OciorMVBA: Near-Optimal Error-Free Asynchronous MVBA

Jinyuan Chen
jinyuan@ocior.com

Abstract

In this work, we propose an error-free, information-theoretically secure, asynchronous multi-valued validated Byzantine agreement (MVBA) protocol, called OciorMVBA. This protocol achieves MVBA consensus on a message $\boldsymbol{w}$ with expected $O(n|\boldsymbol{w}|\log n + n^2 \log q)$ communication bits, expected $O(n^2)$ messages, and expected $O(\log n)$ rounds, under optimal resilience $n \geq 3t+1$ in an n-node network, where up to t nodes may be dishonest. Here, q denotes the alphabet size of the error correction code used in the protocol. When error correction codes with a constant alphabet size (e.g., Expander Codes) are used, q becomes a constant. An MVBA protocol that guarantees all required properties without relying on any cryptographic assumptions, such as signatures or hashing, except for the common coin assumption, is said to be *information-theoretically secure (IT secure)*. Under the common coin assumption, an MVBA protocol that guarantees all required properties in *all* executions is said to be *error-free*.

We also propose another error-free, IT-secure, asynchronous MVBA protocol, called OciorMVBArr. This protocol achieves MVBA consensus with expected $O(n|\boldsymbol{w}|+n^2\log n)$ communication bits, expected $O(1)$ rounds, and expected $O(1)$ common coins, under a relaxed resilience (RR) of $n \geq 5t+1$. Additionally, we propose a hash-based asynchronous MVBA protocol, called OciorMVBAh. This protocol achieves MVBA consensus with expected $O(n|\boldsymbol{w}| + \kappa n^3)$ bits, expected $O(1)$ rounds, and expected $O(1)$ common coins, given $n \geq 3t+1$.

I. Introduction

Multi-valued validated Byzantine agreement (MVBA), introduced by Cachin et al. in 2001 [1], is one of the key building blocks for distributed systems and cryptography. In MVBA, distributed nodes proposes their input values and seek to agree on one of the proposed values, ensuring that the agreed value satisfies a predefined Predicate function (referred to as *External Validity*). MVBA is a variant of Byzantine agreement (BA), which was proposed by Pease, Shostak and Lamport in 1980 [2]. In BA, if all honest nodes input the same value $\boldsymbol{w}$, it is required that every honest node eventually outputs $\boldsymbol{w}$ (referred to as *Validity*). As one can see, MVBA's External Validity is different from BA's Validity.

In this work, we focus on the design of *asynchronous* MVBA protocols. The seminal work by Fischer, Lynch, and Paterson [3] reveals that no deterministic MVBA protocol can exist in the asynchronous setting. Therefore, any asynchronous MVBA protocol must incorporate randomness. A common approach to designing such a protocol is to create a deterministic algorithm supplemented by common coins, which provide the necessary randomness.

Additionally, we primarily focus on the design of error-free, information-theoretically secure (IT secure), asynchronous MVBA protocols. An MVBA protocol that guarantees all required properties without relying on any cryptographic assumptions, such as signatures or hashing, except for the common coin assumption, is said to be *IT secure*. Under the common coin assumption, an MVBA protocol that guarantees all required properties in *all* executions is said to be *error-free*.

Specifically, we propose an error-free, IT secure, asynchronous MVBA protocol, called OciorMVBA. This protocol achieves MVBA consensus on a message $\boldsymbol{w}$ with expected $O(n|\boldsymbol{w}|\log n + n^2 \log q)$ communication bits, expected $O(n^2)$ messages, and expected $O(\log n)$ rounds, under optimal resilience $n \geq 3t+1$ in an n-node network, where up to t nodes may be dishonest. Here, q denotes the alphabet size of the error correction code used in the protocol. When error correction codes with a constant alphabet size (e.g., Expander Codes [4]) are used, q becomes a constant.

The design of OciorMVBA in this MVBA setting builds on the protocols COOL and OciorCOOL, originally designed for the BA setting [5]–[7]. Specifically, COOL and OciorCOOL introduced two

primitives: unique agreement (UA) and honest-majority distributed multicast (HMDM) [5]–[7]. COOL and OciorCOOL achieve the deterministic, error-free, IT secure, synchronous BA consensus with $O(n|\boldsymbol{w}| + nt\log q)$ communication bits and $O(t)$ rounds, under optimal resilience $n \geq 3t+1$. When error correction codes with a constant alphabet size are used, q becomes a constant, and consequently, COOL and OciorCOOL are optimal.

• UA: In UA, distributed nodes input their values and seek to decide on an output of the form $(\boldsymbol{w}, \mathrm{s}, \mathrm{v})$, where $\mathrm{s} \in \{0,1\}$ and $\mathrm{v} \in \{0,1\}$ denote a success indicator and a vote, respectively. UA requires that all honest nodes eventually output the same value $\boldsymbol{w}$ or a default value (*Unique Agreement*). Furthermore, if any honest node votes $\mathrm{v}=1$, then at least $t+1$ honest nodes eventually output $(\boldsymbol{w}, 1, *)$ for the same $\boldsymbol{w}$ (*Majority Unique Agreement*).

• HMDM: In HMDM, there are at least $t+1$ honest nodes acting as senders, multicasting a message to n nodes. HMDM requires that if every honest sender inputs the same message $\boldsymbol{w}$, then every honest node eventually outputs $\boldsymbol{w}$.

Our proposed OciorMVBA is a recursive protocol (see Fig. 1) that consists of the algorithms of strongly-honest-majority distributed multicast (SHMDM), reliable Byzantine agreement (RBA), asynchronous biased binary Byzantine agreement (ABBBA), and asynchronous binary BA (ABBA).

• RBA: Our RBA algorithm, called OciorRBA, is built from UA and HMDM. We note that the Unique Agreement and Majority Unique Agreement properties of UA provide ideal conditions for HMDM.

• SHMDM: SHMDM is slightly different from HMDM, as all honest nodes act as senders in SHMDM.

• ABBBA: This new primitive is introduced here and used as a building block in our protocols. In ABBBA, each honest node inputs a pair of binary numbers (a_1, a_2), for some $a_1, a_2 \in \{0,1\}$. One property is that if $t+1$ honest nodes input the second number as $a_2 = 1$, then any honest node that terminates outputs 1 (Biased Validity). Another property is that if an honest node outputs 1, then at least one honest node inputs $a_1 = 1$ or $a_2 = 1$ (Biased Integrity).

In this work, we also propose another error-free, IT-secure, asynchronous MVBA protocol, called OciorMVBArr. This protocol achieves MVBA consensus with expected $O(n|\boldsymbol{w}| + n^2\log n)$ communication bits, expected $O(1)$ rounds, and expected $O(1)$ common coins, under a relaxed resilience (RR) of $n \geq 5t+1$. Additionally, we propose a hash-based asynchronous MVBA protocol, called OciorMVBAh. This protocol achieves MVBA consensus with expected $O(n|\boldsymbol{w}| + \kappa n^3)$ bits, expected $O(1)$ rounds, and expected $O(1)$ common coins, under optimal resilience $n \geq 3t+1$.

The proposed OciorMVBA protocol is described in Algorithms 1-4 and Section II. The proposed OciorMVBArr protocol is described in Algorithms 5-7 and Section III. The proposed OciorMVBAh protocol is described in Algorithms 8-10 and Section IV. Table I provides a comparison between the proposed protocols and some other MVBA protocols. In the following subsection, we provide some definitions and primitives used in our protocols.

A. Primitives

Asynchronous network. We consider a network of n distributed nodes, where up to t of the nodes may be dishonest. Every pair of nodes is connected via a reliable and private communication channel. The network is considered to be *asynchronous*, i.e., the adversary can arbitrarily delay any message, but the messages sent between honest nodes will eventually arrive at their destinations.

Adaptive adversary. We consider an adaptive adversary, i.e., the adversary can corrupt any node at any time during the course of protocol execution, but at most t nodes in total can be controlled by adversary.

Information-theoretic protocol. A protocol that guarantees all required properties without relying on any cryptographic assumptions, such as signatures or hashing, except for the common coin assumption, is said to be *IT secure*. The proposed protocols OciorMVBA and OciorMVBArr are IT secure.

Signature-free protocol. Under the common coin assumption, a protocol that guarantees all required properties without relying on signature-based cryptographic assumptions is said to be *signature-free*. All of the the proposed protocols are signature-free.

TABLE I

COMPARISON BETWEEN THE PROPOSED PROTOCOLS AND SOME OTHER MVBA PROTOCOLS. HERE q DENOTES THE ALPHABET SIZE OF THE ERROR CORRECTION CODE USED IN THE PROPOSED PROTOCOLS. WHEN ERROR CORRECTION CODES WITH A CONSTANT ALPHABET SIZE (E.G., EXPANDER CODE [4]) ARE USED, q BECOMES A CONSTANT. κ IS A SECURITY PARAMETER. IN OciorMVBA, EACH NODE USES $O(\log n)$ COMMON COINS, BUT THESE COINS ARE ASSOCIATED WITH NETWORK SIZES OF $n/2, n/2^2, n/2^3, \cdots, n/2^{\log n}$, RESPECTIVELY. THIS IMPLIES THAT THE COMMUNICATION COST OF USING THESE COMMON COINS IS EQUIVALENT TO THAT OF A SCHEME USING $O(1)$ COMMON COINS ASSOCIATED WITH A NETWORK SIZE OF n.

Protocols	Resilience	Communication (Total Bits)	# Coin	Rounds	Cryptographic Assumption (Expect for Common Coin)
Cachin et al. [1]	$t < \frac{n}{3}$	$O(n^2\|\boldsymbol{w}\| + \kappa n^2 + n^3)$	$O(1)$	$O(1)$	Threshold Sig
Abraham et al. [8]	$t < \frac{n}{3}$	$O(n^2\|\boldsymbol{w}\| + \kappa n^2)$	$O(1)$	$O(1)$	Threshold Sig
Dumbo-MVBA [9]	$t < \frac{n}{3}$	$O(n\|\boldsymbol{w}\| + \kappa n^2)$	$O(1)$	$O(1)$	Threshold Sig
Duan et al. [10]	$t < \frac{n}{3}$	$O(n^2\|\boldsymbol{w}\| + \kappa n^3)$	$O(1)$	$O(1)$	Hash
Feng et al. [11]	$t < \frac{n}{5}$	$O(n\|\boldsymbol{w}\| + \kappa n^2 \log n)$	$O(1)$	$O(1)$	Hash
Komatovic et al. [12]	$t < \frac{n}{4}$	$O(n\|\boldsymbol{w}\| + \kappa n^2 \log n)$	$O(1)$	$O(1)$	Hash
Proposed OciorMVBAh	$t < \frac{n}{3}$	$O(n\|\boldsymbol{w}\| + \kappa n^3)$	$O(1)$	$O(1)$	Hash
Duan et al. [10]	$t < \frac{n}{3}$	$O(n^2\|\boldsymbol{w}\| + n^3 \log n)$	$O(1)$	$O(1)$	Non
Proposed OciorMVBArr	$t < \frac{n}{5}$	$O(n\|\boldsymbol{w}\| + n^2 \log n)$	$O(1)$	$O(1)$	Non
Proposed OciorMVBA	$t < \frac{n}{3}$	$O(n\|\boldsymbol{w}\| \log n + n^2 \log q)$	$O(\log n)$	$O(\log n)$	Non

Error-free protocol. Under the common coin assumption, a protocol that that guarantees all of the required properties in *all* executions is said to be *error-free*. The proposed protocols OciorMVBA and OciorMVBArr are error-free.

Definition 1 (**Multi-valued validated Byzantine agreement** (MVBA))**.** *In the* MVBA *problem, there is an external* Predicate *function* $\{0,1\}^* \rightarrow \{\text{true}, \text{false}\}$ *known to all nodes. In this problem, each honest node proposes its input value, ensuring that it satisfies the* Predicate *function to be true. The* MVBA *protocol guarantees the following properties:*

- **Agreement:** *If any two honest nodes output* $\boldsymbol{w}'$ *and* $\boldsymbol{w}''$*, respectively, then* $\boldsymbol{w}' = \boldsymbol{w}''$.
- **Termination:** *Every honest node eventually outputs a value and terminates.*
- **External validity:** *If an honest node outputs a value* $\boldsymbol{w}$*, then* $\text{Predicate}(\boldsymbol{w}) = \text{true}$.

Definition 2 (**Byzantine agreement** (BA))**.** *In the* BA *protocol, the distributed nodes seek to reach agreement on a common value. The* BA *protocol guarantees the following properties:*

- **Termination:** *If all honest nodes receive their inputs, then every honest node eventually outputs a value and terminates.*
- **Consistency:** *If any honest node outputs a value* $\boldsymbol{w}$*, then every honest node eventually outputs* $\boldsymbol{w}$.
- **Validity:** *If all honest nodes input the same value* $\boldsymbol{w}$*, then every honest node eventually outputs* $\boldsymbol{w}$.

Definition 3 (**Reliable broadcast** (RBC))**.** *In a reliable broadcast protocol, a leader inputs a value and broadcasts it to distributed nodes, satisfying the following conditions:*

- **Consistency:** *If any two honest nodes output* $\boldsymbol{w}'$ *and* $\boldsymbol{w}''$*, respectively, then* $\boldsymbol{w}' = \boldsymbol{w}''$.
- **Validity:** *If the leader is honest and inputs a value* $\boldsymbol{w}$*, then every honest node eventually outputs* $\boldsymbol{w}$.
- **Totality:** *If one honest node outputs a value, then every honest node eventually outputs a value.*

Definition 4 (**Reliable Byzantine agreement** (RBA))**.** RBA *is a variant of* RBC *problem and is a relaxed version of* BA *problem. The* RBA *protocol guarantees the following properties:*
Consistency: *If any two honest nodes output* $\boldsymbol{w}'$ *and* $\boldsymbol{w}''$*, respectively, then* $\boldsymbol{w}' = \boldsymbol{w}''$.
Validity: *If all honest node input the same value* $\boldsymbol{w}$*, then every honest node eventually outputs* $\boldsymbol{w}$.
Totality: *If one honest node outputs a value, then every honest node eventually outputs a value.*

Definition 5 (**Distributed multicast**). *In the problem of distributed multicast (*DM*), there exits a subset of nodes acting as senders multicasting the message over n nodes, where up to t nodes could be dishonest. Each node acting as an sender has an input message. A protocol is called as a* DM *protocol if the following property is guaranteed:*

- **Validity:** *If all honest senders input the same message $\boldsymbol{w}$, every honest node eventually outputs $\boldsymbol{w}$.*

Honest-majority distributed multicast *(*HMDM*, [5]–[7]): A* DM *problem is called as honest-majority* DM *if at least $t+1$ senders are honest.* HMDM *was used previously as a building block for* COOL *and OciorCOOL protocols [5]–[7].*

Strongly-honest-majority distributed multicast (SHMDM**):** *A* DM *problem is called as strongly-honest-majority* DM *if all honest nodes are acting as senders.*

Definition 6 (**Unique agreement** (UA, [5]–[7])). UA *is a variant of Byzantine agreement problem operated over n nodes, where up to t nodes may be dishonest. In a* UA *protocol, each node inputs an initial value and seeks to make an output taking the form as $(\boldsymbol{w}, \mathrm{s}, \mathrm{v})$, where $\mathrm{s} \in \{0,1\}$ is a success indicator and $\mathrm{v} \in \{0,1\}$ is a vote. The* UA *protocol guarantees the following properties:*

- **Unique Agreement:** *If any two honest nodes output $(\boldsymbol{w}', 1, *)$ and $(\boldsymbol{w}'', 1, *)$, respectively, then $\boldsymbol{w}' = \boldsymbol{w}''$.*
- **Majority Unique Agreement:** *If any honest node outputs $(*, *, 1)$, then at least $t+1$ honest nodes eventually output $(\boldsymbol{w}, 1, *)$ for the same $\boldsymbol{w}$.*
- **Validity:** *If all honest nodes input the same value $\boldsymbol{w}$, then all honest nodes eventually output $(\boldsymbol{w}, 1, 1)$.*

UA *was used previously as a building block for* COOL *and OciorCOOL protocols [5]–[7].*

Asynchronous complete information dispersal (ACID**).** We introduce a new primitive ACID. The goal of an ACID protocol is to disperse information over distributed nodes. Once a leader completes the dispersal of its proposed message, it is guaranteed that each honest node could retrieve the delivered message correctly from distributed nodes via a data retrieval scheme. Two ACID definitions are provided below: one for an ACID instance dispersing a message proposed by a leader, and the other one for a whole ACID protocol of running n parallel ACID instances.

Definition 7 (ACID instance). *An* ACID$[(\mathrm{ID}, i)]$ *protocol with an identity (ID, i), a message is proposed by P_i (i.e., the leader in this case) and is dispersed over n distributed nodes, for $i \in [1:n]$. An* ACID$[(\mathrm{ID}, i)]$ *protocol is complemented by a data retrieval protocol* DR$[(\mathrm{ID}, i)]$ *in which each node retrieves the message proposed by P_i from n distributed nodes. The* ACID$[(\mathrm{ID}, i)]$ *and* DR$[(\mathrm{ID}, i)]$ *protocols guarantee the following properties:*

- **Completeness:** *If P_i is honest, then P_i eventually completes the dispersal (ID, i).*
- **Availability:** *If P_i completes the dispersal for (ID, i), and all honest nodes start the data retrieval protocol for (ID, i), then each node eventually reconstructs some message.*
- **Consistency:** *If two honest nodes reconstruct messages $\boldsymbol{w}'$ and $\boldsymbol{w}''$ respectively for (ID, i), then $\boldsymbol{w}' = \boldsymbol{w}''$.*
- **Validity:** *If an honest P_i has proposed a message $\boldsymbol{w}$ for (ID, i) and an honest node reconstructs a message $\boldsymbol{w}'$ for (ID, i), then $\boldsymbol{w}' = \boldsymbol{w}$.*

Definition 8 (Parallel ACID instances). *An* ACID[ID] *protocol is a protocol involves running n parallel* ACID *instances, $\{\mathrm{ACID}[(\mathrm{ID}, i)]\}_{i=1}^{n}$, over n distributed nodes, where up to t of the nodes may be dishonest. For an* ACID[ID] *protocol, the following conditions must be satisfied:*

- **Termination:** *Every honest node eventually terminates.*
- **Integrity:** *If one honest node terminates, then there exists a set $\mathcal{I}^\star$ such that the following conditions hold: 1) $\mathcal{I}^\star \subseteq [1:n] \setminus \mathcal{F}$, where $\mathcal{F}$ denotes the set of indexes of all dishonest nodes; 2) $|\mathcal{I}^\star| \geq n - 2t$; and 3) for any $i \in \mathcal{I}^\star$, P_i has completed the dispersal* ACID$[(\mathrm{ID}, i)]$.

Definition 9 (**Asynchronous biased binary Byzantine agreement (**ABBBA**)**). *We introduce a new primitive called as* ABBBA. *In an* ABBBA *protocol, each honest node inputs a pair of binary numbers*

(a_1, a_2)*, for some* $a_1, a_2 \in \{0, 1\}$*. The honest nodes seek to reach an agreement on a common value* $a \in \{0, 1\}$*. An* ABBBA *protocol should satisfy the following properties:*

- **Conditional termination:** *Under an input condition—i.e., if one honest node inputs its second number as* $a_2 = 1$ *then at least* $t + 1$ *honest nodes input their first numbers as* $a_1 = 1$*—then every honest node eventually outputs a value and terminates.*
- **Biased validity:** *If* $t+1$ *honest nodes input the second number as* $a_2 = 1$*, then any honest node that terminates outputs* 1*.*
- **Biased integrity:** *If an honest node outputs* 1*, then at least one honest node inputs* $a_1 = 1$ *or* $a_2 = 1$*.*

Definition 10 (Common coin). *The seminal work by Fischer, Lynch, and Paterson in [3] reveals that no deterministic* MVBA *protocol can exist in the asynchronous setting. Therefore, any asynchronous* MVBA *protocol must incorporate randomness. A common approach to designing such a protocol is to create a deterministic algorithm supplemented by common coins, which provide the necessary randomness. Here, we assume the existence of a common coin protocol* $l \leftarrow \text{Election[id]}$ *associated with an identity* id*, which guarantees the following properties:*

- **Termination:** *If* $t+1$ *honest nodes activate* Election[id]*, then each honest node that activates it will output a common value* l*.*
- **Consistency:** *If any two honest nodes output* l' *and* l'' *from* Election[id]*, respectively, then* $l' = l''$*.*
- **Uniform:** *The output* l *from* Election[id] *is randomly generated based on a uniform distribution for* $l \in [1 : n]$*.*
- **Unpredictability:** *The adversary cannot correctly predict the output of* Election[id] *unless at least one honest node has activated it.*

When analyzing the performance of MVBA *protocols, we exclude the cost of the common coin protocol.*

Error correction code (ECC**).** An (n, k) error correction coding scheme consists of an encoding scheme $\text{ECCEnc} : \mathcal{B}^k \rightarrow \mathcal{B}^n$ and a decoding scheme $\text{ECCDec} : \mathcal{B}^{n'} \rightarrow \mathcal{B}^k$, where $\mathcal{B}$ denotes the alphabet of each symbol and $q \triangleq |\mathcal{B}|$ denotes the size of $\mathcal{B}$, for some n'. While $[y_1, y_2, \cdots, y_n] \leftarrow \text{ECCEnc}(n, k, \boldsymbol{w})$ outputs n encoded symbols, $y_j \leftarrow \text{ECCEnc}_j(n, k, \boldsymbol{w})$ outputs the jth encoded symbol.

Reed-Solomon (RS) codes (cf. [13]) are widely used error correction codes. An (n, k) RS error correction code can correct up to t Byzantine errors and simultaneously detect up to e Byzantine errors in n' symbol observations, given the conditions of $2t+e+k \leq n'$ and $n' \leq n$. The (n, k) RS code is operated over Galois Field $GF(q)$ under the constraint $n \leq q - 1$ (cf. [13]). RS codes can be constructed using *Lagrange polynomial interpolation*. The resulting code is a type of RS code with a minimum distance $d = n - k + 1$, which is optimal according to the Singleton bound. Berlekamp-Welch algorithm and Euclid's algorithm are two efficient decoding algorithms for RS codes [13]–[15].

Although RS is a popular error correction code, it has a constraint on the size of the alphabet, namely $n \leq q - 1$. To overcome this limitation, other error correction codes with a constant alphabet size, such as Expander Codes [4], can be used.

Erasure code (EC**).** An (n, k) erasure coding scheme consists of an encoding scheme $\text{ECEnc} : \mathcal{B}^k \rightarrow \mathcal{B}^n$ and a decoding scheme $\text{ECDec} : \mathcal{B}^k \rightarrow \mathcal{B}^k$, where $\mathcal{B}$ denotes the alphabet of each symbol and $q \triangleq |\mathcal{B}|$ denotes the size of $\mathcal{B}$. With an (n, k) erasure code, the original message can be decoded from any k encoded symbols. Specifically, given $[y_1, y_2, \cdots, y_n] \leftarrow \text{ECEnc}(n, k, \boldsymbol{w})$, then $\text{ECDec}(n, k, \{y_{j_1}, y_{j_2}, \cdots y_{j_k}\}) = \boldsymbol{w}$ holds true for any k distinct integers $j_1, j_2, \cdots, j_k \in [1 : n]$.

Online error correction (OEC**).** Online error correction is a variant of traditional error correction [16]. An (n, k) error correction code can correct up to t' Byzantine errors in n' symbol observations, provided the conditions of $2t' + k \leq n'$ and $n' \leq n$. However, in an asynchronous setting, a node might not be able to decode the message with n' symbol observations if $2t' + k > n'$. In such a case, the node can wait for one more symbol observation before attempting to decode again. This process repeats until the node successfully decodes the message. By setting the threshold as $n' \geq k + t$, OEC may perform up to t trials in the worst case before decoding the message.

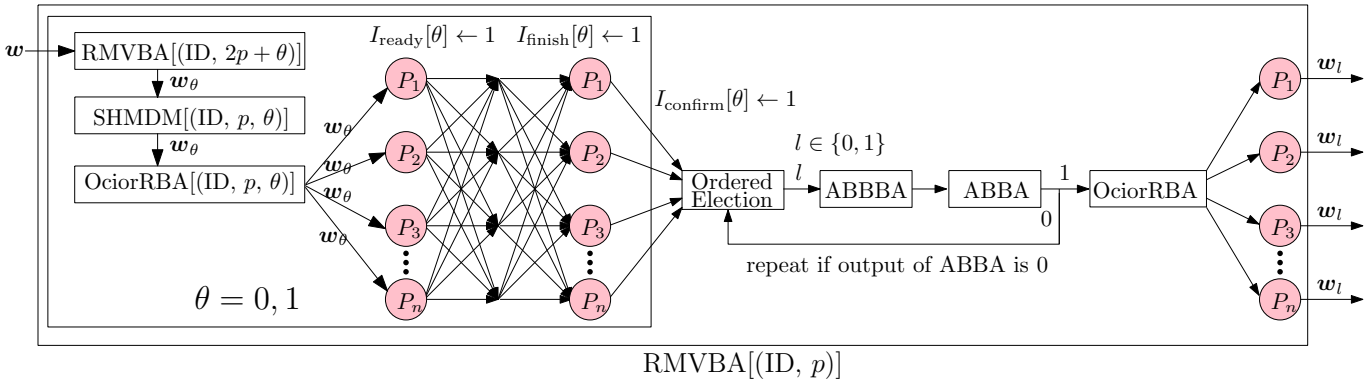

Fig. 1. A block diagram of the proposed OciorMVBA protocol with an identifier ID.

Algorithm 1 OciorMVBA protocol, with identifier ID, for $n \geq 3t+1$. Code is shown for P_i.

*// ** RMVBA: Recursive MVBA, error-free, IT secure ***
*// ** $\mathcal{S}_p$ is partitioned into two disjoint sets $\mathcal{S}_{2p}$ and $\mathcal{S}_{2p+1}$ such that $|\mathcal{S}_{2p}| = \lfloor |\mathcal{S}_p|/2 \rfloor$ and $|\mathcal{S}_{2p+1}| = \lceil |\mathcal{S}_p|/2 \rceil$ ***
*// ** Initially $\mathcal{S}_1$ denotes the set of all nodes with size n ***

1: **procedure** RMVBA$[(\mathrm{ID}, p)](\boldsymbol{w})$
2: let $\tilde{n} \leftarrow |\mathcal{S}_p|; \tilde{t} \leftarrow \lfloor \frac{|\mathcal{S}_p|-1}{3} \rfloor; r_p \leftarrow \lfloor \log p \rfloor + 1$
3: let $\theta \leftarrow 0, \bar{\theta} \leftarrow 1$ if $P_i \in \mathcal{S}_{2p}$, else $\theta \leftarrow 1, \bar{\theta} \leftarrow 0$
4: let $I_{\mathrm{finish}} \leftarrow \{\}; I_{\mathrm{ready}} \leftarrow \{\}; I_{\mathrm{confirm}} \leftarrow \{\}$
5: **for** $j \in \{0,1\}$ **do**
6: $I_{\mathrm{finish}}[j] \leftarrow 0; I_{\mathrm{ready}}[j] \leftarrow 0; I_{\mathrm{confirm}}[j] \leftarrow 0$
7: **upon** receiving input $\boldsymbol{w}$, for $\mathrm{Predicate}(\boldsymbol{w}) = \mathrm{true}$ and $P_i \in \mathcal{S}_p$ **do**:
8: **if** $|\mathcal{S}_p| \leq M$ **then** *// M is a preset finite number*
9: $\hat{\boldsymbol{w}} \leftarrow \mathrm{IneMVBA}[(\mathrm{ID}, p)](\boldsymbol{w})$ *// IneMVBA: inefficient MVBA protocol*
10: **return** $\hat{\boldsymbol{w}}$ and **terminate**
11: **else**
12: **pass** $\boldsymbol{w}$ into RMVBA$[(\mathrm{ID}, 2p+\theta)]$ as input *// From Line 3, it is true that $P_i \in \mathcal{S}_{2p+\theta}$ and $P_i \notin \mathcal{S}_{2p+\bar{\theta}}$*
13: **pass** $\perp$ into SHMDM$[(\mathrm{ID}, p, \bar{\theta})]$ as input
14: **wait** for $(I_{\mathrm{confirm}}[0] = 1) \vee (I_{\mathrm{confirm}}[1] = 1)$
15: **for** $l \in \{0,1\}$ **do**
16: $a \leftarrow \mathrm{ABBBA}[(\mathrm{ID}, p, l, \tilde{n}, \tilde{t})](I_{\mathrm{ready}}[l], I_{\mathrm{finish}}[l])$ *// asynchronous biased binary BA, within $\mathcal{S}_p$, with $\tilde{n}, \tilde{t}$ parameters*
17: $b \leftarrow \mathrm{ABBA}[(\mathrm{ID}, p, l, \tilde{n}, \tilde{t})](a)$ *// asynchronous binary BA, operated within $\mathcal{S}_p$, with $\tilde{n}, \tilde{t}$ parameters*
18: **if** $b = 1$ **then**
19: **pass** b into OciorRBA$[(\mathrm{ID}, p, l)]$ as a binary input (other than the message input)
20: **wait** for OciorRBA$[(\mathrm{ID}, p, l)]$ to output value $\hat{\boldsymbol{w}}$
21: **if** $\mathrm{Predicate}(\hat{\boldsymbol{w}}) = \mathrm{true}$ **then**
22: **output** $\hat{\boldsymbol{w}}$ and **terminate** this RMVBA$[(\mathrm{ID}, p)]$ and all invoked recursive protocols under it.

23: **upon** RMVBA$[(\mathrm{ID}, 2p+\theta)]$ outputs $\hat{\boldsymbol{w}}$, with $\mathrm{Predicate}(\hat{\boldsymbol{w}}) = \mathrm{true}$, and $P_i \in \mathcal{S}_{2p+\theta}$ **do**:
24: **pass** $\hat{\boldsymbol{w}}$ into SHMDM$[(\mathrm{ID}, p, \theta)]$ as a message input
25: **upon** SHMDM$[(\mathrm{ID}, p, j)]$ outputs $\hat{\boldsymbol{w}}$, with $\mathrm{Predicate}(\hat{\boldsymbol{w}}) = \mathrm{true}$, for $j \in \{0,1\}$ and $P_i \in \mathcal{S}_p$ **do**:
26: **pass** $\hat{\boldsymbol{w}}$ into OciorRBA$[(\mathrm{ID}, p, j)]$ as a message input *// OciorRBA$[(\mathrm{ID}, p, j)]$ is a reliable BA protocol operated within $\mathcal{S}_p$*

27: **upon** OciorRBA$[(\mathrm{ID}, p, j)]$ delivers $\mathrm{v}_i = 1$, for $j \in \{0,1\}$ and $P_i \in \mathcal{S}_p$ **do**:
28: $I_{\mathrm{ready}}[j] \leftarrow 1$ *// ready*
29: **send** $(\text{“READY”}, \mathrm{ID}, r_p, j)$ to all nodes within $\mathcal{S}_p$

30: **upon** receiving $\tilde{n} - \tilde{t}$ $(\text{“READY”}, \mathrm{ID}, r_p, j)$ messages from distinct nodes within $\mathcal{S}_p$ for $j \in \{0,1\}$ and $P_i \in \mathcal{S}_p$ **do**:
31: $I_{\mathrm{finish}}[j] \leftarrow 1$ *// finish*
32: **send** $(\text{“FINISH”}, \mathrm{ID}, r_p, j)$ to all nodes within $\mathcal{S}_p$

33: **upon** receiving $\tilde{n} - \tilde{t}$ $(\text{“FINISH”}, \mathrm{ID}, r_p, j)$ messages from distinct nodes within $\mathcal{S}_p$ for $j \in \{0,1\}$ and $P_i \in \mathcal{S}_p$ **do**:
34: $I_{\mathrm{confirm}}[j] \leftarrow 1$ *// confirm*

Algorithm 2 ABBBA protocol with identifier $(\mathrm{ID}, p, l, \tilde{n}, \tilde{t})$, for $\mathrm{id} := (\mathrm{ID}, \lfloor \log p \rfloor + 1, l)$. This protocol operates on a network $\mathcal{S}_p$ of $\tilde{n}$ nodes, up to $\tilde{t}$ of which may be dishonest. Code is shown for P_i.

*// ** Each node inputs a pair of numbers (a_1, a_2), for some $a_1, a_2 \in \{0, 1\}$ ***
*// ** Terminate is guaranteed if the following condition is satisfied: if one honest node inputs $a_2 = 1$, then at least $\tilde{t} + 1$ honest nodes input $a_1 = 1$ ***
*// ** If at least $\tilde{t} + 1$ honest nodes input $a_2 = 1$, then none of the honest nodes will output 0. This is because at most $\tilde{n} - (\tilde{t} + 1)$ nodes input $a_2 = 0$ in this case, indicating that the condition in Line 8 could not be satisfied. ***
*// ** If one honest node outputs 1 (only when $(\mathrm{cnt}_1 \geq \tilde{t} + 1) \vee (\mathrm{cnt}_2 \geq \tilde{t} + 1)$), then at least one honest node has an input as $a_1 = 1$ or $a_2 = 1$ ***

1: **upon** receiving input (a_1, a_2), for some $a_1, a_2 \in \{0, 1\}$ **do**:
2: $\quad \mathrm{cnt}_1 \leftarrow 0; \mathrm{cnt}_2 \leftarrow 0; \mathrm{cnt}_3 \leftarrow 0$
3: $\quad$ **send** ("ABBBA", id, a_1, a_2) to all nodes
4: $\quad$ **if** $(a_1 = 1) \vee (a_2 = 1)$ **then output** 1 and **terminate**
5: $\quad$ **wait** for at least one of the following events: 1) $\mathrm{cnt}_1 \geq \tilde{t} + 1$, 2) $\mathrm{cnt}_2 \geq \tilde{t} + 1$, or 3) $\mathrm{cnt}_3 \geq \tilde{n} - \tilde{t}$
6: $\quad\quad$ **if** $(\mathrm{cnt}_1 \geq \tilde{t} + 1) \vee (\mathrm{cnt}_2 \geq \tilde{t} + 1)$ **then**
7: $\quad\quad\quad$ **output** 1 and **terminate**
8: $\quad\quad$ **else if** $\mathrm{cnt}_3 \geq \tilde{n} - \tilde{t}$ **then**
9: $\quad\quad\quad$ **output** 0 and **terminate**
10: **upon** receiving ("ABBBA", id, a_1, a_2) from P_j for the first time, for some $a_1, a_2 \in \{0, 1\}$ **do**:
11: $\quad \mathrm{cnt}_1 \leftarrow \mathrm{cnt}_1 + a_1; \mathrm{cnt}_2 \leftarrow \mathrm{cnt}_2 + a_2$
12: $\quad$ **if** $a_2 = 0$ **then** $\mathrm{cnt}_3 \leftarrow \mathrm{cnt}_3 + 1$

Algorithm 3 SHMDM protocol with identifier (ID, p, θ), for $\mathrm{id} := (\mathrm{ID}, \lfloor \log p \rfloor + 1, \theta)$, and for $\theta \in \{0, 1\}$. Code is shown for P_i, where P_i denotes the ith node within $\mathcal{S}_p$.

*// ** SHMDM: Strongly-honest-majority distributed multicast ***
*// ** $\mathcal{S}_p$ is partitioned into two disjoint sets $\mathcal{S}_{2p}$ and $\mathcal{S}_{2p+1}$ with $|\mathcal{S}_{2p}| = \lfloor |\mathcal{S}_p|/2 \rfloor$ and $|\mathcal{S}_{2p+1}| = \lceil |\mathcal{S}_p|/2 \rceil$ ***
1: Initially set $\mathbb{Z}_{\mathrm{oec}} \leftarrow \{\}, n^\star \leftarrow |\mathcal{S}_{2p+\theta}|; t^\star \leftarrow \lfloor \frac{|\mathcal{S}_{2p+\theta}|-1}{3} \rfloor; k^\star \leftarrow t^\star + 1$
2: **upon** receiving input $\boldsymbol{w}$ **do**:
3: $\quad$ **if** $P_i \in \mathcal{S}_{2p+\theta}$ **then**
4: $\quad\quad i^\star \leftarrow i - \theta \cdot |\mathcal{S}_{2p}|$ *// $i^\star$ is the position of this node within $\mathcal{S}_{2p+\theta}$*
5: $\quad\quad z_{i^\star} \leftarrow \mathrm{ECCEnc}_{i^\star}(n^\star, k^\star, \boldsymbol{w})$ *// $\mathrm{ECCEnc}_{i^\star}$ outputs the $i^\star$th encoded symbol only*
6: $\quad\quad$ **send** ("INITIAL", id, $z_{i^\star}$) to all nodes in $\mathcal{S}_p \setminus \mathcal{S}_{2p+\theta}$ *// broadcast coded symbol to other set for decoding initial message*
7: $\quad\quad$ **output** $\boldsymbol{w}$ and **terminate**
8: **upon** receiving ("INITIAL", id, z) from P_j for the first time for some z, for $P_j \in \mathcal{S}_{2p+\theta}$, and $P_i \in \mathcal{S}_p \setminus \mathcal{S}_{2p+\theta}$ **do**:
9: $\quad j^\star \leftarrow j - \theta \cdot |\mathcal{S}_{2p}|; \quad \mathbb{Z}_{\mathrm{oec}}[j^\star] \leftarrow z$ *// $j^\star$ is the position of P_j within $\mathcal{S}_{2p+\theta}$*
10: $\quad$ **if** $|\mathbb{Z}_{\mathrm{oec}}| \geq k^\star + t^\star$ **then** *// online error correcting (OEC)*
11: $\quad\quad \tilde{\boldsymbol{w}} \leftarrow \mathrm{ECCDec}(n^\star, k^\star, \mathbb{Z}_{\mathrm{oec}})$
12: $\quad\quad [z'_1, z'_2, \cdots, z'_{n^\star}] \leftarrow \mathrm{ECCEnc}(n, k, \tilde{\boldsymbol{w}})$
13: $\quad\quad$ **if** at least $k^\star + t^\star$ symbols in $[z'_1, z'_2, \cdots, z'_{n^\star}]$ match with those in $\mathbb{Z}_{\mathrm{oec}}$ **then**
14: $\quad\quad\quad$ **output** $\tilde{\boldsymbol{w}}$ and **terminate**

Algorithm 4 OciorRBA protocol with identifier (ID, p, θ), for $\mathrm{id} := (\mathrm{ID}, \lfloor \log p \rfloor + 1, \theta)$. Code is shown for P_i, where P_i denotes the ith node within $\mathcal{S}_p$. This protocol is operated within $\mathcal{S}_p$.

*// ** OciorRBA is an error-free reliable Byzantine agreement (*RBA*) protocol, extended from OciorCOOL and OciorRBC [7]. ***

Initially set $\tilde{n} \leftarrow |\mathcal{S}_p|; \tilde{t} \leftarrow \lfloor \frac{|\mathcal{S}_p|-1}{3} \rfloor$, $\tilde{k} \leftarrow \lfloor \frac{\tilde{t}}{5} \rfloor + 1$; $I_{\mathrm{oecfinal}} \leftarrow 0; \mathbb{Y}_{\mathrm{oec}} \leftarrow \{\}; \mathbb{U}_0 \leftarrow \{\}; \mathbb{U}_1 \leftarrow \{\}; \mathbb{S}_0^{[1]} \leftarrow \{\}; \mathbb{S}_1^{[1]} \leftarrow \{\}; \mathbb{S}_0^{[2]} \leftarrow \{\}; \mathbb{S}_1^{[2]} \leftarrow \{\}; I_{\mathrm{ecc}} \leftarrow 0; I_{\mathrm{SI2}} \leftarrow 0; I_3 \leftarrow 0$

Phase 1

upon receiving a non-empty message input $\boldsymbol{w}_i$ **do**:
  $\boldsymbol{w}^{(i)} \leftarrow \boldsymbol{w}_i$
  $[y_1^{(i)}, y_2^{(i)}, \cdots, y_{\tilde{n}}^{(i)}] \leftarrow \mathrm{ECCEnc}(\tilde{n}, \tilde{k}, \boldsymbol{w}_i)$
  send $(\text{“SYMBOL”}, \mathrm{id}, (y_j^{(i)}, y_i^{(i)}))$ to P_j, $\forall j \in [1:\tilde{n}]$, and then set $I_{\mathrm{ecc}} \leftarrow 1$ *// exchange coded symbols*
upon receiving $(\text{“SYMBOL”}, \mathrm{id}, (y_i^{(j)}, y_j^{(j)}))$ from P_j for the first time **do**:
  wait until $I_{\mathrm{ecc}} = 1$
  if $(y_i^{(j)}, y_j^{(j)}) = (y_i^{(i)}, y_j^{(i)})$ **then**
    $\mathbb{U}_1 \leftarrow \mathbb{U}_1 \cup \{j\}$ *// update the set of link indicators*
  else
    $\mathbb{U}_0 \leftarrow \mathbb{U}_0 \cup \{j\}$
upon $|\mathbb{U}_1| \geq \tilde{n} - \tilde{t}$, and $(\text{“SI1”}, \mathrm{id}, *)$ not yet sent **do**:
  set $\mathrm{s}_i^{[1]} \leftarrow 1$, and **send** $(\text{“SI1”}, \mathrm{id}, \mathrm{s}_i^{[1]})$ to all nodes *// set success indicator*
upon $|\mathbb{U}_0| \geq \tilde{t} + 1$, and $(\text{“SI1”}, \mathrm{id}, *)$ not yet sent **do**:
  set $\mathrm{s}_i^{[1]} \leftarrow 0$, and **send** $(\text{“SI1”}, \mathrm{id}, \mathrm{s}_i^{[1]})$ to all nodes
upon receiving $(\text{“SI1”}, \mathrm{id}, \mathrm{s}_j^{[1]})$ from P_j for the first time **do**:
  if $\mathrm{s}_j^{[1]} = 1$ **then** $\mathbb{S}_1^{[1]} \leftarrow \mathbb{S}_1^{[1]} \cup \{j\}$ **else** $\mathbb{S}_0^{[1]} \leftarrow \mathbb{S}_0^{[1]} \cup \{j\}$

Phase 2

upon $(\mathrm{s}_i^{[1]} = 0) \vee (|\mathbb{S}_0^{[1]} \cup \mathbb{U}_0| \geq \tilde{t} + 1)$, and $(\text{“SI2”}, \mathrm{id}, \mathrm{s}_i^{[2]})$ not yet sent **do**:
  set $\mathrm{s}_i^{[2]} \leftarrow 0$, **send** $(\text{“SI2”}, \mathrm{id}, \mathrm{s}_i^{[2]})$ to all nodes *// update success indicator*
upon $(\mathrm{s}_i^{[1]} = 1) \wedge (|\mathbb{S}_1^{[1]} \cap \mathbb{U}_1| \geq \tilde{n} - \tilde{t})$, and $(\text{“SI2”}, \mathrm{id}, \mathrm{s}_i^{[2]})$ not yet sent **do**:
  set $\mathrm{s}_i^{[2]} \leftarrow 1$, $I_{\mathrm{SI2}} \leftarrow 1$, and **send** $(\text{“SI2”}, \mathrm{id}, \mathrm{s}_i^{[2]})$ to all nodes
upon receiving $(\text{“SI2”}, \mathrm{id}, \mathrm{s}_j^{[2]})$ from P_j for the first time **do**:
  if $\mathrm{s}_j^{[2]} = 1$ **then** $\mathbb{S}_1^{[2]} \leftarrow \mathbb{S}_1^{[2]} \cup \{j\}$ **else** $\mathbb{S}_0^{[2]} \leftarrow \mathbb{S}_0^{[2]} \cup \{j\}$

24: **upon** $|\mathbb{S}_{\mathrm{v}}^{[2]}| \geq \tilde{n} - \tilde{t}$, for a $\mathrm{v} \in \{1, 0\}$, and $(\text{“READY”}, \mathrm{id}, *)$ not yet sent **do**:
25: set $\mathrm{v}_i \leftarrow \mathrm{v}$, and **deliver** v_i *// the value of v_i is delivered out to the protocol in Algorithm 1*
26: **send** $(\text{“READY”}, \mathrm{id}, \mathrm{v}_i)$ to all nodes
27: **upon** receiving $\tilde{t} + 1$ $(\text{“READY”}, \mathrm{id}, \mathrm{v})$ messages from different nodes for the same v and $(\text{“READY”}, \mathrm{id}, *)$ not yet sent **do**:
28: **send** $(\text{“READY”}, \mathrm{id}, \mathrm{v})$ to all nodes
29: **upon** receiving $2\tilde{t} + 1$ $(\text{“READY”}, \mathrm{id}, \mathrm{v})$ messages from different nodes for the same v **do**:
30: **if** $(\text{“READY”}, \mathrm{id}, *)$ not yet sent **then**
31: **send** $(\text{“READY”}, \mathrm{id}, \mathrm{v})$ to all nodes
32: set $\mathrm{v}_o \leftarrow \mathrm{v}$
33: **if** $\mathrm{v}_o = 0$ **then**
34: set $\boldsymbol{w}^{(i)} \leftarrow \perp$, then **output** $\boldsymbol{w}^{(i)}$ and **terminate** *// $\perp$ is a default value*
35: **else**
36: set $I_3 \leftarrow 1$
37: **upon** receiving a binary input $b = 1$ (other than the message input $\boldsymbol{w}_i$) **do**: *// b is delivered from the protocol in Algorithm 1*
38: set $I_3 \leftarrow 1$

Phase 3

39: **upon** $I_3 = 1$ **do**: *// only after executing Line 36 or Line 38*
40: **if** $I_{\mathrm{SI2}} = 1$ **then**
41: **output** $\boldsymbol{w}^{(i)}$ and **terminate**
42: **else**
43: **wait** until receiving $\tilde{t} + 1$ $(\text{“SYMBOL”}, \mathrm{id}, (y_i^{(j)}, *))$ messages, $\forall j \in \mathbb{S}_1^{[2]}$, for the same $y_i^{(j)} = y^\star$, for some $y^\star$
44: $y_i^{(i)} \leftarrow y^\star$ *// update coded symbol based on majority rule*
45: **send** $(\text{“CORRECT”}, \mathrm{id}, y_i^{(i)})$ to all nodes
46: **wait** until $I_{\mathrm{oecfinal}} = 1$
47: **output** $\boldsymbol{w}^{(i)}$ and **terminate**
48: **upon** receiving $(\text{“CORRECT”}, \mathrm{id}, y_j^{(j)})$ from P_j for the first time, $j \notin \mathbb{Y}_{\mathrm{oec}}$, and $I_{\mathrm{oecfinal}} = 0$ **do**:
49: $\mathbb{Y}_{\mathrm{oec}}[j] \leftarrow y_j^{(j)}$
50: **if** $|\mathbb{Y}_{\mathrm{oec}}| \geq \tilde{k} + \tilde{t}$ **then** *// online error correcting (OEC)*
51: $\hat{\boldsymbol{w}} \leftarrow \mathrm{ECCDec}(\tilde{n}, \tilde{k}, \mathbb{Y}_{\mathrm{oec}})$
52: $[y_1, y_2, \cdots, y_{\tilde{n}}] \leftarrow \mathrm{ECCEnc}(\tilde{n}, \tilde{k}, \hat{\boldsymbol{w}})$
53: **if** at least $\tilde{k} + \tilde{t}$ symbols in $[y_1, y_2, \cdots, y_{\tilde{n}}]$ match with those in $\mathbb{Y}_{\mathrm{oec}}$ **then**
54: $\boldsymbol{w}^{(i)} \leftarrow \hat{\boldsymbol{w}}$; $I_{\mathrm{oecfinal}} \leftarrow 1$
55: **upon** having received both $(\text{“SYMBOL”}, \mathrm{id}, (y_i^{(j)}, y_j^{(j)}))$ and $(\text{“SI2”}, \mathrm{id}, 1)$ messages from P_j, and $j \notin \mathbb{Y}_{\mathrm{oec}}$, and $I_{\mathrm{oecfinal}} = 0$ **do**:
56: $\mathbb{Y}_{\mathrm{oec}}[j] \leftarrow y_j^{(j)}$
57: run the OEC steps as in Lines 50-54

II. OciorMVBA

This proposed OciorMVBA is an error-free, information-theoretically secure asynchronous MVBA protocol. OciorMVBA doesn't rely on any cryptographic assumptions, such as signatures or hashing, except for the common coin assumption. The design of OciorMVBA in this MVBA setting builds on the protocols COOL and OciorCOOL [5]–[7].

A. Overview of the proposed OciorMVBA protocol

The proposed OciorMVBA is described in Algorithm 1, along with Algorithms 2-4. Fig. 1 presents a block diagram of the proposed OciorMVBA protocol. For a network $\mathcal{S}_p$, we define the network size and the faulty threshold as $\tilde{n}_p := |\mathcal{S}_p|$ and $\tilde{t}_p := \lfloor \frac{|\mathcal{S}_p|-1}{3} \rfloor$, for $p \in \{1, 2, 3, \cdots\}$. The original network is defined as $\mathcal{S}_1 := \{P_i : i \in [1 : n]\}$, where P_i denotes the ith node in the network. $\mathcal{S}_p$ is partitioned into two disjoint sets $\mathcal{S}_{2p}$ and $\mathcal{S}_{2p+1}$ such that $|\mathcal{S}_{2p}| = \lfloor |\mathcal{S}_p|/2 \rfloor$ and $|\mathcal{S}_{2p+1}| = \lceil |\mathcal{S}_p|/2 \rceil$. Below, we provide an overview of the proposed protocol.

OciorMVBA is a recursive protocol. As shown in Fig. 1, the protocol $\mathrm{RMVBA}[(\mathrm{ID}, p)]$ invokes two sub-protocols: $\mathrm{RMVBA}[(\mathrm{ID}, 2p)]$ and $\mathrm{RMVBA}[(\mathrm{ID}, 2p{+}1)]$. Each sub-protocol, in turn, invokes two additional sub-protocols. This process continues until the size of network on which a sub-protocol operates on is smaller than a predefined finite threshold. In the final protocol invoked, any inefficient MVBA protocol

(referred to as $\mathrm{IneMVBA}$) can be used without impacting overall performance, as the size of the operated network is finite.

The general protocol $\mathrm{RMVBA}[(\mathrm{ID}, p)]$ comprises several steps, as illustrated in Fig. 1. It operates over the network $\mathcal{S}_p$, which is partitioned into two disjoint sets, $\mathcal{S}_{2p}$ and $\mathcal{S}_{2p+1}$, of balanced size. The two invoked protocols operate on these partitioned network sets. The key steps involved in $\mathrm{RMVBA}[(\mathrm{ID}, p)]$ are described below.

- *Step 1:* Upon an invoked sub-protocol $\mathrm{RMVBA}[(\mathrm{ID}, 2p+\theta)]$, for $\theta \in \{0, 1\}$, outputs a message $\boldsymbol{w}_\theta$, the nodes within $\mathcal{S}_{2p+\theta}$ input $\boldsymbol{w}_\theta$ into $\mathrm{SHMDM}[(\mathrm{ID}, p, \theta)]$.
- *Step 2:* Upon $\mathrm{SHMDM}[(\mathrm{ID}, p, \theta)]$ outputs a message $\boldsymbol{w}_\theta$, the nodes within $\mathcal{S}_p$ input $\boldsymbol{w}_\theta$ into OciorRBA$[(\mathrm{ID}, p, \theta)]$.
- *Step 3:* Upon OciorRBA$[(\mathrm{ID}, p, \theta)]$ delivers $\mathrm{v}_i = 1$, the nodes within $\mathcal{S}_p$ set $I_{\mathrm{ready}}[\theta] \leftarrow 1$ and send $(\text{“READY”}, \mathrm{ID}, r_p, \theta)$ to all nodes within $\mathcal{S}_p$, where $r_p := \lfloor \log p \rfloor + 1$.
- *Step 4:* Upon receiving $\tilde{n}_p - \tilde{t}_p$ $(\text{“READY”}, \mathrm{ID}, r_p, \theta)$ messages from distinct nodes within $\mathcal{S}_p$, the nodes within $\mathcal{S}_p$ set $I_{\mathrm{finish}}[\theta] \leftarrow 1$ and send $(\text{“FINISH”}, \mathrm{ID}, r_p, \theta)$ to all nodes within $\mathcal{S}_p$.
- *Step 5:* Upon receiving $\tilde{n}_p - \tilde{t}_p$ $(\text{“FINISH”}, \mathrm{ID}, r_p, \theta)$ messages from distinct nodes within $\mathcal{S}_p$, the nodes within $\mathcal{S}_p$ set $I_{\mathrm{confirm}}[\theta] \leftarrow 1$.
- *Step 6:* After setting $I_{\mathrm{confirm}}[\theta] \leftarrow 1$, the nodes within $\mathcal{S}_p$ proceed to the Ordered Election step, which outputs l, taking a value from $\{0, 1\}$ in order.
- *Step 7:* The nodes within $\mathcal{S}_p$ run the ABBBA protocol with inputs $(I_{\mathrm{ready}}[l], I_{\mathrm{finish}}[l])$.
- *Step 8:* After ABBBA outputs a value a, the nodes within $\mathcal{S}_p$ run an asynchronous binary BA (ABBA) protocol with input a.
- *Step 9:* If ABBA outputs a value 1, the nodes within $\mathcal{S}_p$ input 1 into OciorRBA$[(\mathrm{ID}, p, l)]$ and wait for its output. If OciorRBA$[(\mathrm{ID}, p, l)]$ outputs a value $\boldsymbol{w}_l$ such that $\mathrm{Predicate}(\boldsymbol{w}_l) = \mathrm{true}$, the nodes within $\mathcal{S}_p$ output $\boldsymbol{w}_l$ and terminate the protocol $\mathrm{RMVBA}[(\mathrm{ID}, p)]$. If ABBA outputs a value 0, the nodes go back to Step 6.

In our design, by combining the the Ready-Finish-Confirm process in Steps 4-6 with ABBBA in Steps 7, we ensure that if one honest node has set $I_{\mathrm{confirm}}[l] \leftarrow 1$, eventually every honest node outputs 1 from $\mathrm{ABBBA}[(\mathrm{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ in Step 7, due to the Biased Validity property of ABBBA. It is worth noting that the design of OciorRBA protocol in the RBA setting builds upon the protocols COOL and OciorCOOL, which utilize UA and HMDM as building blocks [5]–[7].

B. Analysis of $\mathrm{OciorMVBA}$

Definition 11 (Good resilience). *A network $\mathcal{S}_p$ is said to have good resilience if the number of dishonest nodes within $\mathcal{S}_p$, denoted by $t_p := |\mathcal{S}_p \cap \mathcal{F}|$, satisfies the condition $t_p < \frac{|\mathcal{S}_p|}{3}$, where $\mathcal{F}$ denotes the set of all dishonest nodes. This condition $t_p < \frac{|\mathcal{S}_p|}{3}$ is equivalent to $t_p \leq \frac{|\mathcal{S}_p|-1}{3}$, and also equivalent to the condition $t_p \leq \lfloor \frac{|\mathcal{S}_p|-1}{3} \rfloor$ due to the integer nature of t_p.*

Definition 12 (Network tree). *In our setting, $\mathcal{S}_p$ is partitioned into two disjoint sets $\mathcal{S}_{2p}$ and $\mathcal{S}_{2p+1}$. For example, $\mathcal{S}_1$ (at Layer 1) is partitioned into two disjoint sets: $\mathcal{S}_2$ and $\mathcal{S}_3$ (at Layer 2). The two sets $\mathcal{S}_2$ and $\mathcal{S}_3$ are partitioned into $\mathcal{S}_4$, $\mathcal{S}_5$ and $\mathcal{S}_6$, $\mathcal{S}_7$, respectively, at Layer 3. We define the network tree as comprising all layers of sets: $\mathcal{S}_1$ at Layer 1; $\mathcal{S}_2$ and $\mathcal{S}_3$ at Layer 2; $\mathcal{S}_4$, $\mathcal{S}_5$, $\mathcal{S}_6$ and $\mathcal{S}_7$ at Layer 3; and so on.*

Definition 13 (Network chain). *By selecting one network set at each layer from a network tree, where the set selected at Layer r is partitioned from the set selected at Layer $r-1$, for $r = 2, 3, \cdots$, then the selected network sets form a network chain. One example of a network chain is $\mathcal{S}_1 \rightarrow \mathcal{S}_3 \rightarrow \mathcal{S}_7 \rightarrow \mathcal{S}_{15} \rightarrow \cdots$.*

Definition 14 (Network chain with good resilience). *A network chain is considered to have good resilience if every network set it includes has good resilience. For example, if all of $\mathcal{S}_1, \mathcal{S}_3, \mathcal{S}_7, \mathcal{S}_{15}, \cdots$*

have good resilience, then the network chain $\mathcal{S}_1 \to \mathcal{S}_3 \to \mathcal{S}_7 \to \mathcal{S}_{15} \to \cdots$ is considered to have good resilience.

Theorem 1 (Agreement and Termination)**.** *In* OciorMVBA*, given $n \geq 3t+1$, every honest node eventually outputs a consistent value and terminates.*

Proof. In our setting, $\mathcal{S}_p$ is partitioned into two disjoint sets $\mathcal{S}_{2p}$ and $\mathcal{S}_{2p+1}$. From Lemma 1, if $\mathcal{S}_p$ has good resilience, i.e., $|\mathcal{S}_p \cap \mathcal{F}| < \frac{|\mathcal{S}_p|}{3}$, then at least one of the two sets, $\mathcal{S}_{2p}$ or $\mathcal{S}_{2p+1}$, also has good resilience. From Lemma 2, given $n \geq 3t+1$, there exists a network chain with good resilience.

Let us focus on a network chain $\mathcal{S}_1 \to \cdots \to \mathcal{S}_p \to \mathcal{S}_{2p+\theta} \to \cdots$ with good resilience, for some $\theta \in \{0,1\}$. From Lemma 4, it is true that if $\mathrm{RMVBA}[(\mathrm{ID}, 2p+\theta)]$ outputs a consistent value at all honest nodes within $\mathcal{S}_{2p+\theta}$ and terminates, then $\mathrm{RMVBA}[(\mathrm{ID}, p)]$ eventually outputs a consistent value at all honest nodes within $\mathcal{S}_p$ and terminates. Based on a recursive argument, and given that the last invoked RMVBA protocol eventually outputs a consistent value at all honest nodes within the last network set in the chain, it is concluded that every honest node within $\mathcal{S}_1$ eventually outputs a consistent value and terminates, given that $n \geq 3t+1$. □

Theorem 2 (External Validity)**.** *In* OciorMVBA*, if an honest node outputs a value $\boldsymbol{w}$, then* $\mathrm{Predicate}(\boldsymbol{w}) = \mathrm{true}$*.*

Proof. In OciorMVBA, if an honest node outputs a value $\boldsymbol{w}$, it has verified that $\mathrm{Predicate}(\boldsymbol{w}) = \mathrm{true}$ at Line 21 of Algorithm 1. □

Theorem 3 (Communication and Round Complexities)**.** *The communication complexity of* OciorMVBA *is $O(n|\boldsymbol{w}|\log n + n^2 \log q)$ bits, while the round complexity of* OciorMVBA *is $O(\log n)$ rounds, given $n \geq 3t+1$.*

Proof. The protocol $\mathrm{RMVBA}[(\mathrm{ID}, p)]$ is operated over $\mathcal{S}_p$ with a network size of $\tilde{n}_p := |\mathcal{S}_p|$. The total communication complexity in bits of the protocol $\mathrm{RMVBA}[(\mathrm{ID}, p)]$, defined by $f_{\mathrm{TB}}(\tilde{n}_p)$ is expressed as

$$f_{\mathrm{TB}}(\tilde{n}_p) = \begin{cases} O(|\boldsymbol{w}|) & \text{if } m \leq M \\ \beta_1 \tilde{n}_p |\boldsymbol{w}| + \beta_2 \tilde{n}_p^2 \log q + f_{\mathrm{TB}}\big(\lfloor \frac{\tilde{n}_p}{2} \rfloor\big) + f_{\mathrm{TB}}\big(\lceil \frac{\tilde{n}_p}{2} \rceil\big) & \text{otherwise} \end{cases}$$

where M is a finite constant; β_1 and β_2 are finite constants; and $|\boldsymbol{w}|$ denotes the size of each input message. It is worth mentioning that each coded symbol transmitted in DRBC-COOL protocol carries at least $\log q$ bits due to the alphabet size of error correction code. We ignore the cost of the index $r_p = \lfloor \log p \rfloor + 1$ in transmitted messages (using $\log \log n$ bits), as it can be redesigned such that the total indexing cost related to r_p becomes negligible compared to the cost of coded symbols. When $n = 2^J M$ for some J, then the total communication complexity in bits of the proposed OciorMVBA is

$$\begin{aligned}
f_{\mathrm{TB}}(n) =& \beta_1 n|\boldsymbol{w}| + \beta_2 n^2 \log q + 2 f_{\mathrm{TB}}\big(\frac{n}{2}\big) \\
=& \beta_1 n|\boldsymbol{w}| + \beta_2 n^2 \log q + 2\big(\beta_1 \cdot \frac{n}{2} \cdot |\boldsymbol{w}| + \beta_2 \frac{n^2}{4} \log q + 2 f_{\mathrm{TB}}\big(\frac{n}{2^2}\big)\big) \\
=& \beta_1 n|\boldsymbol{w}| + \beta_2 n^2 \log q + \beta_1 n|\boldsymbol{w}| + \beta_2 \frac{n^2}{2} \log q + 2^2 \cdot f_{\mathrm{TB}}\big(\frac{n}{2^2}\big) \\
& \vdots \\
=& \beta_1 n|\boldsymbol{w}| + \beta_2 n^2 \log q + \beta_1 n|\boldsymbol{w}| + \beta_2 \frac{n^2}{2} \log q + \cdots + \beta_1 n|\boldsymbol{w}| + \beta_2 \frac{n^2}{2^{J-1}} \log q + 2^J \cdot f_{\mathrm{TB}}\big(\frac{n}{2^J}\big) \\
=& J\beta_1 n|\boldsymbol{w}| + \beta_2 n^2 \log q \ \cdot (1 + \frac{1}{2} + \cdots + \frac{1}{2^{J-1}}) + 2^J \cdot f_{\mathrm{TB}}\big(M\big) \\
=& O(n|\boldsymbol{w}|\log n + n^2 \log q).
\end{aligned}$$

The round complexity of OciorMVBA is $O(\log n)$ rounds. □

Lemma 1. *If $\mathcal{S}_p$ has good resilience, i.e., $|\mathcal{S}_p \cap \mathcal{F}| < \frac{|\mathcal{S}_p|}{3}$, then at least one of the two sets, $\mathcal{S}_{2p}$ or $\mathcal{S}_{2p+1}$, also has good resilience.*

Proof. If $\mathcal{S}_p$ has good resilience, i.e., $|\mathcal{S}_p \cap \mathcal{F}| < \frac{|\mathcal{S}_p|}{3}$, then at least one of the following two conditions is satisfied: $|\mathcal{S}_{2p} \cap \mathcal{F}| < \frac{|\mathcal{S}_{2p}|}{3}$ or $|\mathcal{S}_{2p+1} \cap \mathcal{F}| < \frac{|\mathcal{S}_{2p+1}|}{3}$. □

Lemma 2. *Given $n \geq 3t+1$, there exists a network chain with good resilience.*

Proof. Given $n \geq 3t+1$, $\mathcal{S}_1$ has good resilience. $\mathcal{S}_1$ is partitioned into two disjoint sets, $\mathcal{S}_2$ and $\mathcal{S}_3$. From Lemma 1, at least one of the sets $\mathcal{S}_2$ and $\mathcal{S}_3$ has good resilience. Let us assume that $\mathcal{S}_3$ has good resilience and include it into a network chain. Next, $\mathcal{S}_3$ is partitioned into two disjoint sets $\mathcal{S}_6$ and $\mathcal{S}_7$. Again, from Lemma 1, at least one of the sets $\mathcal{S}_6$ and $\mathcal{S}_7$ has good resilience. Let us assume that $\mathcal{S}_7$ has good resilience and include it into the network chain. By repeating this process, we construct a network chain $\mathcal{S}_1 \rightarrow \mathcal{S}_3 \rightarrow \mathcal{S}_7 \rightarrow \cdots$ such that all network sets it includes have good resilience. This implies that the selected network chain has good resilience. □

Without loss of generality, we will focus on the network set $\mathcal{S}_p$, such that all other protocols $\mathrm{RMVBA}[(\mathrm{ID}, p')]$ invoking $\mathrm{RMVBA}[(\mathrm{ID}, p)]$ haven't outputted a value at any honest node yet, where the network sets $\mathcal{S}_{p'}$ and $\mathcal{S}_p$ are within the same network chain with good resilience and $p' < p$.

Lemma 3. *Let us assume that $\mathcal{S}_p$ has good resilience. If one honest node within $\mathcal{S}_p$ outputs a value $\boldsymbol{w}$ from $\mathrm{RMVBA}[(\mathrm{ID}, p)]$, then all other honest nodes within $\mathcal{S}_p$ eventually outputs a consistent value $\boldsymbol{w}$ from $\mathrm{RMVBA}[(\mathrm{ID}, p)]$.*

Proof. Given that $\mathcal{S}_p$ has good resilience, if one honest node within $\mathcal{S}_p$ outputs a value $\boldsymbol{w}$ from $\mathrm{RMVBA}[(\mathrm{ID}, p)]$, then all hones nodes within $\mathcal{S}_p$ must have output $1 \leftarrow \mathrm{ABBA}[(\mathrm{ID}, p, l, \tilde{n}, \tilde{t})](a)$ in Line 17 of Algorithm 1, for some $l \in \{0, 1\}$. Then, from Lemma 10 (Consistency and Totality properties of OciorRBA), all hones nodes within $\mathcal{S}_p$ eventually output the same value $\boldsymbol{w}$ from $\mathrm{OciorRBA}[(\mathrm{ID}, p, l)]$ in Line 20 of Algorithm 1. □

Lemma 4. *Let us assume that $\mathcal{S}_1 \rightarrow \cdots \rightarrow \mathcal{S}_p \rightarrow \mathcal{S}_{2p+\theta} \rightarrow \cdots$ forms a network chain with good resilience, for some $\theta \in \{0, 1\}$. If $\mathrm{RMVBA}[(\mathrm{ID}, 2p+\theta)]$ outputs a consistent value at all honest nodes within $\mathcal{S}_{2p+\theta}$ and terminates, then $\mathrm{RMVBA}[(\mathrm{ID}, p)]$ eventually outputs a consistent value at all honest nodes within $\mathcal{S}_p$ and terminates.*

Proof. Assume that $\mathcal{S}_p$ and $\mathcal{S}_{2p+\theta}$ have good resilience. If $\mathrm{RMVBA}[(\mathrm{ID}, 2p+\theta)]$ outputs a consistent value $\boldsymbol{w}$ at all honest nodes within $\mathcal{S}_{2p+\theta}$, then eventually $\mathrm{SHMDM}[(\mathrm{ID}, p, \theta)]$ outputs a consistent value $\boldsymbol{w}$ at all honest nodes within $\mathcal{S}_p$ (see Lines 23 and 24 of Algorithm 1). Consequently, $\mathrm{OciorRBA}[(\mathrm{ID}, p, \theta)]$ eventually outputs the same value $\boldsymbol{w}$ at all hones nodes within $\mathcal{S}_p$ (See Lines 25 and 26 of Algorithm 1). Therefore, all honest nodes within $\mathcal{S}_p$ eventually set $I_{\mathrm{ready}}[\theta] \leftarrow 1, I_{\mathrm{finish}}[\theta] \leftarrow 1, I_{\mathrm{confirm}}[\theta] \leftarrow 1$. In this case, all honest nodes within $\mathcal{S}_p$ eventually go to Line 15 and execute the steps in Lines 15-22 of Algorithm 1. Based on the result of Lemma 3, if $\theta = 0$, then all honest nodes within $\mathcal{S}_p$ eventually output a consistent value $\boldsymbol{w}$. For the case where $\theta = 1$, if $1 \leftarrow \mathrm{ABBA}[(\mathrm{ID}, p, 0, \tilde{n}_p, \tilde{t}_p)]$, then all honest nodes within $\mathcal{S}_p$ eventually output a consistent value $\boldsymbol{w}'$ for some $\boldsymbol{w}'$. Otherwise, they eventually output a consistent value $\boldsymbol{w}$. □

Lemma 5. *Assume that $\mathcal{S}_p$ has good resilience. If $\mathrm{ABBA}[(\mathrm{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ outputs 1, then at least one honest Node i within $\mathcal{S}_p$ has set $\mathrm{v}_i = 1$ from OciorRBA$[(\mathrm{ID}, p, l)]$, for $l \in \{0, 1\}$.*

Proof. In this setting, if $\mathrm{ABBA}[(\mathrm{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ outputs $b = 1$, then at least one honest node must have provided an input $a = 1$ to $\mathrm{ABBA}[(\mathrm{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ (see Line 17 of Algorithm 1), due to the validity property of Byzantine agreement. This means that at least one honest node must have produced an output $a = 1$ from $\mathrm{ABBBA}[(\mathrm{ID}, p, l, \tilde{n}_p, \tilde{t}_p)](I_{\mathrm{ready}}[l], I_{\mathrm{finish}}[l])$ (see Line 16 of Algorithm 1). If an honest node outputs 1 from ABBBA, then at least one honest node must have provided at least one input as 1 to ABBBA (see Line 7 of Algorithm 2, biased integrity property). Thus, if $\mathrm{ABBA}[(\mathrm{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ outputs

$b = 1$, then at least one honest node must have set $I_{\text{ready}}[l] = 1$ or $I_{\text{finish}}[l] = 1$. If one honest Node i sets $I_{\text{ready}}[l] = 1$, it must have set $\text{v}_i = 1$ from OciorRBA$[(\text{ID}, p, l)]$ (see Line 27 of Algorithm 1). If one honest Node i sets $I_{\text{finish}}[l] = 1$, then at least $\tilde{n}_p - 2\tilde{t}_p$ honest nodes must have set $\text{v}_i = 1$ from OciorRBA$[(\text{ID}, p, l)]$ (see Line 30 of Algorithm 1). Therefore, if $\text{ABBA}[(\text{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ outputs $b = 1$, then at least one honest Node i within $\mathcal{S}_p$ has set $\text{v}_i = 1$ from OciorRBA$[(\text{ID}, p, l)]$. □

Lemma 6. *Assume that $\mathcal{S}_p$ has good resilience. If* $\text{ABBA}[(\text{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ *outputs* 1*, then no honest Node i within $\mathcal{S}_p$ will set* $\text{v}_i = 0$ *from OciorRBA*$[(\text{ID}, p, l)]$*, for* $l \in \{0, 1\}$.

Proof. Lemma 5 reveals that if $\text{ABBA}[(\text{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ outputs $b = 1$, then at least one honest Node i within $\mathcal{S}_p$ has set $\text{v}_i = 1$ from OciorRBA$[(\text{ID}, p, l)]$ in this setting. If one honest node has set $\text{v}_i = 1$, then at least $\tilde{n}_p - 2\tilde{t}_p$ honest nodes have sent $(\text{“SI2”}, \text{id}, 1)$ (see Line 24 of Algorithm 4). In this case, the size of $\mathbb{S}_0^{[2]}$ is bounded by $|\mathbb{S}_0^{[2]}| \leq \tilde{n}_p - (\tilde{n}_p - 2\tilde{t}_p) < \tilde{n}_p - \tilde{t}_p$ from the view of any honest node, which indicates that no honest node will set $\text{v}_i = 0$ from OciorRBA$[(\text{ID}, p, l)]$. □

Lemma 7. *Assume that $\mathcal{S}_p$ has good resilience. If* $\text{ABBA}[(\text{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ *outputs* 1*, then eventually every honest node within $\mathcal{S}_p$ will set $I_3 = 1$ from OciorRBA*$[(\text{ID}, p, l)]$*, for* $l \in \{0, 1\}$.

Proof. From the result of Lemma 6, if $\text{ABBA}[(\text{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ outputs 1, then no honest Node i within $\mathcal{S}_p$ will set $\text{v}_i = 0$ from OciorRBA$[(\text{ID}, p, l)]$. This suggests that, in this case, no honest node will set $\text{v}_o = 0$ and Line 34 of Algorithm 4 will never be executed in OciorRBA$[(\text{ID}, p, l)]$. On the other hand, if $\text{ABBA}[(\text{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ outputs $b = 1$, then eventually every honest node within $\mathcal{S}_p$ will input $b = 1$ into OciorRBA$[(\text{ID}, p, l)]$ and set $I_3 = 1$ (see Line 38 of Algorithm 4). □

Lemma 8. *Assume that $\mathcal{S}_p$ has good resilience. If* $\text{ABBA}[(\text{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ *outputs* 1*, then at least* $\tilde{n}_p - 2\tilde{t}_p \geq \tilde{t}_p + 1$ *honest nodes within $\mathcal{S}_p$ have set* $\text{s}_i^{[2]} = 1$ *from OciorRBA*$[(\text{ID}, p, l)]$*, for* $l \in \{0, 1\}$.

Proof. From the result of Lemma 5, if $\text{ABBA}[(\text{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ outputs 1, then at least one honest Node i within $\mathcal{S}_p$ has set $\text{v}_i = 1$ from OciorRBA$[(\text{ID}, p, l)]$. When one honest Node i within $\mathcal{S}_p$ has set $\text{v}_i = 1$, it means that at least $\tilde{n}_p - 2\tilde{t}_p \geq \tilde{t}_p + 1$ honest nodes within $\mathcal{S}_p$ have set $\text{s}_i^{[2]} = 1$ from OciorRBA$[(\text{ID}, p, l)]$ (see Line 24 of Algorithm 4). □

Lemma 9. *[7, Lemma 11] Assume that $\mathcal{S}_p$ has good resilience. If* $\text{ABBA}[(\text{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ *outputs* 1*, then all of the honest nodes who set* $\text{s}_i^{[2]} = 1$ *in Phase* 2 *of OciorRBA*$[(\text{ID}, p, l)]$ *should have the same input message $\boldsymbol{w}^\star$ at the beginning of Phase* 1 *of OciorRBA*$[(\text{ID}, p, l)]$*, for some $\boldsymbol{w}^\star$, for* $l \in \{0, 1\}$.

Proof. The result is directly derived from [7, Lemma 11]. □

Lemma 10 (Consistency and Totality Properties of OciorRBA)**.** *Assume that $\mathcal{S}_p$ has good resilience. If* $\text{ABBA}[(\text{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ *outputs* 1*, then all honest nodes within $\mathcal{S}_p$ eventually output the same message $\boldsymbol{w}^\star$ from OciorRBA*$[(\text{ID}, p, l)]$*, for some $\boldsymbol{w}^\star$, for* $l \in \{0, 1\}$.

Proof. The proof is similar to [7, Theorem 5]. If $\text{ABBA}[(\text{ID}, p, l, \tilde{n}_p, \tilde{t}_p)]$ outputs 1, then we have the following facts:

- Fact 1: Eventually every honest node within $\mathcal{S}_p$ will set $I_3 = 1$ and go to Phase 3 of OciorRBA$[(\text{ID}, p, l)]$ (Lemma 7).
- Fact 2: All of the honest nodes who set $\text{s}_i^{[2]} = 1$ in Phase 2 of OciorRBA$[(\text{ID}, p, l)]$ should have the same input message $\boldsymbol{w}^\star$ at the beginning of Phase 1 of OciorRBA$[(\text{ID}, p, l)]$ for some $\boldsymbol{w}^\star$ (Lemma 9).
- Fact 3: At least $\tilde{t}_p + 1$ honest nodes within $\mathcal{S}_p$ have set $\text{s}_i^{[2]} = 1$ from OciorRBA$[(\text{ID}, p, l)]$ (Lemma 8).

From Fact 2, if an honest node sets $\text{s}_i^{[2]} = 1$, then this node outputs the value $\boldsymbol{w}^\star$ in OciorRBA$[(\text{ID}, p, l)]$ (see Line 41 of Algorithm 4). From Facts 2 and 3, if an honest Node i sets $\text{s}_i^{[2]} = 0$, then it will eventually receives at least $\tilde{t}_p + 1$ matching $(\text{“SYMBOL”}, \text{id}, (\text{ECCEnc}_i(\tilde{n}_p, \tilde{k}_p, \boldsymbol{w}^\star), *))$ messages from the honest nodes within $\mathbb{S}_1^{[2]}$, where $\text{ECCEnc}_i()$ denotes the ith encoded symbol and $\tilde{k}_p$ is an encoding parameter. In this case, Node i will set $y_i^{(i)} \leftarrow \text{ECCEnc}_i(\tilde{n}_p, \tilde{k}_p, \boldsymbol{w}^\star)$ in Line 44 of Algorithm 4, and send

("CORRECT", id, $y_i^{(i)}$) to all nodes in Line 45. Therefore, every symbol $y_j^{(j)}$ sent from honest nodes and collected in $\mathbb{Y}_{\text{oec}}$ should be the symbol encoded from the same message $\boldsymbol{w}^\star$. Thus, every honest node who sets $\text{s}_i^{[2]} = 0$ will eventually decode the message $\boldsymbol{w}^\star$ with OEC decoding and output $\boldsymbol{w}^\star$ in Line 47 of Algorithm 4. □

Algorithm 5 OciorMVBArr protocol, with identifier ID, for $n \geq 5t + 1$. Code is shown for P_i.

*// ** OciorMVBArr: without any cryptographic assumption (other than common coin), with a relaxed resilience $n \geq 5t + 1$***
*// ** ACID[ID]: a protocol for n parallel asynchronous complete information dispersal (ACID) instances; once an ACID instance is complete, there exists a retrieval scheme to correctly retrieve its delivered message. ***
*// ** Election[(ID, r)]: an election protocol, requiring at least $t+1$ inputs from distinct nodes to generate an output l, for $r \in [1 : n]$***
*// ** ABA*rr*[(ID, l)] calls the asynchronous Byzantine agreement (ABA) protocol by Li-Chen [17], for $n \geq 5t + 1$, using only $O(1)$ common coins, $O(n|\boldsymbol{w}| + n^2 \log q)$ total bits, and $O(1)$ rounds, without any cryptographic assumption (other than common coin) ***

1: **upon** receiving MVBA input message $\boldsymbol{w}_i$ and $\text{Predicate}(\boldsymbol{w}_i) = \text{true}$ **do**:
2: $\quad\mathcal{S}_{\text{shares}} \leftarrow \text{ACIDrr[ID]}(\boldsymbol{w}_i)$ *// a protocol for n parallel ACID instances*
3: $\quad$**for** $r \in [1 : n]$ **do**
4: $\quad\quad l \leftarrow \text{Election}[(\text{ID}, r)]$ *// an election protocol*
5: $\quad\quad \bar{\boldsymbol{w}} \leftarrow \text{DRrr}[(\text{ID}, l)](\mathcal{S}_{\text{shares}}[l])$ *// shuffle the code symbols originally sent from Node l and decode*
6: $\quad\quad \hat{\boldsymbol{w}} \leftarrow \text{ABArr}[(\text{ID}, l)](\bar{\boldsymbol{w}})$ *// call the asynchronous BA protocol by Li-Chen [17]*
7: $\quad\quad$**if** $\text{Predicate}(\hat{\boldsymbol{w}}) = \text{true}$ **then**
8: $\quad\quad\quad$**output** $\hat{\boldsymbol{w}}$ and **terminate**

Algorithm 6 ACIDrr subprotocol with identifier ID for $t < \frac{n}{5}$. Code is shown for P_i

*// ** ACID[ID] is a protocol for n parallel ACID instances $\text{ACID}[(\text{ID}, 1)], \text{ACID}[(\text{ID}, 2)], \cdots, \text{ACID}[(\text{ID}, n)]$ ***
*// ** ACID[(ID, j)] is an ACID instance for delivering the message proposed from Node j ***
*// ** Once Node j completes ACID[(ID, j)], there exists a retrieval scheme to correctly retrieve the message. ***
*// ** When an honest node returns and stops this protocol, then at least $n - t$ ACID instances have been completed. ***

1: Initially set $\mathcal{S}_{\text{shares}}[j] \leftarrow \perp$, $\forall j \in [1 : n]$

*// ** ACID-share ***
2: **upon** receiving input message $\boldsymbol{w}_i$ **do**:
3: $\quad[y_1, y_2, \cdots, y_n] \leftarrow \text{ECCEnc}(n, t + 1, \boldsymbol{w}_i)$
4: $\quad$**send** ("SHARE", ID, y_j) to P_j, $\forall j \in [1 : n]$ *// exchange coded symbols*

*// ** ACID-vote ***
5: **upon** receiving ("SHARE", ID, y) from P_j for the first time **do**:
6: $\quad\mathcal{S}_{\text{shares}}[j] \leftarrow y$
7: $\quad$**send** ("VOTE", ID) to P_j

*// ** vote for election ***
8: **upon** receiving $n - t$ ("VOTE", ID) messages from distinct nodes **do**:
9: $\quad$**send** ("ELECTION", ID) to all nodes *// ACID[(ID, i)] is complete at this point*

*// ** confirm for election ***
10: **upon** receiving $n - t$ ("ELECTION", ID) messages from distinct nodes and ("CONFIRM", ID) not yet sent **do**:
11: $\quad$**send** ("CONFIRM", ID) to all nodes
12: **upon** receiving $t + 1$ ("CONFIRM", ID) messages from distinct nodes and ("CONFIRM", ID) not yet sent **do**:
13: $\quad$**send** ("CONFIRM", ID) to all nodes

*// ** return and stop ***
14: **upon** receiving $2t + 1$ ("CONFIRM", ID) messages from distinct nodes **do**:
15: $\quad$**if** ("CONFIRM", ID) not yet sent **then**
16: $\quad\quad$**send** ("CONFIRM", ID) to all nodes
17: $\quad$**return** $\mathcal{S}_{\text{shares}}$

Algorithm 7 DRrr subprotocol for $t < \frac{n}{5}$, with identifier $\mathrm{id} = (\mathrm{ID}, l)$. Code is shown for P_i.

Initially set $\mathbb{Y}_{\mathrm{Symbols}}[l] \leftarrow \{\}$
upon receiving input $\mathcal{S}_{\mathrm{shares}}[l]$, for $\mathcal{S}_{\mathrm{shares}}[l] := y^\star$ **do**:
    send ("ECHOSHARE", ID, $l, y^\star$) to all nodes
    wait for $|\mathbb{Y}_{\mathrm{Symbols}}[l]| = n - t$
        $\hat{\boldsymbol{w}} \leftarrow \mathrm{ECCDec}(n, t+1, \mathbb{Y}_{\mathrm{Symbols}}[l])$
        return $\hat{\boldsymbol{w}}$
upon receiving ("ECHOSHARE", ID, l, y) from P_j for the first time **do**:
    $\mathbb{Y}_{\mathrm{Symbols}}[l] \leftarrow \mathbb{Y}_{\mathrm{Symbols}}[l] \cup \{j : y\}$

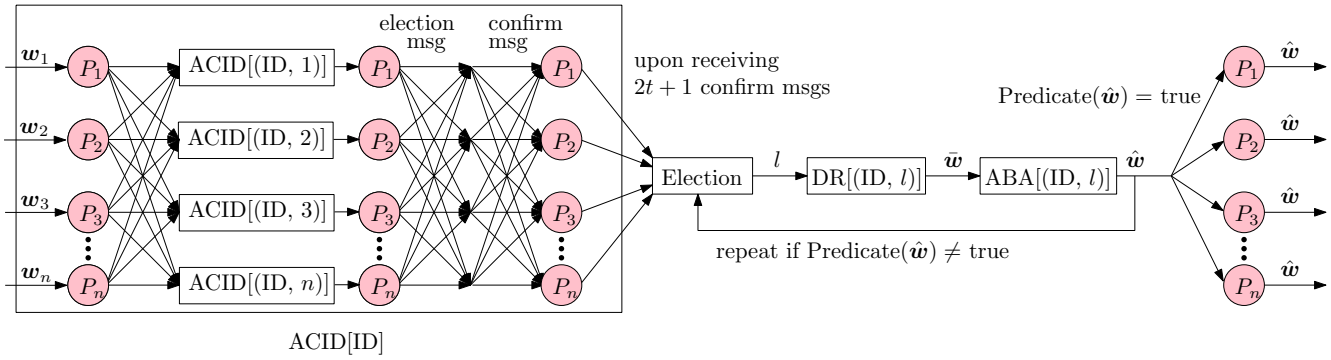

Fig. 2. A block diagram of the proposed OciorMVBArr protocol with an identifier ID.

III. OciorMVBArr

This proposed OciorMVBArr is an error-free, information-theoretically secure asynchronous MVBA protocol, with relaxed resilience $n \geq 5t + 1$. OciorMVBArr does not rely on any cryptographic assumptions, such as signatures or hashing, except for the common coin assumption.

A. Overview of the proposed OciorMVBArr protocol

The proposed OciorMVBArr is described in Algorithm 5, along with Algorithms 6 and 7. Fig. 2 presents a block diagram of the proposed OciorMVBArr protocol. The proposed OciorMVBArr consists of the algorithms ACIDrr[ID], Election$[(\mathrm{ID}, r)]$, DRrr$[(\mathrm{ID}, l)]$, and ABArr$[(\mathrm{ID}, l)]$ for $r, l \in [1 : n]$.

- ACIDrr[ID]: This is a protocol for n parallel ACID instances: ACID$[(\mathrm{ID}, 1)]$, ACID$[(\mathrm{ID}, 2)]$, $\cdots$, ACID$[(\mathrm{ID}, n)]$. Once an ACID instance is complete, there exists a retrieval scheme to correctly retrieve its delivered message.
- Election$[(\mathrm{ID}, r)]$: This is an election protocol that requires at least $t + 1$ inputs from distinct nodes to generate a random value l, where $r \in [1 : n]$.
- DRrr$[(\mathrm{ID}, l)]$: An ACID$[(\mathrm{ID}, l)]$ protocol is complemented by a data retrieval protocol DR$[(\mathrm{ID}, l)]$, in which each node retrieves the message proposed by P_l from n distributed nodes.
  - If Node l is honest and has completed ACID$[(\mathrm{ID}, l)]$, then during DRrr$[(\mathrm{ID}, l)]$, each node will receive at least $2t + 1$ shares generated from Node l, given $n \geq 5t + 1$. In this case all honest node eventually output the same message from DRrr$[(\mathrm{ID}, l)]$.
  - Even if Node l is dishonest, ABArr$[(\mathrm{ID}, l)]$ ensures that all honest nodes output the same message.
- ABArr$[(\mathrm{ID}, l)]$: This is an asynchronous Byzantine agreement protocol that calls the protocol by Li-Chen [17] for $n \geq 5t+1$. It uses only $O(1)$ common coins, $O(n|\boldsymbol{w}| + n^2 \log q)$ total bits, and $O(1)$ rounds, without relying on any cryptographic assumptions, except for the common coin assumption.

B. Analysis of OciorMVBArr

Theorem 4 (Agreement)**.** *In* OciorMVBArr*, given $n \geq 5t + 1$, if any two honest nodes output $\boldsymbol{w}'$ and $\boldsymbol{w}''$, respectively, then $\boldsymbol{w}' = \boldsymbol{w}''$.*

Proof. In $\mathrm{OciorMVBArr}$, if any two honest nodes output values at Rounds r and r' (see Line 3 of Algorithm 5), respectively, then $r = r'$, due to the consistency property of the protocols $\mathrm{Election}[(\mathrm{ID}, r)]$ and $\mathrm{ABArr}[(\mathrm{ID}, l)]$. Moreover, at the same round r, if any two honest nodes output $\boldsymbol{w}'$ and $\boldsymbol{w}''$, respectively, then $\boldsymbol{w}' = \boldsymbol{w}''$, due to the consistency property of the protocol $\mathrm{ABArr}[(\mathrm{ID}, l)]$. □

Theorem 5 (Termination)**.** *In* $\mathrm{OciorMVBArr}$*, given* $n \geq 5t + 1$*, every honest node eventually outputs a value and terminates.*

Proof. In this setting, every honest node eventually returns $\mathcal{S}_{\mathrm{shares}}$ and terminates from the protocol $\mathrm{ACIDrr}[\mathrm{ID}]$, due to the Termination property of this protocol. Furthermore, by the Integrity property of $\mathrm{ACIDrr}[\mathrm{ID}]$, if an honest node returns $\mathcal{S}_{\mathrm{shares}}$ and terminates from the protocol $\mathrm{ACIDrr}[\mathrm{ID}]$, then there exists a set $\mathcal{I}^\star$ such that the following conditions hold:) $\mathcal{I}^\star \subseteq [1:n] \setminus \mathcal{F}$, where $\mathcal{F}$ denotes the set of indexes of all dishonest nodes; 2) $|\mathcal{I}^\star| \geq n - 2t$; and 3) for any $i \in \mathcal{I}^\star$, P_i has completed the dispersal $\mathrm{ACID}[(\mathrm{ID}, i)]$.

Subsequently, every honest node eventually runs $l \leftarrow \mathrm{Election}[(\mathrm{ID}, r)]$ in Line 4 of Algorithm 5, at the same round r. If Node l is honest and $l \in \mathcal{I}^\star$, then during $\mathrm{DRrr}[(\mathrm{ID}, l)]$, each node will receive at least $2t + 1$ shares generated from Node l, given $n \geq 5t + 1$. In this case all honest node eventually output the same message from both $\mathrm{DRrr}[(\mathrm{ID}, l)]$ and $\mathrm{ABArr}[(\mathrm{ID}, l)]$, and then terminate. If Node l is dishonest and the message $\hat{\boldsymbol{w}}$ output by $\mathrm{ABArr}[(\mathrm{ID}, l)]$ in Line 6 does not satisfy $\mathrm{Predicate}(\hat{\boldsymbol{w}}) = \mathrm{true}$, then all honest nodes proceed to the next round. All honest node eventually terminates if, at some round r, $\mathrm{Election}[(\mathrm{ID}, r)]$ outputs a value l such that Node l is honest and $l \in \mathcal{I}^\star$. □

Theorem 6 (External Validity)**.** *In* $\mathrm{OciorMVBArr}$*, if an honest node outputs a value* $\boldsymbol{w}$*, then* $\mathrm{Predicate}(\boldsymbol{w}) = \mathrm{true}$*.*

Proof. In $\mathrm{OciorMVBArr}$, if an honest node outputs a value $\boldsymbol{w}$, it has verified that $\mathrm{Predicate}(\boldsymbol{w}) = \mathrm{true}$ at Line 7 of Algorithm 5. □

Algorithm 8 $\mathrm{OciorMVBAh}$ protocol, with identifier ID, for $n \geq 3t + 1$. Code is shown for P_i.

*// ** Merkle tree is implemented here for vector commitment based on hashing ***
*// ** $\mathrm{VcCom}()$ outputs a commitment, i.e., Merkle root, with $O(\kappa)$ bits ***
*// ** $\mathrm{VcOpen}()$ returns a proof that the targeted value is the committed element of the vector ***
*// ** $\mathrm{VcVerify}(j, C, y, \omega)$ returns true only if ω is a valid proof that C is the commitment of a vector whose jth element is y ***

1: **upon** receiving MVBA input message $\boldsymbol{w}_i$ and $\mathrm{Predicate}(\boldsymbol{w}_i) = \mathrm{true}$ **do**:
2: $[\mathcal{L}_{\mathrm{lock}}, \mathcal{R}_{\mathrm{ready}}, \mathcal{F}_{\mathrm{finish}}, \mathcal{S}_{\mathrm{shares}}] \leftarrow \mathrm{ACIDh}[\mathrm{ID}](\boldsymbol{w}_i)$ *// a protocol for n parallel* ACID *instances*
3: **for** $r \in [1:n]$ **do**
4: $l \leftarrow \mathrm{Election}[(\mathrm{ID}, r)]$ *// an election protocol*
5: $a \leftarrow \mathrm{ABBBA}[(\mathrm{ID}, l)](\mathcal{R}_{\mathrm{ready}}[l], \mathcal{F}_{\mathrm{finish}}[l])$ *// asynchronous biased binary Byzantine agreement (*ABBBA*)*
6: $b \leftarrow \mathrm{ABBA}[(\mathrm{ID}, l)](a)$ *// an asynchronous binary Byzantine agreement (*ABBA*)*
7: **if** $b = 1$ **then**
8: $\hat{\boldsymbol{w}}_l \leftarrow \mathrm{DRh}[(\mathrm{ID}, l)](\mathcal{L}_{\mathrm{lock}}[l], \mathcal{S}_{\mathrm{shares}}[l])$ *// Data Retrieval (DR)*
9: **if** $\mathrm{Predicate}(\hat{\boldsymbol{w}}_l) = \mathrm{true}$ **then**
10: **output** $\hat{\boldsymbol{w}}_l$ and **terminate**

Algorithm 9 ACIDh subprotocol with identifier ID, based on hashing. Code is shown for P_i

*// ** ACID[ID] is a protocol for n parallel ACID instances $\mathrm{ACID}[(\mathrm{ID},1)], \mathrm{ACID}[(\mathrm{ID},2)], \cdots, \mathrm{ACID}[(\mathrm{ID},n)]$ ***
*// ** $\mathrm{ACID}[(\mathrm{ID},j)]$ is an ACID instance for delivering the message proposed from Node j ***
*// ** Once Node j completes $\mathrm{ACID}[(\mathrm{ID},j)]$, there exists a retrieval scheme to correctly retrieve the message ***
*// ** When an honest node returns and stops this protocol, then at least $n-t$ ACID instances have been completed ***
*// ** ECEnc() and ECDec() are encoding function and decoding function of (n,k) erasure code ***

// initialization:
$\mathcal{L}_{\mathrm{lock}} \leftarrow \{\}; \mathcal{R}_{\mathrm{ready}} \leftarrow \{\}; \mathcal{F}_{\mathrm{finish}} \leftarrow \{\}; \mathcal{S}_{\mathrm{shares}} \leftarrow \{\}; \mathcal{H}_{\mathrm{hash}} \leftarrow \{\}$
for $j \in [1:n]$ **do**
  $\mathcal{S}_{\mathrm{shares}}[j] \leftarrow (\perp, \perp, \perp); \mathcal{L}_{\mathrm{lock}}[j] \leftarrow 0; \mathcal{R}_{\mathrm{ready}}[j] \leftarrow 0; \mathcal{F}_{\mathrm{finish}}[j] \leftarrow 0$

*// ** ACID-share ***
upon receiving input message $\boldsymbol{w}_i$ **do**:
  $[y_1, y_2, \cdots, y_n] \leftarrow \mathrm{ECEnc}(n, t+1, \boldsymbol{w}_i)$
  $C \leftarrow \mathrm{VcCom}([y_1, y_2, \cdots, y_n])$
  for $j \in [1:n]$ **do**
    $\omega_j \leftarrow \mathrm{VcOpen}(C, y_j, j)$
    send $(\text{“SHARE”}, \mathrm{ID}, C, y_j, \omega_j)$ to P_j

*// ** ACID-vote ***
upon receiving $(\text{“SHARE”}, \mathrm{ID}, C, y, \omega)$ from P_j for the first time **do**:
  if $\mathrm{VcVerify}(i, C, y, \omega) = \mathrm{true}$ **then**
    $\mathcal{S}_{\mathrm{shares}}[j] \leftarrow (C, y, \omega); \mathcal{H}_{\mathrm{hash}}[C] \leftarrow j$
    send $(\text{“VOTE”}, \mathrm{ID}, C)$ to all nodes

*// ** ACID-lock ***
upon receiving $n-t$ $(\text{“VOTE”}, \mathrm{ID}, C)$ messages from distinct nodes, for the same C **do**:
  wait until $C \in \mathcal{H}_{\mathrm{hash}}$
  $j^\star \leftarrow \mathcal{H}_{\mathrm{hash}}[C]; \mathcal{L}_{\mathrm{lock}}[j^\star] \leftarrow 1$
  send $(\text{“LOCK”}, \mathrm{ID}, C)$ to all nodes

*// ** ACID-ready ***
upon receiving $n-t$ $(\text{“LOCK”}, \mathrm{ID}, C)$ messages from distinct nodes, for the same C **do**:
  wait until $C \in \mathcal{H}_{\mathrm{hash}}$
  $j^\star \leftarrow \mathcal{H}_{\mathrm{hash}}[C]; \mathcal{R}_{\mathrm{ready}}[j^\star] \leftarrow 1$
  send $(\text{“READY”}, \mathrm{ID}, C)$ to all nodes

*// ** ACID-finish ***
upon receiving $n-t$ $(\text{“READY”}, \mathrm{ID}, C)$ messages from distinct nodes, for the same C **do**:
  wait until $C \in \mathcal{H}_{\mathrm{hash}}$
  $j^\star \leftarrow \mathcal{H}_{\mathrm{hash}}[C]; \mathcal{F}_{\mathrm{finish}}[j^\star] \leftarrow 1$
  send $(\text{“FINISH”}, \mathrm{ID})$ to $P_{j^\star}$

*// ** vote for election ***
upon receiving $n-t$ $(\text{“FINISH”}, \mathrm{ID})$ messages from distinct nodes **do**:
  send $(\text{“ELECTION”}, \mathrm{ID})$ to all nodes *// $\mathrm{ACID}[(\mathrm{ID}, i)]$ is complete at this point*

*// ** confirm for election ***
upon receiving $n-t$ $(\text{“ELECTION”}, \mathrm{ID})$ messages from distinct nodes and $(\text{“CONFIRM”}, \mathrm{ID})$ not yet sent **do**:
  send $(\text{“CONFIRM”}, \mathrm{ID})$ to all nodes
upon receiving $t+1$ $(\text{“CONFIRM”}, \mathrm{ID})$ messages from distinct nodes and $(\text{“CONFIRM”}, \mathrm{ID})$ not yet sent **do**:
  send $(\text{“CONFIRM”}, \mathrm{ID})$ to all nodes

*// ** return and stop ***
upon receiving $2t+1$ $(\text{“CONFIRM”}, \mathrm{ID})$ messages from distinct nodes **do**:
  if $(\text{“CONFIRM”}, \mathrm{ID})$ not yet sent **then**
    send $(\text{“CONFIRM”}, \mathrm{ID})$ to all nodes
  return $[\mathcal{L}_{\mathrm{lock}}, \mathcal{R}_{\mathrm{ready}}, \mathcal{F}_{\mathrm{finish}}, \mathcal{S}_{\mathrm{shares}}]$

Algorithm 10 DRh subprotocol, with identifier $\mathrm{id} = (\mathrm{ID}, l)$, based on hashing. Code is shown for P_i.

Initially set $\mathbb{Y}_{\mathrm{Symbols}}[l] \leftarrow \{\}$
upon receiving input $(\mathrm{lock\_indicator}, \mathrm{share})$ **do**:
  if $(\mathrm{lock\_indicator} = 1) \wedge (\mathrm{share} \neq (\perp, \perp, \perp))$ **then**
    $(C^\star, y^\star, \omega^\star) \leftarrow \mathrm{share}$
    send ("ECHOSHARE", $\mathrm{ID}, l, C^\star, y^\star, \omega^\star$) to all nodes
  wait for $|\mathbb{Y}_{\mathrm{Symbols}}[l][C]| = t + 1$ for some C
    $\hat{\boldsymbol{w}} \leftarrow \mathrm{ECDec}(n, t+1, \mathbb{Y}_{\mathrm{Symbols}}[l][C])$
    if $\mathrm{VcCom}(\mathrm{ECEnc}(n, t+1, \hat{\boldsymbol{w}})) = C$ **then**
      return $\hat{\boldsymbol{w}}$
    else
      return $\perp$
upon receiving ("ECHOSHARE", $\mathrm{ID}, l, C, y, \omega$) from P_j for the first time, for some C, y, ω **do**:
  if $\mathrm{VcVerify}(j, C, y, \omega) = \mathrm{true}$ **then**
    if $C \notin \mathbb{Y}_{\mathrm{Symbols}}[l]$ **then**
      $\mathbb{Y}_{\mathrm{Symbols}}[l][C] \leftarrow \{j : y\}$
    else
      $\mathbb{Y}_{\mathrm{Symbols}}[l][C] \leftarrow \mathbb{Y}_{\mathrm{Symbols}}[l][C] \cup \{j : y\}$

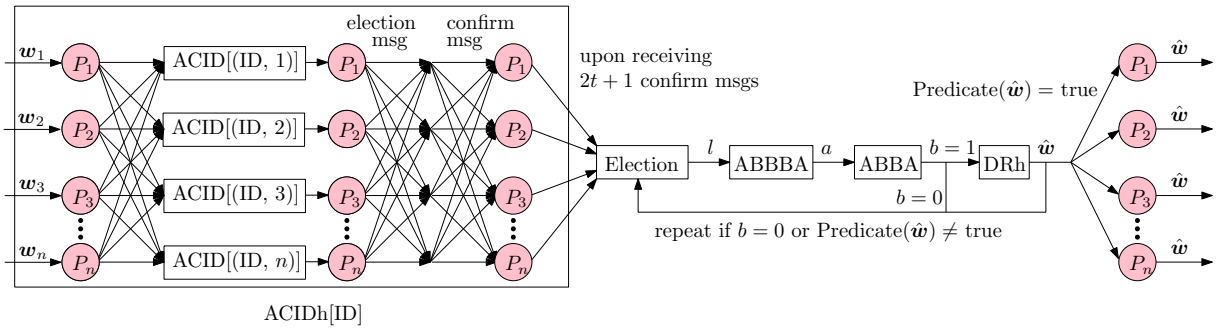

Fig. 3. A block diagram of the proposed OciorMVBAh protocol with an identifier ID.

IV. OciorMVBAh

This proposed OciorMVBAh is a hash-based asynchronous MVBA protocol. OciorMVBAh achieves consensus with a communication complexity of $O(n|\boldsymbol{w}| + \kappa n^3)$ bits, an expected round complexity of $O(1)$ rounds, and an expected $O(1)$ number of common coins, given $n \geq 3t + 1$.

A. *Overview of the proposed* OciorMVBAh *protocol*

The proposed OciorMVBAh is described in Algorithm 8, along with Algorithms 2, 9, and 10. Fig. 3 presents a block diagram of the proposed OciorMVBAh protocol. In this protocol, we use a vector commitment implemented with a Merkle tree based on hashing.

Vector commitment. A vector commitment consists of the following algorithms:

- $\mathrm{VcCom}(\boldsymbol{y}) \rightarrow C$: Given an input vector $\boldsymbol{y} = [y_1, y_2, \cdots, y_n]$ of size n, this algorithm outputs a commitment C, i.e., Merkle root, with $O(\kappa)$ bits.
- $\mathrm{VcOpen}(C, y_j, j) \rightarrow \omega_j$: Given inputs (C, y_j, j), this algorithm returns a value ω_j to prove that the targeted value y_j is the jth committed element of the vector.
- $\mathrm{VcVerify}(j, C, y_j, \omega_j) \rightarrow \mathrm{true/false}$: This algorithm returns true only if ω_j is a valid proof that C is the commitment of a vector whose jth element is y_j.

The proposed OciorMVBAh consists of the algorithms ACIDh[ID], Election[(ID, r)], ABBBA[(ID, l)], ABBA[(ID, l)], and DRh[(ID, l)], for $r, l \in [1 : n]$.

- ACIDrr[ID]: This is a protocol for n parallel ACID instances: $\mathrm{ACID}[(\mathrm{ID}, 1)]$, $\mathrm{ACID}[(\mathrm{ID}, 2)], \cdots, \mathrm{ACID}[(\mathrm{ID}, n)]$. Once an ACID instance is complete, there exists a retrieval scheme to correctly retrieve its delivered message.
- $\mathrm{Election}[(\mathrm{ID}, r)]$: This is an election protocol that requires at least $t+1$ inputs from distinct nodes to generate a random value l, where $r \in [1:n]$.
- $\mathrm{ABBBA}[(\mathrm{ID}, l)]$: This is an asynchronous biased binary BA protocol. It has two inputs (a_1, a_2), for some $a_1, a_2 \in \{0, 1\}$. It guarantees the following properties:
  - Conditional termination: Under an input condition—i.e., if one honest node inputs its second number as $a_2 = 1$ then at least $t+1$ honest nodes input their first numbers as $a_1 = 1$—then every honest node eventually outputs a value and terminates.
  - Biased validity: If $t+1$ honest nodes input the second number as $a_2 = 1$, then any honest node that terminates outputs 1.
  - Biased integrity: If an honest node outputs 1, then at least one honest node inputs $a_1 = 1$ or $a_2 = 1$.
- $\mathrm{ABBA}[(\mathrm{ID}, l)]$: This is an asynchronous binary BA protocol.
- $\mathrm{DRh}[(\mathrm{ID}, l)]$: This is a data retrieval protocol associated with an $\mathrm{ACID}[(\mathrm{ID}, l)]$ protocol. It is activated only if $b = 1$ (see Line 7 of Algorithm 8), where b is the output of $\mathrm{ABBA}[(\mathrm{ID}, l)]$.
  - The instance of $b = 1$ reveals that at least one honest node outputs $a = 1$ from $\mathrm{ABBBA}[(\mathrm{ID}, l)]$, which further suggests that at least one honest node inputs $\mathcal{R}_{\mathrm{ready}}[l] = 1$ or $\mathcal{F}_{\mathrm{finish}}[l] = 1$ into $\mathrm{ABBBA}[(\mathrm{ID}, l)]$, based on the biased integrity of ABBBA.
  - When one honest node inputs $\mathcal{R}_{\mathrm{ready}}[l] = 1$ or $\mathcal{F}_{\mathrm{finish}}[l] = 1$, it is guaranteed that at least $n - 2t$ honest nodes have stored correct shares sent from Node l (see Lines 15 and 17 of Algorithm 9), which implies that every honest node eventually retrieves the same message from $\mathrm{DRh}[(\mathrm{ID}, l)]$.

B. Analysis of OciorMVBAh

Theorem 7 (Agreement)**.** *In* OciorMVBAh*, given $n \geq 3t+1$, if any two honest nodes output $\boldsymbol{w}'$ and $\boldsymbol{w}''$, respectively, then $\boldsymbol{w}' = \boldsymbol{w}''$.*

Proof. In OciorMVBAh, if any two honest nodes output values at Rounds r and r' (see Line 3 of Algorithm 8), respectively, then $r = r'$. This follows from the consistency property of the protocols $\mathrm{Election}[(\mathrm{ID}, r)]$ and $\mathrm{ABBA}[(\mathrm{ID}, l)]$, as well as the consistency property of the $\mathrm{DRh}[(\mathrm{ID}, l)]$ protocol when $\mathrm{ABBA}[(\mathrm{ID}, l)]$ outputs 1 (see Lemma 11).

Moreover, at the same round r, if any two honest nodes output $\boldsymbol{w}'$ and $\boldsymbol{w}''$, respectively, then $\boldsymbol{w}' = \boldsymbol{w}''$, due to the consistency property of the protocol $\mathrm{DRh}[(\mathrm{ID}, l)]$ when $\mathrm{ABBA}[(\mathrm{ID}, l)]$ outputs 1 (see Lemma 11). It is worth noting that $\mathrm{DRh}[(\mathrm{ID}, l)]$ is activated only if $\mathrm{ABBA}[(\mathrm{ID}, l)]$ outputs 1 (see Line 7 of Algorithm 8). □

Theorem 8 (Termination)**.** *In* OciorMVBAh*, given $n \geq 3t+1$, every honest node eventually outputs a value and terminates.*

Proof. In this setting, every honest node eventually returns values and terminates from the protocol ACIDh[ID], due to the Termination property of this protocol. Furthermore, by the Integrity property of ACIDh[ID], if an honest node returns values and terminates from the protocol ACIDh[ID], then there exists a set $\mathcal{I}^\star$ such that the following conditions hold:) $\mathcal{I}^\star \subseteq [1:n] \setminus \mathcal{F}$, where $\mathcal{F}$ denotes the set of indexes of all dishonest nodes; 2) $|\mathcal{I}^\star| \geq n - 2t$; and 3) for any $i \in \mathcal{I}^\star$, P_i has completed the dispersal $\mathrm{ACID}[(\mathrm{ID}, i)]$.

Subsequently, every honest node eventually runs $l \leftarrow \mathrm{Election}[(\mathrm{ID}, r)]$ in Line 4 of Algorithm 8, at the same round r. If Node l is honest and $l \in \mathcal{I}^\star$, then $\mathrm{ABBA}[(\mathrm{ID}, l)]$ eventually outputs 1 (due to the biased validity property of $\mathrm{ABBBA}[(\mathrm{ID}, l)]$) and then during $\mathrm{DRh}[(\mathrm{ID}, l)]$ each node will receive at least $t+1$ correct shares generated from Node l, given $n \geq 3t+1$. In this case all honest node eventually

output the same message from $\mathrm{DRh}[(\mathrm{ID}, l)]$, and then terminate. If Node l is dishonest or $\mathrm{ABBA}[(\mathrm{ID}, l)]$ outputs 0, then all honest nodes proceed to the next round. All honest node eventually terminates if, at some round r, $\mathrm{Election}[(\mathrm{ID}, r)]$ outputs a value l such that Node l is honest and $l \in \mathcal{I}^\star$. □

Theorem 9 (External Validity)**.** *In* $\mathrm{OciorMVBAh}$*, if an honest node outputs a value* $\boldsymbol{w}$*, then* $\mathrm{Predicate}(\boldsymbol{w}) = \mathrm{true}$*.*

Proof. In $\mathrm{OciorMVBAh}$, if an honest node outputs a value $\boldsymbol{w}$, it has verified that $\mathrm{Predicate}(\boldsymbol{w}) = \mathrm{true}$ at Line 9 of Algorithm 8. □

Lemma 11. *In* $\mathrm{OciorMVBAh}$*, if* $\mathrm{ABBA}[(\mathrm{ID}, l)]$ *outputs* 1*, then every honest node eventually retrieves the same message from* $\mathrm{DRh}[(\mathrm{ID}, l)]$*, for* $l \in [1 : n]$*.*

Proof. In $\mathrm{OciorMVBAh}$, $\mathrm{DRh}[(\mathrm{ID}, l)]$ is activated only if $b = 1$ (see Line 7 of Algorithm 8), where b is the output of $\mathrm{ABBA}[(\mathrm{ID}, l)]$. The instance of $b = 1$ reveals that at least one honest node outputs $a = 1$ from $\mathrm{ABBBA}[(\mathrm{ID}, l)]$, which further suggests that at least one honest node inputs $\mathcal{R}_{\mathrm{ready}}[l] = 1$ or $\mathcal{F}_{\mathrm{finish}}[l] = 1$ into $\mathrm{ABBBA}[(\mathrm{ID}, l)]$, based on the biased integrity of ABBBA. When one honest node inputs $\mathcal{R}_{\mathrm{ready}}[l] = 1$ or $\mathcal{F}_{\mathrm{finish}}[l] = 1$, it is guaranteed that at least $n - 2t$ honest nodes have stored correct shares sent from Node l (see Lines 15 and 17 of Algorithm 9), which implies that every honest node eventually retrieves the same message from $\mathrm{DRh}[(\mathrm{ID}, l)]$. □

REFERENCES

[1] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup, "Secure and efficient asynchronous broadcast protocols," in *Advances in Cryptology–CRYPTO 2001. Lecture Notes in Computer Science*, vol. 2139, Aug. 2001.

[2] M. Pease, R. Shostak, and L. Lamport, "Reaching agreement in the presence of faults," *Journal of the ACM*, vol. 27, no. 2, pp. 228–234, Apr. 1980.

[3] M. Fischer, N. Lynch, and M. Paterson, "Impossibility of distributed consensus with one faulty process," *Journal of the ACM*, vol. 32, no. 2, pp. 374–382, Apr. 1985.

[4] M. Sipser and D. Spielman, "Expander codes," *IEEE Trans. Inf. Theory*, vol. 42, no. 6, pp. 1710–1722, Nov. 1996.

[5] J. Chen, "Fundamental limits of Byzantine agreement," 2020, available on ArXiv: https://arxiv.org/pdf/2009.10965.pdf.

[6] ——, "Optimal error-free multi-valued Byzantine agreement," in *International Symposium on Distributed Computing (DISC)*, Oct. 2021.

[7] ——, "OciorCOOL: Faster Byzantine agreement and reliable broadcast," Sep. 2024, available on ArXiv: https://arxiv.org/abs/2409.06008.

[8] I. Abraham, D. Malkhi, and A. Spiegelman, "Asymptotically optimal validated asynchronous Byzantine agreement," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, Jul. 2019, pp. 337–346.

[9] Y. Lu, Z. Lu, Q. Tang, and G. Wang, "Dumbo-MVBA: Optimal multi-valued validated asynchronous Byzantine agreement, revisited," in *Proceedings of the ACM Symposium on Principles of Distributed Computing (PODC)*, Jul. 2020, pp. 129–138.

[10] S. Duan, X. Wang, and H. Zhang, "FIN: Practical signature-free asynchronous common subset in constant time," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, 2023, pp. 815–829.

[11] H. Feng, Z. Lu, T. Mai, and Q. Tang, "Making hash-based MVBA great again," Mar. 2024, available on: https://eprint.iacr.org/2024/479.

[12] J. Komatovic, J. Neu, and T. Roughgarden, "Toward optimal-complexity hash-based asynchronous MVBA with optimal resilience," Oct. 2024, available on ArXiv: https://arxiv.org/abs/2410.12755.

[13] I. Reed and G. Solomon, "Polynomial codes over certain finite fields," *Journal of the Society for Industrial and Applied Mathematics*, vol. 8, no. 2, pp. 300–304, Jun. 1960.

[14] R. Roth, *Introduction to coding theory*. Cambridge University Press, 2006.

[15] E. Berlekamp, "Nonbinary BCH decoding (abstr.)," *IEEE Trans. Inf. Theory*, vol. 14, no. 2, pp. 242–242, Mar. 1968.

[16] M. Ben-Or, R. Canetti, and O. Goldreich, "Asynchronous secure computation," in *Proceedings of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, 1993, pp. 52–61.

[17] F. Li and J. Chen, "Communication-efficient signature-free asynchronous Byzantine agreement," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Jul. 2021.