Making Hash-based MVBA Great Again

Hanwen Feng*, Zhenliang Lu*, Tiancheng Mai*, and Qiang Tang*

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

ABSTRACT

Multi-valued Validated Asynchronous Byzantine Agreement (MVBA) is one essential primitive for many distributed protocols, such as asynchronous Byzantine fault-tolerant scenarios like atomic broadcast (ABC), asynchronous distributed key generation, and many others. Recent efforts (Lu et al, PODC' 20) have pushed the communication complexity of MVBA to optimal $O(\ell n + \lambda n^2)$, which, however, heavily rely on "heavyweight" cryptographic tools, such as non-interactive threshold signatures. The computational cost of algebraic operations, the susceptibility to quantum attacks, and the necessity of a trusted setup associated with threshold signatures present significant remaining challenges. There is a growing interest in information-theoretic or hash-based constructions (historically called signature-free constructions). Unfortunately, the state-of-the-art hash-based MVBA (Duan et al., CCS'23) incurs a large $O(\ell n^2 + \lambda n^3)$-bits communication, which in turn makes the hash-based MVBA inferior performance-wise comparing with the "classical" ones. Indeed, this was clearly demonstrated in our experimental evaluations.

To make hash-based MVBA actually realize its full potential, in this paper, we introduce an MVBA with adaptive security, and $\widetilde{O}(\ell n + \lambda n^2)$ communication, exclusively leveraging conventional hash functions. Our new MVBA achieves nearly optimal communication, devoid of heavy operations, surpassing both threshold signature-based schemes and the hash-based scheme in many practical settings, as demonstrated in our experiments. For example, in scenarios with a network size of $n = 201$ and an input size of 1.75 MB, our MVBA exhibits a latency that is 81% lower than that of the existing hash-based MVBA and 47% lower than the threshold signature-based MVBA. Our new construction also achieves optimal parameters in other metrics such as $O(1)$ rounds and $O(n^2)$ message complexity, except with a sub-optimal resilience, tolerating up to 20% Byzantine corruptions (instead of 33%). Given its practical performance advantages, our new hash-based MVBA naturally leads to better asynchronous distributed protocols, by simply plugging it into existing frameworks.

1 INTRODUCTION

Byzantine fault-tolerant (BFT) consensus is the foundation of distributed computing, providing a means for individuals to establish a consistent view in a distributed environment, and facilitating the execution of higher-level functionalities. With the surge of distributed applications over the global Internet in the last decade, asynchronous BFT protocols are gaining re-surged attention and substantial progress in recent years, due to their resilience to network churns and ease of implementation.

Multi-Valued Validated Byzantine Agreement (MVBA), introduced in the seminal work of Cachin et al. [17], stands out as one of the most critical tools for asynchronous BFTs. In an MVBA protocol, each node provides a multi-bit value as input, collectively deciding on one of the input values to output, which has to satisfy a predefined condition. MVBA remained as theoretical study, [4, 17, 42], until Dumbo [37] re-established its critical importance to construct practical asynchronous BFT protocols. Since then, it starts to play a pivotal role in many distributed protocols, including asynchronous distributed key generation [3, 32, 39], dynamic-committee proactive secret sharing [38, 53], optimistic asynchronous consensus protocols [33, 41], and network agnostic distributed protocols [7].

The original MVBA of Cachin et al. [17] is with $O(\ell n^2 + \lambda n^2 + n^3)$ bits of communication. Here, ℓ is the bit length of input, n is the number of participants, and λ is the security parameter that captures the signature size etc. Its large communication complexity becomes the major bottleneck of its practical use. After 20 years, the communication complexity was reduced by the MVBA protocol of Abraham et al. [4] to be $O(\ell n^2 + \lambda n^2)$. However, when the input size is moderate, such as $O(n)$ (e.g., a vector of input bits), the ℓn^2 term becomes n^3 and dominates again. For this reason, Lu et al. [42] gave an extension framework that finally led to an optimal MVBA in Dumbo-MVBA* [42], with $O(\ell n + \lambda n^2)$ bits of communication, that matches the lower bound [2, 4]. Subsequently, Guo et al.[36] gave further concrete optimizations on rounds, and constructed Speeding MVBA (sMVBA), which eventually led to Dumbo-NG [31], a fast asynchronous BFT with very high throughput.

On the other hand, those recent communication efficient MVBA protocols made use of some "heavyweight" cryptography, particularly non-interactive threshold signatures like BLS [8, 14, 15]. Relying on those tools raises *both* security and performance (particularly computation) concerns. Specifically, the underlying algebraic assumptions are vulnerable to quantum attackers, the pairing operations are considerably expensive (e.g., 10^5 times slower than computing hash), and they may require a private setup for decentralized application scenarios like blockchains. Despite the trusted setup possibly being eliminated by using distributed key generation [27] or recently introduced transparent threshold signatures [6, 50], more communication and computation will be incurred, further hindering the performance.

Considering practical concerns associated with using threshold signatures and motivated by a desire to minimize cryptographic assumptions for enhanced security (e.g., plausible post-quantum security), there is renewed interest in exploring MVBA in the

Table 1: Comparison of the Multi-valued Validated BA protocols [1]

Protocol	Resilience	Adaptive? [2]	Communication	Message	#coin	Cryptographic Tools	Trivial Hash-based Version [3]
CKPS01-MVBA [17]	$f < n/3$	✓	$O(\ell n^2 + \lambda n^2 + n^3)$	$O(n^2)$	$O(1)$	threshold signature	$O(\ell n^2 + \lambda n^3)$
VABA [4]	$f < n/3$	✓	$O(\ell n^2 + \lambda n^2)$	$O(n^2)$	$O(1)$	threshold signature	$O(\ell n^2 + \lambda n^3)$
Dumbo-MVBA [42]	$f < n/3$	✓	$O(\ell n + \lambda n^2)$	$O(n^2)$	$O(1)$	threshold signature	$O(\ell n + \lambda n^3)$
sMVBA [36]	$f < n/3$	✓	$O(\ell n^2 + \lambda n^2)$	$O(n^2)$	$O(1)$	threshold signature	$O(\ell n^2 + \lambda n^3)$
sMVBA*-BLS [36] [5]	$f < n/3$	✓	$O(\ell n + \lambda n^2)$	$O(n^2)$	$O(1)$	threshold signature	$O(\ell n + \lambda n^3)$
sMVBA*-ECDSA [36] [6]	$f < n/3$	✓	$O(\ell n + \lambda n^3)$	$O(n^2)$	$O(1)$	ECDSA	$O(\ell n + \lambda n^3)$
FIN-MVBA [29]	$f < n/3$	✓	$O(\ell n^2 + \lambda n^3)$	$O(n^3)$	$O(1)$	hash	-
ELV-HMVBA (Sec.1.2 Warm-up)	$f < n/3$	✗	$O(\ell\kappa n + \lambda\kappa n^2 \log n)$ [4]	$O(n^2\kappa)$	$O(\kappa)$	hash	-
Our HMVBA (Sect. 5)	$f < n/5$	✓	$O(\ell n + \lambda n^2 \log n)$	$O(n^2)$	$O(1)$	hash	-

[1] Following the standard practice in asynchronous consensus literature, we assume a common coin and consistently omit its cost. Throughout this paper, We use f to denote the maximal number of nodes that an adversary can corrupt.
[2] The "classical" MVBA schemes [4, 17, 29, 36, 42] were not paired with detailed security proofs for adaptive security, which do hold if their all components are adaptively secure, particularly, as BLS [15] has been proved to be adaptively secure in a recent work [8].
[3] The trivial hash-based version means the hash-based MVBA constructions obtained by naively replacing the threshold signature used in the corresponding MVBA scheme by a concatenation of $n - f$ hash-based signatures. The asymptotic communication complexity may only get slightly worse, but the actual cubic term gets a larger coefficient too for much worse concrete complexity.
[4] κ is the statistical security parameter, which is usually chosen as a few tens.
[5] sMVBA*-BLS is the MVBA scheme obtained by plugging sMVBA into Dumbo-MVBA*'s framework to reduce $O(\ell n^2)$ term.
[6] sMVBA*-ECDSA replaces the threshold signature in sMVBA*-BLS with the catenation of $n - f$ ECDSA signatures.

information-theoretical (IT) setting [24, 27, 29] [1], or in solely using collision-resistant hash functions for better performance than their IT-secure analogs. This is in contrast with the above "classical" MVBA protocols. However, while enjoying the obvious benefits of using hash functions rather than heavy cryptographic tools like threshold signatures, the state-of-the-art design, FIN-MVBA by Duan et al. [29], suffers from high communication complexity $O(\ell n^2 + \lambda n^3)$, which is even asymptotically worse than the early construction from Cachin et al.'s [17]. It follows that current hash-based MVBA achieves post-quantum security, but at the cost of larger communication (thus inferior performance), which did not realize its full power. This leads to the natural question:

Can we develop an MVBA *that is free of heavy cryptographic operations (using collision resistant hash functions only), while at the same time, demonstrates performance benefits?*
Specifically, can we develop a hash based MVBA *with close to optimal communication (and other metrics), while maintaining adaptive security?*

1.1 Our Results

Hash-based MVBA with (Nearly) Optimal Communication and Adaptive Security. In this paper, we answer the question affirmatively by repeating the successful developments in "classical" MVBA protocols, i.e., match the complexity, and solely making blackbox use of conventional hash functions. Specifically, we present the first hash based MVBA protocol HMVBA with adaptive security, $O(1)$ rounds, and $O(\ell n + \lambda n^2 \log n)$ communication. This is achieved via a new "Dispersal-Elect-Agree" paradigm, which make use of an overlooked primitive of asynchronous multi-valued byzantine agreement with weak validity. We compare our results with existing ones in Table 1. As an immediate implication, we can just plug-in our new HMVBA protocol to existing frameworks such as Dumbo-NG [31] to get a better asynchronous BFT protocol, and many more.

As a caveat, our scheme tolerates up to 20% Byzantine corruption rather than being optimally resilient against 33% corrupted nodes. We view our result as a stepping stone towards the a hash-based MVBA achieving optimal resilience and maintaining all other benefits, while our result itself is practically useful as an end result. See Sect.1.3 for more discussions.

Implementation and Evaluation. We implemented our HMVBA in Python 3 and deployed it on AWS EC2 `t2.medium` instances evenly distributed across 13 AWS regions. For a fair comparison, we further developed Python implementations of FIN-MVBA [29] and the actual state of the art "classical" MVBA protocol sMVBA*, which is obtained by trivially instantiating the Dumbo-MVBA* framework in [42] with sMVBA in [36].

We tested the three MVBA protocols with various input sizes L and network sizes N. The results demonstrate that (1) The current hash-based MVBA FIN-MVBA is indeed sacrificing performance, as it is consistently worse than sMVBA*. (2) our new HMVBA consistently outperforms the other two MVBAs when the input size is fixed and the scale increases starting from moderate N. Moreover, our MVBA establishes a clearly better throughput-latency trade-off at a reasonable scale. See Sect.7 for details.

1.2 Challenges and Our Techniques

Recall that the goal of MVBA is to decide on a "valid" input, when all honest nodes provide valid inputs. A natural idea is to have all nodes agree on an input from a random node such that, with a constant probability, the value is valid and the protocol can be expected to be completed after a few repetitions. We found that

[1] also known as the signature-free setting in the literature [21, 44], subsuming the error-free setting [19, 48] as a special case.

existing MVBA schemes follow a common approach, which we call "Lock-Elect-Vote" (LEV) to realize this natural idea.

The existing "Lock-Elect-Vote" (LEV) approach. FIN-MVBA [29] employs a reliable broadcast protocol RBC [16] (which ensures agreement among all honest nodes even if the sender is malicious; see section 4) to "lock" input messages. Then, it invokes a leader election protocol, which is usually realized by a common coin, to decide whose input is going to be the prospective output. Finally, an ABA (asynchronous binary agreement), actually a variant called reproposable ABA [54], is applied to "vote" on the status of the elected leader's RBC instance. When ABA outputs one, it means at least one honest node inputs 1, which implies that the honest node has received the corresponding value from the elected RBC instance. RBC's property will then ensure all honest nodes will eventually receive that same value. FIN-MVBA gives a hash-based instantiation, as components have efficient hash-based instantiations [16, 44]. However, since using RBC to disseminate an ℓ-bit value to n nodes incurs $O(\ell n+\lambda n^2)$ bits of communication, invoking n parallel RBC instances leads to the communication complexity of $O(\ell n^2 + \lambda n^3)$.

With non-interactive threshold signatures, we can employ a "cheaper" broadcast primitive, provable broadcast PB [4, 37] (with communication cost of $O(\ell n + \lambda n)$) to replace RBC. Roughly, in PB, the sender, which first multicasts its input to all receivers, will collect enough partial signatures on the input from receivers and aggregate them into a threshold signature on the message to be multicasted. Since each honest receiver will sign at most one message for each sender, in each broadcast there will be at most one message having a valid threshold signature, which thus can serve as a proof showing the signed message is "the unique message" of the broadcast, facilitating a consistent view. In doing so, we get rid of the cubic term, since n parallel PB instances only cost us $O(\ell n^2 + \lambda n^2)$. Unfortunately, non-interactive threshold signatures usually require some algebraic structures that are unavailable in conventional hash functions (or any other lightweight tools), let alone the reliance on a trusted setup. [2]

Why hash-based quadratic MVBA is hard? In the LEV approach, for locking n messages we use n instances of broadcast with a strong consistency guarantee. If we use n instances of RBC, whether it has hash-based instantiations, it necessitates $O(n^3)$ communication cost. On the other hand, PB requires a *succinct* proof showing the endorsements from $n - f$ (f is the number of faulty nodes) receivers, which is highly non-trivial in hash-based setting, due to the lack of suitable algebraic structures. An alternative method is to use a concatenation of $n - f$ hash-based signature as the proof, which however blows up the communication cost to $O(n^3)$ again. MVBA with quadratic communication without using expensive cryptographic tools appears to be beyond the LEV paradigm. [3]

On the other hand, we find it easy to construct an MVBA protocol with quadratic communication complexity and optimal resilience, if we only focus on *static adversaries*. At a high-level, we can select a few nodes in the beginning, such that at least one of them is honest with an overwhelming probability; Then, we let all selected nodes broadcast their inputs (via RBC), and let all nodes agree on the broadcasted values (via ABA) and finally decide on a valid input through some deterministic rules. We term this construction by "Elect-Lock-Vote"(ELV), and discuss more details in Appendix A.

Our new approach towards adaptive security: "Disseminate-Elect-Agree". The adaptive security failure of ELV is largely due to the failure in the "locking" step. Specifically, as we are using very few ($< f$) parallel RBC instances to lock messages, an adaptive adversary can target all the senders to stop the protocol. We are facing a dilemma: on one hand, avoiding cubic communication (while not using threshold signatures) prevents us from locking $O(n)$ inputs. On the other hand, if there are only $o(n)$ inputs to be locked, an adaptive adversary may simply corrupt all the selected senders and make the protocol stuck.

To break out of this dilemma, we start with a "dissemination" (everyone initiates an instance) step (which does not use RBC thus avoiding the high communication). Now we do not have the strong insurance of RBC that if one honest node receives a value, all other honest nodes will also receive the same value. After "election" using coin, if the selected value is disseminated by a malicious node, then there is no consistency guarantee. We need a more powerful "Vote" technique to pair with the efficient "dissemination" to compensate for the absence of RBC for locking. To illustrate, let us assume the following "ideal" dissemination to design the remaining techniques, then we discuss how to realize the dissemination part efficiently.

- **Ideal dissemination.** All nodes are engaged in this phase to disseminate their inputs to the entire network. At the end of this phase, every honest node should have the inputs provided by all other honest nodes.

Now after the ideal dissemination and election, if the elected node (as sender in dissemination) is honest, every honest node is holding a same value; otherwise, they may have different values (or some may not have one). To conquer the challenge for agreement on the final output, a binary agreement (ABA) to vote as in previous constructions seems insufficient. Instead, we observe that a classical yet overlooked BFT primitive, Multi-Valued Byzantine Agreement with Weak Validity (MBA) [40, 49], is closer to our need as the more powerful "Vote" step. *weak validity* means if all honest nodes have the same input x, then they will output that value; no guarantee otherwise. Now, if the selected input is from an honest party, such that every other honest node already has it, then they must decide on this value; if the selected one is malicious, since now MBA has no guarantee, we should at least let the honest nodes be aware of the failure such that they can restart from the coin step. So before invoking an MBA protocol to vote, each honest node will multi-cast its current "notification" (either a received fragment, or nothing, just using $\perp$). More care is needed for the details (see Sec.5). We rephrase this new paradigm as "Disseminate-Elect-Agree".

Realizing dissemination with quadratic communication. We now turn to the construction of dissemination. Note that the ideal dissemination can be trivially realized in a synchronous network; Every node simply multicasts its input such that every other honest node can have it at the end of the round. The communication cost

[2] Remark that from the feasibility point of view, as inspired by recent works [6, 50], one may construct such a threshold signature by making non-blackbox use of a hash-based signature [11] and a hash-based succinct argument system [5], which, however, is much heavier than BLS [8].

[3] Note that slightly trading resilience (as long as $f = O(n)$) does not make the above problem easier. With a smaller f, we may just lock fewer (but still more than f) inputs in the first phase, which cannot improve asymptotic performance.

of such dissemination will be merely $O(\ell n^2)$. However, subtleties arise due to asynchrony: some honest nodes may have not finished the multicast step when the election starts. We will have to relax the requirement of ideal dissemination and only ask for a constant fraction of honest inputs to be disseminated to all honest nodes.

Even for the relaxed dissemination, there are subtleties around. Let us consider a straightforward approach as a baseline, where each node performs the following tasks: (1) multicast its input; (2) when receiving an input value from node $\mathcal{P}_i$ for the first time, respond OK to $\mathcal{P}_i$; (3) whenever receiving OK from $n-f$ distinct nodes, multicast DONE; (4) whenever receiving DONE from $n-f$ distinct nodes, move to the next phase. In doing so, we can guarantee there are at least $n-2f$ honest nodes (we call them good senders hereafter) who managed to deliver their input messages to at least $n-2f$ honest nodes. However, even when a good sender is elected, there can still be f honest nodes (we call them unfortunate receivers) that have not received the corresponding message. What is worse, an adaptive adversary may corrupt the elected good sender, retract the message that has not been delivered, and send different messages to these unfortunate receivers.

We rectify this situation by letting all nodes exchange information about the value received from the elected node, such that honest nodes shall use the value endorsed by the majority of other nodes as input for MBA. When $n \geq 5f+1$, there could be $3f+1$ honest nodes having received the message from an elected good sender. In the step for exchanging information, their voice will form a majority in every honest node's view.

Pushing to optimal communication using erasure code. All the above discussions assume that every node multicasts its entire input, which results in a communication cost of $O(\ell n^2)$. However, as in MVBA each node outputs only one value, so it is unnecessary for every node to keep track of all input values from other nodes. We therefore adopt the dispersal-then-recast methodology introduced in [42]. In the MVBA design of [42], instead of having each node directly send its input to everyone, they consider that each node first disperses fragments of its input to all nodes. These fragments have a smaller size compared to the full input value. What's more, all other honest nodes can reconstruct the input of the elected node using a sufficient number of fragments. We bend the methodology into our design of dissemination, using a hash-based Merkle tree to help nodes identify the correct fragments, resulting in $O(\ell n + \lambda n^2 \log n)$ bit complexity. More details can be found in Section 5.

Applying non-intrusion secure MBA and further optimizations. Our new HMVBA makes novel use of MBA to achieve agreement on messages. However, employing MBA directly for consensus on the entire message can be still costly in terms of communication, potentially squandering previous efforts. Nevertheless, our dissemination phase already ensures a certain level of data availability: if an honest node uses a message as the input of MBA, all other honest nodes also can obtain this message during recast. Hence, rather than using MBA to agree on the entire message, we leverage it to agree on a short hash **digest**. In conventional MBA, the output of MBA might be a digest provided by the adversary, resulting in the unavailability of the original value for some honest nodes and causing an agreement issue. Fortunately, the *non-intrusion security* property in MBA [45] addresses this concern. This property ensures that if the output of MBA is not $\perp$, it must be the input of an honest node, ensuring data availability for the agreed digest.

We can instantiate the MBA (actually IT-secure) from [45] in our HMVBA, featuring $O(1)$ rounds, $O(n^2)$ messages, and $O(\ell n^2)$-bit communication cost. However, for practical efficiency, the MBA in [45] requires 6 additional rounds of multicast beyond its ABA components, which we aim to optimize. By leveraging the fact that our MVBA operates with $n \geq 5f+1$, we design a more efficient IT-secure MBA with only 2 extra rounds of multicast alongside ABA in the same setting, while maintaining the same asymptotic performance as [45]. Further details are provided in Section 6.

1.3 Further Discussions

Challenges in Achieving Optimal Resilience. Constructing a hash-based MVBA with optimal resilience and quadratic communication complexity is an ideal goal, yet it presents significant challenges and remains an important open problem in the field. Specifically, to maintain quadratic communication complexity in the hash based setting, as mentioned earlier, we must forego the RBC while avoid using provable broadcast which relies on threshold signatures. The only option left to us is to disseminate each message at a linear communication cost without proof, as we did in our dissemination phase. Consequently, since we cannot lock these messages, we only ensure that, at the end of our dissemination phase, at least $n-2f$ honest nodes (forming a set S) deliver their input messages to at least $n-2f$ honest nodes. When $n=3f+1$, there are $f+1$ honest nodes that have received the initial value broadcasted by the nodes in set S. This necessitates amplifying the value received by these $f+1$ honest nodes to ensure that all honest nodes receive the same value. Note that other f honest nodes to receive a different value, thereby making achieving "agreement" more challenging. Asynchronous data dissemination [25] studied a simpler variant where the other honest nodes do not have inputs, but a much stronger multi-value "agreement" will be needed when there are substantial "noises" from other nodes.

Consider that our goal is to obtain the practical benefits of hash-based MVBA, while, as we demonstrated, existing hash-based MVBA [29] is less efficient than classic ones (e.g., sMVBA [36]). Given the challenges in obtaining an optimally resilient construction, even if it is feasible, such a construction is likely to be more complex. We conjecture that our MVBA protocol may offer better performance in practice compared to a future optimal construction (if it exists).

Leveraging our MVBA as a More Robust Optimistic Path. Our MVBA protocol can maintain agreement even when up to $n/3$ nodes are corrupted, by leveraging optimal-resilient MBA [45] in our construction. Consequently, if more than $n/5$ nodes are corrupted, sub-optimal resilience in our design primarily impacts liveness, potentially causing the protocol to become stuck. However, we can mitigate this liveness issue by employing generic frameworks of optimistic asynchronous consensus, such as Bolt-Dumbo Transfer [41], Abraxas [13], or ParBFT [23]. In cases where our protocol gets stuck, nodes can invoke the fallback mechanism and execute a conventional MVBA with optimal resilience instead. Importantly, existing optimistic paths typically exhibit zero fault tolerance, resulting in more frequent fallback invocations. In contrast, our current

construction provides a more robust optimistic path, where the fallback mechanism is less likely to be triggered.

2 RELATED WORK

Common-coin-aided consensus. The well-known FLP impossibility [30] ruled out the deterministic asynchronous Byzantine consensus. To get around this impossibility, the state-of-the-art asynchronous consensus [44] protocols rely on "common coins" to provide randomness but are deterministic otherwise. A common coin is an unpredictable and unbiased randomness, for which all honest nodes in the network should have a consistent view and can obtain the coin when a sufficient number of nodes have requested it. When designing asynchronous consensus, an idealized common coin oracle is usually assumed, such that we can focus on the consensus part. In practice, the common coin oracle can be realized by a trusted third party who distributes uniformly sampled coins to all nodes, as considered in Rabin's pioneering work [51]. There are continuing efforts to replace a trusted dealer with distributed protocols, such as the threshold signature/PRF based ones [18] and dedicated common coin protocols [27, 32]. These coin protocols can be plugged into any asynchronous common-coin-aided consensus protocol, if not worrying about the setup introduced or computational assumptions. This work follows this design paradigm.

(Multi-valued) Asynchronous Byzantine Agreement. The most basic form of asynchronous Byzantine consensus is asynchronous binary agreement (ABA), where each node has a binary value as input and will agree on a binary value. The validity of ABA is defined in an *unanimous* manner, *i.e.*, if all honest nodes input the same binary value, they will agree on this value. As there are only two candidate values, the validity of ABA implies the so-called *strong validity*, *i.e.*, the output is always the input of some honest node, which is a very useful property for applications. In the perspective of constructions, Mostefaoui et al. 's seminal work [44] presented an ABA protocol with $O(1)$ rounds and $O(n^2)$ communication complexity, not relying on any cryptographic tools beyond the coin. There are follow-up works [22] for improving the concrete performance of [44].

Multi-valued byzantine agreement (MBA) is a natural extension to ABA for handling multi-bit inputs. There is a straightforward reduction from MBA to ABA, by applying multiple ABA instances to agree on each bit. However, for an ℓ-bit input, the expected running time and the message complexity of ℓ parallel ABA instances will be blown up to $O(\log \ell)$ and $O(\ell n^2)$, respectively. Mostefaoui and Raynal [45] presented an optimized MBA with $O(1)$ rounds, $O(n^2)$ messages, and $O(\ell n^2)$ communication complexity. For large-size inputs, say $\ell \gg \lambda$, Nayak et al. [46] presented a general framework based on MBA for λ bits. Based on pairing-based cryptography, their framework can give a MBA with $O(1)$ rounds, $O(n^2)$ messages, and $O(\ell n + \lambda n^2)$ communication complexity. If only using hash functions, the communication complexity of the MBA in [46] will be $O(\ell n + \lambda n^2 \log n)$. Alternatively, Li and Chen [40] presented an MBA with communication complexity of $O(\ell n + n^2 \log n)$, achieving perfect security without using any cryptographic tools. However, the scheme in [40] requires $n \geq 5f + 1$.

Note that the MBA discussed above focuses on the unanimous style of validity, also called *weak validity*, which guarantees that when all honest nodes have the same input value, they will agree on that value. But for other cases, there is no guarantee on what value they will agree on; the output could be a default value $\perp$. Some works, including [45] considered a slightly stronger validity called *non-intrusion validity*, which means if the output $v \neq \perp$, then v must be an input of an honest node. The non-intrusion property has been leveraged and explored in consequent works [20, 52]. Compared with the non-intrusion MBA in [45], our MBA in Section 6 improves the concrete communication and rounds cost while assuming $n \geq 5f + 1$ which aligns with our MVBA.

Multi-valued Validated Asynchronous Byzantine Agreement. The weak validity of MBA is insufficient for many natural cases. A dream version of validity would be that the nodes always agree on the input of an honest node, which, often called *strong validity*, is known to be out of reach for large input sizes [47]. To address this issue, Cachin [17] introduced *external validity*, which guarantees that the nodes can always agree on a "valid" value satisfying a predefined predicate function, as long as all honest nodes input valid values. A multi-valued Byzantine agreement with external validity is often called MVBA. Note that MVBA and MBA (with weak validity) are generally incomparable, as MVBA's output may be controlled by the adversary in any input case. However, this issue can be mitigated by carefully designing a predicate that the output should satisfy. Moreover, Cachin [17] gave a simple framework for building atomic broadcast (ABC) by using MVBA. Note that ABC is directly useful in practice as it can provide a distributed public ledger, which testifies to the usefulness of MVBA.

Cachin [17] presented the first MVBA construction with $O(1)$ rounds, $O(n^2)$ message complexity, and $O(\ell n^2 + \lambda n^2 + n^3)$ communication complexity. Abraham et al. [4] improved the communication complexity to $O(\ell n^2 + \lambda n^2)$. Lu et al. [42] finally achieved the optimal communication complexity of $O(\ell n + \lambda n^2)$. Particularly, Lu et al. [42] essentially gave a general framework that can build an MVBA with the optimal communication complexity, $O(\ell n + \lambda n^2)$, from any MVBA with the communication complexity of $O(\ell n^2 + \lambda n^2)$. Recently, Guo et al. [36] presented an MVBA with the communication complexity of $O(\ell n^2 + \lambda n^2)$, featuring concretely fewer rounds, which can also be plugged into Lu et al. [42]'s framework, leading to an MVBA with optimal communication complexity and better concrete performance. All these MVBA schemes require threshold signatures, whose current constructions rely on algebraic assumptions that cannot resist quantum attackers. Meanwhile, threshold signatures require a trusted setup, which may be problematic in many settings.

Due to the drawbacks of using heavy cryptographic tools like threshold signatures, several recent works [1, 27, 29] shifted their focus on studying MVBA in the information-theoretical setting (also known as signature free setting in the literature) or the hash-based setting, where the hash function is the only cryptographic tool and used in a blackbox manner. These works actually give a framework that could be instantiated in both settings, while their hash-based instantiations enjoy better performance. Particularly, Das et al. [26, 27] essentially presented MVBA schemes as a component of their distributed key generation protocols, and its hash-based instantiation has $O(\log n)$ rounds, $O(n^3)$ messages, and the

communication complexity of $O(\ell n^2 + \lambda n^3)$. Duan et al. [29] improved Das et al.'s result to $O(1)$ rounds. Nonetheless, there exists a significant asymptotic efficiency gap between current hash-based MVBA constructions and classical constructions such as [4, 17, 42].

Other Hash-based Consensus. Hash-based constructions for other BFT primitives, like asynchronous common subset (ACS) and atomic broadcast (ABC), are also attracting attention. Building upon their MVBA, Duan et al. [29] introduced ACS in both the IT-setting and the hash-based setting, each with $O(1)$ rounds and $O(n^3)$ message complexity. However, the former has $O(\ell n^2+\lambda n^2+n^3 \log n)$ bit complexity, while the latter has $O(\ell n^2 + \lambda n^3)$ communication complexity. Prior to [29], Zhang et al. [54] gave an ACS with $O(\log n)$ rounds in the same setting, by using n parallel instances of ABA. Very recently, Sui and Duan [52] presented ABC with both IT-secure and hash-based instantiations, which feature $O(n^2)$ message complexity. However, the communication complexity of [52] is still $O(\ell n^2 + \lambda n^3)$. Nonetheless, with hash-based MVBA in this paper, it is trivial to build hash-based ACS and ABC with quadratic communication complexity.

3 MODEL AND GOAL

3.1 System model

The system involves a set of n known nodes labeled as $\{\mathcal{P}_1, \ldots, \mathcal{P}_n\}$ which are connected through pairwise authenticated channels.

We consider an adaptive and computationally bounded adversary ($\mathcal{A}$), capable of corrupting up to $f < n/5$ nodes at any point during the protocol execution. Nodes not corrupted by the adaptive adversary at a certain stage of the protocol are termed "so-far-uncorrupted." However, once the adaptive adversary corrupts a node $\mathcal{P}_i$, that node $\mathcal{P}_i$ becomes fully controlled by the adaptive adversary and can act maliciously. A node is considered honest if it has never been corrupted by the adaptive adversary. Specifically, if a "so-far-uncorrupted" node $\mathcal{P}_i$ sent a message to $\mathcal{P}_j$ and then got corrupted by $\mathcal{A}$ before it was delivered, we allow the adversary to retract this message, preventing its delivery. Security against such an adversary is also known as strongly adaptive security in the literature [2], and previous MVBA works [4, 42] consider the same security.

Throughout this paper, we concentrate on asynchronous networks, where no assumptions are made about the timing of message transmissions. Moreover, the adversary can intentionally delay messages but must eventually deliver all messages sent among honest nodes. We do not require a PKI setup and do not use digital signatures in any form, aligning with the unauthenticated setting. The only cryptographic tool employed in our scheme is a collision-resistant hash function.

3.2 Goal: hash-based asynchronous multi-valued validated Byzantine agreement

Our goal is to design an efficient multi-valued validated Byzantine agreement (MVBA) protocol [4, 17, 42] in the setting we described above. We recall the definition of MVBA in the following.

Definition 3.1. In an MVBA protocol involving an external global predicate function denoted as Predicate : $\{0,1\}^\ell \to \{1,0\}$, each honest node has its input value that conforms to the predefined global function Predicate. The objective is to generate a unanimous output value from these n inputs, ensuring that the resulting output also adheres to the predetermined global function Predicate. Formally, the protocol strives to attain the following properties, with all but negligible probability:

- **Termination.** If each honest node $\mathcal{P}_i$ takes an input value v_i such that Predicate$(v_i) = 1$, then the protocol ensures that every honest node outputs a value v.
- **External-Validity.** If an honest node $\mathcal{P}_i$ outputs a value v, it guarantees that Predicate$(v) = 1$.
- **Agreement.** If one honest node $\mathcal{P}_i$ outputs v and another honest node $\mathcal{P}_j$ outputs v', it guarantees that v is equal to v', i.e., $v = v'$.

"Quality" was introduced by Abraham et al. [4]. It indicates that the probability of the output value is determined by the adversary. If this probability is less than 1, it can prevent the adversary from entirely determining the output. In this paper, our focus is on the situation where $n \geq kf + 1$. In such cases, we introduce an optimal quality property that achieves the upper bound for the probability that the output value was provided by an adversary node.

(1) **Quality.** If an honest node output an value v, then it is ensured that the probability of v being input by the adversary is at most $\frac{f}{n-f}$ [35].

4 PRELIMINARIES

In this section, we introduce the definitions of several fundamental building blocks that are employed in our paper. For simplicity, we provide a concise overview of their high-level abstractions along with their formal definitions.

Collision-resistant Hash Function. A cryptographic collision-resistant hash function ensures that a computationally limited adversary cannot find two distinct inputs that produce the same hash value, except for a negligible probability. The size of hash value is counted as λ throughout this paper.

Erasure code scheme. The (k, n)-erasure code scheme [12] consists of two deterministic algorithms, referred to as Enc and Dec. The Enc algorithm takes a vector $\mathbf{v} = (v_1, \cdots, v_k)$ consisting of k data fragments and maps it into a vector $\mathbf{m} = (m_1, \cdots, m_n)$ containing n code fragments. Crucially, the Dec algorithm allows for the reconstruction of $\mathbf{v}$ using any set of k elements from the code vector $\mathbf{m}$. Throughout the paper, we consider a $(f + 1, n)$-erasure code scheme, and we employ the terms fragment and codeword interchangeably when the context is unambiguous.

Vector commitment (VC). A VC scheme is specified as a tuple of algorithms denoted as (VCom, Open, VerifyOpen). The VCom algorithm produces a commitment vc for an input vector $\mathbf{m}$. When provided with both (m_i, i) and vc, the Open algorithm generates a succinct proof π_i. This proof is designed to verify that the element m_i corresponds to the i-th committed element in the vector $\mathbf{m}$. The position proof can be verified by the VerifyOpen algorithm. Specifically, given m_i, i, the commitment vc, and an opening proof π_i, it outputs 0/1 to determine the validity of the proof.

Remark: In this VC scheme, the hiding property is not required, and the VCom algorithm is deterministic. To implement the VC protocol, we consider using a Merkle tree based on hash functions,

as described in [43]. In this instantiation, the size of commitment vc is $O(\lambda)$ bits, and the openness proof π is $O(\lambda \log n)$ bits in size.

Common coin & **Election**. In our protocol design, we follow the approach of prior works [10, 22, 44] by assuming the existence of common coins—a concept introduced by Rabin in [51]. This common coin provides an unpredictable and unbiased random value that is shared among all nodes in the network. We model this common coin as an oracle, denoted as $random()$, which all nodes can query using a string `event`. To prevent the adversary from preemptively knowing the coin values of honest nodes, we impose the condition that $random()$ responds to a query with `event` only when at least $f+1$ nodes have previously queried $random()$ with the same `event`. This requirement ensures that at least one non-faulty node has requested the coin.

The common coin is employed in the random leader election protocol, denoted as Election$[id]$ [4], which is designed to output a uniformly sampled index $\ell \in [n]$.

Reliable broadcast (RBC**).** In RBC [16], there exists a sender whose goal is to broadcast a value to all nodes. More formally, an RBC protocol satisfies the following properties:

- *Totality.* If an honest node outputs v, then all honest nodes output v.
- *Agreement.* If any two honest nodes output v and v' respectively, then $v = v'$.
- *Validity.* If the sender is honest and inputs v, then all honest nodes output v.

Asynchronous binary agreement (ABA**).** In an ABA protocol, as defined in [44], honest nodes provide a single bit as input and output a common bit value $b \in \{0, 1\}$ that is the input of at least one honest node. Formally, an ABA protocol aims to fulfill the following properties, with all but negligible probability:

- **Termination**. If all honest nodes input a bit, either 0 or 1, then every honest node outputs a bit $b \in \{0, 1\}$.
- **Agreement**. If any two honest nodes output b and b' receptively, then $b = b'$.
- **Validity**. If any honest node outputs a bit $b \in \{0, 1\}$, then at least one honest node had b as its input.

Note: Throughout this paper, we always assume that randomness is generated using the $random()$ function for the ABA protocol.

Multi-valued Byzantine agreement (MBA**).** In ABA, the input values are restricted to either 0 or 1. In contrast, in MBA, honest nodes provide input values $v_i \in \{0, 1\}^\ell \cup \{\perp\}$, where the values are not limited to binary values $\{0, 1\}$. Formally, MBA satisfies the following properties except with negligible probability:

- **Termination.** If all honest nodes input a value v_i, then every honest node output a value v.
- **Agreement.** If any two honest nodes output v and v' receptively, then $v = v'$.
- **Weak Validity.** If all honest nodes input the same value v, then all honest nodes output v.

Besides the above conventional properties of an MBA, we further require it to satisfy the following *non-intrusion validity*, which was introduced in [9] and named in [45].

- **Non-intrusion.** If one honest node outputs v and $v \neq \perp$, then v is the input of some honest node.

Notations: The notation $\Pi[\mathsf{ID}]$ is employed to denote an instance of the protocol Π with the identifier ID. Additionally, the notation $y \leftarrow \Pi[\mathsf{ID}](x)$ signifies the action of invoking $\Pi[\mathsf{ID}]$ with input x and obtaining y as the output. We sometimes use $[k]$ to represent the integers from 1 to k, for some positive integer k.

5 ASYNCHRONOUS MULTI-VALUED VALIDATED BYZANTINE AGREEMENT

In this section, we introduce our hash-based MVBA protocol, designed to withstand adaptive adversaries and denoted as HMVBA. We presume the availability of collision-resistant hash functions and require $n \geq 5f + 1$ participants. The HMVBA protocol achieves an optimal time complexity of $O(1)$ and an optimal message complexity of $O(n^2)$. Additionally, it achieves optimal communication complexity, specifically $O(n\ell)$ when $\ell \geq \lambda n \log n$, where ℓ represents the size of the input values.

5.1 Overview of the HMVBA protocol

As we discussed in Introduction, our HMVBA follows an informal paradigm which we call "Disseminate-Elect-Agree". Now we turn to explain how HMVBA realizes each part of the framework. The workflow of our HMVBA is delineated in Figure 1.

To attain the optimal communication complexity of $O(n\ell)$, we let each node disseminate their input via an erasure-code-based *Dispersal phase*, rather than through simply multicasting. The Merkle tree is utilized to help the network identify the correct codewords. Specifically, as illustrated in Figure 1, when an honest node receives a valid value v, it computes the codewords $\{m_1, m_2, \cdots, m_n\}$ using the deterministic Enc algorithm and the vector commitment vc via the VCom algorithm. Subsequently, it generates the corresponded opening π_j for node $\mathcal{P}_j$ and sends a Diff message to $\mathcal{P}_j$, containing $(\mathsf{vc}, m_j, \pi_j)$. Upon receiving a valid codeword, i.e., the codeword corresponding to the received vector commitment, $\mathcal{P}_j$ sends an Echo back to the sender. If an honest node receives $n - f$ Echo messages from distinct nodes, it implies that its codeword has been received by at least $n - f$ distinct nodes. In this case, it multicasts a Done message to all nodes. Once an honest node receives $n - f$ Done messages from distinct nodes, indicating that at least $n - f$ distinct nodes have completed their dispersal, it multicasts a Finish message to all. Upon receiving $n - f$ Finish messages from distinct nodes, it attempts to output a value.

Election can be realized by the underlying common coin. After the sender $\mathcal{P}_s$ is chosen by Election, the network starts to recast the input value of $\mathcal{P}_s$ and enters the "Agree" part, trying to agree on the input of $\mathcal{P}_s$. Particularly, if any honest node $\mathcal{P}_i$ receives a valid Diff message from $\mathcal{P}_s$, $\mathcal{P}_i$ multicasts it to all nodes through a Value message. Otherwise, it will multicast a Value message that carries some $\perp$. Upon receiving $n - f$ Value messages from distinct nodes, it obtains a value M_i and checks if M_i is valid. At this stage, if $\mathcal{P}_s$ was honest before Election, all honest nodes should have the same M_i. Otherwise, different nodes may see different valid M_i. Then, the nodes run a MBA to agree on the vector commitment of M_i. If the output of MBA is not $\perp$, all nodes output the message w.r.t. the agreed commitment value. In this case, the non-intrusion security of MBA ensures all honest nodes can eventually obtain the

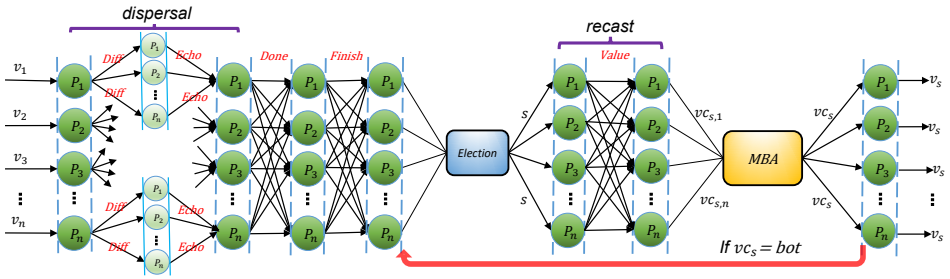


Figure 1: The execution flow of our HMVBA

corresponding message. Otherwise, the protocol is repeated from the Election phase.

5.2 Details of the HMVBA protocol

Our HMVBA protocol is detailed in Algorithm 1. Below is a detailed description of the process of the HMVBA protocol:

(1) *Dispersal phase* (lines 1-16). Nodes disperse their input value v through Diff messages; each Diff message includes a vector commitment, a codeword, and a position proof. When a node receives a valid Diff message from a sender for the first time, it responses an Echo message to the sender. Once a node receives $n-f$ Echo messages from distinct nodes, it informs all nodes that its dispersal is completed via Done messages. When a node receives $n-f$ Done messages from distinct nodes, it will send a Finish message to all nodes. If an honest node receives $f+1$ Finish messages from distinct nodes, it will also send a Finish message to all nodes if it has not done so already.

(2) Election *& Recast phase* (lines 17-33). Once a node receives $n-f$ Finish messages from distinct nodes, it initiates a common coin protocol Election to randomly select a leader node $\mathcal{P}_s$. Nodes exchange the Diff messages received from the elected node and attempt to reconstruct a value. Specifically, upon receiving the result $\mathcal{P}_s$ from Election, each node $\mathcal{P}_i$ checks if it has previously received a valid Diff message from the sender $\mathcal{P}_s$. If $\mathcal{P}_i$ has received it, $\mathcal{P}_i$ multicasts (Value, $k, \mathcal{P}_s$, vc, m_i, π_i). If $\mathcal{P}_i$ has not received a valid message, it multicasts (Value, $k, \mathcal{P}_s, \perp, \perp, \perp$). Honest nodes wait for $n-f$ Value messages from distinct nodes. If there are at least $n-3f \geq 2f+1$ messages carrying the same vc, each node randomly selects $f+1$ messages from these $2f+1$ messages and tries to decode the $f+1$ codewords to generate an output value M_i. If the value M_i satisfies the condition Predicate$(M_i) = 1$, it will set VCom_i equal to the vector commitment VCom of the value M_i. Otherwise, it will set VCom_i equal to $\perp$.

(3) *MBA phase* (lines 34-43). All honest nodes invoke MBA with the vector commitment vc_i as input. Suppose MBA returns a value vc′. If $\mathsf{vc}' \neq \perp$ and vc′ is the input of MBA, then output the M_i generated in the recast phase. If $\mathsf{vc}' \neq \perp$ and vc′ is not the input of MBA, then wait for $f+1$ valid Value messages from distinct nodes such that $|store[\mathsf{vc}']| = f+1$, and then decode it to output M_i. If $\mathsf{vc}' = \perp$, repeat the Election process until an externally valid value is obtained.

5.3 Security and Complexity analysis

We now prove that Algorithm 1 satisfies the properties of MVBA as defined in Definition 3.1, as established in the following theorem.

Theorem 5.1. *Assuming the underlying hash function is collision-resistant and the underlying* MBA *satisfies termination, weak-validity, agreement, and non-intrusion security, and aided by a common coin, our* HMVBA *in Algorithm 1 achieves the security properties of termination, external-validity, agreement, and quality (cf. Definition 3.1) with all but negligible probability. This holds against any adaptive and computationally bounded adversary corrupting up to f among $n \geq 5f+1$ nodes.*

Proof outline. Recall that our HMVBA adheres to the "Disseminate-Elect-Agree" paradigm, as discussed in the Introduction. Intuitively, the security proof commences by demonstrating that our design indeed fulfills the expected properties of each part, especially the dissemination part, which informally requires at least a constant fraction of honest inputs that can be received or recasted by the network. Subsequently, building on the intermediate guarantees, we establish the individual properties.

First, in Lemma 5.2, we establish that if a "so-far-uncorrupted" node $\mathcal{P}_s$ successfully disperses its input (i.e., multicasts (Done, 1)), then all honest nodes can recover its original input value M, even if the node later becomes corrupted by an adaptive adversary. This property is non-trivial, particularly in the context of both an asynchronous network and adaptive corruption. When $\mathcal{P}_s$ multicasts Done, at least $n-f$ nodes have received fragments of message M from $\mathcal{P}_s$, with at least $n-2f$ of them being honest nodes. Meanwhile, there are up to f honest nodes may not have received any fragments. Subsequently, an adaptive adversary can later corrupt $\mathcal{P}_s$ and send incorrect fragments (not corresponding to the fragments in M) to these f nodes that have not received the correct fragments. In doing so, the adversary can provide up to $2f$ incorrect fragments in the recasting phase. To successfully reconstruct the original message M, it is necessary to receive at least $4f+1$ correct fragments, which is why we need to require $n \geq 5f+1$.

Then, in Lemma 5.3, we show that when an honest node initiates the leader election, a sufficient number of "so-far-uncorrupted" nodes have already completed their dispersal, and every honest node can enter the leader election phase. This is crucial for ensuring that the network can reconstruct a valid value from the selected node with high probability and that the protocol can progress.

Building on the intermediate results established by Lemma 5.2 and 5.3, we proceed to prove termination in Lemma 5.4, external

Algorithm 1 HMVBA protocol with external Predicate, for each party $\mathcal{P}_i$: $n \geq 5f + 1$

let $S[i] \leftarrow \perp$ for $i \in [n]$, $store \leftarrow \{\}$, $flag \leftarrow 0$, $abandons \leftarrow 0$ ▷ Dispersal

1: **upon** receiving an input value v s.t. Predicate$(v) = 1$ **do**
2: $\boldsymbol{m} := \{m_1, \cdots, m_n\} \leftarrow \text{Enc}(n, f, v)$, where v is parsed as a $f + 1$ vector
3: $\text{vc} \leftarrow \text{VCom}(\boldsymbol{m})$;
4: **for** each $j \in [n]$ **do**
5: $\pi_j \leftarrow \text{Open}(\text{vc}, m_j, j)$
6: **send** $(\text{Diff}, \text{vc}, m_j, \pi_j)$ to $\mathcal{P}_j$
7: **upon** receiving $(\text{Diff}, \text{vc}, m_i, \pi_i)$ from $\mathcal{P}_j$ for the first time **do**
8: **if** $abandons = 0$ and $\text{VerifyOpen}(\text{vc}, m_i, i, \pi_i) = 1$ **then**
9: $S[j] \leftarrow (j, \text{vc}, m_i, \pi_i)$ ▷ store fragment
10: **send** $(\text{Echo}, 1)$ to $\mathcal{P}_j$
11: **upon** receiving $(\text{Echo}, 1)$ from $n - f$ nodes **do**
12: **multicast** $(\text{Done}, 1)$
13: **upon** receiving $(\text{Done}, 1)$ from $n - f$ nodes **do**
14: **multicast** $(\text{Finish}, 1)$
15: **upon** receiving $(\text{Finish}, 1)$ from $f + 1$ nodes **do**
16: **multicast** $(\text{Finish}, 1)$ if it has not yet been sent
17: **upon** receiving $(\text{Finish}, 1)$ from $n - f$ nodes **do**
18: $abandons \leftarrow 1$ ▷ abandon all Dispersal
19: **for** each $k \in \{1, 2, 3, \dots\}$ **do**
20: $s \leftarrow \text{Election}[k]$ ▷ threshold $f + 1$
21: **if** $S[s] := (j, \text{vc}, m_i, \pi_i)$ **then**
22: **multicast** $(\text{Value}, k, s, \text{vc}, m_i, \pi_i)$
23: **else**
24: **multicast** $(\text{Value}, k, s, \perp, \perp, \perp)$ ▷ $S[s] = \perp$
25: **upon** receiving $(\text{Value}, k, s, \text{vc}, m_j, \pi_j)$ from $\mathcal{P}_j$ for the first time **do**
26: **if** $m_j \neq \perp$ and $\text{VerifyOpen}(\text{vc}, m_j, j, \pi_j) = 1$ **then**
27: $store[\text{vc}] \leftarrow store[\text{vc}] \cup (j, m_j)$
28: **upon** $|store[\text{vc}]| = n - 3f$ **do**
▷ pick $f + 1$ elements in $store[\text{vc}]$ when decoding
29: $M_i \leftarrow \text{Dec}(store[\text{vc}])$
30: **if** $\text{Predicate}(M_i) = 1$ **then**
31: $\text{vc}_i \leftarrow \text{vc}$, $flag \leftarrow 1$
32: **upon** receiving Value from $n - f$ nodes and $flag = 0$ **do**
33: $\text{vc}_i \leftarrow \perp$; $flag \leftarrow 1$
34: **upon** $flag = 1$ **do**
35: $\text{vc}' \leftarrow \text{MBA}[k](\text{vc}_i)$ ▷ see Algorithm 2
36: **if** $\text{vc}' \neq \perp$ **then**
37: **if** $\text{vc}' = \text{vc}_i$ **then**
38: output M_i
39: **else**
40: **wait** until $|store[\text{vc}']| = f + 1$
41: output $M_i \leftarrow \text{Dec}(store[\text{vc}'])$
42: **else**
43: $M_i \leftarrow \perp$; $flag \leftarrow 0$; $store \leftarrow \{\}$

validity in Lemma 5.5, agreement in Lemma 5.6, and quality in Lemma 5.7, respectively.

Detailed proofs. In the following, we present detailed proofs for all lemmas mentioned above.

Lemma 5.2. *Suppose a "so-far-uncorrupted" node $\mathcal{P}_s$ has a valid input value v_s and multicasts $(\text{Done}, 1)$, then if all honest nodes recast $\mathcal{P}_s$'s input, then all honest nodes can recast the same valid value M, and it holds that $M = v_s$.*

Proof. According to the pseudocode of Algorithm 1, if a "so-far-uncorrupted" node $\mathcal{P}_s$ multicasts $(\text{Done}, 1)$, then it implies that at least $n - 2f$ honest nodes $\mathcal{P}_j$ received $(\text{Diff}, \text{vc}, m_j, \pi_j)$, which are the correct fragments of v_s. If all honest nodes recast $\mathcal{P}_s$'s input, then in this case, at least $n - 2f$ honest nodes will multicast $(\text{Value}, k, s, \text{vc}, m_i, \pi_i)$ to all, and $m_i \neq \perp$.

Since all honest nodes need to wait for $n-f$ Value messages from distinct nodes, then all honest nodes must see at least $n-3f$ Value messages that carry the same vc and valid fragment m_i. Given that there are at most $2f$ incorrect fragments when facing an adaptive adversary and $n \geq 5f + 1$, it is impossible for one honest node to see $n - 3f$ Value messages carrying the same vc, while another honest node sees $n - 3f$ Value messages carrying a different vc' where $\text{vc} \neq \text{vc}'$. Therefore, all honest nodes recast the same value based on the same vector commitment vc. Hence, if the sender is "so-far-uncorrupted" at the moment of multicasting $(\text{Done}, 1)$, the recast value will be the same as the original encoded value due to the deterministic nature of the decoding algorithm (Dec), resulting in $M = v_s$. □

Lemma 5.3. *If an honest node invokes* $\text{Election}[k]$, *then at least $n-2f$ distinct "so-far-uncorrupted" nodes have completed their dispersal and multicast $(\text{Done}, 1)$, and all honest nodes have also invoked* $\text{Election}[k]$.

Proof. Suppose an honest node $\mathcal{P}_i$ invokes $\text{Election}[k]$ for $k = 1$. This implies that $\mathcal{P}_i$ has received $n - f$ $(\text{Finish}, 1)$ messages. Furthermore, it implies that at least $n - 2f$ "so-far-uncorrupted" nodes have multicast $(\text{Finish}, 1)$ messages. Firstly, this indicates that at least one honest node has received $n-f$ $(\text{Done}, 1)$ messages from distinct nodes, meaning that at least $n - 2f$ distinct "so-far-uncorrupted" nodes have completed their dispersal and multicast $(\text{Done}, 1)$. Secondly, it also implies that all honest nodes can receive at least $f + 1$ $(\text{Finish}, 1)$ messages, resulting all honest nodes will multicast $(\text{Finish}, 1)$ messages, ensuring that all honest nodes can receive at least $n - f$ $(\text{Finish}, 1)$ messages, therefore, all honest nodes will also invoke $\text{Election}[k]$.

For $k > 1$, it is evident that at least $n - 2f$ distinct honest nodes have completed their dispersal and multicast $(\text{Done}, 1)$ based on the analysis for $k = 1$. If an honest node $\mathcal{P}_i$ invokes $\text{Election}[k]$, it implies that $\text{MBA}[k - 1]$ output an invalid value. According to the agreement property of MBA, all honest nodes will output the same value, resulting in all honest nodes also invoking $\text{Election}[k]$. □

Lemma 5.4. **Termination**. *If each honest node $\mathcal{P}_i$ takes an input value v_i such that* $\text{Predicate}(v_i) = 1$, *then the protocol ensures that every honest node outputs a value v.*

Proof. According to Algorithm 1, all honest nodes start with externally valid values. If no honest nodes abandon all dispersal, then

all messages sent among honest nodes have been delivered. Therefore, any honest node can know that at least $n-f$ dispersal processes have completed successfully and they will invoke Election$[k]$. If any honest node abandons all dispersal, it means that this node has seen $n-f$ (Finish, 1) messages. Hence, this honest node will invoke Election$[k]$ to elect a random number s. From Lemma 5.3, we know that at least $n-2f$ distinct "so-far-uncorrupted" nodes have completed their dispersal and multicast (Done, 1) messages, and all honest nodes have also invoked Election$[k]$.

Let Q be the identifier set of these $n-2f$ "so-far-uncorrupted" nodes. Next, let us consider two following cases:

- **Case 1:** If $s \in Q$, then according to Lemma 5.2, all honest nodes can recast the same value M, and this value satisfies the global Predicate, leading to all honest nodes inputting the same vector commitment vc to MBA. Following the validity of MBA, all honest nodes output the vc in this round; after that, all honest nodes immediately output the corresponding value M.
- **Case 2:** If $s \notin Q$, then it is possible that the honest nodes could recast different values. However, thanks to the agreement and termination properties of MBA, all honest nodes will eventually terminate and output the same value from MBA. If the output value does not satisfy Predicate, they will repeat the election process.

For case 1, the probability that $\mathcal{P}_s$ is "so-far-uncorrupted" and has completed its dispersal before the Election$[k]$ returns its output is at least $p = (n-2f)/n$. Let the event E_k represent that the protocol does not terminate when MBA$[k]$ has been invoked. Therefore, the probability of the event E_k, denoted as $\Pr[E_k]$, is bounded by $(1-p)^k$. When $n \geq 5f+1$, it is clear that $\Pr[E_k] \leq (1-p)^k \to 0$ as $k \to \infty$, indicating that the protocol eventually halts. Moreover, let K be the random variable representing when MBA$[K]$ outputs a valid value that satisfies the Predicate. So $\mathbb{E}[K] \leq \sum_{K=1}^{\infty} K(1-p)^{K-1}p = 1/p$, indicating that the protocol terminates in expected constant time. □

Lemma 5.5. **External-Validity**. *If an honest node $\mathcal{P}_i$ outputs a value v, it guarantees that* Predicate$(v) = 1$.

Proof. According to Algorithm 1, if an honest node produces an output value v, according to the code, all honest nodes receive the same vector commitment vc of value v from the output of MBA. Following the non-intrusion property of MBA, the vc is the input of some honest node, implying that the corresponding value v satisfies the condition Predicate$(v) = 1$. Consequently, external validity is inherently guaranteed. □

Lemma 5.6. **Agreement**. *If one honest node $\mathcal{P}_i$ outputs v and another honest node $\mathcal{P}_j$ outputs v', it guarantees that v is equal to v', i.e., $v = v'$.*

Proof. As outlined in Algorithm 1, when an honest node generates an output value v, it signifies that v is the corresponding value of the vector commitment vc, which is the outcome of the MBA. With the assurance of the agreement and termination properties of MBA, all other honest nodes also reach a consensus to output the same vector commitment vc. Due to the deterministic nature of the Dec algorithm, all honest nodes output the same value v. □

Lemma 5.7. **Quality**. *If an honest node outputs v, the probability that v was proposed by the adversary is at most 1/4 when facing an adaptive adversary with $n \geq 5f+1$.*

Proof. Due to Lemma 5.3, whenever an honest node initiates Election, it implies that at least $n-2f$ distinct "so-far-uncorrupted" nodes have successfully completed their dispersal and multicast (Done, 1) messages. Furthermore, if any honest node invokes election protocol Election$[k]$, all other honest nodes will eventually invoke Election$[k]$ as well. Let's assume that Election$[k]$ returns s. If the sender $\mathcal{P}_s$ is 'so-far-uncorrupted" and multicasts (Done, 1) before any honest nodes invoke Election$[k]$, according to Lemma 5.2, all honest nodes can collectively reconstruct the same valid value M. Following the code, after that, all honest nodes compute a vector commitment of value M via the deterministic Dec algorithm and obtain the same vc. They take the vc as the input for MBA$[k]$. Following the validity of MBA, the MBA returns vc to all honest nodes, resulting in all honest nodes outputting the corresponding value M.

In any other case, if MBA$[k]$ outputs an invalid value, all nodes will proceed to the next iteration and engage in Election$[k+1]$. However, if MBA$[k]$ returns a valid value, all honest nodes will promptly output this value as the final result. Therefore, we proceed to consider the following three worst-case scenarios:

1. If the sender $\mathcal{P}_s$ has not completed his dispersal yet, even if the adversary corrupts the sender, at least $n-2f$ honest nodes have already abandoned all Diff messages. Consequently, the adversary can influence at most $2f$ Diff messages among all nodes.
   When $n \geq 5f+1$, an honest node attempts to recast a value only if they have received at least $n-f$ fragments, and at least $n-3f > 2f$ of which are not $\perp$ and correspond to the same vector commitment. It is apparent that the adversary cannot ensure that all honest nodes recast a value determined by the adversary. Therefore, the worst-case scenario is that MBA returns $\perp$, and we proceed to repeat the Election process. The probability of this case occurring is at most f/n.
2. If the sender $\mathcal{P}_s$ has completed his dispersal, and the sender's input was determined by the adversary, the probability of this case happening is at most f/n.
3. If the sender $\mathcal{P}_s$ has completed his dispersal, and the sender's input was not determined by the adversary, the probability of this case occurring is at least $(n-2f)/n$.

The probability of deciding an output value v proposed by the adversary is at most $\sum_{k=1}^{\infty}(f/n)^k$, which is 1/4 when $n \geq 5f+1$. □

Efficiency analysis. We have the following efficiency for the HMVBA in Algorithm 1.

Theorem 5.8. *The* HMVBA *implemented in Algorithm 1 has constant running time, $O(n^2)$ message complexity, and $O(n\ell+\lambda n^2 \log n)$ communication complexity, where ℓ is the input size.*

Proof. As depicted in Algorithm 1, the cost breakdown of the HMVBA protocol can be summarized into three distinct phases: (i) the dispersal phase: During this phase, each node sends $O(n)$ messages, including Diff, Echo, Done, and Finish messages. (ii)

Election & Recast phase: This phase involves invoking the Election protocol, and an all-to-all multicast of VALUE messages takes place. (iii) the MBA phase: This phase is initiated by invoking the MBA protocol instance.

Assuming the use of the MBA protocol [44], where the time complexity of MBA is $O(1)$, message complexity is $O(n^2)$, and communication complexity is $O(n^2\ell)$, the overall complexities of the HMVBA protocol can be summarized as follows:

- **Time complexity:** Following Lemma 5.4 and considering that the time complexity of the MBA blackbox is $O(1)$, the overall time complexity of MVBA is $O(1)$.
- **Message complexity:** In the dispersal phase, which incurs $O(n^2)$ messages, where each node sends a total of $O(n)$ messages. In the Election & Recast phase, beyond a single common coin invocation, the recast phase requires one all-to-all multicast, incurring $O(n^2)$ messages. In the MBA phase, there is only one MBA instance. Moreover, the Election & Recast phase and the MBA phase are expected to be repeated two times. To summarize, the overall message complexity of the HMVBA protocol is $O(n^2)$.
- **Communication complexity:** In the dispersal phase, the communication complexity incurs $O(n\ell+\lambda n^2 \log n)$ bit complexity. In the Election & Recast phase, there is one common coin and one all-to-all multicast, incurring $O(n\ell+\lambda n^2 \log n)$ bits of communication. In the MBA phase, the communication complexity is $O(\lambda n^2)$ bits. Moreover, the time complexity shows the protocol will terminate in constant time. Hence, the overall communication complexity of the HMVBA protocol is $O(n\ell + \lambda n^2 \log n)$.

□

6 ASYNCHRONOUS MULTI-VALUED BYZANTINE AGREEMENT WITH WEAK VAILIDTY

In Section 5, we introduce HMVBA which uses MBA as its core component. While the MBA presented in [45] exhibits optimal asymptotic performance, it was originally designed for optimal tolerance resilience, introducing a complicated process throughout the entire protocol and necessitating a higher number of required rounds. In this section, given that this paper operates under $n \geq 5f+1$, we propose a novel MVBA that significantly outperforms the one in [45]. Specifically, our MBA reduces the all-to-all broadcast step by at least four compared to the MBA in [45]. Simultaneously, it maintains the same asymptotic complexity performance.

To design the MBA protocol, we begin by assuming the existence of an ABA protocol against an adaptive adversary. For the instantiation of these protocols, we adopt the IT ABA protocol [44], which does not necessitate any cryptographic assumptions beyond the common coin. Given that the coin is derived from a common source, effectively instantiating the coin with the provided oracle, this IT ABA protocol [44] is also robust against adaptive adversaries.

6.1 Overview of the MBA protocol

Our construction is primarily founded on the following: If all honest nodes input the same value v, then all honest nodes output v. Therefore, we initially perform a "filter" procedure to retain only the "good cases", where the good case implies that at least the majority of nodes have the same input. Following the filtering process, all honest nodes invoke ABA to determine whether to output a non-$\perp$ value based on the output of ABA. If ABA outputs 1, it indicates the existence of a good case. To achieve agreement, we need to ensure that all honest nodes output the same value under the good case. To maintain weak validity, we also need to guarantee that the output value aligns with the input of the majority of nodes.

It is crucial to ensure that when all honest nodes have the same input, they can collectively identify the occurrence of a good case. This leads to all honest nodes inputting 1 when invoking ABA. Following the validity property of ABA, this ensures that ABA must output 1. Consequently, all honest nodes output the same non-$\perp$ value, i.e., they output the input value. In our MBA, if all honest nodes input the same value v, where vc $\neq 0$, it is evident that they will multicast (ECHO, v). As a result, all honest nodes can receive (ECHO, v) messages from at least $n-2f$ distinct nodes. Hence, all honest nodes will invoke ABA with input 1. Following the validity property of ABA, all honest nodes will output 1 from ABA.

Even if not all honest nodes input the same values, the protocol will not get stuck. They will either multicast (ECHO, v') or (ECHO, 0), ensuring they still have input for ABA. Hence, in any case, all honest nodes always have a value as input for ABA. Following the termination and agreement properties of ABA, all honest nodes produce the same output value. If ABA outputs 1, then all honest nodes wait for $f+1$ identical (ECHO, v') messages, where $v' \neq 0$. We can ensure that all honest nodes receive $f+1$ identical (ECHO, v') messages from distinct nodes, where $v' \neq 0$. Additionally, we can guarantee that if two different honest nodes $\mathcal{P}_i$ and $\mathcal{P}_j$ multicast (ECHO, v') and (ECHO, v''), respectively, moreover, if $v' \neq 0$ and $v'' \neq 0$, then $v' = v''$. Hence, all nodes can output the same value v.

6.2 Details of the MBA protocol

In this section, we present a comprehensive description of the construction of our MBA protocol. The detailed procedure for MBA is outlined in Algorithm 1. The protocol consists of three distinct logical phases, following these sequential steps:

(1) *Filter phase* (lines 1-7). When a node $\mathcal{P}_i$ receives an input value v, it multicasts (VALUE, v) to all nodes. All nodes wait for VALUE messages from $n-f$ distinct nodes. Once the node $\mathcal{P}_i$ receives (VALUE, v') from $n-2f$ distinct nodes, where $v' \neq 0$, it multicasts (ECHO, v') message to all nodes. This (ECHO, v') message serves as the signal that at least $n-2f$ honest nodes have the same input. Otherwise, it multicasts (ECHO, 0) message to all nodes.
(2) *ABA phase* (lines 8-13). For any node $\mathcal{P}_i$, upon receiving ECHO messages from $n-f$ distinct nodes, if at least $n-2f$ ECHO messages carry the same non-zero value v', then it will consider 1 as the input for ABA; otherwise, it takes 0 as its input.
(3) *Output phase* (lines 14-18). In this phase, all nodes output a value based on the output result of ABA. If the output of

Algorithm 2 MBA protocol, for each party $\mathcal{P}_i$: $n \geq 5f + 1$

let $flag \leftarrow 0$
1: **upon** receiving an input value v **do**
2: **multicast** (Value, v)
3: **upon** receiving (Value, $*$) from $n - f$ nodes **do**
4: **if** (Value, v') was received from $n - 2f$ nodes **then**
5: **multicast** (Echo, v')
6: **else**
7: **multicast** (Echo, 0)
8: **upon** receiving (Echo, $*$) from $n - f$ nodes **do**
9: **if** (Echo, v') was received from $n - 2f$ nodes and $v' \neq 0$ **then**
10: $flag \leftarrow 1$
11: **else**
12: $flag \leftarrow 0$
13: **wait** $b \leftarrow$ ABA($flag$)
14: **if** $b = 0$ **then**
15: **output** $\perp$
16: **if** $b = 1$ **then**
17: **wait** until receiving (Echo, v') from f+1 nodes and $v' \neq 0$
18: **output** v'

ABA is 0, then all nodes output $\perp$. If the output of ABA is 1, then all nodes output a non-$\perp$ value.
Specifically, if ABA outputs 1, then all honest nodes wait until receiving (Echo, v') from $f + 1$ distinct nodes, where $v' \neq 0$. Then, all nodes output value v'.

6.3 Security and Complexity analysis

Security analysis. We establish the security of MBA in the following theorem.

Theorem 6.1. *Assuming the underlying* ABA *satisfies termination, validity, and agreement, our* MBA *in Algorithm 2 achieves the security properties of termination, weak validity, agreement, and non-intrusion security with all but negligible probability. This holds against any adaptive and* ***computationally unbounded*** *adversary corrupting up to f among $n \geq 5f + 1$ nodes.*

Sketched proof. If one honest node $\mathcal{P}_i$ received (Value, v') from $n-2f$ distinct nodes, and another honest node $\mathcal{P}_j$ also received $n - 2f$ (Value, v'') messages from distinct nodes. Since $n \geq 5f + 1$, at least $n - 3f \geq 2f + 1$ honest nodes multicast (Value, v') and (Value, v''). If $v' \neq v''$, then it implies one honest node multicast two different messages (Value, v') and (Value, v''), which is a clear contradiction. As a result, if two different honest nodes $\mathcal{P}_i$ and $\mathcal{P}_j$ multicast (Echo, v') and (Echo, v''), respectively, and if $v' \neq 0$ and $v'' \neq 0$, then $v' = v''$.

Suppose that all honest nodes have an input value, resulting in all honest nodes receiving $n - f$ Value messages. Consequently, all honest nodes can also multicast a Echo message, from which it follows that all honest nodes will have an input value for ABA. According to the termination of ABA, all honest nodes receive an output from ABA. If ABA outputs 1, according to the validity of ABA, at least one honest node's input is 1, implying that this honest node received at least $n - 2f$ (Echo, v') messages and $v' \neq 0$. Due to $n \geq 5f + 1$, at least $n - 3f \geq 2f + 1$ honest nodes multicast (Echo, v'). Again, because of the uniqueness of (Echo, v') among honest nodes, all honest nodes can learn the same v. Hence, all honest nodes have the same output. □

Detailed proofs. Below, we provide detailed proofs for the sketched proof mentioned above.

Lemma 6.2. *Suppose one honest node $\mathcal{P}_i$ multicasts (Echo, v') and another honest node $\mathcal{P}_j$ multicasts (Echo, v''). If $v' \neq 0$ and $v'' \neq 0$, then $v' = v''$.*

Proof. If one honest node $\mathcal{P}_i$ multicasts (Echo, v'), where $v' \neq 0$, then, by the code, $\mathcal{P}_i$ received (Value, v') from $n - 2f$ distinct nodes. Similarly, if another honest node $\mathcal{P}_i$ also multicasts (Echo, v''), where $v'' \neq 0$, then it also implies $\mathcal{P}_i$ received (Value, v'') from $n - 2f$ distinct nodes. Since there are at most f malicious nodes, hence at least $n - 3f \geq 2f + 1$ honest nodes multicast (Value, v') and (Value, v''). If $v' \neq v''$, based on the assumption $n \geq 5f + 1$, it implies that one honest node multicasts two different messages (Value, v') and (Value, v''), leading to a contradiction. Therefore, $v' = v''$. □

Lemma 6.3. *Suppose all honest nodes have an initial input value v, then all honest nodes have an input value $flag$ for* ABA.

Proof. If all honest nodes have an initial input value v, then all honest nodes will multicast a Value message. Based on the network assumption that all messages sent by honest nodes will eventually be received by all honest nodes, all honest nodes can receive at least $n - f$ Value messages. This triggers the multicast of a Echo message, which further implies that all honest nodes can receive at least $n - f$ Echo messages. As a result, all honest nodes will invoke ABA with an input value $flag$. □

Lemma 6.4. *Suppose* ABA *outputs 1, then all honest nodes will output the same value.*

Proof. If ABA outputs 1, according to the validity property of ABA, at least one honest node's input is 1. By the code, this honest node has received at least $n - 2f$ (Echo, v') messages, and $v' \neq 0$. Due to $n \geq 5f + 1$, at least $n - 3f \geq 2f + 1$ honest nodes multicast (Echo, v'), where $v' \neq 0$. According to Lemma 6.2, if $v' \neq 0$, then the honest nodes will multicast the same (Echo, v') message. Hence, all honest nodes will output the same value v'. □

Lemma 6.5. **Termination**. *If all honest nodes have an initial input value v, then the protocol ensures that every honest node will output a value v.*

Proof. If all honest nodes have an initial input value v, according to Lemma 6.3, all honest nodes have an input value $flag$ for ABA. According to the termination and agreement properties, all honest nodes can output the same value from ABA. Hence, if the output value is 0, then all honest nodes output $\perp$. In contrast, if the output value is 1, following Lemma 6.4, all honest nodes have the same output. □

Lemma 6.6. **Weak-Validity**. *If all honest nodes input the same value v, then all honest nodes output v.*

Proof. If all honest nodes input the same value v, then all honest nodes will multicast the same value v through Value and Echo messages. By the code, all honest nodes have 1 as the input for ABA. According to the validity of ABA, ABA will return 1 to all. Because (Echo, v) messages sent by honest nodes are the same, where $v \neq 0$, hence, all honest nodes output v. □

Lemma 6.7. **Agreement**. *If any two honest nodes output v and v' receptively, then $v = v'$.*

Proof. As outlined in Algorithm 2, if an honest node generates an output value v, it means that ABA has output a value. According to the agreement property of ABA, all honest nodes have the same output from ABA. If ABA outputs 0, then all honest nodes output $\perp$. If ABA outputs 1, following Lemma 6.4, all honest nodes also output the same value v. □

Lemma 6.8. **Non-intrusion.** *If one honest node outputs v and $v \neq \perp$, then v is the input of some honest node.*

Proof. If an honest node $\mathcal{P}_i$ outputs v and $v \neq \perp$, according to the code, it implies that ABA returns 1. This also indicates that $\mathcal{P}_i$ received $f + 1$ (Echo, v) messages from distinct nodes. Since there are at most f malicious nodes, it follows that at least one honest node multicasts (Echo, v). Additionally, this implies that the honest node received at least $n - 2f$ (Value, v) messages from distinct nodes. Given that $n \geq 5f + 1$, it follows that v must be the input of some honest node. □

Theorem 6.9. *In the IT model, Algorithm 2 achieves asynchronous* MBA *among n parties in the presence of an adaptive adversary controlling up to $f < n/5$ nodes.*

Proof. Lemma 6.5,6.6, 6.7 and 6.8 complete the proof. □

Efficiency of MBA. we have the following efficiency for MBA in Algorithm 2.

Theorem 6.10. *The communication complexity is $O(n\ell + \lambda n^2)$ bits per transaction, which is optimal when $\ell \geq \lambda n$, where ℓ is the size of the transaction and λ is the security parameter.*

Proof. The cost breakdown of Algorithm 2, as depicted in Section 6.2, can be summarized into three distinct phases: the Filter phase, the ABA phase, and the Output phase. The overall complexities of the MBA protocol can be summarized as follows:

- **Time complexity:** The protocol terminates in expected constant running time, as supported by Lemma 6.5, and this is further ensured by the fact that the time complexity of ABA is $O(1)$.
- **Message complexity:** In the filter phase, incurring $O(n^2)$ messages, each node sends a total of n messages. In the ABA phase, beyond common coin invocation, it needs to exchange $O(n^2)$ messages. The output phase occurs without any message exchange. To summarize, the overall message complexity of the MBA protocol is $O(n^2)$.
- **Communication complexity:** In the filter phase, the communication complexity incurs $O(n^2\ell)$ bit complexity due to the size of value being ℓ. In the ABA phase, it costs $O(n^2)$ bit complexity. The output phase incurs no communication cost. Consequently, the overall communication complexity of the MBA protocol is $O(n^2\ell)$.

□

7 IMPLEMENTATION AND EVALUATIONS

We implemented and evaluated the performance of HMVBA in a Wide Area Network (WAN) setting. Along the way, we conducted systematic comparisons with several typical MVBA protocols, including sMVBA* [36] and the hash-based FIN-MVBA proposed in [29]. Specifically, sMVBA* utilizes sMVBA [36] as the underlying MVBA to instantiate the Dumbo-MVBA* [42], while replacing the threshold BLS signature with the catenation of $n - f$ ECDSA signatures. Remark that we use the ECDSA-based version rather than the standard BLS-based version, as the former has been shown to be concretely more efficient than the latter in almost all cases in[36]. Besides, we emphasize that our experiments did not employ any network layer optimizations.

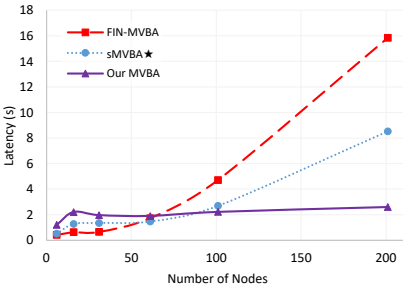
Test environment. The experiments are conducted among AWS EC2 `t2.medium` instances evenly distributed in 13 AWS regions: N. Virginia, Ohio, N. California, Oregon, Canada Central, Mumbai, Tokyo, Seoul, Osaka, Singapore, Sydney, Ireland, and São Paulo. Each `t2.medium` instance is equipped with 2 Intel Xeon processors of speed up to 3.4GHz Turbo CPU clock and 4 GB memory. It also provides a baseline bandwidth of 256 Mbps and a peak bandwidth of 1024 Mbps.

The one-shot agreement is evaluated with different input sizes (L) and network sizes (N). Each input L consists of B batches of transactions. In our experiments, a single transaction is represented as a string of 250 bytes, which approximates the size of a typical Bitcoin transaction with one input and two outputs. Hence, we express L as $250 \times B$, where the batch size B ranges from 1 to 7×10^3 in our experiments. Moreover, we conducted tests with six different network sizes, specifically $N = 6$, 16, 31, 61, 101, and 201. The parameter f, denoting the number of corrupted nodes, is set as the optimal threshold. It is consistently set to the maximum integer that satisfies the condition $N \geq 5f + 1$, where N represents the network size.

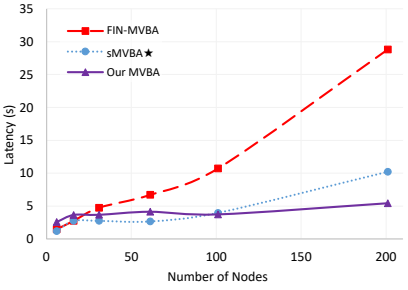
In our test, all N nodes take an input and participate in the instantiation simultaneously. The latency for each node is defined as the time difference between receiving an input and outputting all transactions. To measure the latency among all nodes, we utilize the 20% trimmed mean. Initially, we obtained ten latency measurements by conducting repeated assessments ten times for each specific test configuration under a fixed network size and a fixed input size. Subsequently, the average latency is determined by applying the 20% trimmed mean to the collected latency values.

Implementation details. All asynchronous protocols are written as multi-process Python 3 programs, and are developed upon the open source code of *Dumbo_NG* [31] [4]. The pair-to-pair communication channels between every two nodes are set up using unauthenticated TCP sockets. The Python program initiates three processes on each node, comprising a protocol process responsible for protocol execution, a client process managing data transmission,

[4] https://github.com/yylluu/Dumbo_NG

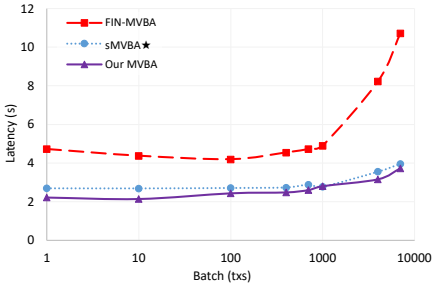


(a) Basic Latency $B = 1$

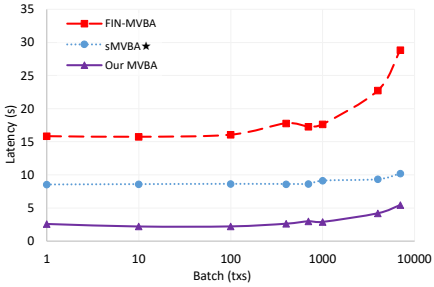


(b) Batch Size $B = 7000$

Latency of Several MVBA Protocols with Fixed Batch Sizes

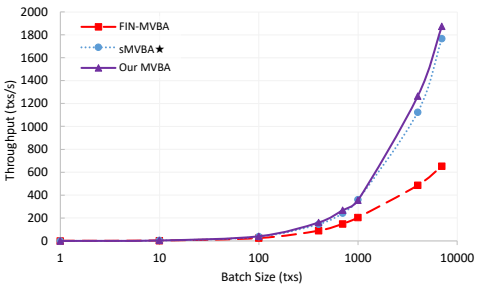


(c) Network Size $N = 101$

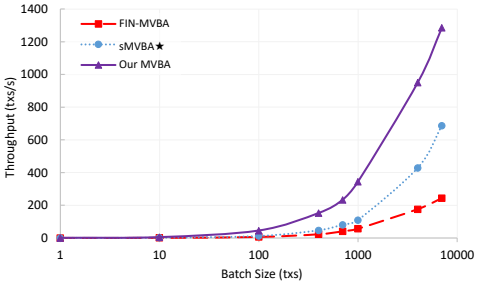


(d) Network Size $N = 201$

Latency of Several MVBA Protocols with Fixed Scale Sizes



(e) Network Size $N = 101$



(f) Network Size $N = 201$

Throughput of Several MVBA Protocols with Fixed Scale Sizes

Figure 2: Performance of Several MVBA Protocols

and a server process managing data reception. Concurrent tasks within each process are managed by *gevent* [5] Python library.

We adapt the source code of sMVBA* [42] in *Dumbo_NG* [31] [6] to fit our testing framework. All components in FIN-MVBA [29] are implemented from scratch, including its WRBC and RABA constructions. For our own HMVBA, we adopted the ABA component from *ADKG* [7] in [27]. Everything else is newly implemented, except for the basic cryptographic components present in *Dumbo_NG*, such as erasure code, Merkle tree, and ECDSA signature. Common coins and leader election protocols are needed by all tested MVBA protocols. They are implemented by hashing the session ID, which is shared across the entire network. As a result, the implementation of these sub-protocols does not introduce any fairness issues.

Highlighting Results. Key information from our experiments are:

- Our HMVBA will consistently outperform the other two MVBA protocols when the input size is fixed and the scale increases beyond $N = 101$. This holds even when the input size is reasonably large, e.g., $B = 7000$;
- With the scale of $N = 201$, our HMVBA outperforms the other two MVBA protocols for all tested input sizes, ranging from batch size $B = 1$ to 7000.
- Our HMVBA enjoys a better throughput-latency trade-off with a reasonably large scale, e.g., $N \geq 101$.

[5] http://www.gevent.org/
[6] https://github.com/yylluu/Dumbo_NG/tree/main/dumbomvbastar
[7] https://github.com/sourav1547/adkg/blob/adkg/adkg/broadcast/binaryagreement.py

Table 2: Improvements of basic latency ($B = 1$)

Scale (N)	Basic Latency (milisec)			Our Improvement	
	FIN-MVBA	sMVBA*	Our HMVBA	FIN-MVBA	sMVBA*
101	4721	2692	2215	↓53%	↓18%
201	15834	8522	2592	↓84%	↓70%

Latency Improvements with Fixed Input Sizes. Our HMVBA achieves a substantial reduction in basic latency (e.g., batch size $B = 1$) compared to FIN-MVBA by 53% and 84% when $N = 101$ and 202, respectively. Similarly, it also demonstrates latency reductions of 18% and 70% compared to sMVBA* under the same conditions of $N = 101$ and 202. Refer to Table 2 & Fig. 2(a). This outcome aligns with our asymptotic comparison, as our MVBA effectively reduces the $O(\lambda n^3)$ term in the communication cost of FIN-MVBA.

For larger input sizes (e.g., batch size $B = 7000$), our HMVBA continues to outperform FIN-MVBA and achieves latency reductions starting from an even smaller scale (e.g., $N = 31$). This reflects our asymptotic improvement by reducing the $O(\ell n^2)$ term in the communication cost of FIN-MVBA to $O(\ell n)$. In contrast, since sMVBA* already benefits from the $O(\ell n)$ term, our HMVBA does not lower the latency of sMVBA* on a smaller scale. However, it still can reduce latency under the same conditions that enable it to outperform sMVBA* in basic latency comparisons. This can be indicative of the impact of computational complexity, as observed when $N = 101$ and 202. Refer to Fig. 2(b).

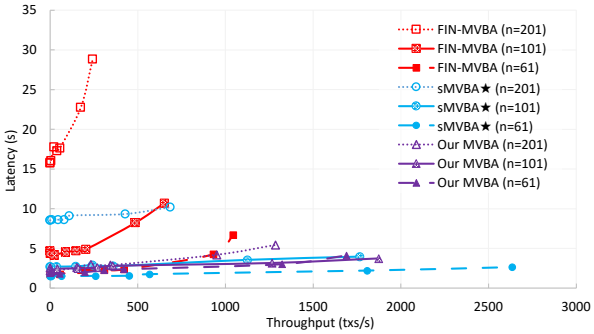


Figure 3: Latency vs. Throughput of Several MVBA Protocols

Latency & Throughput Improvements with Fixed Network Size. To illustrate our advantages in handling various input sizes in a fixed-size network, we conducted tests on all three MVBA protocols with a fixed two network sizes of $N = 101$ and 201, and various input batch sizes ranging from $B = 1$ to 7000. As depicted in Fig. 2(c) & Fig. 2(d), our HMVBA consistently reduces latency across all tested input sizes. Furthermore, with fixed network sizes of $N = 101$ and 201, our HMVBA demonstrates greater throughput across all tested input sizes, cf Fig. 2(e) & Fig. 2(f). This matches our advantages in improving latency illustrated in Fig. 2(c) & Fig. 2(d).

Better Latency-throughput Trade-off. In Fig. 3, the throughput-latency trade-offs in the three MVBA protocols are demonstrated at three reasonable network scales: $N = 61$, 101 and 201. These curves do not exhibit an L-shape, suggesting that the tested MVBA protocols have not reached the network-bound, and the peak throughput is yet to be observed. Despite not exhausting their bandwidths, our HMVBA has already demonstrated a significant advantage over FIN-MVBA in all three scales, and sMVBA* in the larger two scales. Specifically, when $N = 61$, our HMVBA achieves a maximum throughput of 1690 transactions per second under the current testing condition, whereas FIN-MVBA has never reached this throughput under the same testing conditions. On the other hand, our HMVBA is yet to outpace sMVBA* at this scale. When $N = 101$, our HMVBA exhibits slightly better latency compared to sMVBA* at the same throughput. Moreover, its latency is approximately half of that observed in FIN-MVBA. Furthermore, as the network size N increases to 201, the latency gap between our HMVBA and sMVBA* significantly widens. In this case, sMVBA* experiences a latency that is twice that of our HMVBA. Additionally, FIN-MVBA incurs an even greater latency, surpassing four times the latency of our HMVBA to achieve the same throughput.

ACKNOWLEDGEMENT

This work was supported in part by research awards from Ethereum Foundation, Protocol Labs, Stellar Development Foundation, and SOAR Prize from the University of Sydney.

REFERENCES

[1] Ittai Abraham, Gilad Asharov, Arpita Patra, and Gilad Stern. 2023. Perfectly Secure Asynchronous Agreement on a Core Set in Constant Expected Time. *Cryptology ePrint Archive* (2023).
[2] Ittai Abraham, TH Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. 2019. Communication complexity of byzantine agreement, revisited. In *Proc. ACM PODC 2019*. ACM, 317–326.
[3] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Reaching consensus for asynchronous distributed key generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. 363–373.
[4] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. In *PODC*. ACM, 337–346.
[5] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. 2017. Ligero: Lightweight Sublinear Arguments Without a Trusted Setup. In *CCS*. ACM, 2087–2104.
[6] Thomas Attema, Ronald Cramer, and Matthieu Rambaud. 2021. Compressed $\varSigma $-Protocols for Bilinear Group Arithmetic Circuits and Application to Logarithmic Transparent Threshold Signatures. In *ASIACRYPT (4) (Lecture Notes in Computer Science, Vol. 13093)*. Springer, 526–556.
[7] Renas Bacho, Daniel Collins, Chen-Da Liu-Zhang, and Julian Loss. 2023. Network-Agnostic Security Comes (Almost) for Free in DKG and MPC. In *CRYPTO (1) (Lecture Notes in Computer Science, Vol. 14081)*. Springer, 71–106.
[8] Renas Bacho and Julian Loss. 2022. On the Adaptive Security of the Threshold BLS Signature Scheme. In *CCS*. ACM, 193–207.
[9] Michael Ben-Or and Ran El-Yaniv. 2003. Resilient-optimal interactive consistency in constant time. *Distributed Computing* 16, 4 (2003), 249–262.
[10] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. 1994. Asynchronous secure computations with optimal resilience. In *Proc. ACM PODC 1994*. ACM, 183–192.
[11] Daniel J. Bernstein, Daira Hopwood, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, Louiza Papachristodoulou, Michael Schneider, Peter Schwabe, and Zooko Wilcox-O'Hearn. 2015. SPHINCS: Practical Stateless Hash-Based Signatures. In *EUROCRYPT (1) (Lecture Notes in Computer Science, Vol. 9056)*. Springer, 368–397.
[12] Richard E Blahut. 1983. *Theory and practice of error control codes*. Addison-Wesley.
[13] Erica Blum, Jonathan Katz, Julian Loss, Kartik Nayak, and Simon Ochsenreither. 2023. Abraxas: Throughput-Efficient Hybrid Asynchronous Consensus. In *CCS*. ACM, 519–533.
[14] Alexandra Boldyreva. 2003. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. In *Public Key Cryptography (Lecture Notes in Computer Science, Vol. 2567)*. Springer, 31–46.
[15] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short Signatures from the Weil Pairing. In *ASIACRYPT (Lecture Notes in Computer Science, Vol. 2248)*. Springer, 514–532.
[16] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Information and Computation* 75, 2 (1987), 130–143.
[17] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*. Springer, 524–541.
[18] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology* 18, 3 (2005), 219–246.
[19] Jinyuan Chen. 2021. Optimal error-free multi-valued byzantine agreement. *Leibniz international proceedings in informatics* (2021).
[20] Ran Cohen, Pouyan Forghani, Juan A. Garay, Rutvik Patel, and Vassilis Zikas. 2023. Concurrent Asynchronous Byzantine Agreement in Expected-Constant Rounds, Revisited. In *TCC (4) (Lecture Notes in Computer Science, Vol. 14372)*. Springer, 422–451.
[21] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. 2006. From consensus to atomic broadcast: Time-free Byzantine-resistant protocols without signatures. *Comput. J.* 49, 1 (2006), 82–96.
[22] Tyler Crain. 2020. Two More Algorithms for Randomized Signature-Free Asynchronous Binary Byzantine Consensus with $t < n/3$ and $O(n^2)$ Messages and $O(1)$ Round Expected Termination. *arXiv preprint arXiv:2002.08765* (2020).
[23] Xiaohai Dai, Bolin Zhang, Hai Jin, and Ling Ren. 2023. ParBFT: Faster Asynchronous BFT Consensus with a Parallel Optimistic Path. In *CCS*. ACM, 504–518.
[24] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. 2023. Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In *32nd USENIX Security Symposium (USENIX Security 23)*. 5359–5376.
[25] Sourav Das, Zhuolun Xiang, and Ling Ren. 2021. Asynchronous Data Dissemination and its Applications. In *CCS*. ACM, 2705–2721.
[26] Sourav Das, Zhuolun Xiang, Alin Tomescu, Alexander Spiegelman, Benny Pinkas, and Ling Ren. 2023. A New Paradigm for Verifiable Secret Sharing. *IACR Cryptol. ePrint Arch.* (2023), 1196.
[27] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2518–2534.
[28] Bernardo David, Bernardo Magri, Christian Matt, Jesper Buus Nielsen, and Daniel Tschudi. 2022. GearBox: Optimal-size Shard Committees by Leveraging the Safety-Liveness Dichotomy. In *CCS*. ACM, 683–696.

[29] Sisi Duan, Xin Wang, and Haibin Zhang. 2023. FIN: Practical Signature-Free Asynchronous Common Subset in Constant Time. In *CCS*. ACM, 815–829.

[30] Michael J Fischer, Nancy A Lynch, and Michael S Paterson. 1985. Impossibility of Distributed Consensus with One Faulty Process. *JACM* 32, 2 (1985), 374–382.

[31] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Dumbo-ng: Fast asynchronous bft consensus with throughput-oblivious latency. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 1187–1201.

[32] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Efficient Asynchronous Byzantine Agreement without Private Setups. In *ICDCS*. IEEE, 246–257.

[33] Rati Gelashvili, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman, and Zhuolun Xiang. 2022. Jolteon and ditto: Network-adaptive efficient consensus with asynchronous fallback. In *International Conference on Financial Cryptography and Data Security*. Springer, 296–315.

[34] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *SOSP*. ACM, 51–68.

[35] Guy Goren, Yoram Moses, and Alexander Spiegelman. 2022. Probabilistic indistinguishability and the quality of validity in byzantine agreement. In *Proceedings of the 4th ACM Conference on Advances in Financial Technologies*. 111–125.

[36] Bingyong Guo, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2022. Speeding Dumbo: Pushing Asynchronous BFT Closer to Practice. In *NDSS*. 1–18.

[37] Binyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster Asynchronous BFT Protocols. In *Proc. ACM CCS 2020*. ACM.

[38] Bin Hu, Zongyang Zhang, Han Chen, You Zhou, Huazu Jiang, and Jianwei Liu. 2022. DyCAPS: Asynchronous Proactive Secret Sharing for Dynamic Committees. *Cryptology ePrint Archive* (2022).

[39] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. 2020. Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures. In *ACM CCS*. 1751–1767.

[40] Fan Li and Jinyuan Chen. 2021. Communication-efficient signature-free asynchronous Byzantine agreement. In *2021 IEEE International Symposium on Information Theory (ISIT)*. IEEE, 2864–2869.

[41] Yuan Lu, Zhenliang Lu, and Qiang Tang. 2022. Bolt-Dumbo Transformer: Asynchronous Consensus As Fast As the Pipelined BFT. In *CCS*. ACM, 2159–2173.

[42] Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. 2020. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th symposium on principles of distributed computing*. 129–138.

[43] Ralph C Merkle. 1987. A digital signature based on a conventional encryption function. In *Eurocrypt 1987*. Springer, 369–378.

[44] Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. 2015. Signature-Free Asynchronous Binary Byzantine Consensus with t < n/3, O(n2) Messages, and O(1) Expected Time. *J. ACM* 62, 4 (2015), 31:1–31:21.

[45] Achour Mostéfaoui and Michel Raynal. 2017. Signature-free asynchronous Byzantine systems: from multivalued to binary consensus with t< n/3, $O(n^2)$ messages, and constant time. *Acta Informatica* 54, 5 (2017), 501–520.

[46] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. 2020. Improved Extension Protocols for Byzantine Broadcast and Agreement. In *DISC (LIPIcs, Vol. 179)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 28:1–28:17.

[47] Gil Neiger. 1994. Distributed consensus revisited. *Information processing letters* 49, 4 (1994), 195–201.

[48] Arpita Patra. 2011. Error-free multi-valued broadcast and Byzantine agreement with optimal communication complexity. In *International Conference On Principles Of Distributed Systems*. Springer, 34–49.

[49] Arpita Patra and C Pandu Rangan. 2011. Communication optimal multi-valued asynchronous byzantine agreement with optimal resilience. In *Information Theoretic Security: 5th International Conference, ICITS 2011, Amsterdam, The Netherlands, May 21-24, 2011. Proceedings 5*. Springer, 206–226.

[50] Tian Qiu and Qiang Tang. 2023. Predicate Aggregate Signatures and Applications. In *ASIACRYPT (2) (Lecture Notes in Computer Science, Vol. 14439)*. Springer, 279–312.

[51] Michael O Rabin. 1983. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. IEEE, 403–409.

[52] Xiao Sui and Sisi Duan. 2023. Signature-Free Atomic Broadcast with Optimal $O(n^2)$ Messages and O(1) Expected Time. *IACR Cryptol. ePrint Arch.* (2023), 1549.

[53] Thomas Yurek, Zhuolun Xiang, Yu Xia, and Andrew Miller. 2023. Long Live The Honey Badger: Robust Asynchronous DPSS and its Applications. In *32nd USENIX Security Symposium (USENIX Security 23)*. 5413–5430.

[54] Haibin Zhang and Sisi Duan. 2022. Pace: Fully parallelizable bft from reproposable byzantine agreement. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 3151–3164.

A "ELECT-LOCK-VOTE" MVBA WITH STATIC SECURITY

As a warm-up towards efficient hash-based MVBA, we first examine candidates with static security.

There is a trivial folklore idea for feasibility: we can first use the common coin to elect a committee with at least 2/3 members are honest, then run any MVBA protocol within the committee and decide on a value, and finally have committee members disseminate the value to the whole population. However, this approach is only useful for super large networks, since the committee size needs to be considerably large (see [28]) to ensure an honest-majority with a high probability. Moreover, this approach has to trade resilience as well, since the ratio of honest parties in the committee is always smaller than the ratio in the whole population.

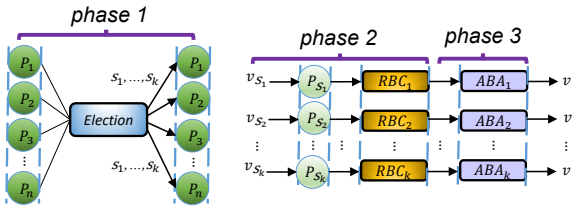


Figure 4: The execution flow of ELV with static security

We observe a simple static construction that is free of all drawbacks of the above folklore one. Recall that the bottleneck of LEV paradigm is "locking" n input messages, which appears to be unnecessary as we only need *one* output in MVBA. In a naive attempt, we may just sample a random party (using a common coin), let it lock its input, and have all parties vote on the status of the provided input. In doing so, parties can decide on a valid output when the selected party is honest. However, when the selected party is malicious, it may choose to not initiate the "lock" phase in the first place, such that the protocol cannot proceed, causing a termination issue. Fortunately, we can address this by sampling κ (which is the statistical security parameter, usually just a few of tens) inputs to be locked, such that at least one is from an honest party with an overwhelming probability. We call the construction "Elect-Lock-Vote" (ELV). For completeness, a brief description of the ELV construction is included below, which is based on RBC and ABA, with $O(\log \kappa)$ rounds and $O(\kappa \ell n + \kappa \lambda n^2)$ communication. Its execution flow is outlined in Figure 4.

- **Step 1:** Run leader election to decide on κ distinct random values $\{s_1, \cdots, s_\kappa\}$ from $[n]$.
- **Step 2:** Each $\mathcal{P}_{s_i}$ for $i \in [\kappa]$ uses RBC to broadcast its input to the whole network.
- **Step 3:** For each $\mathcal{P}_i$, $i \in [n]$, when it receives a valid value from the i-th RBC for $i \in [\kappa]$, it inputs 1 to the i-th ABA instances. Upon receiving 1 from any of the κ ABA instances, it inputs 0 to all other ABA instances if it hasn't provided input. It waits all κ ABA to be completed, determines the smallest $i^* \in [\kappa]$ such that i^*-th ABA outputs 1, and decides on the value received from i^*-th RBC.

Security Analysis. The agreement property directly stems from the agreement of the underlying RBC and ABA. External validity

and termination rely on the fact that at least one honest node, denoted as $\mathcal{P}_{i^*}$, will be elected with high probability. Specifically, $\mathcal{P}_{i^*}$ broadcasts its valid input v_{i^*} to the network, ensuring all honest nodes eventually receive it. Upon receiving v_{i^*}, an honest node should vote for it, unless the network has already decided on another valid value, satisfying the external validity condition. Consequently, the i^*-th ABA instance outputs 1, leading to the termination of other ABA instances, as all honest nodes provide inputs to them after receiving the output of the i^*-th ABA. This ensures the termination of the whole protocol.

Unfortunately, an adaptive adversary can corrupt all κ selected parties, making above construction completely fail. One may wonder whether it's possible to hide the committee (e.g, using VRF-based sampling technique [34]) to mitigate the adaptive corruption. Besides that VRF is already hard to obtain from hash, the best we can hope is a weak version of adaptive security which puts non-standard restrictions on the adversary, as discussed by Abraham et al. in [2]. Throughout this paper, we focus on *strongly* adaptive security where adversary can even retract messages after an honest node sends it out (but not yet delivered).