-vote phase, there is one all-to-all multicast and one ABA instance, which incurs $\mathcal{O}(n^2)$ messages. In recast phase, there is only one RC instance, thus yielding $\mathcal{O}(n^2)$ messages. Moreover, the elect phase, the rc-vote-phase, and the recast phase would be repeated for expected 3 times. To sum up, the overall message complexity of the Dumbo-MVBA protocol is $\mathcal{O}(n^2)$.
- **Communication complexity:** In the dispersal phase, there are n PD instances, each of which incurs $\mathcal{O}(L + n\lambda)$ bits. In the finish phase, there are three all-to-all multicasts, which corresponds to $\mathcal{O}(n^2)$ λ-bit messages. In the elect-ID phase, there is one common coin, that incurs $\mathcal{O}(n^2\lambda)$ bits. In the recast-vote phase, there is one all-to-all multicast and one ABA instance, thus incurring $\mathcal{O}(n^2)$ messages, each of which contains at most λ bits. In the recast phase, there is only one RC instance, thus yielding $\mathcal{O}(n^2)$ messages, each of which contains at most $\mathcal{O}(L + n\lambda)$ bits. Moreover, the elect phase and the rc-vote phase would be repeated for expected 3 times, and the recast phase would be repeated for expected 2 times. Hence, the communication complexity of the Dumbo-MVBA protocol is $\mathcal{O}(nL + n^2\lambda)$.

Note if considering $L \geq \mathcal{O}(n\lambda)$, the Dumbo-MVBA protocol realizes optimal communication complexity $\mathcal{O}(nL)$. Later in Section, we will show that $L \geq \mathcal{O}(n\lambda)$ represents many typical applications of MVBA protocols, in particular when constructing asynchronous common subset (which is a critical building block that can bridge MVBA to a broader array of applications like asynchronous atomic broadcast and asynchronous multiparty computation).

6 Dumbo-MVBA⋆: Generic Communication-Efficient MVBA Framework

The *dispersal-then-recast* methodology can also be applied to bootstrap any existing MVBA to realize optimal communication for sufficiently large input. We call this extension protocol Dumbo-MVBA⋆. The key idea is to invoke the underlying MVBA with taking as input the small-size proofs of APDB. Though Dumbo-MVBA⋆ is a "reduction" from MVBA to MVBA itself, an advanced module instead of more basic building block such as binary agreement, this self-bootstrap technique can better utilize MVBA to achieve a simple modular design as explained in Figure 2, and we note it does not require the full power of APDB (and thus can potentially remove the rounds of communication generating the *done* proof).

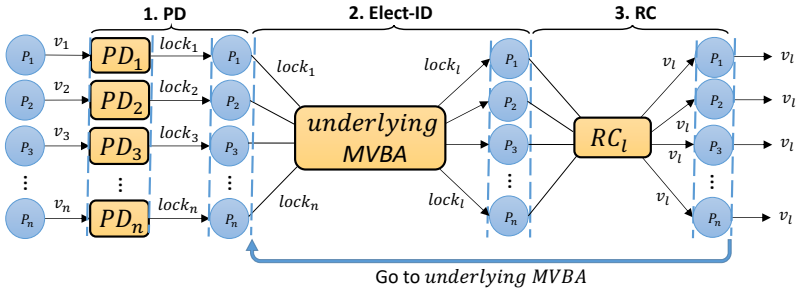

Fig. 2. The execution flow of Dumbo-MVBA⋆.

6.1 Overview of Dumbo-MVBA⋆ framework

As shown in Figure 2, the generic framework still follows the idea of *dispersal-then-recast*: (i) each party disperses its own input value and obtains a *lock* proof attesting the recast-ability of its own dispersal; (ii) then, the parties can invoke any existing MVBA as a black-box to "elect" a

valid *lock* proof, and then recover the already-dispersed value, until all parties recast and decide an externally valid value.

This generic Dumbo-MVBA⋆ framework presents a simple modular design that can enhance any existing MVBA protocol to achieve optimal communication for sufficiently large input, as long as the underlying MVBA protocol has quality property to ensure a constant probability of deciding some honest parties' input as output. In particular, when instantiating the framework with using the MVBA protocol due to Abraham et al. [4] or Guo et al. [42], we can obtain a couple of other communication-optimal MVBA instantiations that outperform the state-of-the-art and achieve only $\mathcal{O}(nL + n^2\lambda)$ communication complexity. Moreover, our Dumbo-MVBA⋆ framework also ensures that the resulting MVBA instantiations can preserve asymptotically optimal round and message complexities.

Algorithm 5 The Dumbo-MVBA⋆ protocol with identification id and external Predicate(), for each party $\mathcal{P}_i$

let $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k\rangle]$ to be an MVBA instance which takes as input string *lockproof* and is parameterized by the next external predicate:

$$\mathsf{Predicate}_{\mathsf{Election}}(lockproof) \equiv (lockproof \text{ can be parsed as } \langle i, lock_i\rangle) \wedge (\mathsf{ValidateLock}(\langle \mathsf{id}, i\rangle, lock_i) \ \wedge \ i \in [n])$$

for each $j \in [n]$ **do**
  let $store[j] \leftarrow \emptyset$ and **initialize** an instance $\mathsf{PD}[\langle \mathsf{id}, j\rangle]$

1: **upon** receiving input v_i s.t. $\mathsf{Predicate}(v_i) = 1$ **do**
2: pass v_i into $\mathsf{PD}[\langle \mathsf{id}, i\rangle]$ as input ▷ provable dispersal phase
3: **wait** for $\mathsf{PD}[\langle \mathsf{id}, i\rangle]$ delivers $lock_i$
4: **for** each $k \in \{1, 2, 3, \dots\}$ **do**
5: $\langle l, lock_l\rangle \leftarrow \mathsf{MVBA}_{under}[\langle \mathsf{id}, k\rangle](\langle i, lock_i\rangle)$ ▷ elect a finished dispersal to recast
6: $v_l \leftarrow \mathsf{RC}[\langle \mathsf{id}, l\rangle](store[l], lock_l)$
7: **if** $\mathsf{Predicate}(v_l) = true$ **then** output v_l

8: **upon** $\mathsf{PD}[\langle \mathsf{id}, j\rangle]$ delivers *store* **do** ▷ record $store[j]$ for each $\mathsf{PD}[\langle \mathsf{id}, j\rangle]$
9: $store[j] \leftarrow store$

6.2 Construction of Dumbo-MVBA⋆

Here is our generic Dumbo-MVBA⋆ framework. Informally, a Dumbo-MVBA⋆ instance with identification id (as illustrated in Algorithm 5) proceeds as:

1. *Dispersal phase* (line 1-3, 8-9). n concurrent PD instances are activated. Each party $\mathcal{P}_i$ is the designated sender of the instance $\mathsf{PD}[\langle \mathsf{id}, i\rangle]$, through which $\mathcal{P}_i$ disperses its input's fragments across the network.
2. *Elect-ID phase* (line 4-5). As soon as the party $\mathcal{P}_i$ delivers $lock_i$ during its dispersal instance $\mathsf{PD}[\langle \mathsf{id}, i\rangle]$, it takes the proof $lock_i$ as input to invoke a concrete MVBA instance with identifier $\langle \mathsf{id}, k\rangle$, where $k \in \{1, 2, \dots\}$. The external validity of underlying MVBA instance is specified to output a valid $lock_l$ for any PD instance $\mathsf{PD}[\langle \mathsf{id}, l\rangle]$.
3. *Recast phase* (line 6-7). Eventually, the $\mathsf{MVBA}[\langle \mathsf{id}, k\rangle]$ instance returns to all parties a common $lock_l$ proof for the $\mathsf{PD}[\langle \mathsf{id}, l\rangle]$ instance, namely, MVBA elects a party $\mathcal{P}_l$ to recover its dispersal. Then, all honest parties invoke $\mathsf{RC}[\langle \mathsf{id}, l\rangle]$ to recover a common value v_l. If the recast v_l is not valid, every party $\mathcal{P}_i$ can realize locally due to the same global Predicate, so each $\mathcal{P}_i$ can consistently go back the *elect-ID phase* to repeat the election by running another $\mathsf{MVBA}[\langle \mathsf{id}, k+1\rangle]$ instance with still passing $lock_i$ as input, until a valid v_l can be recovered by an elected $\mathsf{RC}[\langle \mathsf{id}, l\rangle]$ instance.

6.3 Analyses of Dumbo-MVBA⋆

Here we present the detailed proofs along with the complexity analyses for our Dumbo-MVBA⋆ construction.

Security intuition. The Dumbo-MVBA⋆ protocol described by Algorithm 5 realizes (optimal) MVBA among n parties against adaptive adversary controlling $f \leq \lfloor \frac{n-1}{3} \rfloor$ parties, given (i) f-resilient APDB protocol against adaptive adversary (with all properties but abandon-ability and provability), (ii) adaptively secure f-resilient MVBA protocol. The key intuitions of Dumbo-MVBA⋆ as follows:

- The repetition of the phase (2) and the phase (3) can terminate in expected constant time, as the quality of every underlying MVBA instance ensures that there is at least 1/2 probability of electing a PD instance whose sender was not corrupted before invoking MVBA.
- As such, the probability of not recovering any externally valid value to halt exponentially decreases with the repetition of *elect-ID* and *recast*. Hence only few (i.e., two) underlying MVBA instances and RC instances will be executed on average.

Security proof of Dumbo-MVBA⋆. Now we prove that Algorithm 5 satisfies all properties of MVBA except with negligible probability.

Lemma 13. *Suppose a party $\mathcal{P}_i$ delivers $\langle l, lock_l \rangle$ in any $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k \rangle]$ that $k \in [n]$, then all honest parties would invoke $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ and recover a common value from $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$. Besides, if $\mathcal{P}_l$ (i.e., the sender of $\mathsf{PD}[\langle \mathsf{id}, l \rangle]$) was not corrupted before $lock_l$ was delivered, then the $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ returns a validated value.*

Proof. If any honest party delivers $\langle l, lock_l \rangle$ in any MVBA_{under} instance, all honest parties deliver the same $\langle l, lock_l \rangle$ in this MVBA_{under} instance, so all honest parties would invoke $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$. Moreover, due to the specification of $\mathsf{Predicate}_{\mathsf{Election}}$ shown in Algorithm 5, all honest parties deliver $\langle l, lock_l \rangle$, s.t. $\mathsf{ValidateLock}(\langle \mathsf{id}, l \rangle, lock_l) = 1$. According to the *recast-ability* property of APDB, all honest parties (that invoke $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$) will terminate and output the same value (or the same $\perp$). In addition, the *recast-ability* property also states: conditioned on that $\mathcal{P}_l$ was not corrupted before delivering $lock_l$ and it took as input a valid value v_l to disperse in $\mathsf{PD}[\langle \mathsf{id}, l \rangle]$, the $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ will return to all parties the valid value v_l.

Lemma 14. Termination. *If every honest party $\mathcal{P}_i$ activates the protocol on identification* id *with proposing an input value v_i such that $\mathsf{Predicate}(v_i) = true$, then every honest party outputs a value v for* id*. Moreover, if the expected running time of the underlying MVBA_{under} is $\mathcal{O}(\mathsf{poly}_{rt}(n))$, Dumbo-MVBA⋆ is expected to run in $\mathcal{O}(\mathsf{poly}_{rt}(n))$ time.*

Proof. According to Algorithm 5, Dumbo-MVBA⋆ firstly executes n concurrent PD instance. From the *termination* of APDB: if a sender $\mathcal{P}_s$ is honest and all honest parties activate $\mathsf{PD}[\langle \mathsf{id}, s \rangle]$ without abandoning, then the honest sender $\mathcal{P}_s$ can deliver $lock_s$ for identification $\langle \mathsf{id}, s \rangle$ s.t. $\mathsf{ValidateLock}(\langle \mathsf{id}, s \rangle, lock_s) = 1$.

In case every honest party $\mathcal{P}_i$ passes an input to its PD instance, all honest parties can deliver a lock proof *lock* from PD, which satisfies the $\mathsf{Predicate}_{\mathsf{Election}}$ of $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k \rangle]$. Hence, each honest party $\mathcal{P}_i$ will pass a valid $\langle i, lock_i \rangle$ as input into $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k \rangle]$ for each iteration $k \in [n]$. Following the *agreement* and *termination* of MVBA, all honest parties can get the same output $\langle l, lock_l \rangle$ from each $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k \rangle]$ instance.

Due to the *external-validity* of the underlying MVBA, the output $\langle l, lock_l \rangle$ of each $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k \rangle]$ shall satisfy $\mathsf{Predicate}_{\mathsf{Election}}(\mathsf{id}, l, lock_l) = 1$. After $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k \rangle]$ returns $\langle l, lock_l \rangle$, the $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ will be invoked and return a same value v_l to all in constant time due to Lemma 13. Let us consider two cases for any $k \in \{1, 2, \dots\}$ as follows:

- **Case 1:** If the value v_l returned by $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ is valid, then output the value.
- **Case 2:** If the value v_l returned by $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ is not valid, the parties will go back to the elect-ID phase to execute $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k+1 \rangle]$, until a valid value will be decided.

Now, we prove that the honest parties would terminate in expected constant time, except with negligible probability. Due to the *quality* properties of the MVBA, the probability that $\langle l, lock_l \rangle$ was proposed by the adversary is at most $1/2$ for each MVBA_{under} instance with different identification $\langle \mathsf{id}, k \rangle$. In addition, due to the *recast-ability* of APDB, whenever $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k \rangle]$'s output $\langle l, lock_l \rangle$ was not proposed by the adversary, a valid value can be collectively recovered by all honest parties due to $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$. So the probability that an externally valid v_l is recover after invoking each $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k \rangle]$ is at least $p = 1/2$. Let the event E_k represent that the protocol does not terminate when $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k \rangle]$ has been invoked for k times, so the probability of the event E_k, $\Pr[E_k] \leq (1-p)^k$. It is clear to see $\Pr[E_k] \leq (1-p)^k \to 0$ when $k \to \infty$, so the protocol eventually halts. Moreover, let K to be the random variable that the protocol just terminates when $k = K$, so $\mathbb{E}[K] \leq \sum_{K=1}^{\infty} K(1-p)^{K-1}p = 1/p = 2$, indicating the protocol is expected to terminate after sequentially invoking $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k \rangle]$ twice.

Lemma 15. External-Validity. *If an honest party outputs a value v for* id*, then* $\mathsf{Predicate}(v) = true$.

Proof. According to Algorithm 5, when an honest party outputs a value, $\mathsf{Predicate}(v) = true$. Therefore, the external-validity trivially follows.

Lemma 16. Agreement. *If any two honest parties output v and v' for* id *respectively, then $v = v'$.*

Proof. From the *agreement* property of MVBA, all honest parties get the same output $\langle l, lock_l \rangle$. Hence, all honest parties will participate in the common $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ instance. Moreover, due to the *recast-ability* property of APDB, all honest parties will recover the same value from each invoked $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$. In addition, all honest parties have the same a-priori known predicate, and they output only when the recast value from $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ satisfying this global predicate. Thus the decided output of any two honest parties must be the same.

Lemma 17. Quality. *If an honest party outputs v for* id*, the probability that v was proposed by the adversary is at most 1/2.*

Proof. Due to the *external-validity* and *agreement* properties of the underlying MVBA_{under}, every honest party can get the same output $\langle l, lock_l \rangle$ from $\mathsf{MVBA}_{under}[\langle \mathsf{id}, k \rangle]$ which satisfies the *external* $\mathsf{Predicate}_{\mathsf{Election}}$, namely, $lock_i$ is the valid lock proof for the sender $\mathcal{P}_l$'s dispersal instance due to $\mathsf{ValidateLock}(\langle \mathsf{id}, l \rangle, lock_l) = true$.

Then, all honest parties will participate in the same $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ instance, according to Algorithm 5. From Lemma 13, we know the $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ will terminate and output a common value v_l to all. Because of the *quality* properties of the MVBA, the probability that $\langle l, lock_l \rangle$ was proposed by the adversary is at most $1/2$. So $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ returns a value v_l that might correspond the next two cases:

- **Case 1:** The sender $\mathcal{P}_l$ was corrupted by $\mathcal{A}$ (before delivering $lock_l$);
- **Case 2:** The sender $\mathcal{P}_l$ was not corrupted by $\mathcal{A}$ (before delivering $lock_l$), and executing $\mathsf{RC}[\langle \mathsf{id}, l \rangle]$ must output the valid value proposed by this sender (when it was not corrupted), due to the *recast-ability* of APDB;

Due to the *fairness* of underlying MVBA_{under}, the probability of Case 1 is at most $1/2$, while the probability of Case 2 is at least $1/2$, so the probability of deciding a value v_l was proposed by the adversary is at most $1/2$.

Theorem 4. *The protocol described by Algorithm 5 (*$\mathsf{Dumbo\text{-}MVBA}\star$*) realizes asynchronous validate Byzantine agreement among n parties against adaptive adversary controlling $f < n/3$ parties, given (i) f-resilient* APDB *protocol against adaptive adversary, and (ii) f-resilient* MVBA *protocol against adaptive adversary.*

Proof. Lemma 14, 15, 16, and 17 complete the proof.

Table 3. Asymptotic performance of MVBA when it compiles any underlying MVBA protocol with L-bit input (here the underlying MVBA has round, message and communication complexities of $\mathcal{O}(\mathsf{poly}_{rc}(n))$, $\mathcal{O}(\mathsf{poly}_{mc}(n))$ and $\mathcal{O}(\mathsf{poly}_{cc}(L,\lambda,n))$, respectively)

	Round Compl.	Message Compl.	Comm. Compl. (bits)
underlying MVBA to compile	$\mathcal{O}(\mathsf{poly}_{rc}(n))$	$\mathcal{O}(\mathsf{poly}_{mc}(n))$	$\mathcal{O}(\mathsf{poly}_{cc}(L,\lambda,n))$
Dumbo-MVBA$\star$ instantiation	$\mathcal{O}(\mathsf{poly}_{rc}(n))$	$\mathcal{O}(\mathsf{poly}_{mc}(n))$	$\mathcal{O}(Ln+\lambda n^2+\mathsf{poly}_{cc}(\lambda,\lambda,n))$

Complexity analysis of Dumbo-MVBA$\star$. According to the pseudocode of Algorithm 5, the cost of Dumbo-MVBA$\star$ is incurred in the next three phase: (i) the dispersal phase consisting of n concurrent PD instances; (ii) the elect-ID phase consisting of few expected constant number (i.e., two) of underlying MVBA instances; (iii) the recast phase consisting of few expected constant number (i.e., two) of RC instances.

Recall the complexities of PD and RC protocols: PD costs $\mathcal{O}(n)$ messages, $\mathcal{O}(L+n\lambda)$ bits, and $\mathcal{O}(1)$ running time; RC costs $\mathcal{O}(n^2)$ messages, $\mathcal{O}(nL+n^2\lambda)$ bits, and $\mathcal{O}(1)$ running time. Suppose the underlying MVBA module incurs expected $\mathcal{O}(\mathsf{poly}_{rt}(n))$ running time, expected $\mathcal{O}(\mathsf{poly}_{mc}(n))$ messages, and expected $\mathcal{O}(\mathsf{poly}_{cc}(L,\lambda,n))$ bits, where $\mathcal{O}(\mathsf{poly}_{mc}(n)) \geq \mathcal{O}(n^2)$ and $\mathcal{O}(\mathsf{poly}_{cc}(L,\lambda,n)) \geq \mathcal{O}(Ln+n^2)$ due to the lower bounds of adaptively secure MVBA. Thus, the complexities of Dumbo-MVBA$\star$ can be summarized as:

- **Running time:** Since PD and RC are deterministic protocols with constant running timing, the running time of Dumbo-MVBA$\star$ is dominated by the underlying MVBA module, namely, $\mathcal{O}(\mathsf{poly}_{rt}(n))$.
- **Message complexity:** The message complexity of n PD instances (or a RC instance) is $\mathcal{O}(n^2)$. The message complexity of the underlying MVBA is $\mathcal{O}(\mathsf{poly}_{mc}(n))$, where $\mathcal{O}(\mathsf{poly}_{mc}(n)) \geq \mathcal{O}(n^2)$. As such, the messages complexity of Dumbo-MVBA$\star$ is dominated by the underlying MVBA protocol, namely, $\mathcal{O}(\mathsf{poly}_{mc}(n))$.
- **Communication complexity:** The communication of n concurrent PD instances (or a RC instance) is $\mathcal{O}(nL+n^2\lambda)$. The underlying MVBA module incurs $\mathcal{O}(\mathsf{poly}_{cc}(\lambda,\lambda,n))$ bits. So the overall communication complexity of Dumbo-MVBA$\star$ is $\mathcal{O}(Ln+\lambda n^2+\mathsf{poly}_{cc}(\lambda,\lambda,n))$.

As shown in Table 3, Dumbo-MVBA$\star$ reduces the communication of the underlying MVBA from $\mathcal{O}(\mathsf{poly}_{cc}(L,\lambda,n))$ to $\mathcal{O}(Ln+\lambda n^2+\mathsf{poly}_{cc}(\lambda,\lambda,n))$, which removes all superlinear terms factored by L in the communication complexity. In particular, for sufficiently large input whose length $L \geq \max(\lambda n, \mathsf{poly}_{cc}(\lambda,\lambda,n)/n)$, Dumbo-MVBA$\star$ coincides with the asymptotically *optimal* $\mathcal{O}(nL)$ communication.

Concrete instantiation. Dumbo-MVBA$\star$ can be instantiated by extending any existing MVBA protocols with quality, for example, the ones due to Abraham et al. [4] and Guo et al. [42]. Moreover, both mentioned MVBA protocols cost expected $\mathcal{O}(1)$ rounds, $\mathcal{O}(n^2)$ messages, and $\mathcal{O}(n^2L+n^2\lambda)$ bits, it is clear that our Dumbo-MVBA$\star$ framework can extend them to attain $\mathcal{O}(nL+n^2\lambda)$ bits without scarifying the asymptotically optimal round and message complexities. Note if considering $\mathcal{O}(L) \geq \mathcal{O}(n\lambda)$, the above instantiation of Dumbo-MVBA$\star$ also realizes asymptotically optimal $\mathcal{O}(nL)$ communication complexity.

7 Application to Asymptotically Optimal ACS Instantiation

Given Dumbo-MVBA protocols at hand, it becomes straightforward to realize a communication-efficient asynchronous common subset (ACS) protocol with only quadratic communication cost of $\mathcal{O}(\ell n^2+\lambda n^2)$. Here ℓ is the bit-length of ACS input, and when $\ell \geq \lambda$, the communication cost $\mathcal{O}(\ell n^2)$ becomes asymptotically optimal, as the output syntax of ACS explicitly implies a trivial communication lower bound of $\Omega(\ell n^2)$ because every party has to output a $(n-f)$-sized subset of all parties' inputs. This Section would elaborate on how to achieve the result by using Dumbo-MVBA to improve the ACS construction of Cachin et al. in [21] and discuss its further applications.

7.1 Asymptotically optimal ACS instantiation from Dumbo-MVBA

As discussed in Introduction, ACS is usually the intermediate "bridge" to realize ABC [21,59] and AMPC [14,56,27]. To construct ACS, a few efficient reductions to MVBA were studied [21,43,42] and have demonstrated real-world practicality. Hinted by those relevant studies, it becomes enticing to improve these MVBA-based ACS protocols (e.g., CKPS-ACS in [21]) by plug in our Dumbo-MVBA protocols to replace earlier burdensome building block, which might also cause immediate improvements to ABC and AMPC. Let us begin with briefly reviewing the syntax and properties of ACS.

Definition 4. *A protocol among n parties with maximal tolerance up to f adaptive corruption is said to be an asynchronous common subset (ACS) protocol, if it allows each parties to take as input a value and then collectively output a common subset of all the parties' input values. In addition, it satisfies the following properties, in the asynchronous authenticated message-passing model (c.f. Section 2), with all but except negligible probability:*

- ***Agreement.*** *If any two honest parties output, then their output sets must be same;*
- ***Validity.*** *If an honest party outputs a set S, then $|S| \geq n - f$ and S contains the input values from at least $n - 2f$ honest parties;*
- ***Totality.*** *If $n - f$ honest parties invoke the protocol with taking an input, then all honest parties can output.*

We will take the ACS construction due to Cachin et al. [21] (CKPS-ACS) as an example to explain the application of Dumbo-MVBA to ACS. Recall that CKPS-ACS requires a bulletin public key infrastructure, such that the corresponding public key pk_i of each $\mathcal{P}_i$ is known by everyone in the system. Also, let Sign and Verify to be the signing and verification algorithms of some digital signature scheme that is of existential unforgeability under adaptive chosen message attacks [41]. Then, CKPS-ACS can take advantage of MVBA's external validity to solicit a set of $n-f$ message-signature pairs from distinct parties. As illustrated in Algorithm 6, the protocol has two main phases that proceed as follows:

- *Diffuse signed messages* (line 1-6). Once a party receives an input value, it signs the value and broadcasts the value-signature pair to all parties; each party would wait for $2f + 1$ such value-signature pairs sent from distinct parties;
- *Decide output subset* (line 7-9). Each party proposes the set Q of value-signature pairs to an adaptively secure MVBA instance with a properly defined external predicate (e.g., denoted by MVBA_{acs}), and waits this MVBA instance to return a set Q' of $n - f$ value-signature pairs from distinct parties; then it can output S, namely, the values in Q'.

Algorithm 6 CKPS-ACS with identifier ID (for each party $\mathcal{P}_i$), excerpted from Fig 3 in [21]

let $Q = \emptyset$
let $\mathsf{MVBA}_{acs}[\mathsf{ID}]$ to be an MVBA instance which takes as input Q and is parameterized by the next external predicate:

$$\mathsf{Predicate}(Q) \equiv (Q \text{ can be parsed as } \{(j, v_j, \sigma_j)\}) \ \wedge \ (|Q| = n - f) \ \wedge$$
$$(\forall\ (j, v_j, \sigma_j) \in Q,\ \mathsf{Verify}_j(\sigma_j, \langle \mathsf{ID}, v_j \rangle)) \wedge (\forall \text{ two } (j_1, v_{j_1}\sigma_{j_1}) \text{ and } (j_2, v_{j_2}, \sigma_{j_2}) \in Q,\ j_1 \neq j_2).$$

1: **upon** receiving input v_i **do**
2: $\quad \sigma_i \leftarrow \mathsf{Sign}_i(\langle \mathsf{ID}, v_i \rangle)$
3: $\quad$ **multicast** $(\textsc{Diffuse}, \mathsf{ID}, v_i, \sigma_i)$ to all parties
4: **upon** receive $(\textsc{Diffuse}, \mathsf{ID}, v_j, \sigma_j)$ message from $\mathcal{P}_j$ for the first time **do**
5: $\quad$ **if** $\mathsf{Verify}_j(\sigma_j, \langle \mathsf{ID}, v_j \rangle)$=1 **then**
6: $\quad\quad Q = Q \cup (j, v_j, \sigma_j)$
7: **upon** $|Q| = n - f$ **do**
8: $\quad Q' \leftarrow \mathsf{MVBA}_{acs}[\mathsf{ID}](Q)$ ▷ Here MVBA_{acs} is instantiated by one of Dumbo-MVBA protocols
9: $\quad$ output $S = \{v_j \mid (\cdot, v_j, \cdot) \in Q'\}$

The concrete performance of CKPS-ACS heavily depends on the actual instantiation of underlying MVBA_{acs}. Prior to this study, existing MVBA protocols [21,4] have a $\mathcal{O}(Ln^2)$-term in communication cost (where L is the bit length of MVBA input), thus resulting in burdensome cubic communicated bits during CKPS-ACS's execution. Nevertheless, thanks to the improvements achieved by our Dumbo-MVBA protocols, we can use Dumbo-MVBA directly to instantiate CKPS-ACS, realizing a better CKPS-ACS instantiation (denoted by CKPS-ACS-D for short through the paper). CKPS-ACS-D improves the communication cost of earlier CKPS-ACS instantiations by an $\mathcal{O}(n)$ factor, so only expected quadratic bits would be sent (by honest parties). Besides, CKPS-ACS-D remains the asymptotically optimal message and round complexities and attains the maximal tolerance against up to $n/3$ adaptive Byzantine corruption.

7.2 Analyses of the ACS instantiation from Dumbo-MVBA

Here we present detailed analyses of our efficient ACS instantiation CKPS-ACS-D.

Security proof of CKPS-ACS-D. Noticeably, CKPS [21] did not explicitly abstract a functionality of ACS and hence did not prove their implicit ACS construction satisfies the properties of ACS according to Definition 4, we thus also give such proofs for sake of completeness.

Lemma 18. ***Agreement and totality.*** *Algorithm 6 satisfies the agreement and totality properties of* ACS *except with negligible probability.*

Proof. We prove agreement through proof by contradiction: due to lines 8 and 9, if Algorithm 6 does not satisfy agreement, the agreement of underlying MVBA_{acs} is also broken, which leads to a contradiction since MVBA_{acs} satisfies Definition 1.

Totality can also be proven by proof by contradiction. There are at least $n-f$ parties that are honest through the course of the protocol. Conditioned on all honest parties start ACS, every honest party must receive a set of value-signature pairs satisfying MVBA_{acs}'s external validity condition. Hence, all honest parties would invoke MVBA_{acs} with externally valid input. Assuming Algorithm 6 might not satisfy totality, it would break the termination property of underlying MVBA_{acs}, leading to contradiction.

Lemma 19. ***Validity.*** *If an honest party outputs a set S, then $|S| \geq n-f$ and S contains inputs from at least $n-2f$ honest parties.*

Proof. Proof by contradiction: according to the external validity condition of MVBA_{acs} and the pseudocode of lines 8 and 9, if Algorithm 6 does not satisfy the validity property of ACS, then either the external validity of MVBA_{acs} or the unforgeability of digital signature is broken.

Theorem 5. *In the authenticated setting, the protocol described by Algorithm 6 solves asynchronous common subset (*ACS*) among n parties against adaptive adversary controlling $f < n/3$ parties, given f-resilient* MVBA *protocol against adaptive adversary.*

Proof. Lemmas 18 and 19 complete the proof.

Complexity analysis of CKPS-ACS-D. The complexity of Algorithm 6 is incurred in the following two phases: (i) everyone multicasts its digitally signed ACS input to all parties; (ii) all parties collectively execute a specific MVBA_{acs} instance with taking a set of $n-f$ message-signature pairs as input. We let MVBA_{acs} to be instantiated by our Dumbo-MVBA protocols. Considering the input length of ACS to be ℓ, the complexities of CKPS-ACS-D can be analyzed as follows:

- **Round complexity:** Considering that the multicasts of input values are deterministic process with constant rounds, the round complexity of Algorithm 6 would be dominated by its underlying MVBA_{acs} protocol. Recall we instantiate MVBA_{acs} by Dumbo-MVBA protocols, which can terminate in expected constant rounds. As such, CKPS-ACS-D enjoys expected constant round complexity.

- **Message complexity:** The multicasts of input values need $\mathcal{O}(n^2)$ messages, and MVBA_{acs} costs expected $\mathcal{O}(n^2)$ messages if being instantiated by Dumbo-MVBA protocols. Hence, the messages complexity of CKPS-ACS-D is $\mathcal{O}(n^2)$.
- **Communication complexity:** The communication cost of input multicasts is $\mathcal{O}(\lambda n^2 + \ell n^2)$. The input length L of MVBA_{acs} is $\mathcal{O}(n\lambda + n\ell)$, so the underlying MVBA_{acs} incurs $\mathcal{O}(\lambda n^2 + Ln) = \mathcal{O}(\lambda n^2 + \ell n^2)$ bits. The overall communication complexity of CKPS-ACS-D is, therefore, $\mathcal{O}(\lambda n^2 + \ell n^2)$.

7.3 Further applications to ABC and AMPC

Application to ABC (i.e., distributed ledger consensus). Our MVBA protocols and resulting ACS instantiation can be found applicable in various ABC implementations to reduce their communication overhead with preserving expected constant round confirmation latency. For example, our result can immediately reduce the commutation of CKPS ABC protocol by an $\mathcal{O}(n)$ order. For some more recent ACS protocols like HBBFT variants [59,33,75], they are centered around reductions from ABC to ACS, but their ACS instantiations suffers from expected $\mathcal{O}(\log n)$ rounds and $\tilde{\mathcal{O}}(\lambda n^3)$ communication overhead. Using our ACS instantiation CKPS-ACS-D, their confirmation latency and communication overhead can be reduced to expected $\mathcal{O}(1)$ rounds and $\mathcal{O}(\lambda n^2)$ bits, respectively.

Moreover, some very recent practical ABC protocol [39] relies on more efficient direct reduction from ACS to MVBA without the detour to ACS, showing much better performance in term of achieving throughput close to line rate (when the system scale n is several dozens). Nevertheless, [39] has a scalability bottleneck that prevents it preserving throughput in larger scales: it requires each party to take a vector of n quorum certificates as MVBA input, and when instantiating this by earlier MVBA protocols, a large communication overhead of $\mathcal{O}(\lambda n^3)$ is incurred. Our Dumbo-MVBA protocols immediately provide better building blocks, reducing the cubic communication overhead of MVBA components in [39] to only $\mathcal{O}(\lambda n^2)$.

Application to the online phase of AMPC-as-a-Service. Lu et al. recently proposed the first implementation of AMPC-as-a-Service called HoneyBadgerMPC (hbMPC) [56]. The paradigm of hbMPC consists of an offline pre-processing phase and an online computing phase: (i) the offline phase is continuously executed among parties to distributively generate Shamir's secret shares of random numbers and Beaver's multiplication triples [11]; (ii) given the pre-processed randomness shares and multiplication triple shares, the offline phase allows parties to solicit a common subset of their private inputs, and then confidentially evaluate a circuit f on the solicited inputs in a distributed manner.

The first step of hbMPC's online phase is an ACS protocol, which solicits a sufficient number of private inputs [2] used to evaluate f. Nevertheless, hbMPC adopts a sub-optimal ACS protocol [59] with expected $O(\log n)$ rounds and $O(\lambda n^3 \log n)$ communication, causing an expected online latency of $O(\log n + f_D)$ rounds and an expected online communication cost of $O(\lambda n^3 \log n + \lambda f_M \cdot n + \lambda f_D \cdot n^2)$ bits, where f_D is the depth of circuit f and f_M represents the number of multiplication gates of circuit f. Our ACS instantiation CKPS-ACS-D can immediately improve the latency and communication of online phase to $O(f_D)$ and $O(\lambda n^2 + \lambda f_M \cdot n + \lambda f_D \cdot n^2)$, respectively. Namely, the circuit-independent $\mathcal{O}(\log n)$ term of online round complexity is removed, and the circuit-independent online communication overhead is reduced to $O(\lambda n^2)$ from $O(\lambda n^3)$.

8 Conclusion and open problems

We present two optimal-resilient MVBA protocols that can reduce the communication cost of prior art [4,21] by an $\mathcal{O}(n)$ factor, where n is the number of parties. These communication-efficient

[2] Note that the input of a party can be private because: (i) the party can reconstruct a randomness that is collectively generated and shared among all parties during the offline phase, then (ii) it uses the reconstructed randomness as a one-time pad added to it plaintext input, thus obtaining the ciphertext form of input.

MVBA protocols also attain optimal round and message complexities, asymptotically. Our result complements the recent breakthrough of Abraham et al. at PODC '19 [4], which solves the remaining part of the long-standing open problem from Cachin et al. at CRYPTO '01 [21] on designing communication-efficient MVBA protocols.

Our MVBA protocols can immediately be applied to instantiation more efficient asynchronous common subset protocol with reduced communication blow-up, thus providing better consensus building block for asynchronous multiparty computation and asynchronous atomic broadcast.

Despite the progress, there are still a few interesting open problems left in the topic of MVBA. First, it is interesting to explore various trade-offs in security model to further reduce the communication complexity. For example, can we design MVBA protocols using only $o(n^2)$ messages in the setting of weak adaptive adversaries and/or near-optimal resilience? Second, it is also critical in many applications like asynchronous distributed key generation [49,3,31,40] to consider MVBA without trusted setup. Can we design trusted-setup free MVBA protocols that can attain similar communication complexity? Finally, our results rely on heavy public-key cryptography, and a natural question is can we extend the results into the setting using only lightweight cryptography with taking only minicrypt assumptions?

References

1. Abraham, I., Chan, T., Dolev, D., Nayak, K., Pass, R., Ren, L., Shi, E.: Communication complexity of byzantine agreement, revisited. In: Proc. ACM PODC 2019. pp. 317–326. ACM (2019)
2. Abraham, I., Dolev, D., Halpern, J.Y.: An almost-surely terminating polynomial protocol for asynchronous byzantine agreement with optimal resilience. In: Proceedings of the twenty-seventh ACM symposium on Principles of distributed computing. pp. 405–414 (2008)
3. Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Reaching consensus for asynchronous distributed key generation. In: Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing. pp. 363–373 (2021)
4. Abraham, I., Malkhi, D., Spiegelman, A.: Asymptotically optimal validated asynchronous byzantine agreement. In: Proc. ACM PODC 2019. pp. 337–346. ACM (2019)
5. 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)
6. Aspnes, J.: Lower bounds for distributed coin-flipping and randomized consensus. Journal of the ACM (JACM) **45**(3), 415–450 (1998)
7. Attiya, H., Censor, K.: Tight bounds for asynchronous randomized consensus. Journal of the ACM (JACM) **55**(5), 1–26 (2008)
8. Bacho, R., Loss, J.: On the adaptive security of the threshold bls signature scheme. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. pp. 193–207 (2022)
9. Bangalore, L., Choudhury, A., Patra, A.: The power of shunning: efficient asynchronous byzantine agreement revisited. Journal of the ACM (JACM) **67**(3), 1–59 (2020)
10. Bar-Joseph, Z., Ben-Or, M.: A tight lower bound for randomized synchronous consensus. In: Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing. pp. 193–199 (1998)
11. Beaver, D.: Efficient multiparty protocols using circuit randomization. In: Advances in Cryptology–CRYPTO'91: Proceedings 11. pp. 420–432. Springer (1992)
12. Ben-Or, M.: Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols. In: Proc. ACM PODC 1983. pp. 27–30. ACM (1983)
13. Ben-Or, M., El-Yaniv, R.: Resilient-optimal interactive consistency in constant time. Distributed Computing **16**(4), 249–262 (2003)
14. Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience. In: Proc. ACM PODC 1994. pp. 183–192. ACM (1994)
15. Blahut, R.E.: Theory and practice of error control codes. Addison-Wesley (1983)
16. 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 (2020)
17. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: PKC 2003. pp. 31–46. Springer (2003)

18. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: International conference on the theory and application of cryptology and information security. pp. 514–532. Springer (2001)
19. Bracha, G.: Asynchronous byzantine agreement protocols. Information and Computation **75**(2), 130–143 (1987)
20. Cachin, C., Kursawe, K., Lysyanskaya, A., Strobl, R.: Asynchronous verifiable secret sharing and proactive cryptosystems. In: Proc. ACM CCS 2002. pp. 88–97 (2002)
21. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Annual International Cryptology Conference. pp. 524–541. Springer (2001)
22. Cachin, C., Kursawe, K., Shoup, V.: Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. Journal of Cryptology **18**(3), 219–246 (2005)
23. Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: Proc. IEEE SRDS 2005. pp. 191–201. IEEE (2005)
24. Cachin, C., Vukolic, M.: Blockchain consensus protocols in the wild (keynote talk). In: 31st International Symposium on Distributed Computing (DISC 2017). Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2017)
25. Canetti, R., Rabin, T.: Fast asynchronous byzantine agreement with optimal resilience. In: Proc. ACM STOC 1993. pp. 42–51. ACM (1993)
26. Catalano, D., Fiore, D.: Vector commitments and their applications. In: PKC 2013
27. Choudhury, A., Hirt, M., Patra, A.: Asynchronous multiparty computation with linear communication complexity. In: International Symposium on Distributed Computing. pp. 388–402 (2013)
28. Cohen, S., Keidar, I., Spiegelman, A.: Not a coincidence: Sub-quadratic asynchronous byzantine agreement whp. In: 34th International Symposium on Distributed Computing (2020)
29. Correia, M., Neves, N.F., Veríssimo, P.: From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. The Computer Journal **49**(1), 82–96 (2006)
30. Das, S., Xiang, Z., Ren, L.: Asynchronous data dissemination and its applications. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 2705–2721 (2021)
31. 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 (2022)
32. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM Journal on Computing **12**(4), 656–666 (1983)
33. Duan, S., Reiter, M.K., Zhang, H.: Beat: Asynchronous bft made practical. In: Proc. ACM CCS 2018. pp. 2028–2041. ACM (2018)
34. Duan, S., Wang, X., Zhang, H.: Practical signature-free asynchronous common subset in constant time. Cryptology ePrint Archive (2023)
35. Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: Proceedings of the twentieth annual ACM symposium on Theory of computing. pp. 148–161 (1988)
36. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. JACM **32**(2), 374–382 (1985)
37. Fitzi, M., Garay, J.A.: Efficient player-optimal protocols for strong and differential consensus. In: Proc. ACM PODC 2003. pp. 211–220. ACM (2003)
38. Ganesh, C., Patra, A.: Optimal extension protocols for byzantine broadcast and agreement. Distributed Computing pp. 1–19 (2020)
39. 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)
40. Gao, Y., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Efficient asynchronous byzantine agreement without private setups. In: 2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS). pp. 246–257 (2022)
41. Goldwasser, S., Micali, S., Rivest, R.L.: A digital signature scheme secure against adaptive chosen-message attacks. SIAM Journal on computing **17**(2), 281–308 (1988)
42. Guo, B., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Speeding dumbo: Pushing asynchronous bft closer to practice. In: Proc. NDSS 2022 (2022)
43. Guo, B., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo: Faster asynchronous bft protocols. In: Proc. ACM CCS 2020. ACM (2020)
44. Hendricks, J., Ganger, G.R., Reiter, M.K.: Verifying distributed erasure-coded data. In: Proc. ACM PODC 2007. pp. 139–146. ACM (2007)
45. Huang, S.E., Pettie, S., Zhu, L.: Byzantine agreement in polynomial time with near-optimal resilience. In: Proceedings of the 54th Annual ACM SIGACT Symposium on Theory of Computing. pp. 502–514 (2022)

46. Kapron, B.M., Kempe, D., King, V., Saia, J., Sanwalani, V.: Fast asynchronous byzantine agreement and leader election with full information. ACM Transactions on Algorithms (TALG) **6**(4), 1–28 (2010)
47. Kate, A., Goldberg, I.: Distributed key generation for the internet. In: Proc. IEEE ICDCS 2009. pp. 119–128. IEEE (2009)
48. King, V., Saia, J.: Byzantine agreement in polynomial expected time. In: Proceedings of the forty-fifth annual ACM symposium on Theory of computing. pp. 401–410 (2013)
49. Kokoris Kogias, E., Malkhi, D., Spiegelman, A.: Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security. pp. 1751–1767 (2020)
50. Kursawe, K., Shoup, V.: Optimistic asynchronous atomic broadcast. In: International Colloquium on Automata, Languages, and Programming. pp. 204–215. Springer (2005)
51. Lamport, L.: The weak byzantine generals problem. JACM **30**(3), 668–676 (1983)
52. Lamport, L., Shostak, R., Pease, M.: The byzantine generals problem. ACM Transactions on Programming Languages and Systems (TOPLAS) **4**(3), 382–401 (1982)
53. Libert, B., Joye, M., Yung, M.: Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. Theoretical Computer Science **645**, 1–24 (2016)
54. Libert, B., Yung, M.: Concise mercurial vector commitments and independent zero-knowledge sets with short proofs. In: Theory of Cryptography Conference. pp. 499–517 (2010)
55. Loss, J., Moran, T.: Combining asynchronous and synchronous byzantine agreement: The best of both worlds. IACR Cryptology ePrint Archive **2018**, 235 (2018)
56. 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)
57. Lu, Y., Lu, Z., Tang, Q., Wang, G.: Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In: Proc. PODC 2020. pp. 129–138 (2020)
58. Merkle, R.C.: A digital signature based on a conventional encryption function. In: Eurocrypt 1987. pp. 369–378. Springer (1987)
59. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of bft protocols. In: Proc. ACM CCS 2016. pp. 31–42. ACM (2016)
60. Mostéfaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous byzantine consensus with $t < n/3$ and $o(n^2)$ messages. In: Proc. ACM PODC 2014. pp. 2–9. ACM (2014)
61. Nayak, K., Ren, L., Shi, E., Vaidya, N.H., Xiang, Z.: Improved extension protocols for byzantine broadcast and agreement. In: 34st International Symposium on Distributed Computing (DISC 2020)
62. Neiger, G.: Distributed consensus revisited. Information processing letters **49**(4), 195–201 (1994)
63. Patra, A.: Error-free multi-valued broadcast and byzantine agreement with optimal communication complexity. In: International Conference On Principles of Distributed Systems. pp. 34–49. Springer (2011)
64. Pease, M., Shostak, R., Lamport, L.: Reaching agreement in the presence of faults. JACM **27**(2), 228–234 (1980)
65. Rabin, M.O.: Randomized byzantine generals. In: 24th Annual Symposium on Foundations of Computer Science (sfcs 1983). pp. 403–409. IEEE (1983)
66. Rabin, M.O.: Efficient dispersal of information for security, load balancing, and fault tolerance. JACM **36**(2), 335–348 (1989)
67. Ramasamy, H.V., Cachin, C.: Parsimonious asynchronous byzantine-fault-tolerant atomic broadcast. In: International Conference On Principles Of Distributed Systems. pp. 88–102. Springer (2005)
68. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics **8**(2), 300–304 (1960)
69. Reiter, M.K.: Secure agreement protocols: Reliable and atomic group multicast in rampart. In: Proc. ACM CCS 1994. pp. 68–80. ACM (1994)
70. Shoup, V.: Practical threshold signatures. In: Eurocrypt 2000. pp. 207–220. Springer (2000)
71. Toueg, S.: Randomized byzantine agreements. In: Proceedings of the third annual ACM symposium on Principles of distributed computing. pp. 163–178 (1984)
72. Turpin, R., Coan, B.A.: Extending binary byzantine agreement to multivalued byzantine agreement. Information Processing Letters **18**(2), 73–76 (1984)
73. Wensley, J.H., Lamport, L., Goldberg, J., Green, M.W., Levitt, K.N., Melliar-Smith, P.M., Shostak, R.E., Weinstock, C.B.: Sift: Design and analysis of a fault-tolerant computer for aircraft control. Proc. the IEEE **66**(10), 1240–1255 (1978)

74. Yurek, T., Xiang, Z., Xia, Y., Miller, A.: Long live the honey badger: Robust asynchronous {DPSS} and its applications. In: 32nd USENIX Security Symposium (USENIX Security 23). pp. 5413–5430 (2023)
75. 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)