Signature-Free Atomic Broadcast with Optimal $O(n^2)$ Messages and $O(1)$ Expected Time

Xiao Sui[1] and Sisi Duan[2]

[1] Shandong University <suixiao@mail.sdu.edu.cn>
[2] Tsinghua University <duansisi@tsinghua.edu.cn>

Abstract. Byzantine atomic broadcast (ABC) is at the heart of permissioned blockchains and various multi-party computation protocols. We resolve a long-standing open problem in ABC, presenting the first information-theoretic (IT) and signature-free asynchronous ABC protocol that achieves optimal $O(n^2)$ messages and $O(1)$ expected time. Our ABC protocol adopts a new design, relying on a reduction from—perhaps surprisingly—a somewhat neglected primitive called multivalued Byzantine agreement (MBA).

1 Introduction

Byzantine atomic broadcast (ABC) protocols, or Byzantine fault-tolerant (BFT) protocols, are at the core of state machine replication, permissioned blockchains, and various cryptographic protocols such as multi-party computation (MPC). Among these ABC protocols, completely asynchronous ABC protocols with no timing assumptions [3,7,9,16,24,28,29,35,37,47] have been receiving considerable attention, due to their intrinsic robustness against performance and denial-of-service (DoS) attacks.

IT and signature-free settings. The celebrated FLP impossibility result rules out the possibility of deterministic asynchronous consensus protocols [26], so asynchronous consensus protocols must be randomized to be probabilistically live. In practice, one can use either local coins (flipping a coin locally and independently at each replica) or common coins (using a common coin available for all replicas) [43]. Consensus protocols using local coins, however, terminate in exponential expected time [18, 38, 48]. Thus, to avoid exponential running time, asynchronous consensus protocols need to use common coins.

We follow a long line of work in consensus [7, 9, 18, 19, 36, 39–42, 47] and call the setting using *common coins only* the information-theoretical (IT) setting, the signature-free setting, or the cryptography-free setting (which we will use interchangeably in the paper).

Known results in the signature-free setting. In the consensus problem, every replica holds a message, and all replicas want to agree on one (or a set of) message(s). Notable asynchronous consensus primitives include asynchronous binary agreement (ABA), asynchronous multivalued Byzantine agreement (MBA), and asynchronous ABC. Informally speaking, ABA reaches agreement on binary values and MBA reaches agreement on values from an arbitrary domain, while ABC reaches agreement on the order of a sequence of messages.

As one of the most celebrated (and also surprising) results in consensus, Mostéfaoui, Moumen, and Raynal (MMR) demonstrated that by relying on common coins only, one can build a signature-free ABA protocol with optimal resilience, optimal $O(n^2)$ messages and $O(1)$ expected time [39, 40]. The work is enormously impactful in both theory and practice: the state-of-the-art ABC protocols either use their ABA protocols or their derivatives (such as Cobalt ABA [36], Crain's ABA [19], Pillar [47]). In the same setting, Mostéfaoui and Raynal (MR) presented the first signature-free asynchronous multivalued Byzantine agreement (MBA) with optimal $O(n^2)$ messages and $O(1)$ expected time [42] by reducing MBA to ABA.

The open problem. Unlike ABA and MBA, the following problem remains open for ABC:

Does there exist a signature-free ABC protocol with the optimal $O(n^2)$ messages and $O(1)$ expected time?

Note that the problem for ABC *appears* harder than that of ABA and MBA. Intuitively, ABC is concerned about ordering a sequence of messages, while ABA and MBA aim to achieve consensus for one-shot messages.

To the best of our knowledge, no solutions are known for the open problem for ABC, even if we relax it to consider sublinear time complexity. Indeed, as surveyed in Table 1, existing ABC protocols in the signature-free setting have $O(n^3)$ messages, and $O(1)$ or $O(\log n)$ expected time. This is in sharp contrast to the computational setting (that uses threshold signatures and trusted setup), the paradigm proposed by Cachin, Kursawe, Petzold, and Shoup [16]—using multivalued validated Byzantine agreement (MVBA)—leads to ABC protocols with $O(1)$ expected time and optimal $O(n^2)$ messages.

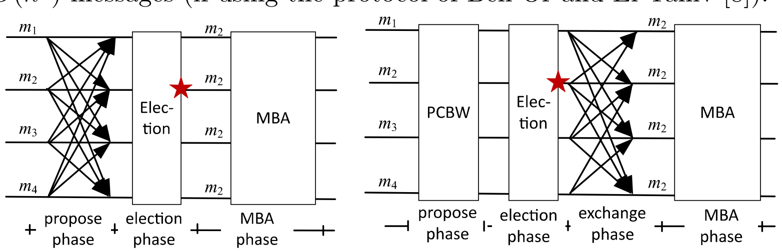
This paper solves this long-standing open problem, demonstrating the first signature-free ABC protocol called SQ with the optimal $O(n^2)$ messages and $O(1)$ expected time.

Our approach: ABC from MBA. Despite being a natural and classic primitive in consensus, multivalued Byzantine agreement (MBA) does not seem to be as "useful" as its binary counterpart (ABA). Indeed, while there exist transformations from MBA to ABC [18, 38], these ABC protocols have $O(n)$ time and $O(n^4)$ messages (even if we instantiate them using best-available subprotocols)—far more expensive than any of the ABC protocols in Table 1. Note that the situation for MBA is also in sharp contrast to its computational and validated version—multivalued validated Byzantine agreement (MVBA) [16] which can be used to build various high-level protocols such as state-of-the-art ABC protocols [3, 35]. Indeed, despite the similarities between MBA and MVBA, they are fundamentally different primitives: MBA does not directly imply MVBA, and MVBA does not directly imply MBA either[3].

[3] In particular, there exist MVBA protocols such that their non-validated versions do not satisfy the validity property of the MBA (see Sec. 2 for the definition of validity).

	paradigm	protocol	time	message
Computational	MVBA-based	CKPS [16]	$O(1)$	$O(n^2)$
		Dumbo [29]	$O(1)$	$O(n^3)$
		Speeding-Dumbo [28]	$O(1)$	$O(n^2)$
		AMS [3]	$O(1)$	$O(n^2)$
		Dumbo-MVBA [35]	$O(1)$	$O(n^2)$
Information-Theoretic (by design)	ABA-based	BKR [9]	$O(\log n)/O(1)$	$O(n^3)/O(n^4)$
		HoneyBadger [37]	$O(\log n)$	$O(n^3)$
		BEAT [24]	$O(\log n)$	$O(n^3)$
		EPIC [33]	$O(\log n)$	$O(n^3)$
	RABA-based	PACE [47]	$O(\log n)$	$O(n^3)$
		FIN [25]	$O(1)$	$O(n^3)$
	DAG-based	DAG-Rider [31]	$O(1)$	$O(n^3)$
	MBA-based	**SQ (this work)**	$\boldsymbol{O(1)}$	$\boldsymbol{O(n^2)}$

Table 1: Comparison of ABC protocols with sublinear time complexity. RABA denotes reproposable ABA [25, 47]. DAG denotes directed acyclic graph. Note that the implemented systems in the information-theoretic (IT) category (HoneyBadger, BEAT, EPIC, PACE, FIN) are not IT-secure systems, but they—"by design"—are IT-secure; here in this table we mean the underlying, "ideal" protocols in these systems by assuming ideal building blocks such as reliable broadcast (RBC), ABA, and common coins. As mentioned in BKR [9], their protocol can have either $O(\log n)$ expected time and $O(n^3)$ messages, or $O(1)$ expected time and $O(n^4)$ messages (if using the protocol of Ben-Or and El-Yaniv [8]).



(a) Our toy construction. (b) Overview of SQ.

Fig. 1: Overview of our approach.

In this paper, we challenge the conventional wisdom and show that we can use MBA to build a signature-free ABC protocol with optimal message and time complexity. Our starting point, as illustrated in Figure 1a, is a toy construction attempting to reduce ABC to MBA. In this construction, replicas proceed in epochs. In an epoch r, each replica p_i broadcasts its proposed message m_i. After receiving $n - f$ proposed messages, replicas run a random leader election protocol which outputs a random leader p_{k_r}. If a replica has previously received the proposed message from p_{k_r}, it provides the received proposed message as input to MBA. Otherwise, the replica simply waits until it receives the proposed message from p_{k_r}. If MBA outputs some value m, p_i delivers m as the ABC output. Meanwhile, a replica can start a new epoch before the MBA instance in the current epoch terminates.

Note that if the leader p_{k_r} is correct, every correct replica eventually receives the proposed message from p_{k_r}, provides the same input to MBA, and MBA will eventually output m_{k_r}. However, if a faulty replica p_{k_r} is selected, we cannot guarantee the termination of the protocol. Indeed, under the scenario, correct replicas in asynchronous environments cannot decide whether they should input $\perp$ (and complete the epoch) or simply wait for m_{k_r} (and stay in the epoch).

In our SQ protocol, we further develop the above idea, as depicted in Figure 1b. At the core of our fully-fledged protocol is ensuring the *existence* of a *key set* consisting of at least $f + 1$ correct replicas for each epoch r, such that if any replica in the key set is selected by the random leader election protocol, MBA will output a non-$\perp$ value.[4] Meanwhile, we ensure that if a replica outside the key set is selected, every correct replica will provide some input to MBA, so every MBA instance will terminate (and we are done). For this purpose, we introduce a new primitive called *parallel consistent broadcast with weak agreed set (PCBW)* and an *exchange* phase between the election phase and the MBA phase. PCBW has a nice feature we need for building ABC: once at least one correct replica terminates the PCBW instance in epoch r, a key set must have existed. If a replica in the key set is elected as the leader, the exchange phase further allows correct replicas that have not received the proposed message from the leader to provide the *correct* input to MBA, so MBA outputs a non-$\perp$ value!

In summary, we reduce the problem of ABC to PCBW and MBA. By providing an efficient PCBW construction with $O(n^2)$ messages and $O(1)$ time and using the state-of-the-art MBA construction, we are able to build an ABC protocol with $O(n^2)$ messages and $O(1)$ time. Additionally, the PCBW primitive itself might be of independent interest.

Summary of our contributions. In this paper, we present SQ, the first IT-secure and signature-free asynchronous ABC protocol that achieves optimal resilience, $O(n^2)$ messages, and $O(1)$ expected time (Sec. 4). In light of the lower bound result [3], our protocol is optimal in both time and message complexity.

We also suggest a communication-efficient variant of our SQ protocol, SQ_h, by additionally using hash functions (Sec. 5).

Additional remarks. We provide additional remarks about our protocols on quantum safety and communication complexity.

Quantum safety. In practice, signature-free ABC may instantiate the underlying common coins using threshold PRF [5, 17]. In this case, in contrast to computational ABC protocols using threshold signatures, signature-free ABC protocols achieve the desirable quantum safety property as defined in [31] (but not quantum liveness), where the safety of the protocols is always attained even in the presence of a quantum adversary.

SQ, to our knowledge, is the first quantum-safe ABC protocol with optimal $O(n^2)$ messages and $O(1)$ expected time. SQ_h (Sec. 5) achieves quantum safety too, as hash functions—with appropriately chosen parameters—are believed to defend against quantum adversaries.

[4] Note here that we only need to ensure the existence of such a set instead of finding such a set.

Communication complexity. We discuss the communication complexity of SQ and its hash-based variant SQ_h.

- If directly assuming the existence of the common coin object (Rabin dealer):
  $\triangleright$ SQ achieves $O(Ln^3)$ communication, where L is the length of a replica's input. The cost is the same as all other signature-free ABC protocols if instantiating the underlying RBC (reliable broadcast) using Bracha's RBC [12,13] (an IT-secure RBC).
  $\triangleright$ SQ_h achieves $O(Ln^2 + \kappa n^3)$ communication, where κ is the security parameter (i.e., the output length of hash functions). The cost is the same as all signature-free ABC protocols if instantiating the underlying RBC using the most efficient hash-based RBC protocol—CCBRB [4].
- If we instantiated common coins using a classic threshold PRF (e.g., the scheme based on the CDH assumption [17]), SQ achieves $O(Ln^2 + \kappa n^2)$ communication and SQ_h achieves $O(Ln^2 + \kappa n^3)$ communication, matching the communication of state-of-the-art signature-free ABC protocols [25, 37, 47] using the above-mentioned CCBRB protocol.

Additional related work. This and prior works [7,9,18,19,36,39–42] assume the common coin object providing global random coins that are visible to all replicas. The common coin object was originally proposed in Rabin's pioneering work [43], where a trusted dealer distributes coins to replicas. The common coin object can also be realized in various other ways, such as threshold PRF [5,17], threshold signatures [6,17,45], random beacons [23,30], dedicated common coin protocols [10,27], and ones based on trusted execution environments (TEEs).

Recently, some signature-free MVBA protocols have been proposed [1,20,21, 25], but they all have $O(n^3)$ message complexity and the ABC protocols relying on them would at least have $O(n^3)$ messages. In particular, Duan, Wang, and Zhang (DWZ) recently proposed a new asynchronous common subset (ACS) protocol that leads to a signature-free ABC protocol with $O(n^3)$ messages and $O(1)$ expected time [25]. We compare our work with theirs in the following:

- DWZ is an ACS protocol and also leads to an ABC protocol, but SQ is not an ACS protocol. We do not know how to efficiently transform a signature-free ABC protocol to a signature-free ACS protocol without increasing the time or message complexity.
- The underlying techniques of DWZ and ours are different: DWZ uses the conventional n parallel RBC instances for replicas to disseminate their proposed messages, while SQ cannot use parallel RBC for the goal of achieving $O(n^2)$ messages, and to reach consensus, SQ uses MBA.

2 Model and Definitions

2.1 System and Threat Model

We consider protocols with n replicas $\{p_1, \cdots, p_n\}$ running over authenticated channels. Among the n replicas, at most f of them may fail arbitrarily (Byzantine failures). Replicas that are not faulty are correct. We consider an asynchronous network with no timing assumptions. We assume $n \geq 3f+1$, which is optimal in this setting. For simplicity, we may let $n = 3f+1$.

Our protocol is secure under an adaptive adversary, where an adaptive adversary can choose the set of corrupted replicas at any moment during the execution of the protocol (as long as we assume an adaptively secure common coin protocol available or directly assume adaptively secure common coins).

Throughout the paper, we use the term *broadcast* to represent best-effort broadcast, i.e., the sender multicasts a message to all replicas.

2.2 Definitions and Building Blocks

Atomic Broadcast (ABC). Atomic broadcast allows replicas to reach an agreement on the order of messages (values). An atomic broadcast protocol Π is specified by *a-broadcast* and *a-deliver*. When a replica is provided (by an adversary) with a queue of payload messages of the form $m \in \{0,1\}^*$, we say the replica *a-broadcasts* the messages. Correct replicas should *a-deliver* the same sequence of messages in the same order.

Definition 1 (ABC). *Let Π be a protocol executed by replicas $p_1, \cdots, p_n$, where each replica a-broadcasts a queue of payload messages and a-delivers messages in a particular order. Π should achieve the following properties:*

- ***Agreement**: If any correct replica a-delivers a message m, then every correct replica a-delivers m.*
- ***Total order**: If a correct replica a-delivers a message m before a-delivering m', then no correct replica a-delivers m' without first a-delivering m.*
- ***Liveness**: If a correct replica a-broadcasts a message m, then it eventually a-delivers m.*
- ***Integrity**: Every correct replica a-delivers a message at most once. If a correct replica a-delivers m, then m was previously a-broadcast by some replica.*

The size of the *a-delivered* messages depends on the concrete constructions. In some protocols, every correct replica *a-delivers* the payload message *a-broadcast* by one replica at a time. Meanwhile, in some protocols, every correct replica *a-delivers* a union of several payload messages *a-broadcast* by some replicas. Our work considers the former case. Our protocols, however, can be transformed to the latter case.

Multivalued Byzantine Agreement (MBA). MBA allows replicas to reach an agreement on a value $v \in \{0,1\}^*$. An MBA protocol is specified by *mba-propose* and *mba-decide*. For a protocol instance, each replica is provided an input value $v \in \{0,1\}^*$ or $\perp$ (a distinguished symbol), where we say the replica *mba-proposes* v or $\perp$. When a replica terminates the protocol and outputs a non-empty value v or $\perp$, we say the replica *mba-decides* v or $\perp$.

Definition 2 (MBA). *Let Π be a protocol executed by replicas $p_1, \cdots, p_n$, where each replica mba-proposes a value $v \in \{0,1\}^* \cup \{\perp\}$, and each correct replica mba-decides a value $v \in \{0,1\}^*$ or $\perp$. Π should satisfy the following properties:*

– ***Agreement****: If a correct replica mba-decides v, then any correct replica that terminates mba-decides v.*
– ***Termination****: If all correct replicas mba-propose some value, every correct replica eventually mba-decides.*
– ***Integrity****: Every correct replica mba-decides at most once.*
– ***Validity****: If all correct replicas mba-propose v, then any correct replica that terminates mba-decides v.*
– ***Non-intrusion****: If a correct replica mba-decides v such that $v \neq \perp$, then v is mba-proposed by a correct replica.*

Due to the validity property, if all correct replicas *mba-propose* the same non-$\perp$ value, $\perp$ cannot be decided. Meanwhile, the non-intrusion property is a strong "validity" property as defined in [18, 41, 42]: a decided value must be a value proposed by a correct replica or a default value denoted $\perp$. The two properties prevent a value proposed only by faulty replicas from being decided.

Asynchronous binary Byzantine agreement (ABA) [14] can be viewed as a special case of MBA by restricting the input to a binary value.

(Strong) common coins. We consider a common coin primitive, a notion first introduced by Rabin [43]. Following the definitions in prior works [15, 19, 40, 43], we distinguish (regular) common coins (corresponding to a low $f+1$ threshold) from strong common coins (corresponding to a high $2f+1$ threshold). A regular common coin primitive is invoked by triggering a *release* event at every correct replica. Here we say a correct replica "releases" the coin, as we require that the coin's value should be unpredictable before the first replica invokes the coin. The common coin protocol *outputs* a coin value $b \in \mathcal{B}$ at each correct replica. We define the common coin primitive as follows.

Definition 3 (Common coin). *Let Π be a protocol executed by replicas $p_1, \cdots, p_n$, where each replica releases the coin and outputs a coin value $b \in \mathcal{B}$. Π should satisfy the following properties.*

- ***Termination.*** *Every correct replica eventually outputs a coin value.*
- ***Agreement.*** *If a correct replica outputs b and another correct replica outputs b', $b = b'$.*
- ***Bias-resistance.*** *If any correct replica outputs b, the distribution of the coin is uniform over $\mathcal{B}$.*
- ***Unpredictability.*** *Unless at least one correct replica has released the coin, no replica has any information about the coin output by a correct replica.*

The definition of the strong common coin differs in the unpredictability only, requiring that unless at least $f+1$ correct replicas have released the coin, no replica has any information about the coin output by a correct replica.

The common coin abstraction encapsulates various ways of concrete implementations, e.g., by assuming a cryptographic trusted setup, where a trusted dealer prepares a one-time setup for a cryptographic threshold common coin protocol (e.g., [17] for static security, [6, 32, 34] for adaptive security). In this case, for each common coin instance, each replica broadcasts a κ-bit string and the total communication is κn^2, where κ is a security parameter.

Leader election from common coins. Our protocol uses a leader election protocol $\mathsf{Election}()$ that can be built from a common coin object or a strong common coin object. Each time the $\mathsf{Election}()$ function is queried, the function outputs a random leader $p_k \in \{p_1, \cdots, p_n\}$. When calculating the communication complexity, we assume that the $\mathsf{Election}()$ function is instantiated from a Rabin dealer [43], where the dealer sends a $\log n$-bit random coin to each replica. The dealer in total sends at most $n \log n$ bits.

Consistent Broadcast (CBC). In consistent broadcast (CBC) [16, 44, 46], a designated replica broadcasts a message to a group of replicas. A CBC protocol is specified by *c-broadcast* and *c-deliver*.

Definition 4 (CBC). *Let Π be a protocol executed by replicas $p_1, \cdots, p_n$, where a replica p_s c-broadcasts a message $m \in \{0,1\}^*$ or $\perp$, and each correct replica c-delivers $m \in \{0,1\}^* \cup \{\perp\}$. Π should satisfy the following properties:*

- ***Validity**: If a correct replica p_s c-broadcasts a message m, then any correct replica p_i eventually c-delivers m.*
- ***Consistency**: If two correct replicas c-deliver two messages m and m', then $m = m'$.*
- ***Integrity**: For any message m, every correct replica p_i c-delivers m at most once. Moreover, if p_i c-delivers m, m was previously c-broadcast by p_s.*

CBC guarantees only that the delivered message is the same for all receivers, but it does not ensure totality (a property requiring either all correct replicas to deliver some message or none to deliver any message) needed for reliable broadcast (RBC). Therefore, it is easier to implement CBC than RBC. For instance, Bracha's RBC [12, 13] requires three communication rounds, while the corresponding CBC requires two rounds only.

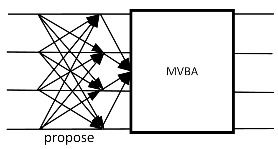
3 Review of Existing ABC Protocols and Overview of Our Approach

3.1 Review of ABC Approaches

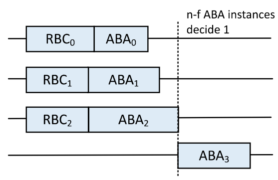
As depicted in Figure 2, we divide existing ABC protocols into four categories: 1) MVBA-based; 2) ABA-based; 3) RABA-based; and 4) DAG-based. From the security model perspective, MVBA-based ABC protocols are sharply distinguished from the rest of ABC protocols: MVBA based protocols rely on threshold signatures that require trusted setup and strong models such as random oracles, while the rest of them assume common coins only.

MVBA-based (Figure 2a). All MVBA-based ABC leverage (non-interactive) threshold signatures to achieve $O(n^2)$ messages and $O(1)$ expected time. However, threshold signatures require trusted setup, strong models (e.g., random oracles), and assume the hardness of computational problems [6, 11, 32, 45].

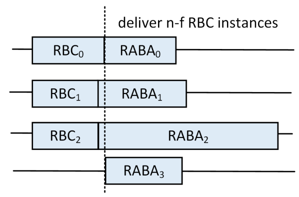
ABA-based (Figure 2b). The BKR paradigm due to Ben-Or, Kelmer, and Rabin relies on n parallel reliable broadcast (RBC) instances and n paral-



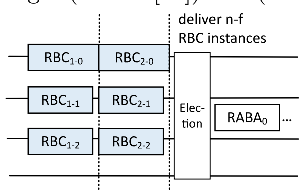
(a) MVBA-based paradigm (CKPS [16]).



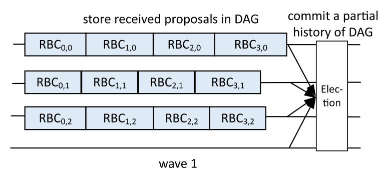
(b) ABA-Based paradigm (BKR [9]).



(c) RABA-based paradigm (PACE [47]).



(d) RABA-based paradigm (FIN [25]).



(e) DAG-based paradigm (DAG-Rider [31]).

Fig. 2: Comparison of asynchronous atomic broadcast paradigms. The figures are best viewed in color. Primitives that are computational are represented in bold boxes. Primitives that make the paradigm achieve $O(n^3)$ complexity are represented in blue boxes.

lel asynchronous binary agreement (ABA) instances.[5] HoneyBadgerBFT [37], BEAT [24], EPIC [33] follow the BKR paradigm. ABA-based ABC has $O(n^3)$ messages and $O(\log n)$ expected time (due to the n parallel ABA instances).

RABA-based. Zhang and Duan [47] improved the BKR framework and proposed PACE. As shown in Figure 2c, PACE replaces ABA using a variant of the ABA primitive called reproposable asynchronous binary agreement (RABA) and makes the RABA instances fully parallel. Very recently, Duan, Wang, and Zhang use (two consecutive) parallel RBC instances and a constant number of RABA instances to build a new ABC protocol achieving $O(n^3)$ messages and $O(1)$ expected time, as illustrated in Figure 2d.

DAG-based (Figure 2e). The DAG-Rider paradigm relies on RBC and DAG-based data structures to build ABC [31]. The paradigm builds two layers. In the first layer, replicas reliably broadcast their proposals and use DAG to store the received proposals. In the second layer, replicas deliver the proposals accordingly. DAG-Rider achieves $O(n^3)$ message complexity and $O(1)$ time.

In summary, there is a mismatch in the message and time complexity between the MVBA-based approach and the other three signature-free approaches. The common characteristic of all signature-free ABC approaches is that they all use parallel RBC protocols, which leads to $O(n^3)$ message complexity for these protocols. We aim to remove this message complexity bottleneck.

[5] Prior to the construction in BKR, Ben-Or, Canetti, and Goldreich proposed an ABC protocol using n^2 RBC instances and achieving $O(n^4)$ message complexity [7].

3.2 Pathway to Our MBA-based ABC

A recap of our toy construction in Figure 1a. As described in our toy construction in the introduction, the major challenge is to handle the case where p_{k_r} is faulty. Indeed, if p_{k_r} is faulty, we cannot guarantee that every epoch will complete. It is possible that none of the correct replicas will *a-deliver* any value, as the termination property of MBA requires *all* correct replicas to *mba-propose*.

As an alternative, we could ask replicas that have not received the proposed messages from p_{k_r} to directly *mba-propose* $\perp$ for MBA after the election phase. However, this alternative solution has a liveness issue as well: replicas may *a-deliver* $\perp$ in *all* epochs and make no progress. We demonstrate the issue via an example in Figure 3 with four replicas, where p_4 is faulty and broadcasts inconsistent messages to the replicas. In the figure, each element indexed by (i, j) represents whether p_i has received the proposed message from p_j right before the election phase, after receiving $n - f$ messages. We observe from the figure that if *any* correct replica p_j (i.e., p_1, p_2, or p_3) is selected, at least one correct replica fails to receive the message from p_j and provides $\perp$ as input to MBA, and other replicas provide the same non-$\perp$ value as input. In this case, MBA may output $\perp$. Meanwhile, the same claim holds if the faulty replica p_4 is selected: as correct replicas provide inconsistent inputs to MBA, MBA may output $\perp$. In both cases, correct replicas may *a-deliver* $\perp$ for all epochs.

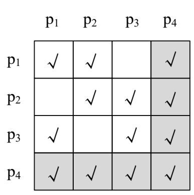


Fig. 3: A livenes issue for the alternative construction.

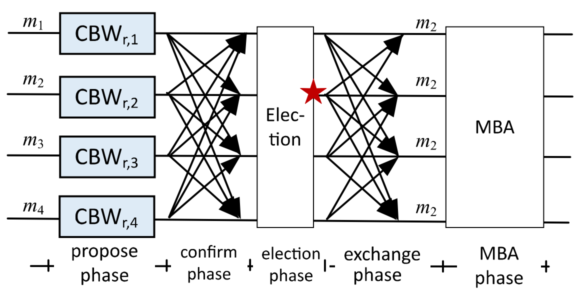
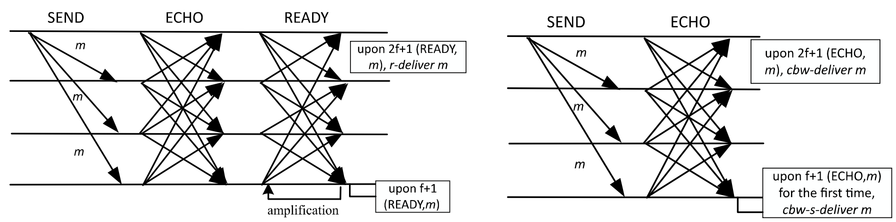


Fig. 4: The SQ_0 protocol.

The crux: ensuring the existence of a key set for each epoch. In SQ, based on our toy construction, we will ensure the existence of a key set consisting of at least f + 1 correct replicas for each epoch. Our goal is that if any replica p_{k_r} in the key set is selected by the random leader election protocol, any correct replica *mba-proposes* m_{k_r} and hence epoch r completes with a non-$\perp$ output. (In SQ, a correct replica *mba-proposes* m_{k_r}, either because it has received m_{k_r} directly from p_{k_r}, or has received m_{k_r} from other replicas.) Meanwhile, if any replica outside the key set is selected, we need to ensure that all correct replicas still *mba-propose* some values. Thus, every MBA instance will terminate and our protocol is live. Below we first introduce a warm-up protocol SQ_0 and then



(a) Bracha's broadcast. (b) Our CBW construction.

Fig. 5: RBC vs. CBW.

briefly describe how we transform it into our fully-fledged protocol SQ.

A warm-up protocol SQ$_0$ with $O(n^3)$ messages and $O(1)$ time. In SQ$_0$ (described in Figure 4), we introduce two new building blocks: a new primitive called *consistent broadcast with weak agreed set (CBW)* and an additional *exchange phase*. We comment that SQ$_0$ is of independent interest and (already) outperforms the state-of-the-art MBA-based ABC protocol that has $O(n^4)$ messages and $O(n)$ time [18, 38].

▷ *Consistent broadcast with weak agreed set (CBW).* As introduced in Sec. 2.2, the classical CBC primitive is a weaker version of reliable broadcast. CBW further extends CBC by introducing an additional output satisfying "weak agreement." The primitive is specified by three events: *cbw-broadcast*, *cbw-deliver*, and *cbw-s-deliver*. Specifically, a designated sender p_s *cbw-broadcasts* a message m. Every correct replica p_i may output two values: it *cbw-delivers* a primary output m and *cbw-s-delivers* a *secondary* output v. Correct replicas that *cbw-deliver* some value always *cbw-deliver* the same value. However, they do not necessarily *cbw-s-deliver* the same value.

Definition 5 (CBW). *Let Π be a protocol executed by replicas $p_1, \cdots, p_n$, where a sender p_s cbw-broadcasts a message $m \in \{0,1\}^*$ or $\perp$ to all replicas. Every correct replica p_i may cbw-deliver $m \in \{0,1\}^*$ or $\perp$ and cbw-s-deliver $v \in \{0,1\}^*$ or $\perp$. Π should achieve the following properties:*

- ***Validity**: If a correct replica p_s cbw-broadcasts a message m, then every correct replica p_i eventually cbw-delivers m and cbw-s-delivers m.*
- ***Consistency**: If a correct replica p_i cbw-delivers message m, another correct replica p_j cbw-delivers message m', then $m = m'$.*
- ***Weak agreement**: If a correct replica p_i cbw-delivers message m, then every correct replica p_j eventually cbw-s-delivers some value.*
- ***Integrity**: Every correct replica cbw-delivers a message at most once. If a correct replica cbw-delivers a message m or cbw-s-delivers m, then m was previously cbw-broadcast by some replica.*

An IT-secure CBW protocol can be built as follows, as shown in Figure 5b. First, the sender p_s broadcasts a $(\textsc{Send}, m)$ message. Second, upon receiving a $(\textsc{Send}, m)$ message from p_s, a correct replica p_i broadcasts an $(\textsc{Echo}, m)$ message. Upon receiving $2f+1$ $(\textsc{Echo}, m)$ messages with the same m, p_i *cbw-delivers* m.

Additionally, upon receiving $f+1$ $(\textsc{Echo}, m)$ messages with the same m for the first time, p_i *cbw-s-delivers* m.

For readers who are familiar with Bracha's RBC (shown in Figure 5a), our CBW protocol can be viewed as its two-phase version yet additionally having a secondary output. Also, our CBW protocol can be viewed as a variant of (authenticated) CBC, but carries "more information" that we need for our purpose.

ABC with $O(n^3)$ messages and $O(1)$ expected time

- Initialize $d \leftarrow \emptyset$ at the beginning of the protocol.
- Initialize the following parameter for each epoch r: $CV_r \leftarrow [\bot]^n$.

Epoch r

- **(Propose)** Upon *a-broadcast*(m_i), *cbw-broadcast* m_i for instance $\mathsf{CBW}_{r,i}$.
- **(Confirm)** Upon *cbw-deliver*(m_j) for instance $\mathsf{CBW}_{r,j}$, set $CV_r[j]$ as m_j. Send a $(\textsc{Confirm}, i)$ message to p_j.
- **(Commit)** Upon receiving $n-f$ $(\textsc{Confirm}, j)$ messages from different p_j, start the election phase.
- Set r as $r+1$ and start the next epoch.

Election, Exchange, and MBA phases

- **(Election)** Query the $\mathsf{Election}(\mathsf{r})$ function and obtain a random value k_r such that $1 \le k_r \le n$.
- **(Exchange)** Broadcast $(\textsc{Send}, r, i, CV_r[k_r])$.
- **(MBA)** If $CV_r[k_r] \neq \bot$, *mba-propose* $CV_r[k_r]$ for instance MBA_r. Otherwise, wait until one of the following conditions is satisfied:
  1) $f+1$ $(\textsc{Send}, r, *, m)$ are received, then *mba-propose* m for instance MBA_r;
  2) $2f+1$ $(\textsc{Send}, r, *, \bot)$ are received, then *mba-propose* $\bot$ for instance MBA_r;
  3) m is *cbw-s-delivered* in CBW_{r,k_r}, then *mba-propose* m for instance MBA_r.

Output conditions

(Event 1) If MBA_r outputs $m \neq \bot$ and $m \notin d$, *a-deliver* m and set d as $d \cup m$.
(Event 2) If MBA_r outputs $\bot$, then *a-deliver* $\bot$.
Clear parameter CV_r.

Fig. 6: SQ_0 that achieves $O(n^3)$ message complexity and $O(1)$ time complexity. The $\mathsf{Election}()$ function is built from strong common coins. Code for replica p_i. We use $*$ to denote any value.

$\triangleright$ *The SQ_0 protocol.* Based on CBW, we present SQ_0 in Figure 6. The protocol proceeds as follows. Every replica p_i first *cbw-broadcasts* its value m_i by starting a CBW instance $\mathsf{CBW}_{r,i}$. Upon *cbw-delivering* some value m_j for instance $\mathsf{CBW}_{r,j}$, p_i sets a local parameter $CV_r[j]$ as m_j and we say m_j is *confirmed* by p_i. Additionally, it also sends p_j a $(\textsc{Confirm})$ message. Meanwhile, p_i waits for $n-f$ $(\textsc{Confirm})$ messages, after which we say the value p_i *cbw-broadcasts* is *committed*. p_i then starts the election phase. Here, we use an $\mathsf{Election}(\mathsf{r})$ function built from strong common coins, where the value k_r is revealed after at least $f+1$ correct replicas have queried $\mathsf{Election(r)}$.

After the $\mathsf{Election(r)}$ function outputs k_r, p_i broadcasts a $(\textsc{Send}, r, i, CV_r[k])$ message. p_i then either directly *mba-proposes* its $CV_r[k_r]$ to MBA instance MBA_r or waits until one of the three conditions occurs: 1) p_i receives $f+1$ $(\textsc{Send})$ messages with the same m and then *mba-proposes* m; 2) p_i receives $2f+1$ $(\textsc{Send})$ message with $\perp$ and then *mba-proposes* $\perp$; 3) p_i has *cbw-s-delivered* some value m in instance CBW_{r,k_r} and then *mba-proposes* v. Finally, after MBA_r outputs some value, p_i *a-delivers* the value output by MBA_r.

$\triangleright$ *Analysis.* We first argue that SQ_0 is live. According to the validity property of CBW, at least $n-f$ CBW instances will complete. Thus, all correct replicas eventually receive $n-f$ $(\textsc{confirm})$ messages and enter the election phase. Here, there are two scenarios for CBW_{r,k_r}, as shown below. In each scenario, we show that every correct replica eventually *mba-proposes* some value to MBA_r, so epoch r eventually completes according to the termination property of MBA.

- Scenario 1: No correct replica *cbw-delivers* any value in CBW_{r,k_r}. In this case, condition 2) or 3) of Figure 6 will eventually be triggered and every correct replica provides some input to MBA_r.
- Scenario 2: At least one correct replica *cbw-delivers* some value in CBW_{r,k_r} and either condition 1) or 3) will eventually be triggered. Condition 1) will be triggered if at least $f+1$ correct replicas *cbw-deliver* the same value. Additionally, the weak agreement property of CBW ensures that every correct replica will eventually *cbw-s-delivers* some value, i.e., condition 3) will be triggered. Every correct replica thus provides some input to MBA_r.

SQ_0 achieves $O(1)$ time because after $f+1$ correct replicas enter the election phase, a key set with at least $f+1$ correct replicas must exist. Specifically, every correct replica p_i waits until $n-f$ replicas have sent a $(\textsc{Confirm})$ message to it before it enters the election phase. Each of the $n-f$ replicas has *cbw-delivered* some value in $\mathsf{CBW}_{r,i}$. Therefore, after $f+1$ correct replicas I enter the election phase, for any $p_{k_r} \in I$, at least $f+1$ correct replicas have *cbw-delivered* some value in CBW_{r,k_r}. They will send a $(\textsc{Send}, r, *, m_{k_r})$ message with the same m_{k_r} according to the consistency property of CBW. Then condition 1) will be eventually satisfied. Additionally, condition 2) will never be triggered. Indeed, as at least $f+1$ correct replicas broadcast $(\textsc{Send}, r, *, m_{k_r})$ messages, no replica can collect more than $2f+1$ $(\textsc{Send}, r, -, \perp)$ messages as there are $3f+1$ replicas in total. Additionally, correct replicas will never provide $m'_{k_r} \neq m_{k_r}$ as input to MBA_r after triggering condition 3). In particular, due to the validity property and the integrity property of CBW, no correct replica will *cbw-s-deliver* m'_{k_r} such that $m'_{k_r} \neq m_{k_r}$. Thus, MBA_r will output a non-$\perp$ value m_{k_r} with at least 1/3 probability.

From SQ_0 to SQ. We transform SQ_0 in Figure 6 to SQ with $O(n^2)$ messages and $O(1)$ time. Additionally, SQ can be built from a leader election object from regular common coins instead of the strong common coins as used in SQ_0. Indeed, SQ ensures that if at least one correct replica enters the election phase, the key set already consists of at least $f+1$ correct replicas. This is achieved by defining a new primitive called *parallel consistent broadcast with weak agreed set (PCBW)* where each epoch r includes one PCBW instance (that has $O(n^2)$ messages). Briefly speaking, PCBW can be viewed as n parallel CBW instances with one additional feature that we need for our final design: if any correct replica terminates the PCBW instance for epoch r, the replica has committed $n-f$ values and each of the values has been confirmed by $n-f$ replicas. Among the $n-f$ committed values, at least $f+1$ of them are proposed by correct replicas which form a key set!

We then provide a PCBW construction with $O(n^2)$ messages. Our PCBW protocol is instantiated using only *one* (PROPOSE) message and two local procedures: an *update procedure* and a *controlling procedure*. As multiple PCBW instances can be started concurrently (one for each epoch in SQ), the (PROPOSE) message together with the update procedure allow replicas to update their local state about the PCBW instances that have not terminated yet. Each replica further uses the controlling procedure to determine whether a PCBW instance (in some epoch r) should terminate, after which we confirm the existence of a key set for epoch r.

4 The SQ Protocol

We are now ready to present the SQ protocol that achieves optimal resilience, $O(1)$ expected time and $O(n^2)$ messages. In this section, we begin with the new *parallel consistent broadcast with weak agreed set (PCBW)* primitive and define its security properties. We then use PCBW in a black-box manner to build SQ. Finally, we present our PCBW construction.

4.1 Parallel Consistent Broadcast with Weak Agreed Set (PCBW)

PCBW is specified by three events: *pcbw-broadcast*, *pcbw-deliver*, and *pcbw-s-deliver*. Every correct replica p_i *pcbw-broadcasts* a message m_i. Meanwhile, every correct replica *pcbw-delivers* a pair of values $(\vec{m}, \vec{cv})$, called primary outputs. For each slot $j \in [n]$, the values $(\vec{m}[j], \vec{cv}[j])$ correspond to the value *pcbw-broadcast* by replica p_j. Meanwhile, $\vec{v}$ is the secondary output of PCBW. The primary outputs of each slot j (i.e., $(\vec{m}[j], \vec{cv}[j])$ satisfy a crusader agreement [2,22]: it is possible that some correct replicas outputs $\vec{m}[j] = m_j$ (resp. $\vec{cv}[j] = cv_j$) while other correct replicas output $\vec{m}[j] = \perp$ (resp. $\vec{cv}[j] = \perp$), but for all correct replicas that output non-$\perp$ values, they output the same value. Meanwhile, correct replicas do not necessarily *pcbw-s-deliver* the same value for each $\vec{v}[j]$. Informally speaking, each $\vec{cv}[j]$ and $\vec{v}[j]$ correspond to the *cbw-delivered* value and the *cbw-s-delivered* value in CBW, respectively. The $\vec{m}[j]$ captures the committed values shown in Figure 6. We now specify the security properties of PCBW as follows.

Definition 6 (PCBW). *Let Π be a protocol executed by replicas $p_1, \cdots, p_n$. Each replica p_i pcbw-broadcasts a message m_i to all replicas. Every correct replica*

p_i may pcbw-deliver $(\vec{m}, \vec{cv})$ where $|\vec{m}| = n$ and $|\vec{cv}| = n$. Additionally, p_i may pcbw-s-delivers $\vec{v}$ where $|\vec{v}| = n$. Π should achieve the following properties:

- ***Validity**: If a correct replica p_i pcbw-broadcasts a message m_i, then every correct replica p_j eventually pcbw-s-delivers $\vec{v}$ where $\vec{v}[i] = m_i$. If p_j pcbw-delivers $(\vec{m}, \vec{cv})$ where $\vec{m}[i] \neq \perp$ and $\vec{cv}[i] \neq \perp$, then $\vec{m}[i] = \vec{cv}[i] = m_i$.*
- ***Consistency**: Suppose that a correct replica p_i pcbw-delivers $(\vec{m}, \vec{cv})$ such that $\vec{cv}[k] = m \neq \perp$ for slot k. For any correct replica p_j:*
  (1) if p_j pcbw-delivers $(\vec{m'}, \vec{cv'})$ where $\vec{cv'}[k] \neq \perp$, then $\vec{cv'}[k] = m$;
  (2) if p_j pcbw-delivers $(\vec{m'}, \vec{cv'})$ where $\vec{m'}[k] \neq \perp$, then $\vec{m'}[k] = m$.
- ***Weak agreement I**: If a correct replica p_i pcbw-delivers $(\vec{m}, \vec{cv})$ where $\vec{cv}[k] \neq \perp$ for slot k, then every correct replica p_j eventually pcbw-s-delivers $\vec{v}$ where $\vec{v}[k] \neq \perp$.*
- ***Weak agreement II**: Consider the first correct replica p_i that pcbw-delivers $(\vec{m}, \vec{cv})$. For any slot k, if $\vec{m}[k] = m_k \neq \perp$, then there exists a set I of at least $f+1$ correct replicas such that for any $p_j \in I$, p_j pcbw-delivers $(\vec{m'}, \vec{cv'})$, where $\vec{cv'}[k] = m_k$.*
- ***Integrity**: Every correct replica pcbw-delivers at most once. Every correct replica pcbw-s-delivers $\vec{v}$ at most $O(n)$ times. For any correct replica p_i:*
  (1) if p_i pcbw-delivers $(\vec{m}, \vec{cv})$, then for any $\vec{m}[k] \neq \perp$ (resp., $\vec{cv}[k] \neq \perp$), $\vec{m}[k]$ (resp., $\vec{cv}[k]$) was previously pcbw-broadcast by replica p_k.
  (2) if p_i pcbw-s-delivers $\vec{v}$, then for any $\vec{v}[k] \neq \perp$, $\vec{v}[k]$ was previously pcbw-broadcast by replica p_k.
- ***Termination**: If every correct replica pcbw-broadcasts, every correct replica eventually pcbw-delivers some values.*

4.2 SQ

Using PCBW in a black-box manner, we show the pseudocode of SQ in Figure 7. Compared to SQ_0 presented in Figure 6, there are two major changes. First, we replace the n parallel CBW instances and the *confirm* round with one PCBW instance PCBW_r. In particular, every replica p_i starts a PCBW instance PCBW_r, using its m_i as input. After receiving $n-f$ *pcbw-broadcast* values in PCBW_r, p_i can start the next epoch. Additionally, after p_i *pcbw-delivers* $(\vec{m}, \vec{cv})$, it starts the election phase. Second, we modify the third condition in the MBA phase, where p_i *pcbw-s-delivers* $\vec{v}$ such that $\vec{v}[k_r]$ is non-$\perp$. In this case, p_i *mba-proposes* $\vec{v}[k_r]$ in MBA_r. We now describe SQ in detail as follows:

Propose phase. Each replica p_i *pcbw-broadcasts* m_i for instance PCBW_r, where m_i is the value it *a-broadcasts* in epoch r. Here we assume each message m_i is unique (and in practice, m_i may consist of a batch of transactions). Upon receiving $n-f$ messages in PCBW_r, p_i enters the next epoch before the current epoch completes.

For the messages replicas *a-broadcast* in each epoch, we follow the approach used in prior ABC protocols (e.g., [16]): in addition to keeping track of the proposed messages, each replica also stores the proposed messages from other

SQ with $O(1)$ expected time and $O(n^2)$ messages

– Initialize $d \leftarrow \emptyset$ at the beginning of the protocol.

Epoch r

– **(Propose)** Upon *a-broadcast*(m_i), *pcbw-broadcast* m_i for instance PCBW_r.
– Upon receiving $n - f$ values in PCBW_r, set r as $r + 1$ and start epoch $f + 1$.

Election, Exchange, and MBA phases

– Wait until p_i *pcbw-delivers* $(\vec{m}, \vec{cv})$ in PCBW_r.
– **(Election)** Query the $\mathsf{Election(r)}$ function and obtain a random value k_r such that $1 \leq k_r \leq n$.
– **(Exchange)** Broadcast $(\textsc{Send}, r, i, \vec{cv}[k_r])$.
– **(MBA)** If $\vec{cv}[k_r] \neq \perp$, *mba-propose* $\vec{cv}[k_r]$ for MBA_r. Otherwise, wait until one of the following conditions is satisfied:
  1) $f + 1$ $(\textsc{Send}, r, *, m)$ messages are received such that $m \neq \perp$, then *mba-propose* m for instance MBA_r.
  2) $2f + 1$ $(\textsc{Send}, r, *, \perp)$ messages are received, then *mba-propose* $\perp$ for instance MBA_r.
  3) $\vec{v}$ is *pcbw-s-delivered* such that $\vec{v}[k_r] \neq \perp$, then *mba-propose* $\vec{v}[k_r]$ for instance MBA_r.

Output conditions for epoch r

(Event 1) If MBA_r outputs $m \neq \perp$ and $m \notin d$, *a-deliver* m and set d as $d \cup m$.
(Event 2) If MBA_r outputs $\perp$, then *a-deliver* $\perp$.

Fig. 7: The SQ protocol for epoch r at replica p_i. The $\mathsf{Election()}$ function is built from regular common coins.

replicas locally in a buffer. After a proposed message is *a-delivered*, the proposed message is removed from the buffer. We set a liveness parameter lp. If some message in the buffer is proposed in epoch r and is not *a-delivered* by epoch $r + lp$, each replica proposes the message until the message is *a-delivered*. This approach ensures that a proposal will eventually be *a-delivered*.

Election phase. Every correct replica p_i waits until it *pcbw-delivers* $(\vec{m_r}, \vec{cv_r})$ in PCBW_r before querying the $\mathsf{Election(r)}$ function.

Exchange phase and MBA phase. After $\mathsf{Election(r)}$ outputs k_r, p_i broadcasts a $(\textsc{Send}, r, i, \vec{cv}[k_r])$ message. p_i then either directly *mba-proposes* its $\vec{cv}[k_r]$ to MBA_r or waits until one of the three conditions occurs: 1) p_i receives $f + 1$ $(\textsc{Send})$ messages with the same m and then *mba-proposes* m; 2) p_i receives $2f+1$ $(\textsc{Send})$ message with $\perp$ and then *mba-proposes* $\perp$; 3) p_i has *pcbw-s-delivered* $\vec{v}$ in instance PCBW_r such that $\vec{v}[k_r] \neq \perp$ and then *mba-proposes* $\vec{v}[k_r]$. Finally, after MBA_r outputs some value, p_i *a-delivers* the value output by MBA_r.

$\triangleright$ *Analysis*. We now briefly argue why SQ is live. First note that due to the termination condition of PCBW, every correct replica eventually enters the election phase. We then distinguish the following two cases:

- No correct replica *pcbw-delivers* $(\vec{m}, \vec{cv})$ in PCBW_r such that $\vec{cv}[k_r] \neq \perp$. In this case, condition 2) or 3) of Figure 7 will eventually be satisfied and replicas provide some input to MBA_r.
- At least one correct replica *pcbw-delivers* $(\vec{m}, \vec{cv})$ such that $\vec{cv}[k_r] \neq \perp$. Then either condition 1) or 3) will eventually be triggered. Condition 1) will be triggered if $f+1$ correct replicas *pcbw-deliver* $(\vec{m'}, \vec{cv'})$ such that $\vec{cv'}[k_r] \neq \perp$. Then due to the consistency property of PCBW, replicas provide $\vec{cv}[k_r]$ as input to MBA_r. Additionally, due to the weak agreement I property of PCBW, every correct replica eventually *pcbw-s-delivers* $\vec{v}$ such that $\vec{v}[k_r]$ is non-$\perp$. Thus, condition 3) will be satisfied.

Thus, every correct replica provides some input to MBA_r. The termination property of MBA thus ensures that epoch r completes.

Now we analyze why SQ achieves $O(1)$ time. Recall that our goal is that if at least one correct replica queries the $\mathsf{Election(r)}$ function, a key set I exists. We consider the first correct replica p_i that queries $\mathsf{Election(r)}$ (after which k_r is revealed). Let $(\vec{m}, \vec{cv})$ be the values p_i *pcbw-delivers*. If we require that $\vec{m}$ has at least $n-f$ non-$\perp$ values, at least $f+1$ components in $\vec{m}$ correspond to the values *pcbw-broadcast* by correct replicas. Now we consider these correct replicas forming the key set I and explain why MBA_r outputs non-$\perp$ if $p_{k_r} \in I$. Let $\vec{m}[k_r] = m_{k_r}$. The weak agreement property II of PCBW ensures that there exist $f+1$ correct replicas and for any p_j among these correct replicas, p_j *pcbw-delivers* $(\vec{m'}, \vec{cv'})$ and $\vec{cv'}[k_r] = m_{k_r}$. Therefore, condition 2) will never be triggered and condition 1) will be eventually triggered. Additionally, the validity property of PCBW further ensures that m_{k_r} is indeed sent by the correct replica p_{k_r} and no other correct replicas will *pcbw-delivers* $(-, \vec{cv''})$ where $\vec{cv''}[k_r] = m'_{k_r} \neq \perp$ and $m'_{k_r} \neq m_{k_r}$. Furthermore, no correct replica will *pcbw-s-deliver* $\vec{v'}$ where $\vec{v'}[k_r] = m'_{k_r}$ and $m'_{k_r} \neq m_{k_r}$. Therefore, correct replicas will never trigger condition 3) and use $m'_{k_r} \neq m_{k_r}$ as input to MBA_r. In all the cases, if $p_{k_r} \in I$, all correct replicas will provide m_{k_r} as input to MBA_r and MBA_r thus outputs m_{k_r} according to the validity property of MBA. Therefore, SQ achieves $O(1)$ time. We provide the proof of the protocol in Sec. 4.4.

Now, we are left to show a secure PCBW protocol and additionally ensure that $\vec{m}$ has at least $n-f$ non-$\perp$ values.

4.3 The PCBW Construction

We are now ready to present our PCBW construction and we show the pseudocode of PCBW_r in Figure 8. As mentioned in Sec. 3.2, our PCBW protocol involves only one (Propose) message and two procedures: an update procedure and a controlling procedure. Multiple PCBW instances can be started in parallel and the information exchanged in the (Propose) message in PCBW_r may make prior PCBW instances (that have not terminated yet) terminate, with the help of the update procedure. Additionally, for each PCBW_r, the controlling procedure enables the termination of PCBW_r while ensuring that the value $\vec{m}$ each correct replica *pcbw-delivers* has at least $n-f$ non-$\perp$ values.

Notations. We use $*$ to denote any value. We use $||$ to denote the concatenation of values. For instance, $m||*$ represents m concatenating any value. For any matrix $M^{m\times n}$ and $i \in [1,m]$, we use $M[i][-]$ to denote the i-th row of M, represented as a vector. For example, let $\vec{m} = M[i][-]$. Then $|\vec{m}| = n$ and $\vec{m}[j] = M[i][j]$ for any $j \in [1,n]$. To facilitate the exposition of the protocol, we also introduce the following two functions.

Definition 7 **(**$\mathsf{Col\_Sum}$ **function).** *For any matrix* $M^{m\times n}$ *of bits, if* $k \in [1,n]$, *then* $\mathsf{Col\_Sum}(M,k) = \Sigma_{i=1}^{m} M[i][k]$. *Namely,* $\mathsf{Col\_Sum}(M,k)$ *returns the sum of all the elements in the* k^{th} *column of* M.

Definition 8 **(**$\mathsf{Col\_Comp}$ **function).** *For any matrix* $M^{m\times n}$, $\mathsf{Col\_Comp}(M,k,v)$ *returns the number of elements in the* k^{th} *column of* M *that have value* v, *i.e.,* $\Sigma_{i=1}^{m}|M[i][k] = v|$.

Initialization. Each replica p_i initializes three parameters: E, EV, and LE. Here, the values stored in EV are also called *echo values*. Moreover, for each instance PCBW_r, p_i initializes three parameters: V_r, M_r, and CV_r. The three parameters will be cleared when PCBW_r terminates. We call each element in CV_r a *confirmed value* and M_r the *state matrix*.

Broadcast phase and update procedure. In PCBW_r, each replica p_i *pcbw-broadcasts* m_i by broadcasting a $(\text{Propose}, r, i, m_i, E, EV, LE)$ message to all replicas. Upon receiving a message $(\text{Propose}, r, j, m_j, E^j, EV^j, LE^j)$ from p_j, p_i starts the update procedure. Below we describe the intuition behind each step in the procedure with examples on how the local parameters are updated.

- **(i) State update according to received values**. V_r serves two purposes: the i-th row stores the *pcbw-broadcast* messages p_i directly receives from the replicas; the j-th row stores the messages p_j claims to have received. We call the values each replica claims to have received *echo values*. Informally speaking, echo values serve the same purpose as the values carried in the (Echo) messages in our CBW construction. p_i stores its echo values (the *pcbw-broadcast* messages it receives) in EV.

  $\triangleright$ *Example (Figure 9a).* We show an example where p_i updates the parameters using m_j as input. p_i sets $V_r[i][j]$ as m_j, and *pcbw-s-delivers* vector $V_r[i][-]$ in PCBW_r. p_i also sets $EV[j]$ as $EV[j]||m_j$ and $E[j][2]$ as r.
- **(ii) State update according to received echo values**. This step updates the j-th row in V according to the echo values EV^j (and the corresponding epoch numbers in E^j). Note that the echo values in EV^j are values p_j receives in prior PCBW instances. Accordingly, for any PCBW_e where $e < r$, if p_i has seen $f+1$ matching echo values m (in column k of V_e) corresponding to some replica p_k, p_i *pcbw-s-delivers* m. Informally speaking, this matches the *cbw-s-deliver* event in $\mathsf{CBW}_{e,k}$.

  $\triangleright$ *Example (Figure 9b).* In the example, based on row 1 of E^j, $e_{k,1} = r-2$ and $e_{k,2} = r$. Also, $EV^j[1]$ can be parsed as $m_{r-1,1}||m_{r,1}$. p_i sets $V_{r-1}[j][1]$ as $m_{r-1,1}$ and $V_r[j][1]$ as $m_{r,1}$. Then there exists a set S of $f+1$ replicas (i.e.,

Initialization:

- $E \leftarrow [\perp]^{n\times 2}$. Each element of E stores a PCBW instance id.
- $EV \leftarrow [\perp]^n$. Each element of EV stores a constant number of *pcbw-broadcast* messages.
- $LE \leftarrow [\perp]^n$. Each element of LE stores a PCBW instance id.
- Initialize the following parameters for PCBW_r:
  - $V_r \leftarrow [\perp]^{n\times n}$. Each element of V_r is a *pcbw-broadcast* message.
  - $CV_r \leftarrow [\perp]^n$. Each element of CV_r is a confirmed value.
  - $M_r \leftarrow [\perp]^{n\times n}$. Each element of M_r is a binary value.

Let $\mathit{confirm}(r, k, m_k)$ be the following predicate:

$$\mathit{confirm}(r, k, m_k) \equiv (V_r[i][k] = m_k \wedge m_k \neq \perp \wedge \mathsf{Col\_Comp}(V_r, k, m_k) \geq 2f + 1)$$

- **(Broadcast)** Upon *pcbw-broadcast*(m_i) in PCBW_r:
  Broadcast $(\textsc{Propose}, r, i, m_i, E, EV, LE)$. For every $k \in [1, n]$, set $E[k][1]$ as $E[k][2]$, and set $EV[k]$ as $\perp$.
- Upon receiving $(\textsc{Propose}, r, j, m_j, E^j, EV^j, LE^j)$ from p_j:
  Let $\mathsf{PCBW}_{r'}$ be the instance s.t. every $\mathsf{PCBW}_{r''}$ with $r'' < r'$ has completed. If $V_{r-1}[i][j] \neq \perp$ and $V_r[i][j] = \perp$, then start the **update procedure** for $(\textsc{Propose}, r, j, m_j, E^j, EV^j, LE^j)$ as follows:

  (state update according to received values)
  - Set $V_r[i][j]$ as m_j and *pcbw-s-deliver* $V_r[i][-]$ in PCBW_r.
  - Set $EV[j]$ as $EV[j]||m_j$, set $E[j][2]$ as r.

  (state update according to received echo values)
  For $k \in [1, n]$, let $e_{k,1}$ be $E^j[k][1]$ and $e_{k,2}$ be $E^j[k][2]$:
  - Parse $EV^j[k]$ as a set of values $m_{e_{k,1}+1}||...||m_{e_{k,2}}$.
  - For every $e \in [e_{k,1} + 1, e_{k,2}]$, if $V_e[j][k] = \perp$, then:
    - set $V_e[j][k]$ as m_e.
    - if $V_e[i][k] = \perp$ and there exists a set S s.t. $|S| \geq f + 1$ and for every $p_{j'} \in S$, $V_r[j'][k] = m$, then set $V_e[i][k]$ as m and *pcbw-s-deliver* $V_e[i][-]$ in PCBW_e.

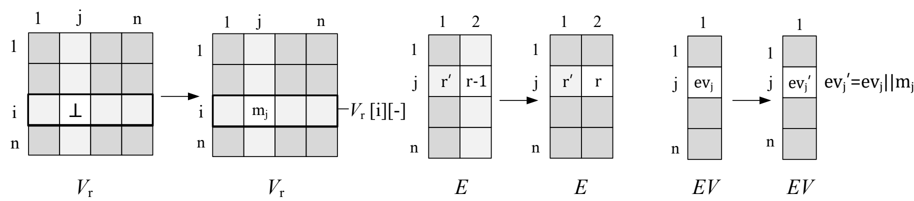
  (state refresh)
  For $k \in [1, n]$, $r'' \in [r', r]$, if $\mathit{confirm}(r'', k, V_{r''}[i][k])$= 1, then
  - Set $CV_{r''}[k]$ as $V_{r''}[i][k]$ and set $M_{r''}[i][k]$ as 1.
  - Set $LE[k]$ as the largest r^* s.t. for every $r'' \in [r', r^*]$, $M_{r''}[i][k] = 1$.

  (state matrix update)
  For $k \in [1, n]$, let e_k denote $LE^j[k]$: for any $r'' \in [r', e_k]$, set $M_{r''}[j][k]$ as 1.

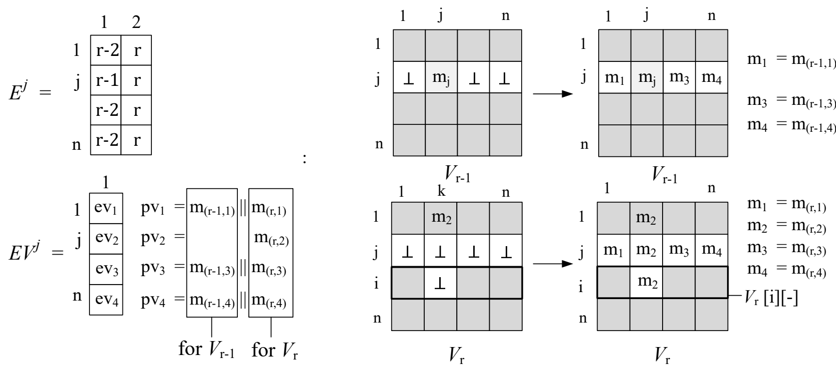
Controlling procedure for PCBW_r

- If there exists a set S of replicas s.t. $|S| \geq 2f + 1$ and for every $p_k \in S$, $\mathsf{Col\_Sum}(M_r, k) \geq 2f + 1$ and $M_r[i][k] = 1$, then the controlling procedure returns 1 and p_i *pcbw-delivers* $(\vec{m}_r, \vec{cv}_r)$ in PCBW_r where:
  - for any $k \in [1, n]$, if $p_k \in S$, then set both $\vec{m}_r[k]$ and $\vec{cv}_r[k]$ as $CV_r[k]$; otherwise set $\vec{cv}_r[k]$ as $CV_r[k]$ and set $\vec{m}_r[k]$ as $\perp$.

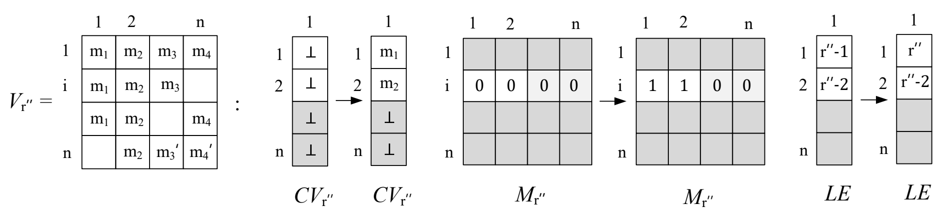
Fig. 8: The PCBW_r protocol at replica p_i. PCBW events are highlighted in blue.



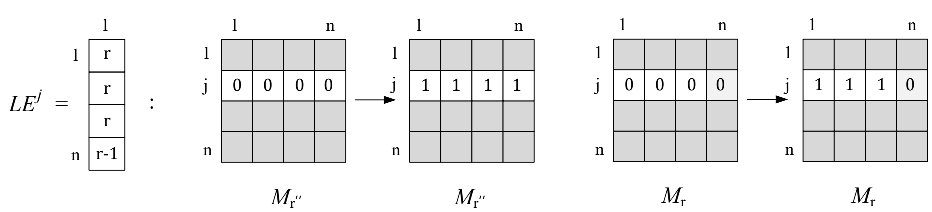
(a) State update according to received values.



(b) State update according to received echo values.



(c) State refresh.



(d) State matrix update. We use r'' to denote any epoch in $[r', r-1]$.

Fig. 9: The update procedure at replica p_i upon receiving a $(\textsc{propose}, r, j, m_j, E^j, EV^j, LE^j)$ message from p_j. In this example, $j = 2$.

p_1 and p_j) such that for any $p_{j'} \in S$, $V_r[j'][k] = m_2$. As $V_r[i][k] = \bot$, p_i sets $V_r[i][k]$ as m_2.

- **(iii) State refresh**. This step further checks whether any value(s) in prior PCBW instances can be confirmed, so CV is updated. In particular, given $\mathsf{PCBW}_{r''}$ where $r'' < r$, if there exist $2f+1$ matching values m in $V_{r''}$ in column k, m is confirmed and $CV_{r''}[k]$ is updated accordingly. Informally speaking, this matches the *cbw-broadcast* event in $\mathsf{CBW}_{r'',k}$. We further update the state matrix M and use M to count the number of confirmed values for each (r'', k) pair.

  ▷ *Example (Figure 9c).* Based on columns 1 and 2 of the $V_{r''}$ matrix, values m_1 and m_2 are confirmed. Then p_i sets $CV_{r''}[1]$ as m_1 and $CV_{r''}[2]$ as m_2. p_i also sets $M_{r''}[i][1]$ and $M_{r''}[i][2]$ as 1. Moreover, since $LE[1] = r'' - 1$, p_i sets $LE[1]$ as r''. For $LE[2]$, as $LE[2] = r'' - 2$, no value from p_2 for $\mathsf{PCBW}_{r''-1}$ has been confirmed by p_i yet, so p_i does not update $LE[2]$.
- **(iv) State matrix update**. Finally, the state matrix $M_{r''}$ for each $\mathsf{PCBW}_{r''}$ (where $r'' < r$) is updated. With the help of the state matrix, we can count the number of replicas that have confirmed each value. As discussed in Sec. 3.2, once $2f+1$ replicas have confirmed a value, the value is committed. Our ultimate goal is to ensure that $2f+1$ values have been committed before any correct replica *pcbw-delivers*.

  ▷ *Example (Figure 9d).* For each row $k = 1, 2, 3$, we have $LE^j[k] = r$. Then p_i sets $M_{r''}[j][k]$ as 1 for any $r'' \in [r', r]$. For $k = n$, as $LE^j[k] = r - 1$, p_i sets $M_{r''}[j][k]$ as 1 for $r'' \in [r', r-1]$.

The controlling procedure. If PCBW_r has not terminated yet, every time replica p_i modifies the local parameters V_r, CV_r, and M_r in the update procedure, p_i also checks whether the controlling procedure is satisfied—after which p_i *pcbw-delivers* $(\vec{m}_r, \vec{cv}_r)$ in PCBW_r and $\vec{m}$ contains at least $n-f$ non-$\bot$ values.

The rule of the controlling procedure is specified as follows: there exists a set S of at least $2f+1$ replicas such that for any $p_k \in S$, column k in M_r has at least $2f+1$ 1's and $M_r[i][k] = 1$ (indicating the corresponding value $CV_r[k]$ is committed). Then p_i *pcbw-delivers* $(\vec{m}_r, \vec{cv}_r)$ such that $\vec{cv}_r$ contains all the confirmed value in CV_r, and $\vec{m}$ contains all the committed values. Here, $\vec{m}_r$ and $\vec{cv}_r$ are two vectors with n components. For any $k \in [1, n]$, if $p_k \in S$, set both $\vec{m}_r[k]$ and $\vec{cv}_r[k]$ as $CV_r[k]$. Otherwise, set $\vec{cv}_r[k]$ as $CV_r[k]$ and $\vec{m}_r$ as $\bot$.

4.4 Proof

We use C to denote the set of correct replicas, where $|C| \geq 2f+1$. Our proof consists of two parts. We first show that our PCBW construction achieves the security properties defined in Sec. 4.1. We then show that for each epoch, using PCBW in a black-box manner, our SQ protocol achieves the security properties of ABC.

Lemma 1. *In* PCBW_r*, if a correct replica* p_j *pcbw-delivers* $(\vec{m}, \vec{cv})$ *where* $\vec{cv}[i] \neq \bot$*, then at least* $f+1$ *correct replicas have received* $\vec{cv}[i]$ *from replica* p_i *and included* $\vec{cv}[i]$ *in their* EV *parameters.*

Proof. If p_j *pcbw-delivers* $(\vec{m}, \vec{cv})$ such that $\vec{cv}[i] \neq \perp$ for epoch r, from the controlling procedure, we know that p_j must have confirmed $\vec{cv}[i]$ and set $CV_r[i]$ as $\vec{cv}[i]$ before it *pcbw-delivers*. Let m_i denote $CV_r[i]$. According to the update procedure, p_j has set $V_r[j][i]$ as m_i and $\mathsf{Col\_Comp}(V_r, i, m_i) \geq 2f+1$. Note that for any $k \neq j$, $V_r[k][i]$ stores the value from p_k as the echo value. Since $\mathsf{Col\_Comp}(V_r, i, m_i) \geq 2f+1$, at least $2f+1$ replicas have included m_i in their $EV[i]$ in the (Propose) messages—indicating that they have received m_i from p_i for epoch r. Therefore, at least $f+1$ correct replicas have received m_i from p_i and included $\vec{cv}[i]$ in their EV parameters in epoch r.

Lemma 2. *In* PCBW_r*, if a correct replica* p_i *pcbw-delivers* $(\vec{m}_i, \vec{cv}_i)$*, another correct replica* p_j *pcbw-delivers* $(\vec{m}_j, \vec{cv}_j)$*, such that for any slot* $k \in [1, n]$*,* $\vec{cv}_i[k] \neq \perp$ *and* $\vec{cv}_j[k] \neq \perp$*, then* $\vec{cv}_i[k] = \vec{cv}_j[k]$*.*

Proof. We prove the lemma by contradiction. Let $\vec{cv}_i[k] = m_{i,k}$ and $\vec{cv}_j[k] = m_{j,k}$. Assume, on the contrary, that $m_{i,k} \neq m_{j,k}$. As p_i is a correct replica, at least $f+1$ correct replicas have received $m_{i,k}$ from p_k and included $m_{i,k}$ in their EV parameters by Lemma 1. Similarly, at least $f+1$ correct replicas have received $m_{i,k}$ from p_k and included $m_{i,k}$ in their EV parameters. As there are $2f+1$ correct replicas, at least one correct replica has stored both $m_{i,k}$ and $m_{j,k}$ in $EV[k]$ for epoch r, a contradiction.

Theorem 1. **(*PCBW-Validity*)***: In* PCBW_r*, if a correct replica* p_i *pcbw-broadcasts a message* m_i*, then every correct replica* p_j *eventually pcbw-s-delivers* $\vec{v}$ *where* $\vec{v}[i] = \{m_i\}$*. If* p_j *pcbw-delivers* $(\vec{m}, \vec{cv})$ *where* $\vec{m}[i] \neq \perp$ *and* $\vec{cv}[i] \neq \perp$*, then* $\vec{m}[i] = \vec{cv}[i] = m_i$*.*

Proof. For each epoch r, if a correct replica p_i *pcbw-broadcasts* a message m_i, p_i broadcasts a $(\text{Propose}, r, i, m_i, -, -, -)$ message pm_i. According to the assumption of the network, every correct replica p_j eventually receives pm_i from p_i. Then p_i executes the state update procedure using pm_i as input. It is not too difficult to see that p_i eventually *pcbw-s-delivers* $\vec{v}$ such that $\vec{v}[i] = \{m_i\}$.

Suppose p_j *pcbw-delivers* $(\vec{m}, \vec{cv})$ such that $\vec{m}[i] \neq \perp$ and $\vec{cv}[i] \neq \perp$ for epoch r. We prove the correctness by contradiction. By Lemma 1, $\vec{cv}[i]$ is *pcbw-broadcast* by p_i and $f+1$ correct replicas have received $\vec{cv}[i]$ from p_i. Since p_i is a correct replica, it *pcbw-broadcasts* only one message m_i in PCBW_r. Then $\vec{cv}[i] = m_i$. In addition, p_j *pcbw-delivers* $(\vec{m}, \vec{cv})$ in the controlling procedure and for any k, $\vec{m}[k]$ is either $\perp$ or $\vec{cv}[k]$. As $\vec{m}[i] \neq \perp$, we have $\vec{m}[i] = \vec{cv}[i] = m_i$.

Theorem 2. **(*PCBW-Consistency*)***: Suppose that a correct replica* p_i *pcbw-delivers* $(\vec{m}, \vec{cv})$ *such that* $\vec{cv}[k] = m \neq \perp$ *for slot* k*. For any correct replica* p_j*:*
(1) if p_j *pcbw-delivers* $(\vec{m'}, \vec{cv'})$ *where* $\vec{cv'}[k] \neq \perp$*, then* $\vec{cv'}[k] = m$*;*
(2) if p_j *pcbw-delivers* $(\vec{m'}, \vec{cv'})$ *where* $\vec{m'}[k] \neq \perp$*, then* $\vec{m'}[k] = m$*.*

Proof. Property (1) follows from Lemma 2. For (2), note that when p_j *pcbw-delivers* $(\vec{m'}, \vec{cv'})$ in the controlling procedure, $\vec{m'}[k]$ is either set as $\vec{cv'}[k]$ or $\perp$. As $\vec{m}[i] \neq \perp$, $\vec{m'}[k] = \vec{cv'}[k] = m$ due to property (1). This completes the proof of the lemma.

Lemma 3. *In* PCBW_r*, if a correct replica* p_i *pcbw-s-delivers* $\vec{v_r}$*, then for any slot* k *such that* $\vec{v_r}[k] = m_k \neq \perp$*,* m_k *is pcbw-broadcast by* p_k*.*

Proof. Based on the update procedure, we distinguish two cases: (1) p_i has received m_k from p_k in a (PROPOSE) message in PCBW_r; (2) p_i has received m_k from $f+1$ replicas as echo values. In case (1), since p_j is a correct replica, m_k is *pcbw-broadcast* by replica p_k. In case (2), at least one correct replica receives m_k from p_k and sends m_k to p_i as an echo value. Then m_k is *pcbw-broadcast* by replica p_k.

Theorem 3. (*PCBW-Weak agreement I*)*: In* PCBW_r*, if a correct replica* p_i *pcbw-delivers* $(\vec{m}, \vec{cv})$ *where* $\vec{cv}[k] \neq \perp$ *for slot* k*, then every correct replica* p_j *eventually pcbw-s-delivers* $\vec{v}$ *where* $\vec{v}[k] \neq \perp$*.*

Proof. Let $\vec{cv}[k] = m_k$. By Lemma 1, at least $f+1$ correct replicas have received m_k from p_k in PCBW_r and will include m_k in their (PROPOSE) messages as echo values in $\mathsf{PCBW}_{r'}$ where $r' > r$. Let S denote the set of $f+1$ correct replicas.

After receiving the (PROPOSE) messages from S in epoch r', every correct replica p_j executes the update procedure. If p_j has not set $V_r[j][k]$ as a non-$\perp$ value before receiving these messages, p_j will update $V_r[j][k]$ to m_k and *pcbw-s-delivers* $V_r[j][-]$ according to our protocol. Otherwise, if p_j sets $V_r[j][k]$ as m'_k and $m'_k \neq \perp$ before p_j receives the (PROPOSE) messages from S, then p_j also has *pcbw-s-delivered* its $V_r[j][-]$. In both cases, the lemma holds.

Theorem 4. (*PCBW-Weak agreement II*)*: In* PCBW_r*, considering the first correct replica* p_i *that pcbw-delivers* $(\vec{m}, \vec{cv})$*. For any slot* k*, if* $\vec{m}[k] = m_k \neq \perp$*, then there exists a set* I *of at least* $f+1$ *correct replicas such that for any* $p_j \in I$*,* p_j *pcbw-delivers* $(\vec{m'}, \vec{cv'})$*, where* $\vec{cv'}[k] = m_k$*.*

Proof. According to the controlling procedure, for any slot k, if $\vec{m}[k] = m_k \neq \perp$, then there exists a set S of $2f+1$ replicas such that for each $p_j \in S$, p_i sets $M_r[j][k]$ as 1 before p_i *pcbw-delivers* $(\vec{m}, \vec{cv})$. Note that $p_i \in S$. Let I denote a set of all correct replicas in S. We have $I \geq f+1$, as there are at most f faulty replicas.

Now we prove that for any $p_j \in I$, p_j *pcbw-delivers* $(\vec{m'}, \vec{cv'})$, where $\vec{cv'}[k] = m_k$. When $j = i$, the statement simply follows, so we consider $j \neq i$. According to our protocol, before *pcbw-delivering* $(\vec{m}, \vec{cv})$, p_i has received a (PROPOSE) message msg from p_j, triggered the update procedure, and set $M_r[j][k]$ as 1 in the state matrix update step. Thus, msg can be parsed as $(\text{PROPOSE}, *, j, m_j, E^j, PV^j, LE^j)$ and $LE^j[k] \geq r$. Additionally, p_j is correct. Before p_j sends msg to p_i, it must have set its $LE[k]$ as a value that is no less than r in the state refresh step. Therefore, p_j has confirmed the *pcbw-broadcast* value from p_k in PCBW_r. Also note p_i *pcbw-delivers* $(\vec{m}, \vec{cv})$ and $\vec{cv}[k] = m_k \neq \perp$. By Lemma 2, p_j has confirmed m_k and set $CV_r[k]$ as m_k before sending msg to p_i. As p_j sent msg to p_i before p_i *pcbw-delivers* $(\vec{m}, \vec{cv})$ and p_i is the first correct replica that *pcbw-delivers* in PCBW_r, p_j already sets its $CV_r[k]$ as m_k before p_i *pcbw-delivers*. By Lemma 6, p_j will eventually *pcbw-delivers* some value. The lemma thus holds.

Theorem 5. (***PCBW-Integrity***)*: Every correct replica pcbw-delivers at most once. Every correct replica pcbw-s-delivers $\vec{v}$ at most $O(n)$ times. For any correct replica p_i:*
(1) if p_i pcbw-delivers $(\vec{m}, \vec{cv})$, then for any $\vec{m}[k] \neq \perp$ (resp., $\vec{cv}[k] \neq \perp$), $\vec{m}[k]$ (resp., $\vec{cv}[k]$) was previously pcbw-broadcast by replica p_k.
(2) if p_i pcbw-s-delivers $\vec{v}$, then for any $\vec{v}[k] \neq \perp$, $\vec{v}[k]$ was previously pcbw-broadcast by replica p_k.

Proof. For any PCBW instance PCBW_r, the controlling procedure returns only once. Therefore, every correct replica *pcbw-delivers* at most once. From the update procedure, each correct replica p_i *pcbw-s-delivers* $\vec{v}$ for epoch r only after p_i sets $V_r[i][k]$ as a non-$\perp$ value for some $k \in [1, n]$. Note that once p_i sets its $V_r[i][k]$ to a non-$\perp$ value, p_i does not change $V_r[i][k]$ anymore. As $|\vec{v}| = n$, p_i *pcbw-s-delivers* $\vec{v}$ at most $O(n)$ times.

Now we prove property (1). Let k_0 denote a slot such that a correct replica *pcbw-delivers* $(\vec{m}, \vec{cv})$ and $\vec{cv}[k_0] \neq \perp$. Then we know that $\vec{cv}[k_0]$ was previously *pcbw-broadcast* by replica p_{k_0} from Lemma 1. Note that for any k such that $\vec{m}[k] \neq \perp$, $\vec{m}[k]$ equals $\vec{cv}[k]$. Therefore, $\vec{m}[k]$ was previously *pcbw-broadcast* by replica p_k.

The correctness of property (2) follows from Lemma 3. This completes the proof.

Theorem 6. (***PCBW-Termination***)*: In PCBW_r, if every correct replica pcbw-broadcasts, every correct replica eventually pcbw-delivers some values.*

Proof. If every correct replica *pcbw-broadcasts*, each correct replica p_i will eventually receive $n-f$ (Propose) messages. We now prove that every correct replica p_i eventually *pcbw-delivers* some values. According to our protocol in Figure 8, p_i *pcbw-delivers* some values if there exists a set S consisting of at least of $2f + 1$ replicas such that for any $p_k \in S$, $M_r[i][k] = 1$ and $\mathsf{Col\_Sum}(M_r, k) \geq 2f + 1$. In the following, we prove that for each correct replica p_i, it will hold that $M_r[i][k] = 1$ and $\mathsf{Col\_Sum}(M_r, k) \geq 2f + 1$ for any k where $p_k \in C$. As $|C| \geq 2f + 1$, p_i eventually *pcbw-delivers* some values.

We first prove that for any $p_k \in C$, p_i will eventually set $M_r[i][k]$ as 1. First note that every correct replica p_i eventually receives the (Propose) messages from any replicas in C. In our protocol, after p_i receives the (Propose$, r, k, m_k, *, *, *, *$) message from $p_k \in C$, p_i sets $V_r[i][k]$ and $EV[k]$ as m_k. The EV vector is included in the (Propose) message in some epoch $r'' > r$. Therefore, every correct replica in C eventually receives the proposed messages for epoch r from every other replica in C, includes them in its EV vector, and then broadcasts them to all replicas. For each $p_j \in C$, after receiving EV^j from p_j, p_i updates its V_r. Eventually, for any $p_j \in C$ and $p_k \in C$, p_i sets $V_r[j][k]$ as m_k, where m_k is proposed by p_k in epoch r. Then p_i eventually sets $M_r[i][k]$ as 1 in the state refresh step.

We now prove that for any $p_k \in C$, the matrix M_r at p_i eventually satisfies the condition that $\mathsf{Col\_Sum}(M_r, k) \geq 2f+1$. As any correct replica p_j eventually sets $M_{r''}[j][k]$ as 1 for every epoch $r'' \leq r$ and $p_k \in C$, p_j will set $LE[k]$ as r, include its LE in the (Propose) message in some epoch, and broadcast the (Propose) message to all replicas. As p_i eventually receives the (Propose) messages from p_j, p_i will set its $M_r[j][k]$ as 1 in the state matrix update step. Therefore, p_i eventually sets its $M_r[j][k]$ as 1 for any $p_j, p_k \in C$.

Therefore, the controlling procedure returns 1 at any correct replica p_i and p_i eventually *pcbw-delivers* some values.

Lemma 4. *In epoch r, if* $\mathsf{Election(r)}$ *returns k and a correct replica p_i mba-proposes m for* MBA_r *where $m \neq \perp$, then m was a-broadcast by p_k for epoch r.*

Proof. Every correct replica p_i *mba-proposes* m if one of the three cases occurs: (1) p_i has *pcbw-delivered* $(\vec{m}, \vec{cv})$ and $\vec{cv}[k] = m$; (2) p_i has received $f+1$ $(\textsc{Send}, r, *, m)$ messages; (3) p_i has *pcbw-s-delivers* v_r such that $\vec{v_r}[k] = m$. We show that in any of the three cases, m was *a-broadcast* by p_k.

- *Case 1:* In this case, the integrity property (1) of PCBW ensures that m was *pcbw-broadcast* by p_k. As every replica *pcbw-broadcasts* its *a-broadcast* value, m was *a-broadcast* by p_k in epoch r.
- *Case 2:* Among the $f+1$ $(\textsc{Send}, r, *, m)$ messages, at least one was sent by a correct replica. The correct replica must have *pcbw-delivered* $(\vec{m_r}, \vec{cv_r})$ such that $\vec{cv_r}[k] = m$. The integrity property (1) of PCBW guarantees that m was *a-broadcast* by p_k in epoch r.
- *Case 3:* The integrity property (2) of PCBW guarantees that m was *a-broadcast* by p_k in epoch r.

Lemma 5. *In epoch r, if* $\mathsf{Election(r)}$ *returns k and a correct replica p_i broadcasts a $(\textsc{Send}, r, i, m)$ message in epoch r, then every correct replica eventually mba-proposes a value or $\perp$ for* MBA_r.

Proof. We show that condition 3) in the MBA phase is eventually satisfied. As p_i broadcasts a $(\textsc{Send}, r, i, m)$ message for epoch r, p_i must have *pcbw-delivered* $(\vec{m_r}, \vec{cv_r})$ such that $\vec{cv_r}[k] = m$. Due to the weak agreement I property of PCBW, every correct replica eventually *pcbw-s-delivers* some value in PCBW_r. Therefore, condition 3) in the MBA phase for epoch r will eventually be satisfied.

Lemma 6. *In epoch r, assuming that the* $\mathsf{Election(r)}$ *function is queried by at least one correct replica and p_i is the first correct replica that queries* $\mathsf{Election(r)}$. *If* $\mathsf{Election(r)}$ *returns k and p_i pcbw-delivers $(\vec{m_r}, \vec{cv_r})$ in* PCBW_r *such that $\vec{m_r}[k] = m \neq \perp$, then all correct replicas mba-propose m for* MBA_r.

Proof. Our proof consists of three parts. First, we show that every correct replica *mba-proposes* some value for MBA_r. Second, we show that no correct replicas *mba-proposes* $\perp$. Last, we show that every correct replica *mba-proposes* m.

We begin with the first part. Since $\mathsf{Election(r)}$ returns k and p_i *pcbw-delivers* $(\vec{m_r}, \vec{cv_r})$ such that $\vec{m_r}[k] = m \neq \perp$, in the exchange phase, p_i will broadcast $(\textsc{Send}, r, i, m)$. By Lemma 5, every correct replica eventually *mba-proposes* some value for MBA_r.

We now show that no correct replica *mba-proposes* $\perp$. As p_i is the first correct replica that queries $\mathsf{Election(r)}$, p_i is also the first replica that *pcbw-delivers* a pair of output $(\vec{m}, \vec{cv})$ in PCBW_r where $\vec{m_r}[k] = m \neq \bot$. Due to the weak agreement II property of PCBW, there exists a set I of $f+1$ correct replicas such that for any $p_j \in I$, p_j *pcbw-delivers* $(\vec{m'}, \vec{cv'})$ where $\vec{cv'}[k] = m$. Hence, at least $f+1$ correct replicas will broadcast $(\textsc{Send}, r, *, m)$ in the exchange phase, and condition 2) for MBA_r will never be satisfied. Thus, no correct replica *mba-proposes* $\bot$ for MBA_r.

Last, from Lemma 4, if a correct replica *mba-proposes* m, m is *a-broadcast* by p_k. As p_k is correct, all correct replicas *mba-propose* the same value m.

Lemma 7. *In epoch r, any correct replica eventually mba-decides for* MBA_r.

Proof. Note there are $n-f$ correct replicas and each correct replica sends a $(\textsc{Propose})$ message in each epoch r. Due to the termination property of PCBW, every correct replica eventually *pcbw-delivers* some values.

Then according to our protocol, correct replicas will query the $\mathsf{Election(r)}$ function. After k is returned by $\mathsf{Election(r)}$, every correct replica broadcasts $CV_r[k]$ in its $(\textsc{Send})$ messages. We now show that every correct replica *mba-proposes* some value.

After obtaining an output for $\mathsf{Election(r)}$, we distinguish two cases: 1) at least one correct replica p_i broadcasts $(\textsc{Send}, r, i, m)$; 2) every correct replica broadcasts $(\textsc{Send}, r, *, \bot)$ for epoch r. We show that every correct replica eventually *mba-proposes* so eventually every correct replica *mba-decides* according to the termination property of MBA.

- *Case 1:* In this case, according to Lemma 5, any correct replica eventually *mba-proposes* a value (or $\bot$) for MBA_r.
- *Case 2:* In this case, after receiving all the $(\textsc{Send})$ messages from correct replicas for epoch r, condition 2) in the MBA phase will eventually be satisfied. Thus, every correct replica will *mba-proposes* some value for MBA_r.

Lemma 8. *In epoch r, if a correct replica p_i a-delivers m and another correct replica p_j a-delivers m', then $m = m'$.*

Proof. We prove the lemma by contradiction. Assume, on the contrary, that $m \neq m'$. According to our protocol, if p_i *a-delivers* m, it *mba-decides* m in MBA_r. If p_j *a-delivers* m', it *mba-decides* $m' \neq m$ in MBA_r, violating the agreement property of MBA. Therefore, it holds that $m = m'$.

Theorem 7 (ABC-Agreement). *If any correct replica a-delivers a message m, then every correct replica a-delivers m.*

Proof. If a correct replica *a-delivers* a message in epoch r, then according to Lemma 7, any correct replica will eventually *mba-decide* for MBA_r and then *a-deliver* some value.

Moreover, if a correct replica p_i *a-delivers* a message m in epoch r, it has *mba-decided* m in MBA_r. The termination and agreement properties of MBA thus guarantee that any correct replica *mba-decides* m and then *a-delivers* m.

Theorem 8 (ABC-Total order). *If a correct replica a-delivers a message m before a-delivering m', then no correct replica a-delivers a message m' without first a-delivering m.*

Proof. We prove the theorem by contradiction. Every correct replica *a-delivers* the messages according to the sequence of epoch numbers. We assume that a correct replica p_i *a-delivers* m in epoch r_1 and m' in epoch r_2 where $r_1 < r_2$. Meanwhile, another correct replica p_j *a-delivers* m' in epoch r_3 and m in epoch r_4 where $r_3 < r_4$. We consider two cases: (1) $r_1 < r_4$ or $r_1 > r_4$; (2) $r_1 = r_4$.

- *Case 1:* Without loss of generality, assume that $r_1 < r_4$. p_i *a-delivers* m in epoch r_1 (and *mba-decides* m in MBA_{r_1}) and p_j *a-delivers* m in epoch r_4. Since p_j *a-delivers* m in epoch r_4, it has not previously *a-delivered* m in any prior epochs (due to the uniqueness of messages). Therefore, it must have *a-delivered* m'' in epoch r_1 such that $m'' \neq m$ and *mba-decided* m'' in MBA_{r_1}, a violation of the agreement property of MBA.
- *Case 2:* Since $r_1 < r_2$ and $r_3 < r_4$, we know that $r_3 < r_2$. Note that p_i *a-delivers* m' in epoch r_2 and p_j *a-delivers* m' in epoch r_3. Similar to case (1), there is a contradiction.

Theorem 9 (ABC-Integrity). *Every correct replica a-delivers a message at most once. If a correct replica a-delivers a message m, then m was previously a-broadcast by some replica.*

Proof. We first prove the first part. Every correct replica *a-delivers* a message after it *mba-decides*. According to the integrity property of MBA, every correct replica *a-delivers* a message once.

We now prove the second part. According to our protocol, if a correct replica *a-delivers* a message m in epoch r, then MBA_r outputs m. The non-intrusion property of MBA ensures that m is *mba-proposed* by a correct replica. By Lemma 4, m was previously *a-broadcast* by some replica.

Lemma 9. *With a probability of at least $1/3$, in every epoch r correct replicas a-deliver a value a-broadcast by a correct replica.*

Proof. According to Lemma 6, for any r, every correct replica eventually *pcbw-delivers* some values and queries the $\mathsf{Election(r)}$ function. Let p_i denote the first correct replica that *pcbw-delivers* $(\vec{m}, \vec{cv})$ and then queries $\mathsf{Election(r)}$. When p_i queries $\mathsf{Election(r)}$, $\vec{m}$ has at least $2f+1$ non-$\perp$ values. Let the replicas that propose these values in PCBW_r be S. The probability that p_k is a correct replica and $p_k \in S$ is at least $1/3$, as

$$\Pr[\mathsf{Election(r)} \in S \cap C] \geq \frac{2f+1+2f+1-(3f+1)}{n} > \frac{1}{3}. \tag{1}$$

Additionally, according to Lemma 6, if p_k is a correct replica and $p_k \in S$, all correct replicas *mba-propose* the value proposed by p_k. Then the validity property of MBA ensures that any correct replica *a-delivers* a value proposed by p_k in epoch r. Therefore, the correct replicas contained in S form a key set.

Let *success* be the event that correct replicas *a-deliver* a value *a-broadcast* by a correct replica in epoch r. We have the following:

$$\begin{aligned}\Pr[success] &= \Pr[success|\mathsf{Election(r)} \in \mathcal{S} \cap \mathcal{C}]\Pr[\mathsf{Election(r)} \in \mathcal{S} \cap \mathcal{C}] + \\ &\quad \Pr[success|\mathsf{Election(r)} \in \overline{\mathcal{S} \cap \mathcal{C}}]\Pr[\mathsf{Election(r)} \in \overline{\mathcal{S} \cap \mathcal{C}}] \\ &\geq \Pr[success|\mathsf{Election(r)} \in \mathcal{S} \cap \mathcal{C}]\Pr[\mathsf{Election(r)} \in \mathcal{S} \cap \mathcal{C}] \\ &= \Pr[\mathsf{Election(r)} \in \mathcal{S} \cap \mathcal{C}] > \frac{1}{3}.\end{aligned} \tag{2}$$

Thus, the probability that the *success* event occurs is at least $1/3$.

Lemma 10 (Efficiency). *If a correct replica a-delivers a message m, the probability that m is either $\perp$ or a-broadcast by a faulty replica is at most $2/3$, i.e., SQ achieves $O(1)$ time complexity.*

Proof. According to Lemma 9, for each epoch r, with a probability of at least $1/3$, a correct replica *a-delivers* a message m *a-broadcast* by a correct replica. Therefore, the probability that m is either $\perp$ or *a-broadcast* by a faulty replica is at most $2/3$.

Theorem 10 (Liveness). *If a correct replica a-broadcasts a message m, then it eventually a-delivers m.*

Proof. If a correct replica p_i *a-broadcasts* m in epoch r, then it *pcbw-broadcasts* m in PCBW_r. The validity property ensures that every correct replica eventually *pcbw-s-delivers* $\vec{v}$ such that $\vec{v}[i] = m$. Furthermore, if a correct replica *pcbw-delivers* $(\vec{m}, \vec{cm})$ such that $\vec{m}[i] = \vec{cm}[i] \neq \perp$, $\vec{m}[i] = \vec{cm}[i] = m$.

Before m is *a-delivered*, any correct replica stores m in its echo buffer in an epoch $r_1 \geq r$. Recall that there exists a predefined liveness parameter lp (epoch number). If all the messages proposed in epochs lower than r have been *a-delivered* and m has not been *a-delivered* by epoch $r + lp$, every replica that stores m in its echo buffer will propose m.

We now prove the theorem by induction on epoch number r. We start from $r = 1$. Let r^* be $max\{r + lp, r_1\}$. Before m is *a-delivered*, all correct replicas will *a-broadcast* m in epochs higher than r^*. According to Lemma 10, p_i will eventually *a-deliver* m in some epoch.

Assume the theorem holds from $r = 1$ to $r = r - 1$. Then any message proposed in an epoch lower than r is eventually *a-delivered*. Assume the messages proposed in epoch 1 to epoch $r - 1$ have been *a-delivered* by a correct replica when it is in epoch r_2. Let r^* be $max\{r + lp, r_1, r_2\}$. Before m is *a-delivered*, all correct replicas will *a-broadcast* m in epochs larger than r^*. According to Lemma 10, p_i will eventually *a-deliver* m in some epoch.

Theorem 11 (Complexity). *SQ achieves $O(n^2)$ message complexity, $O(Ln^3)$ communication complexity, and $O(1)$ time complexity.*

Proof. The first three phases in SQ all have $O(n^2)$ messages. As the MBA phase can be also realized using $O(n^2)$ messages [42], SQ has $O(n^2)$ messages.

We now analyze the communication complexity. Our PCBW construction has $O(Ln^3)$ communication because the (Propose) message includes a proposed value (length L), E (n epoch numbers), PV, and LE (n epoch numbers). For PV, each $PV[k]$ for $k \in [1, n]$ contains a constant number of L-bit values. Hence, the communication of the propose phase is $O(Ln^3)$. For the election phase, assuming a Rabin dealer, the communication complexity is $O(n \log n)$. In the exchange phase, each (Send) message includes at most two proposed messages so the communication complexity is $O(Ln^2)$. In the MBA phase, as the input to MBA is either a proposed message or $\perp$, the MBA phase has $O(Ln^2)$ communication. Therefore, SQ achieves $O(Ln^3)$ communication complexity. Finally, SQ achieves $O(1)$ time complexity according to Lemma 9.

5 A Communication-Efficient Variant of SQ From Hash Functions

In this section, we present SQ_h, a communication-efficient variant of SQ by additionally using hash functions. Recall that SQ has $O(Ln^3)$ communication complexity, because in our PCBW construction every replica broadcasts its received values from all replicas in the (propose) message. In SQ_h, we modify our PCBW construction by replacing the values included in the (propose) messages with their hashes. SQ_h achieves $O(Ln^2 + \kappa n^3)$ communication, where κ is the security parameter, i.e., the length of a hash digest. In this section, we present the PCBW variant and the main protocol remains the same as that in Figure 7.

5.1 The PCBW Protocol

We present the pseudocode of the hash variant of PCBW in Figure 10. Here we highlight the changes from Figure 8 to Figure 10.

First, we modify the parameters. We re-define the V_r parameter: V_r is now a vector instead of a matrix that stores only the proposed message directly received from each replica. For example, $V_r[k]$ stores the proposed message received from p_k in epoch r. Moreover, we define a new vector EH for storing hashes of the received messages (to replace EV). We also introduce a new parameter H_r, an $n \times n$ matrix storing hashes.

Among all the parameters in this variant, the E, EH, and LE parameters are initialized at the beginning of the protocol. Meanwhile, for each PCBW_r, each replica initializes the V_r, H_r, M_r, and CV_r parameters; these parameters are cleared only after epoch r completes.

We explain the two new parameters EH and H_r in detail below.

- EH is an n-value vector that stores the hashes of the proposed messages (also called *echo hashes*). For $k \in [1, n]$, $EH[k]$ contains a constant number of hashes. Intuitively speaking, echo hashes are hashes of the echo values EV used in SQ.

Initialization:

- $E \leftarrow [\perp]^{n\times 2}$. Each element of E stores a PCBW instance id.
- $EH \leftarrow [\perp]^{n}$. Each element of PH stores a constant number of hashes.
- $LE \leftarrow [\perp]^{n}$. Each element of LE stores a PCBW instance id.
- Initialize the following parameters for PCBW_r:
  - $V_r \leftarrow [\perp]^{n}$. Each element of V_r is a proposed message from a replica.
  - $H_r \leftarrow [\perp]^{n\times n}$. Each element of H_r is the hash of a proposed message.
  - $M_r \leftarrow [\perp]^{n\times n}$. Each element of M_r is one bit.
  - $CV_r \leftarrow [\perp]^{n}$. Each element of CV_r is a confirmed value.

Let *confirm*(r, k, m_k) be the following predicate: *confirm*$(r, m_k, k) \equiv (V_r[k] = m_k \wedge m_k \neq \perp \wedge H_r[i][k] = h_r \wedge \mathsf{Col\_Comp}(H_r, k, h_r) \geq 2f+1)$

- **(Broadcast)** Upon *pcbw-broadcast*(m_i) in PCBW_r: Broadcast $(\textsc{Propose}, r, i, m_i, E, EH, LE)$ For every $k \in [1, n]$, set $E[k][1]$ as $E[k][2]$, and set $EH[k]$ as $\perp$.
- Upon receiving $(\textsc{Propose}, r, j, m_j, E^j, EH^j, LE^j)$ from p_j:
  Let $\mathsf{PCBW}_{r'}$ be the instance such that every $\mathsf{PCBW}_{r''}$ with $r'' < r'$ has completed. If $V_{r-1}[i][j] \neq \perp$ and $V_r[i][j] = \perp$, then start the update procedure for $(\textsc{Propose}, r, j, m_j, E^j, EH^j, LE^j)$ as follows:

  (state update according to received values)
  - Set $V_r[j]$ as m_j and *pcbw-s-deliver* V_r in PCBW_r.
  - Set $H_r[i][j]$ as $hash(m_j)$, set $EH[j]$ as $EH[j]||hash(m_j)$, and set $E[j][2]$ as r.

  (state update according to received echo values)
  For $k \in [1, n]$, let $e_{k,1}$ be $E^j[k][1]$ and $e_{k,2}$ be $E^j[k][2]$:
  - Parse $PH^j[k]$ as a vector of hashes $h_{e_{k,1}+1}||...||h_{e_{k,2}}$.
  - For every $e \in [e_{k,1}+1, e_{k,2}]$, if $H_e[j][k] = \perp$, then:
    - $H_e[j][k]$ as h_e.
    - if $H_e[i][k] = \perp$ and there exists a set S such that $|S| \geq f+1$ and for every $p_{j'} \in S$, $H_e[j'][k] = hash(m)$, then set $H_e[i][k]$ as $hash(m)$, set $V_e[k]$ as m, and *pcbw-s-deliver* V_e in PCBW_e.

  (state refresh)
  For $k \in [1, n]$, $r'' \in [r', r]$, if *confirm*$(r'', V_{r''}[k], k)$= 1, then
  - Set $CV_{r''}[k]$ as $V_{r''}[k]$ and set $M_{r''}[i][k]$ as 1.
  - Set $LE[k]$ as the largest r^* such that for every $r'' \in [r', r^*]$, $M_{r''}[i][k] = 1$

  (state matrix update)
  For $k \in [1, n]$, let e_k denote $LE^j[k]$: for any $r'' \in [r', e_k]$, set $M_{r''}[j][k]$ as 1.

Controlling procedure for PCBW_r

- If there exists a set S of replicas such that $|S| \geq 2f+1$ and for every $p_k \in S$, $\mathsf{Col\_Sum}(M_r, k) \geq 2f+1$ and $M_r[i][k] = 1$, then the controlling procedure returns 1 and p_i *pcbw-delivers* $(\vec{m_r}, \vec{cv_r})$ in PCBW_r where:
  - for any $k \in [1, n]$, if $p_k \in S$, then set both $\vec{m_r}[k]$ and $\vec{cv_r}[k]$ as $CV_r[k]$; otherwise set $\vec{cv_r}[k]$ as $CV_r[k]$ and set $\vec{m_r}[k]$ as $\perp$.

Fig. 10: The hash variant of PCBW_r protocol at replica p_i. PCBW events are highlighted in blue.

- H_r is an $n \times n$ matrix and each element is an echo hash. Informally speaking, H_r is a matrix that stores the hashes of the values in V_r used in SQ. For replica p_i, row i stores the hashes of the values p_i receives from other replicas and other rows store the hashes of the received values by other replicas.

Second, we modify the parameters included in the (Propose) message. The (Propose) message now includes E, EH, and LE. The update procedure differs slightly from that in SQ. In particular, the step for *state update according to received values* now updates H_r and EH. The step for *state update according to echo hashes* now updates the H_r matrix using the hashes included in the EH^j parameter.

Finally, we change the definition of the confirm predicate. In PCBW_r, each replica p_i *confirms* a value m_k if p_i has stored a non-$\perp$ value m_k in $V_r[k]$, and there exists a set of at least $2f + 1$ replicas such that for any p_j in the set, $H_r[j][k_r] = hash(m_k)$.

5.2 Proof

Theorem 12. *The hash variant of the PCBW protocol achieves validity, consistency, weak agreement I, weak agreement II, integrity, and termination.*

Proof. We first prove that Lemma 1 still holds for the new PCBW protocol. Since p_i *pcbw-delivers* $(\vec{m}, \vec{cv})$ for PCBW_r and $\vec{cv}[i] = m_i \neq \perp$, p_i has confirmed m_i in PCBW_r before it *pcbw-delivers*. Therefore, p_i has received $\vec{cv}[i]$ from p_j and at least $2f + 1$ replicas included $hash(m_i)$ in their EH parameters. Due to the collision resistance of hash function, these replicas store the same m_i in their V_r parameters with an overwhelming probability. As at least $f+1$ of these replicas are correct, Lemma 1 holds.

Similarly, Lemma 2 and Lemma 3 can be proved for the hash-based variant of PCBW. Accordingly, the proofs for the validity, consistency, weak agreement I, weak agreement II, integrity, and termination properties follow from those of the PCBW presented in Figure 8.

Theorem 13 (Complexity). *SQ_h achieves $O(n^2)$ message complexity and $O(Ln^2 + \kappa n^3)$ communication complexity.*

Proof. As we do not modify the message workflow, the message complexity of the SQ_h remains $O(n^2)$. We focus on the communication complexity. In the propose phase, the (Propose) message now includes a replica's proposed value (length L), E (n epoch numbers), PH, and LE (n epoch numbers). For PH, as each $PH[k]$ for $k \in [1, n]$ contains a constant number of hash values, the communication cost of PH is at most $O(\kappa n)$. Therefore, the communication complexity of the propose phase is $O(Ln^2 + \kappa n^3)$. The communication for other phases remains the same as that in SQ—$O(Ln^2)$. Hence, the hash variant of SQ achieves $O(Ln^2 + \kappa n^3)$ communication complexity.

References

1. Abraham, I., Asharov, G., Patra, A., Stern, G.: Perfectly secure asynchronous agreement on a core set in constant expected time. Cryptology ePrint Archive, Paper 2023/1130 (2023)
2. Abraham, I., Ben-David, N., Yandamuri, S.: Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement. PODC (2022)
3. Abraham, I., Malkhi, D., Spiegelman, A.: Asymptotically optimal validated asynchronous byzantine agreement. In: PODC. pp. 337–346. ACM (2019)
4. Alhaddad, N., Das, S., Duan, S., Ren, L., Varia, M., Xiang, Z., Zhang, H.: Balanced byzantine reliable broadcast with near-optimal communication and improved computation. In: PODC. pp. 399–417 (2022)
5. B. Libert, M.J., Yung, M.: Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. In: Theoretical Computer Science (2016)
6. Bacho, R., Loss, J.: On the adaptive security of the threshold bls signature scheme. In: CCS. pp. 193–207 (2022)
7. Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. STOC (1993)
8. Ben-Or, M., El-Yaniv, R.: Resilient-optimal interactive consistency in constant time. Distrib. Comput. (2003)
9. Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience. In: PODC. pp. 183–192. ACM (1994)
10. Bhat, A., Shrestha, N., Luo, Z., Kate, A., Nayak, K.: Randpiper–reconfiguration-friendly random beacons with quadratic communication. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 3502–3524 (2021)
11. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: PKC (2003)
12. Bracha, G.: An asynchronous [(n-1)/3]-resilient consensus protocol. In: PODC. pp. 154–162. ACM (1984)
13. Bracha, G.: Asynchronous Byzantine agreement protocols. Information and Computation **75**(2), 130–143 (1987)
14. Cachin, C., Guerraoui, R., Rodrigues, L.: Introduction to reliable and secure distributed programming. Springer Science & Business Media (2011)
15. Cachin, C., Guerraoui, R., Rodrigues, L.: Introduction to Reliable and Secure Distributed Programming. 2nd edn. (2011)
16. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: CRYPTO. pp. 524–541. Springer (2001)
17. Cachin, C., Kursawe, K., Shoup, V.: Random oracles in constantinople: Practical asynchronous Byzantine agreement using cryptography. Journal of Cryptology **18**(3), 219–246 (2005)
18. Correia, M., Neves, N.F., Veríssimo, P.: From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. Comput. J. **49**(1), 82–96 (2006)
19. Crain, T.: 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. CoRR **abs/2002.08765** (2020)
20. Das, S., Xiang, Z., Kokoris-Kogias, L., Ren, L.: Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. Cryptology ePrint Archive (2022)

21. Das, S., Yurek, T., Xiang, Z., Miller, A.K., Kokoris-Kogias, L., Ren, L.: Practical asynchronous distributed key generation. In: IEEE Symposium on Security and Privacy. pp. 2518–2534. IEEE (2022)
22. Dolev, D.: The byzantine generals strike again. Journal of Algorithms **3**(1), 14–30 (1982)
23. Drand: Drand - a distributed randomness beacon daemon. `https://github.com/drand/drand` (accessed Oct 2023)
24. Duan, S., Reiter, M.K., Zhang, H.: BEAT: Asynchronous bft made practical. In: CCS. pp. 2028–2041. ACM (2018)
25. Duan, S., Wang, X., Zhang, H.: Fin: Practical signature-free asynchronous common subset in constant time. In: CCS (2023)
26. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of distributed consensus with one faulty process. Tech. rep., Massachusetts Inst of Tech Cambridge lab for Computer Science (1982)
27. Galindo, D., Liu, J., Ordean, M., Wong, J.M.: Fully distributed verifiable random functions and their application to decentralised random beacons. In: European Symposium on Security and Privacy (EuroS&P). pp. 88–102 (2021)
28. Guo, B., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Speeding dumbo: Pushing asynchronous BFT closer to practice. In: NDSS (2022)
29. Guo, B., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo: Faster asynchronous bft protocols. In: CCS (2020)
30. I. T. L. Computer Security Division: Interoperable randomness beacons: Csrc. `https://csrc.nist.gov/projects/interoperable-randomness-beacons` (accessed Oct 2023)
31. Keidar, I., Kokoris-Kogias, E., Naor, O., Spiegelman, A.: All you need is dag. PODC (2021)
32. Libert, B., Joye, M., Yung, M.: Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. Theoretical Computer Science **645**, 1–24 (2016)
33. Liu, C., Duan, S., Zhang, H.: Epic: Efficient asynchronous bft with adaptive security. In: DSN (2020)
34. Loss, J., Moran, T.: Combining asynchronous and synchronous Byzantine agreement: The best of both worlds. IACR Cryptology ePrint Archive **2018**, 235 (2018)
35. Lu, Y., Lu, Z., Tang, Q., Wang, G.: Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In: Proceedings of the 39th Symposium on Principles of Distributed Computing. pp. 129–138 (2020)
36. MacBrough, E.: Cobalt: Bft governance in open networks. arXiv preprint arXiv:1802.07240 (2018)
37. Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of bft protocols. In: CCS. pp. 31–42. ACM (2016)
38. Moniz, H., Neves, N.F., Correia, M., Verissimo, P.: Ritas: Services for randomized intrusion tolerance. TDSC **8**(1), 122–136 (2008)
39. Mostefaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous Byzantine consensus with $t \leq n/3$ and $o(n^2)$ messages. In: PODC. pp. 2–9. ACM (2014)
40. Mostéfaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous binary byzantine consensus with $\mathrm{t} < \mathrm{n}/3$, $\mathrm{o}(\mathrm{n}^2)$ messages, and O(1) expected time. J. ACM **62**(4), 31:1–31:21 (2015)
41. Mostéfaoui, A., Raynal, M.: Signature-free broadcast-based intrusion tolerance: never decide a byzantine value. In: OPODIS. pp. 143–158. Springer (2010)

42. Mostéfaoui, A., Raynal, M.: 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), 501–520 (2017)
43. Rabin, M.O.: Randomized byzantine generals. In: SFCS. pp. 403–409. IEEE (1983)
44. Reiter, M.K.: Secure agreement protocols: Reliable and atomic group multicast in rampart. In: CCS. pp. 68–80 (1994)
45. Shoup, V.: Practical threshold signatures. In: EUROCRYPT (2000)
46. Toueg, S.: Randomized byzantine agreements. In: Proceedings of the third annual ACM symposium on Principles of distributed computing. pp. 163–178 (1984)
47. Zhang, H., Duan, S.: PACE: Fully parallelizable BFT from reproposable byzantine agreement. ACM CCS (2022)
48. Zhang, H., Duan, S., Zhao, B., Zhu, L.: Waterbear: Practical asynchronous bft matching security guarantees of partially synchronous bft. In: Usenix Security (2023)