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

Xiao Sui
Shandong University
suixiao@mail.sdu.edu.cn

Xin Wang
Tsinghua University
wangxin87@tsinghua.edu.cn

Sisi Duan*
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). Completely asynchronous protocols with no timing assumptions [3, 8, 10, 16, 26, 31, 32, 40, 42, 54] 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 [28], 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) [49]. Consensus protocols using local coins, however, terminate in exponential expected time [19, 43, 55]. Thus, to avoid exponential running time, asynchronous consensus protocols need to use common coins.

We follow a long line of work in consensus [8, 10, 19, 20, 41, 44–47, 54] 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), asynchronous common subset (ACS), and asynchronous ABC. Informally speaking, ABA reaches agreement on binary values, MBA reaches agreement on values from an arbitrary domain, ACS reaches agreement on a subset of values, while ABC reaches agreement on the order of a sequence of messages. One can directly obtain an ABC from ACS by running ACS instances sequentially, but additional procedures need to be introduced to obtain ABC from ABA or MBA.

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 [44, 45]. 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 [41], Crain's ABA [20], Pillar [54]). 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 [47] 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. This is in part because almost all known asynchronous ABC protocols are directly built from ACS. 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). {For example,} 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.

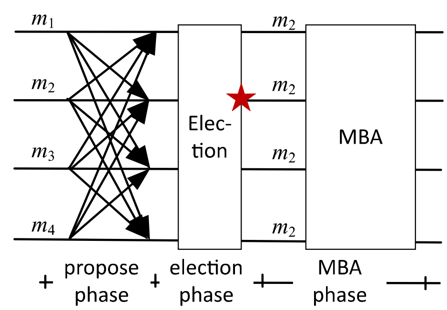
*Corresponding author

	paradigm	protocol	time	message
Computational	MVBA-based	CKPS [16]	$O(1)$	$O(n^2)$
		Dumbo [32]	$O(1)$	$O(n^3)$
		Speeding-Dumbo [31]	$O(1)$	$O(n^2)$
		AMS [3]	$O(1)$	$O(n^2)$
		Dumbo-MVBA [40]	$O(1)$	$O(n^2)$
Information-Theoretic (by design)	ABA-based	BKR [10]	$O(\log n)/O(1)$	$O(n^3)/O(n^4)$
		HoneyBadger [42]	$O(\log n)$	$O(n^3)$
		BEAT [26]	$O(\log n)$	$O(n^3)$
		EPIC [38]	$O(\log n)$	$O(n^3)$
	RABA-based	PACE [54]	$O(\log n)$	$O(n^3)$
		FIN [27]	$O(1)$	$O(n^3)$
	DAG-based	DAG-Rider[34]	$O(1)$	$O(n^3)$
	MBA-based	**SQ (this work)**	$\boldsymbol{O(1)}$	$\boldsymbol{O(n^2)}$

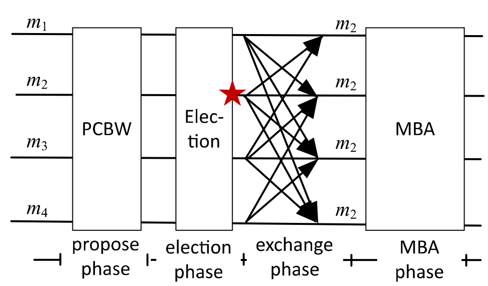
Table 1. Comparison of ABC protocols with sublinear time complexity. RABA denotes reproposable ABA [27, 54]. DAG denotes the directed acyclic graph. Note that the implemented systems in the information-theoretic (IT) category (HoneyBadger, BEAT, EPIC, PACE, FIN) are not IT-secure systems, but they—"by 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 [10], 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 [9]).

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 [19, 43], 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, 40]. 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[1].

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.



(a) Our toy construction. No liveness during failures.



(b) Overview of SQ.

Figure 1. Overview of our approach.

[1]In particular, the non-validated versions of all MVBA protocols we are aware of do not satisfy the validity property of MBA (see Sec. 2 for the definition of validity). One can first use MVBA to build ACS, and then use ACS to build MBA. However, additional procedures need to be introduced. To the best of our knowledge, there does not exist any signature-free ACS with $O(n^2)$ messages so the transformation in the signature-free setting has additional costs.

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 eventually outputs m_{k_r}. However, if p_{k_r} is faulty, 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.[2] 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.

Communication complexity. By assuming the existence of the common coin object (Rabin dealer), we provide two ABC protocols: SQ and its hash-based variant SQ_h. We discuss the communication complexity of the protocols.

- 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 reliable broadcast (RBC) using Bracha's RBC [13, 14] (an IT-secure RBC).
- 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].

Compared to the existing asynchronous signature-free ABC protocol, we only optimize the message complexity and retain the same communication complexity. However, lowering message complexity is both practically useful and theoretically challenging. On the practical side, consider Speeding Dumbo [31] as an example. Speeding Dumb achieves the same communication complexity as Dumbo [32] but lowers the message complexity from $O(n^3)$ to $O(n^2)$. By doing so, the throughput is significantly improved (up to 2x). On the theoretical side, all known ABC protocols with $O(n^2)$ messages are computationally secure by design. Our protocols are thus the first signature-free protocols with optimal messages.

Our contributions. We make the following contributions.

- 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). SQ_h achieves $O(n^2)$ messages and the same communication complexity as the state-of-the-art asynchronous BFT protocols.
- As a by-product, we introduced a warm-up protocol SQ_0 (Sec. 3.2), an MBA-based ABC protocol with $O(n^3)$ messages and $O(1)$ time, which already outperforms the state-of-the-art MBA-based ABC that has $O(n^4)$ messages and $O(n)$ time [19, 43]. SQ_0 can also be used as an MVBA protocol with minor changes and might be of independent interest.
- We implement a prototype of SQ_h and our experiments on up to 61 Amazon EC2 instances show that SQ_h outperforms FIN [27], the state-of-the-art asynchronous protocol. The drawback is that SQ_h can only be used as an ABC protocol instead of an asynchronous common subset protocol (i.e., FIN).

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).

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

Throughout the paper, we use the term *broadcast* to represent best-effort broadcast, i.e., the sender sends 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 2.1 (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 message *a-broadcast* by one replica at a time. In some other protocols, every correct replica *a-delivers* a union of several payload messages *a-broadcast* by some replicas. Our work considers the former case and we show how to transform SQ_h to the latter case.

We may use the term *value* to denote the message some replica *a-broadcasts* or *a-delivers* to differentiate it from the messages in the protocol.

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 2.2 (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 defined in [19, 46, 47]: a decided value must be a value proposed by a correct replica (possibly $\perp$). The two properties prevent a value proposed only by faulty replicas from being decided.

Common coins. We consider a common coin primitive, a notion first introduced by Rabin [49]. Following the definitions in prior works [15, 20, 45, 49], we distinguish low-threshold common coins ($f+1$ threshold) from high-threshold common coins ($2f+1$ threshold). A low-threshold 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 2.3 (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 high-threshold 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, [7, 37, 39] 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 low-threshold common coin object or a high-threshold common coin object. When Election() 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 Election() function is instantiated from a Rabin dealer [49], 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, 50, 52], a designated replica broadcasts a message to a group of replicas. A CBC protocol is specified by *c-broadcast* and *c-deliver*.

Definition 2.4 (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 [13, 14] 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: Most 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). Most MVBA-based ABC protocols leverage (non-interactive) threshold signatures to achieve $O(n^2)$ messages and $O(1)$ expected time. However, threshold signatures require the trusted setup, strong models (e.g., random oracles), and assume the hardness of computational problems [7, 12, 37, 51].

ABA-based (Figure 2b). The BKR paradigm due to Ben-Or, Kelmer, and Rabin relies on n parallel reliable broadcast (RBC) instances and n parallel asynchronous binary agreement (ABA) instances.[3] HoneyBadgerBFT [42], BEAT [26], EPIC [38] 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 [54] 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 used (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. Part of the protocol is also an MVBA. Compared to MVBA in the computational setting, the FIN MVBA has $O(n^3)$ messages.

DAG-based (Figure 2e). The DAG-Rider paradigm relies on RBC and DAG-based data structures to build ABC [34]. 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.

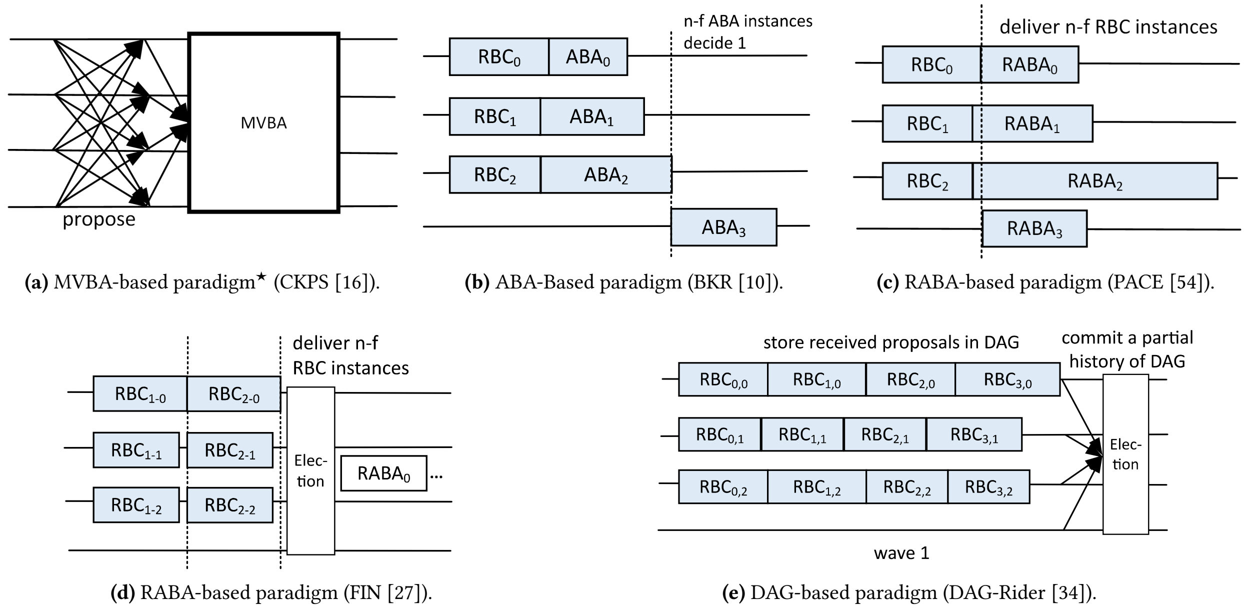
In summary, there is a mismatch in the message and time complexity between the MVBA-based approach and the other three signature-free approaches. The common characteristic of all signature-free ABC approaches is that they all use parallel RBC protocols, which leads to $O(n^3)$ message complexity for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

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

[3] 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 [8].



(a) MVBA-based paradigm★ (CKPS [16]). **(b)** ABA-Based paradigm (BKR [10]). **(c)** RABA-based paradigm (PACE [54]).

(d) RABA-based paradigm (FIN [27]). **(e)** DAG-based paradigm (DAG-Rider [34]).

Figure 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. ★One exception of MVBA-based paradigm is FIN [27], which is signature-free and has $O(n^3)$ messages. We classify FIN as a RABA-based paradigm.

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.

	p_1	p_2	p_3	p_4
p_1	√	√		√
p_2		√	√	√
p_3	√		√	√
p_4	√	√	√	√

Figure 3. A liveness issue for the alternative construction.

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 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 [19, 43].

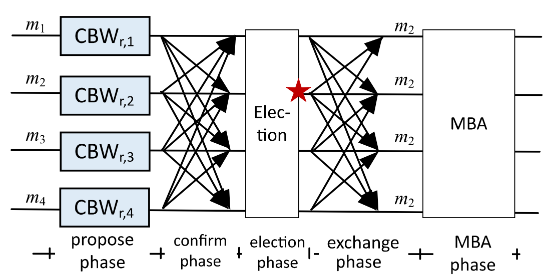


Figure 4. The SQ_0 protocol.

▷ *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.

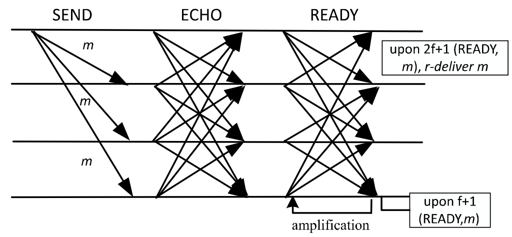
Definition 3.1 (CBW). Let Π be a protocol executed by replicas $p_1, \cdots, p_n$, where a sender p_s *cbw-broadcasts* a message $m \in \{0,1\}^*$ or $\perp$ to all replicas. Every correct replica p_i may *cbw-deliver* $m \in \{0,1\}^*$ or $\perp$ and *cbw-s-deliver* $v \in \{0,1\}^*$ or $\perp$. Π should 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.

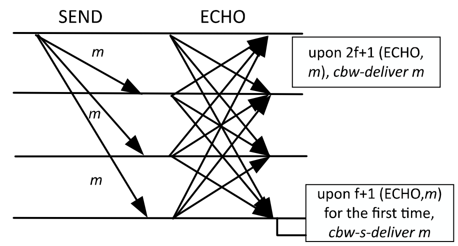
An IT-secure CBW protocol can be built as follows, as shown in Figure 5b. First, the sender p_s broadcasts a (Send, m) message. Second, upon receiving a (Send, m) message from p_s, a correct replica p_i broadcasts an (Echo, m) message. Upon receiving $2f+1$ (Echo, m) messages with the same m, p_i *cbw-delivers* m. Additionally, upon receiving $f+1$ (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 is a two-phase version of RBC yet additionally has 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.

▷ *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 CBW$_{r,i}$. Upon *cbw-delivering* some value m_j for instance 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, p_i also sends p_j a (Confirm) message. Meanwhile, p_i waits for $n-f$ (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 Election(r) function built from high-threshold common coins, where the value k_r is revealed after at least $f+1$ correct replicas query Election(r).



(a) Bracha's broadcast.



(b) Our CBW construction.

Figure 5. RBC vs. CBW.

After Election(r) outputs k_r, p_i broadcasts (Send, $r, i, CV_r[k]$). 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 $f+1$ (Send) messages with the same m and then *mba-proposes* m; 2) p_i receives $2f+1$ (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$_r$.

▷ *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 $n-f$ (confirm) messages and enter the election phase. 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 eventually *cbw-s-delivers* some value, i.e., condition 3) will be triggered and every correct replica *mba-proposes* some value.

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

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

– Initialize $d \leftarrow \emptyset$ at the beginning of the protocol; initialize the confirmed values 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 (Confirm, i) message to p_j.
– **(Commit)** Upon receiving $n - f$ (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 $\mathsf{Election(r)}$ function and obtain a random value k_r such that $1 \le k_r \le n$.
– **(Exchange)** Broadcast (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) $f + 1$ (Send, $r, *, m$) are received, then *mba-propose* m for instance MBA_r;
  2) $2f + 1$ (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 $m \notin d$, then *a-deliver* m, set d as $d \cup m$, and clear parameter CV_r.
(Event 2) If MBA_r outputs $\bot$, then *a-deliver* $\bot$ and clear parameter CV_r.

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

waits until $n - f$ replicas have sent a (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 (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 (Send, $r, *, m_{k_r}$) messages, no replica can collect more than $2f + 1$ (Send, $r, -, \bot$) 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-$\bot$ 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 low-threshold common coins instead of the high-threshold common coins as that in SQ_0. 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, allows 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, 24]: 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 4.1 (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* $(\vec{m}, \vec{cv})$ and $\vec{cv}[k] = m \neq \perp$ for slot k. For any correct replica p_j:
  (1) if p_j *pcbw-delivers* $(\vec{m'}, \vec{cv'})$ and $\vec{cv'}[k] \neq \perp$, then $\vec{cv'}[k] = m$;
  (2) if p_j *pcbw-delivers* $(\vec{m'}, \vec{cv'})$ and $\vec{m'}[k] \neq \perp$, 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**: Let p_i be the first replica that *pcbw-delivers* and p_i *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 either never *pcbw-deliver* or *pcbw-delivers* $(\vec{m'}, \vec{cv'})$ such that $\vec{cv'}[k] = m_k$.
- **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 as a black-box, 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.

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 replicas 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 Election(r) function.

Exchange phase and MBA phase. After 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 $2f+1$ $(\textsc{Send})$ messages 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 m, p_i *a-delivers* m.

▷ **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:

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 $r + 1$.

Election, Exchange, and MBA phases

– Wait until p_i *pcbw-delivers* $(\vec{m}, \vec{cv})$ in PCBW_r.
– **(Election)** Query the Election(r) function and obtain a random value k_r such that $1 \le k_r \le n$.
– **(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 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* $\perp$.

Figure 7. The SQ protocol for epoch r at replica p_i. The Election() function is built from low-threshold common coins.

- 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 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 Election(r) function, a key set I exists. We consider the first correct replica p_i that queries 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. According to the validity property of MBA, MBA_r outputs m_{k_r}. Therefore, SQ achieves $O(1)$ time.

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 show the pseudocode of PCBW_r in Figure 8. As mentioned in Sec. 3.2, our PCBW protocol involves only one $(\textsc{Propose})$ message and two procedures: an update procedure and a controlling procedure. Multiple PCBW instances can be started in parallel. The information exchanged in the $(\textsc{Propose})$ 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.

Initialization. Each replica p_i initializes parameters EV and CV at the beginning of the protocol. Here, the values stored in EV are also called *echo values* and the values stored in CV are also called *confirmed values*. Moreover, for each instance PCBW_r, p_i initializes two parameters to keep track of the

Initialization:

- Initialize echo values $EV \leftarrow \perp$ and confirmed values $CV \leftarrow \perp$ at the beginning.
- Initialize the following parameters for each PCBW$_r$: $\vec{m}_r, \vec{cv}_r \leftarrow [\perp]^n$, $s\text{-}output_r \leftarrow [\perp]^n$, let the *pcbw-delivered* values be $output_r$=$(\vec{m}_r, \vec{cv}_r)$, let the *pcbw-s-delivered* values be $s\text{-}output_r$.

---

let $witness(\textsc{echo}(r, j, v))$ = the number of different replicas from which $\textsc{echo}(r, j, v)$ was received.
let $witness(\textsc{confirm}(r, j, v))$ = the number of different replicas from which $\textsc{confirm}(r, j, v)$ was received.

- **(Broadcast) Upon** *pcbw-broadcast(m_i)* in PCBW$_r$: Broadcast $(\textsc{Propose}, r, i, m_i, EV, CV)$ and clear parameters EV and CV.

Update procedure for PCBW$_r$

- **Upon** receiving $(\textsc{Propose}, r, j, m_j, EV_j, CV_j)$ from p_j:
  - **if** $s\text{-}output_r[j] = \perp$ **then** update $EV \leftarrow EV \cup \textsc{echo}(r, j, m_j)$, $s\text{-}output_r[j] \leftarrow m_j$, *pcbw-s-deliver* $s\text{-}output_r$.
  - **for** $\textsc{echo}(r', k, v) \in EV_j$ **then**
    - **if** $witness(\textsc{echo}(r', k, v)) \geq f + 1$ and $s\text{-}output_{r'}[k] = \perp$ **then** $s\text{-}output_{r'}[k] \leftarrow v$, *pcbw-s-deliver* $s\text{-}output_{r'}$.
    - **if** $witness(\textsc{echo}(r', k, v)) \geq 2f + 1$ **then** update $CV \leftarrow CV \cup \textsc{confirm}(r', k, v)$, $\vec{cv}_{r'}[k] \leftarrow v$.
  - **for** $\textsc{confirm}(r', k, v) \in CV_j$ **then**
    - **if** $witness(\textsc{confirm}(r', k, v)) \geq 2f + 1$ **then** $\vec{m}_{r'}[k] \leftarrow v$.

Controlling procedure for PCBW$_r$

- **if** there exists a set S of replicas s.t. $|S| \geq 2f + 1$ and for every $p_k \in S$, $\vec{m}_r[k] \neq \perp$ **then** the controlling procedure returns 1 and *pcbw-deliver* $output_r$ in PCBW$_r$.

Figure 8. The PCBW$_r$ protocol at replica p_i. PCBW events are highlighted in blue.

outputs: $output_r$ and $s\text{-}output_r$. The parameter $output_r$ contains two vectors $\vec{m}_r$ and $\vec{cv}_r$ and $s\text{-}output_r$ is a vector. These output parameters will be cleared when PCBW$_r$ terminates.

Broadcast phase. In PCBW$_r$, each replica p_i *pcbw-broadcasts* m_i by broadcasting a $(\textsc{Propose}, r, i, m_i, EV, CV)$ message to all replicas. After sending the $(\textsc{Propose})$ message, EV and CV are cleared.

The update procedure. Upon receiving $(\textsc{Propose}, r, j, m_j, EV_j, CV_j)$ message from p_j, p_i starts the update procedure. First, p_i stores m_j as an *echo value* in EV in the form of $\textsc{echo}(r, j, m_j)$. Additionally, p_i also sets $s\text{-}output_r[j]$ as m_j and *pcbw-s-delivers* $s\text{-}output_r$. For any $\textsc{echo}(r', k, v)$ contained in EV_j, p_i checks whether it has received $\textsc{echo}(r', k, v)$ from $f+1$ replicas and $s\text{-}output_{r'}[k] = \perp$. If so, p_i sets $s\text{-}output_{r'}[k]$ as v and *pcbw-s-delivers* $s\text{-}output_{r'}$. In addition, if p_i has received $\textsc{echo}(r', k, v)$ from $2f + 1$ replicas, the value v is *confirmed*. Then, p_i stores v in CV in the form of $\textsc{confirm}(r', k, v)$ and sets $\vec{cv}_{r'}[k]$ as v. For any $\textsc{confirm}(r', k, v)$ contained in CV_j, p_i checks whether it has received $\textsc{confirm}(r', k, v)$ from $2f+1$ replicas. If so, v is a *committed value* and p_i sets $\vec{m}_{r'}[k]$ as v.

The controlling procedure. If PCBW$_r$ has not terminated yet, every time replica p_i modifies the parameter $\vec{m}_r$ for PCBW$_r$ in the update procedure, p_i also checks whether the controlling procedure is satisfied. The controlling procedure returns true if there exists a set S of at least $2f + 1$ replicas such that for any $p_k \in S$, $\vec{m}_r[k]$ is a non-$\perp$ committed value. If so, p_i *pcbw-delivers* $(\vec{m}_r, \vec{cv}_r)$ in PCBW$_r$.

$\triangleright$ **Analysis.** We sketch why our PCBW construction in Figure 8 meets all the properties defined in Sec. 4.1. Consider the instance PCBW$_r$. If a correct replica p_i *pcbw-delivers* $(\vec{m}, \vec{cv})$, according to the update procedure, any non-$\perp$ value in $\vec{cv}$ has been *echoed* by $2f + 1$ replicas (i.e., they include the value in their EV) and any non-$\perp$ value in $\vec{m}$ has been *confirmed* by at least $2f + 1$ replicas (i.e., the $2f + 1$ replicas include the value in their CV). Suppose another correct replica *pcbw-delivers* $(\vec{m}', \vec{cv}')$. According to the quorum intersection rule, for any slot k, $\vec{m}'[k]$ and $\vec{m}[k]$ are either equal or one of them is $\perp$. The same applies to $\vec{cv}'$ and $\vec{cv}$. Therefore, the *consistency* property of PCBW holds.

We now discuss *Weak agreement II*. Consider p_i as the first replica that *pcbw-delivers* some $(\vec{m}, \vec{cv})$. We focus on $\vec{m}[k] = m_k$, where $m_k \neq \perp$. According to the protocol, m_k has been confirmed by $2f + 1$ replicas before p_i sets its $\vec{m}[k]$ as some non-$\perp$ value. Therefore, at least $f+1$ correct replicas (let the set of replicas be I) have previously confirmed m_k. Consider any $p_j \in I$. Replica p_j may never *pcbw-deliver*. But once p_j *pcbw-delivers* some values $(\vec{m}', \vec{cv}')$, p_j must have already confirmed m_k. Therefore, $\vec{cv}'[k] = m_k$ and *Weak agreement II* holds.

For *Weak agreement I*, p_i *pcbw-delivers* a non-$\perp$ $\vec{cv}[k]$ after $\vec{cv}[k]$ has been confirmed by at least $2f+1$ replicas. Let the set of replicas be S. According to the protocol, any correct replica

p_j *pcbw-s-delivers s-output*$_r$ such that *s-output*$_r[k] = m_k$ under two conditions: 1) p_j has received m_k directly from p_k; 2) p_j receives m_k from $f+1$ replicas as echoed values. As any correct replica in S will include $\vec{cv}[k]$ in EV, condition 2) will eventually be satisfied by p_j so *Weak agreement I* is satisfied.

We ignore the discussion on validity, integrity, and termination and show the proof in detail in Appendix A.1.

4.4 Discussion

We now discuss the communication complexity of SQ. In our PCBW construction, the (PROPOSE) message includes a proposed value (length L), EV (echo values), and CV (confirmed values). For EV (resp. CV), each $EV[k]$ (resp. $CV[k]$) contains a constant number of L-bit values, where $k \in [1, n]$. Our PCBW construction thus has $O(Ln^3)$ communication.

Assuming a Rabin dealer, the communication complexity for the election phase is $O(n \log n)$. In the exchange phase, each (SEND) message includes at most two L-bit values, so the communication complexity is $O(Ln^2)$. In the MBA phase, as the input to MBA is either a L-bit value or $\perp$, the MBA phase has $O(Ln^2)$ communication using the most efficient MBA instantiation known so far [47]. Therefore, SQ achieves $O(Ln^3)$ communication complexity.

The "broadcast-echo-confirm-commit" paradigm used in our PCBW construction is similar to the three-phase paradigm for reliable broadcast [13, 14]. Accordingly, the communication complexity of SQ can be further optimized using techniques often employed by asynchronous verifiable information dispersal [5, 18] and reliable broadcast [4, 22] protocols, e.g., erasure codes and online error correction.

SQ achieves $O(n^2)$ messages as only all-to-all communication is involved (and the MBA protocol we consider [47] also achieves $O(n^2)$ messages).

For ease of understanding, in the PCBW construction, every replica needs to include the L-bit value as echo values and the confirmed values. Several optimizations can be used to reduce the concrete cost of communication. For example, we do not need to include the L-bit value in CV as each replica eventually receives them from EV. We provide an optimized PCBW construction with implementation-level details in Appendix B.

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. SQ$_h$ achieves $O(Ln^2 + \kappa n^3)$ communication, where κ is the security parameter (the length of a hash digest). We present the pseudocode of the hash variant of PCBW in Figure 9, and we also provide an implementation-level PCBW construction in Appendix C. The main protocol remains the same as that in Figure 7.

Recall that SQ has $O(Ln^3)$ communication complexity, as every replica broadcasts its received values from all replicas in the (PROPOSE) message. Briefly speaking, we can replace the echoed values and confirmed values with the hashes of the values to optimize the communication. We also modify the update procedure accordingly.

Note that in the ABC protocol, each replica can still obtain the proposed values in the exchange phase. The workflow thus remains exactly the same as SQ. In Figure 9, we still use v to denote *s-output* and *output*$_r$ while $hash(v)$ is exchanged in the (PROPOSE) message. If some replica has not previously received v when it updates *s-output* and *output*$_r$, the replica can obtain v from the exchange phase.

As we only replace some values with their hashes, the correctness of SQ$_h$ follows from SQ and the collision resistance of the hash function. The message complexity of SQ$_h$ is also $O(n^2)$. For the communication complexity, the (PROPOSE) message now includes the proposed value (length L), EH, and CH. Each $EH[k]$ (or $CH[k]$) contains a constant number of hash values, where $k \in [1, n]$. Therefore, the communication complexity of PCBW is $O(Ln^2 + \kappa n^3)$. The communication of other phases remains the same as that in SQ, i.e., $O(Ln^2)$. Hence, SQ$_h$ achieves $O(Ln^2 + \kappa n^3)$ communication.

Batch processing optimization. SQ$_h$ can be further optimized via batch processing, so $O(n)$ proposed values are expected to be *a-delivered* in each epoch. In particular, when every replica sends a (PROPOSE) message in PCBW in epoch r, it includes the hash of its proposed value in epoch $r-1$. Every replica p_i accepts a (PROPOSE) message from p_j in epoch r only after the (PROPOSE) message from p_j for every epoch lower than r has been received. In addition, p_i also locally stores the last epoch r' in which the proposal of p_j has been delivered. The other workflow of the protocol remains the same. In this way, once MBA$_r$ outputs a value m where m was proposed by p_j, replicas deliver the values proposed by p_j between epoch $r'+1$ and r. Accordingly, $O(n)$ proposed values are expected to be delivered in each epoch. Also, thanks to the agreement property of MBA, SQ$_h$ can be implemented using limited memory, unlike unlimited memory required by previous works, e.g., DAG-Rider [34].

Note that replicas may also need to synchronize the proposals between epoch $r'+1$ and $r-1$. We omit the details in this work.

6 Implementation and Evaluation

We implement a prototype of SQ$_h$ with batch processing optimization in Golang, the communication-efficient variant of SQ. Our codebase involves around 7,000 LOC for the protocol and about 1,000 LOC for evaluation. In our implementation, we use gRPC as the communication library, HMAC to realize the authenticated channel, and SHA256 as the hash function. For leader election, we use threshold PRF instead, following the practice of previous works [27, 42, 54].

Initialization:

- Initialize echo hashes $EH \leftarrow \perp$ and confirmed hashes $CH \leftarrow \perp$ at the beginning.
- Initialize the following parameters for each PCBW$_r$: $\vec{m}_r, \vec{cv}_r \leftarrow [\perp]^n$, *s-output*$_r \leftarrow [\perp]^n$, let the *pcbw-delivered* values be *output*$_r$=$(\vec{m}_r, \vec{cv}_r)$, let the *pcbw-s-delivered* values be *s-output*$_r$.

---

let $witness(\textsc{echo}(r, j, hash(v)))$ = the number of different replicas from which $\textsc{echo}(r, j, hash(v))$ was received.
let $witness(\textsc{confirm}(r, j, hash(v)))$ = the number of different replicas from which $\textsc{confirm}(r, j, hash(v))$ was received.

- **(Broadcast) Upon** *pcbw-broadcast*(m_i) in PCBW$_r$: Broadcast ($\textsc{Propose}, r, i, m_i, EH, CH$) and clear parameters EH and CH.

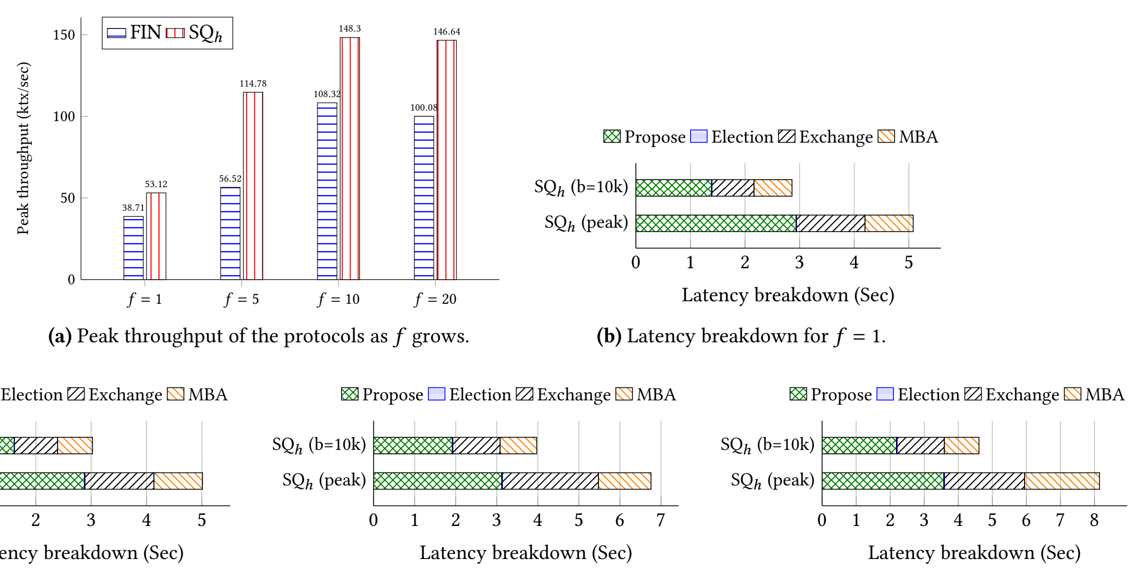
Update procedure for PCBW$_r$

- **Upon** receiving ($\textsc{Propose}, r, j, m_j, EH_j, CH_j$) from p_j:
  if *s-output*$_r[j] = \perp$ **then** update $EH \leftarrow EH \cup \textsc{echo}(r, j, hash(m_j))$, *s-output*$_r[j] \leftarrow m_j$, *pcbw-s-deliver s-output*$_r$.
  for $\textsc{echo}(r', k, hash(v)) \in EH_j$ **then**
  if $witness(\textsc{echo}(r', k, hash(v))) \geq f+1$ and *s-output*$_{r'}[k] = \perp$ **then** set *s-output*$_{r'}[k]$ as v, *pcbw-s-deliver s-output*$_{r'}$.
  if $witness(\textsc{echo}(r', k, hash(v))) \geq 2f+1$ **then** update $CH \leftarrow CH \cup \textsc{confirm}(r', k, hash(v))$, $\vec{cv}_{r'}[k] \leftarrow v$.
  for $\textsc{confirm}(r', k, hash(v)) \in CH_j$ **then**
  if $witness(\textsc{confirm}(r', k, hash(v))) \geq 2f+1$ **then** $\vec{m}_{r'}[k] \leftarrow v$.

Controlling procedure for PCBW$_r$

- **if** there exists a set S of replicas s.t. $|S| \geq 2f+1$ and for every $p_k \in S$, $\vec{m}_r[k] \neq \perp$ **then** the controlling procedure returns 1 and *pcbw-deliver output*$_r$ in PCBW$_r$.

Figure 9. The hash variant of PCBW$_r$ protocol at replica p_i. PCBW events are highlighted in blue. The changes on top of Figure 8 are highlighted in gray.



(a) Peak throughput of the protocols as f grows. (b) Latency breakdown for $f = 1$.

(c) Latency breakdown for $f = 5$. (d) Latency breakdown for $f = 10$. (e) Latency breakdown for $f = 20$.

Figure 10. Peak throughput of SQ$_h$ vs. FIN [27] and latency breakdown of SQ$_h$.

We evaluate the performance of our protocols on Amazon EC2 using up to 61 virtual machines (VMs). We compare the performance of SQ_h with FIN [27], the state-of-the-art asynchronous protocol. We use *m5.xlarge* instances, and each instance has four vCPUs and 16GB of memory. We distribute the replicas evenly in different regions: us-west-2 (Oregon, US), us-east-2 (Ohio, US), ap-southeast-1 (Singapore), and eu-west-1 (Ireland). We evaluate the performance using different network sizes and batch sizes b. We use f to denote the network size where $n = 3f + 1$ replicas are launched. We focus on the peak throughput of different f.

Peak throughput. We show the peak throughput of SQ_h and FIN for $f = 1$ to $f = 20$ in Figure 10a. The peak throughput of SQ_h is 36.90%-103.07% higher than that of FIN. This is expected, and the performance gain of SQ_h is due to three facts: 1) SQ_h can *a-deliver* $O(n)$ proposals in each epoch, same as that in FIN; 2) SQ_h and FIN achieve the same communication complexity; 3) SQ_h has quadratic messages while FIN achieves $O(n^3)$ messages. Note that as mentioned earlier in the paper, we can execute sequential FIN (i.e., ACS) instances to obtain an ABC protocol. SQ_h is an ABC protocol and cannot be directly used as an ACS (i.e., FIN). The performance gain of SQ_h is thus based on the fact that FIN is used as an ABC protocol.

Latency breakdown. We report latency breakdown of SQ_h in Figure 10b-10e. We report the results for the batch size when SQ_h reaches its peak throughput and for $b = 10,000$. In all experiments we launched, the propose phase is the bottleneck of the system. Indeed, the PCBW instance requires each proposal to include several values ($O(n)$ hashes) in addition to the batch of transactions. We believe our protocols can be further optimized to reduce the overhead of the PCBW construction, and we consider the optimization to be future work.

7 Additional Related Works

Signature-free consensus. This and prior works [8, 10, 19, 20, 41, 44–47] 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 [49], where a trusted dealer distributes coins to replicas. The common coin object can also be realized in various other ways, such as threshold PRF [6, 17], threshold signatures [7, 17, 51], randomness beacons [25, 33], dedicated common coin protocols [11, 30], and ones based on trusted execution environments (TEEs).

Recently, some signature-free MVBA protocols have been proposed [1, 21, 23, 27], but they all have $O(n^3)$ message complexity and the ABC protocols relying on them would at least have $O(n^3)$ messages. For example, the FIN protocol [27] is a signature-free ACS protocol that can be directly used as an ABC protocol. In contrast, our protocols are not ACS, and we do not know how to transform our protocols to ACS without increasing the time and message complexity. Additionally, the underlying techniques are fundamentally different. FIN reduces ACS to parallel RBC and MVBA while we reduce ABC to MBA.

MBA. MBA was first introduced in the synchronous assumption [29, 35, 36, 48, 53], where there exists a known upper bound for message transmission and processing. In MBA, every replica holds an (supposedly the same) input and replicas agree on some value or $\perp$ (denoting correct replicas do not hold the same input). In the asynchronous assumption, Mostéfaoui and Raynal (MR) presented the first signature-free MBA with $O(n^2)$ messages and $O(1)$ time [47].

8 Conclusion

We present SQ, the first information-theoretic and signature-free asynchronous Byzantine atomic broadcast protocol with optimal $O(n^2)$ messages and $O(1)$ time. We also show SQ_h, a hash variant of SQ that achieves the same complexities but additionally assumes hashes. We show that the performance of SQ_h is comparable with the state-of-the-art asynchronous protocol FIN.

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] Alhaddad, N., Das, S., Duan, S., Ren, L., Varia, M., Xiang, Z., Zhang, H.: Brief announcement: Asynchronous verifiable information dispersal with near-optimal communication. In: PODC. pp. 418–420 (2022)

[6] 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)

[7] Bacho, R., Loss, J.: On the adaptive security of the threshold bls signature scheme. In: CCS. pp. 193–207 (2022)

[8] Ben-Or, M., Canetti, R., Goldreich, O.: Asynchronous secure computation. STOC (1993)

[9] Ben-Or, M., El-Yaniv, R.: Resilient-optimal interactive consistency in constant time. Distrib. Comput. (2003)

[10] Ben-Or, M., Kelmer, B., Rabin, T.: Asynchronous secure computations with optimal resilience. In: PODC. pp. 183–192. ACM (1994)

[11] 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)

[12] Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: PKC (2003)

[13] Bracha, G.: An asynchronous [(n-1)/3]-resilient consensus protocol. In: PODC. pp. 154–162. ACM (1984)

[14] Bracha, G.: Asynchronous Byzantine agreement protocols. Information and Computation **75**(2), 130–143 (1987)
[15] Cachin, C., Guerraoui, R., Rodrigues, L.: Introduction to reliable and secure distributed programming. Springer Science & Business Media (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] Cachin, C., Tessaro, S.: Asynchronous verifiable information dispersal. In: SRDS. pp. 191–201. IEEE (2005)
[19] 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)
[20] 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)
[21] Das, S., Xiang, Z., Kokoris-Kogias, L., Ren, L.: Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. Cryptology ePrint Archive (2022)
[22] Das, S., Xiang, Z., Ren, L.: Asynchronous data dissemination and its applications. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 2705–2721 (2021)
[23] 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)
[24] Dolev, D.: The byzantine generals strike again. Journal of Algorithms **3**(1), 14–30 (1982)
[25] Drand: Drand - a distributed randomness beacon daemon. https://github.com/drand/drand (accessed Oct 2023)
[26] Duan, S., Reiter, M.K., Zhang, H.: BEAT: Asynchronous bft made practical. In: CCS. pp. 2028–2041. ACM (2018)
[27] Duan, S., Wang, X., Zhang, H.: FIN: Practical signaturbe-free asynchronous common subset in constant time. In: CCS (2023)
[28] 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)
[29] Fitzi, M., Hirt, M.: Optimally efficient multi-valued byzantine agreement. In: Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing. pp. 163–168 (2006)
[30] 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)
[31] Guo, B., Lu, Y., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Speeding Dumbo: Pushing asynchronous BFT closer to practice. In: NDSS (2022)
[32] Guo, B., Lu, Z., Tang, Q., Xu, J., Zhang, Z.: Dumbo: Faster asynchronous bft protocols. In: CCS (2020)
[33] I. T. L. Computer Security Division: Interoperable randomness beacons: Csrc. https://csrc.nist.gov/projects/interoperable-randomness-beacons (accessed Oct 2023)
[34] Keidar, I., Kokoris-Kogias, E., Naor, O., Spiegelman, A.: All you need is dag. PODC (2021)
[35] King, V., Saia, J.: Breaking the $o(n^2)$ bit barrier: scalable byzantine agreement with an adaptive adversary. Journal of the ACM (JACM) **58**(4), 18 (2011)
[36] Liang, G., Vaidya, N.: Error-free multi-valued consensus with byzantine failures. In: PODC. pp. 11–20 (2011)
[37] 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)
[38] Liu, C., Duan, S., Zhang, H.: EPIC: Efficient asynchronous bft with adaptive security. In: DSN (2020)
[39] Loss, J., Moran, T.: Combining asynchronous and synchronous Byzantine agreement: The best of both worlds. IACR Cryptology ePrint Archive **2018**, 235 (2018)
[40] 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)
[41] MacBrough, E.: Cobalt: BFT governance in open networks. arXiv preprint arXiv:1802.07240 (2018)
[42] Miller, A., Xia, Y., Croman, K., Shi, E., Song, D.: The honey badger of bft protocols. In: CCS. pp. 31–42. ACM (2016)
[43] Moniz, H., Neves, N.F., Correia, M., Verissimo, P.: Ritas: Services for randomized intrusion tolerance. TDSC **8**(1), 122–136 (2008)
[44] Mostefaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous Byzantine consensus with $t \le n/3$ and $o(n^2)$ messages. In: PODC. pp. 2–9. ACM (2014)
[45] Mostéfaoui, A., Moumen, H., Raynal, M.: Signature-free asynchronous binary byzantine consensus with t < n/3, o(n^2) messages, and O(1) expected time. J. ACM **62**(4), 31:1–31:21 (2015)
[46] Mostéfaoui, A., Raynal, M.: Signature-free broadcast-based intrusion tolerance: never decide a byzantine value. In: OPODIS. pp. 143–158. Springer (2010)
[47] 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)
[48] Patra, A.: Error-free multi-valued broadcast and byzantine agreement with optimal communication complexity. In: OPODIS. pp. 34–49. Springer (2011)
[49] Rabin, M.O.: Randomized byzantine generals. In: SFCS. pp. 403–409. IEEE (1983)
[50] Reiter, M.K.: Secure agreement protocols: Reliable and atomic group multicast in rampart. In: CCS. pp. 68–80 (1994)
[51] Shoup, V.: Practical threshold signatures. In: EUROCRYPT (2000)
[52] Toueg, S.: Randomized byzantine agreements. In: Proceedings of the third annual ACM symposium on Principles of distributed computing. pp. 163–178 (1984)
[53] Turpin, R., Coan, B.A.: Extending binary byzantine agreement to multivalued byzantine agreement. Information Processing Letters **18**(2), 73–76 (1984)
[54] Zhang, H., Duan, S.: PACE: Fully parallelizable BFT from reproposable byzantine agreement. ACM CCS (2022)
[55] Zhang, H., Duan, S., Zhao, B., Zhu, L.: Waterbear: Practical asynchronous bft matching security guarantees of partially synchronous bft. In: Usenix Security (2023)

A Proofs

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

A.1 Proof of the PCBW Construction

Lemma A.1. *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. □

Lemma A.2. *In* PCBW_r*, if a correct replica* p_j *pcbw-delivers* $(\vec{m}, \vec{cv})$ *where* $\vec{cv}[i] = m_i \neq \perp$*, then at least* $f + 1$ *correct replicas have received* m_i *from replica* p_i *and changed their* s-output$_r[i]$ *parameters from* $\perp$ *to* m_i*.*

Proof. If p_j *pcbw-delivers* $(\vec{m}, \vec{cv})$ such that $\vec{cv}[i] = m_i \neq \perp$ for epoch r, from the controlling procedure, we know that p_j must have confirmed $\vec{cv}[i]$ and set $\vec{cv}[i]$ as m_i before it *pcbw-delivers.* According to the update procedure, p_j has received $\textsc{echo}(r, i, m_i)$ from $2f + 1$ replicas. Therefore, at least $f + 1$ correct replicas have included $\textsc{echo}(r, i, m_i)$ in their EV and broadcast their EV in Propose messages—indicating that they have received m_i from p_i for epoch r. Therefore, by the update procedure, at least $f+1$ correct replicas have received m_i from p_i and changed their *s-output*$_r[i]$ parameters from $\perp$ to m_i. □

Lemma A.3. *In* PCBW_r*, if a correct replica* p_i *pcbw-delivers* $(\vec{m}_i, \vec{cv}_i)$*, another correct replica* p_j *pcbw-delivers* $(\vec{m}_j, \vec{cv}_j)$*, and for a 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 changed their *s-output*$_r[k]$ parameters from $\perp$ to $m_{i,k}$ by Lemma A.2. Similarly, at least $f+1$ correct replicas have received $m_{j,k}$ from p_k and changed their *s-output*$_r[k]$ parameters from $\perp$ to $m_{j,k}$. As there are $2f + 1$ correct replicas, at least one correct replica has stored both $m_{i,k}$ and $m_{j,k}$ in *s-output*$_r[i]$, a contradiction. □

Lemma A.4. *In* PCBW_r*, if a correct replica* p_j *pcbw-delivers* $(\vec{m}, \vec{cv})$ *where* $\vec{m}[i] \neq \perp$ *(resp.* $\vec{cv}[i] \neq \perp$*), then* $\vec{m}[i]$ *(resp.* $\vec{cv}[i]$*) was previously pcbw-broadcast by replica* p_i*.*

Proof. Suppose p_j *pcbw-delivers* $(\vec{m}, \vec{cv})$ such that $\vec{cv}[i] \neq \perp$ for epoch r. Note p_j has confirmed $\vec{v}[i]$. By Lemma A.2, $\vec{cv}[i]$ was *pcbw-broadcast* by p_i.

Suppose p_j *pcbw-delivers* $(\vec{m}, \vec{cv})$ such that $\vec{m}[i] \neq \perp$. According to the update procedure, p_j has received $\textsc{confirm}(r, i, \vec{m}[i])$ from at least $2f + 1$ replicas. Then at least $f + 1$ correct replicas have confirmed $\vec{m}[i]$ and included $\textsc{confirm}(r, i, \vec{m}[i])$ in their CV parameters. Similar to the discussion above, $\vec{m}[i]$ was *pcbw-broadcast* by p_i. This completes the proof. □

Theorem A.5. **(*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 m_i in a Propose 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 update procedure using pm_i as input and *pcbw-s-delivers* $\vec{v}$ such that $\vec{v}[i] = m_i$.

As p_i is a correct replica which *pcbw-broadcasts* only one message in epoch r, the second part of the lemma follows from Lemma A.4.

□

Theorem A.6. **(*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 A.3. For (2), note that when p_j *pcbw-delivers* $(\vec{m'}, \vec{cv'})$ where $\vec{m'}[k] \neq \perp$, p_j has received $\textsc{confirm}(r, k, \vec{m'}[k])$ from $2f+1$ replicas. Then at least $f+1$ correct replicas have confirmed $\vec{m'}[k]$. As $m \neq \perp$, $\vec{m'}[k] = m$ by Lemma A.3. This completes the proof of the lemma. □

Theorem A.7. **(*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 A.2, 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 $\mathrm{PCBW}_{r'}$ 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 *s-output*$_r[k]$ as a non-$\perp$ value before receiving these messages, p_j will update its *s-output*$_r[k]$ to m_k and *pcbw-s-delivers* *s-output*$_r$ according to our protocol.

Otherwise, if p_j sets $s\text{-}output_r[j][k]$ 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 $s\text{-}output_r$. In both cases, the lemma holds. □

Theorem A.8. (***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 *either never pcbw-deliver or 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 has received $\textsc{confirm}(r, k, m_k)$ from p_j before p_i *pcbw-delivers* $(\vec{m}, \vec{cv})$. 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 $p_j \in I$, if p_j *pcbw-delivers* a $(\vec{m'}, \vec{cv'})$, then $\vec{cv'}[k] = m_k$. Note p_j is correct. Before p_j sends $\textsc{confirm}(r, k, m_k)$ to p_i, according to the update procedure, p_j must have confirmed m_k as the *pcbw-broadcast* value from p_k in PCBW_r. As p_j sent $\textsc{confirm}(r, k, m_k)$ 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 $\vec{cv}_r[k]$ as m_k before p_j *pcbw-delivers*. Therefore, when p_j *pcbw-delivers* a $(\vec{m'}, \vec{cv'})$, then $\vec{cv'}[k] = m_k$. The lemma thus holds. □

Theorem A.9. (***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 $s\text{-}output_r[k]$ as a non-$\perp$ value for some $k \in [1, n]$. Note that once p_i sets its $s\text{-}output_r[k]$ to a non-$\perp$ value, p_i does not change $s\text{-}output_r[k]$ anymore. As $|s\text{-}output_r| = n$, p_i *pcbw-s-delivers* $\vec{v}$ at most $O(n)$ times.

The correctness of property (1) follows from Lemma A.4 and the correctness of property (2) follows from Lemma A.1. This completes the proof. □

Theorem A.10. (***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 11, p_i *pcbw-delivers* some values if there exists a set S consisting of at least $2f + 1$ replicas such that for any $p_k \in S$, $\vec{m}_r[k] \neq \perp$. In the following, we prove that for each correct replica p_i, eventually $\vec{m}_r[k] \neq \perp$ for any $p_k \in C$. As $|C| \geq 2f + 1$, p_i eventually *pcbw-delivers* some values.

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 $s\text{-}output_r[k]$ as m_k and includes $\textsc{echo}(r, k, m_k)$ in its EV. 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 parameters, and then broadcasts EV to all replicas. Eventually, for any $p_k \in C$, p_i sets $\vec{cv}_r[k]$ as m_k, where m_k is proposed by p_k in epoch r. Then p_i includes $\textsc{confirm}(r, k, m_k)$ in its CV and broadcasts CV in a (Propose) message in some epoch $r^* > r$. Similarly, every correct replica in C eventually receives the proposed messages for epoch r^* from every other replica in C, then p_i eventually sets its $\vec{m}_r[k]$ as a non-$\perp$ value for any $p_k \in C$. Therefore, the controlling procedure returns 1 at any correct replica p_i and p_i eventually *pcbw-delivers* some values. □

A.2 Proof of SQ

Lemma A.11. *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 $f + 1$ (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 $f + 1$ (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 A.12. *In epoch* r*, if* $\mathsf{Election(r)}$ *returns* k *and a correct replica* p_i *broadcasts a (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 (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 A.13. *In epoch r, assuming that the* Election(r) *function is queried by at least one correct replica and* p_i *is the first correct replica that queries* Election(r)*. If* 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 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 (Send, r, i, m). By Lemma A.12, 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 Election(r), p_i is also the first replica that *pcbw-delivers* a pair of output $(\vec{m}, \vec{cv})$ in PCBW_r where $\vec{m}_r[k] = m \neq \perp$. Due to the weak agreement II and the termination properties 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 (Send, $r, *, m$) in the exchange phase, and condition 2) for MBA_r will never be satisfied. Thus, no correct replica *mba-proposes* $\perp$ for MBA_r.

Last, from Lemma A.11, 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 A.14. *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 (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 Election(r) function. After k is returned by Election(r), every correct replica broadcasts $CV_r[k]$ in its (Send) messages. We now show that every correct replica *mba-proposes* some value.

After obtaining an output for Election(r), we distinguish two cases: 1) at least one correct replica p_i broadcasts (Send, r, i, m); 2) every correct replica broadcasts (Send, $r, *, \perp$) 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 A.12, any correct replica eventually *mba-proposes* a value (or $\perp$) for MBA_r.
- *Case 2:* In this case, after receiving all the (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 A.15. *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 A.16 (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 A.14, 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 A.17 (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 A.18 (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 A.11, m was previously *a-broadcast* by some replica. □

Lemma A.19. *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 A.10, 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 A.13, 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 *suc* be the event that correct replicas *a-deliver* a value *a-broadcast* by a correct replica in epoch r. We have the following:

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

□

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

Lemma A.20 (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 A.19, 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 A.21 (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 A.20, 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 A.20, p_i will eventually *a-deliver* m in some epoch.

□

Theorem A.22 (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 [47], 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), EV (echo values), and CV (confirmed values). For EV, each $EV[k]$ 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 A.19.

□

B Implementation-Level PCBW Construction

We show the pseudocode of PCBW_r with implementatio-level details in Figure 11.

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

Definition B.2 (Col_Comp function)**.** For any matrix $M^{m\times n}$, the function $\text{Col\_Comp}(M, k, v)$ 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 (Propose, r, i, m_i, E, EV, LE) message to all replicas. Upon receiving (Propose, $r, j, m_j, E^j, EV^j, LE^j$) message 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.

  ▷ *Example (Figure 12a).* 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 $\text{CBW}_{e,k}$.

  ▷ *Example (Figure 12b).* 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., p_1 and p_j) such that for any $p_{j'} \in S$, $V_r[j'][k] = m_2$. As $V_r[i][k] = \perp$, 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 $\text{PCBW}_{r''}$ 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 $\text{CBW}_{r'',k}$. We further update the state matrix M and use M to count the number of confirmed values for each (r'', k) pair.

  ▷ *Example (Figure 12c).* 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 $\text{PCBW}_{r''-1}$ 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 $\text{PCBW}_{r''}$ (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 12d).* 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-$\perp$ 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 $\perp$.

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

- **(Broadcast)** Upon *pcbw-broadcast*(m_i) in PCBW$_r$:
  Broadcast (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 (Propose, $r, j, m_j, E^j, EV^j, LE^j$) from p_j:
  Let PCBW$_{r'}$ be the instance s.t. every 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 (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 $confirm(r'', k, V_{r''}[i][k])$= 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 $\perp$.

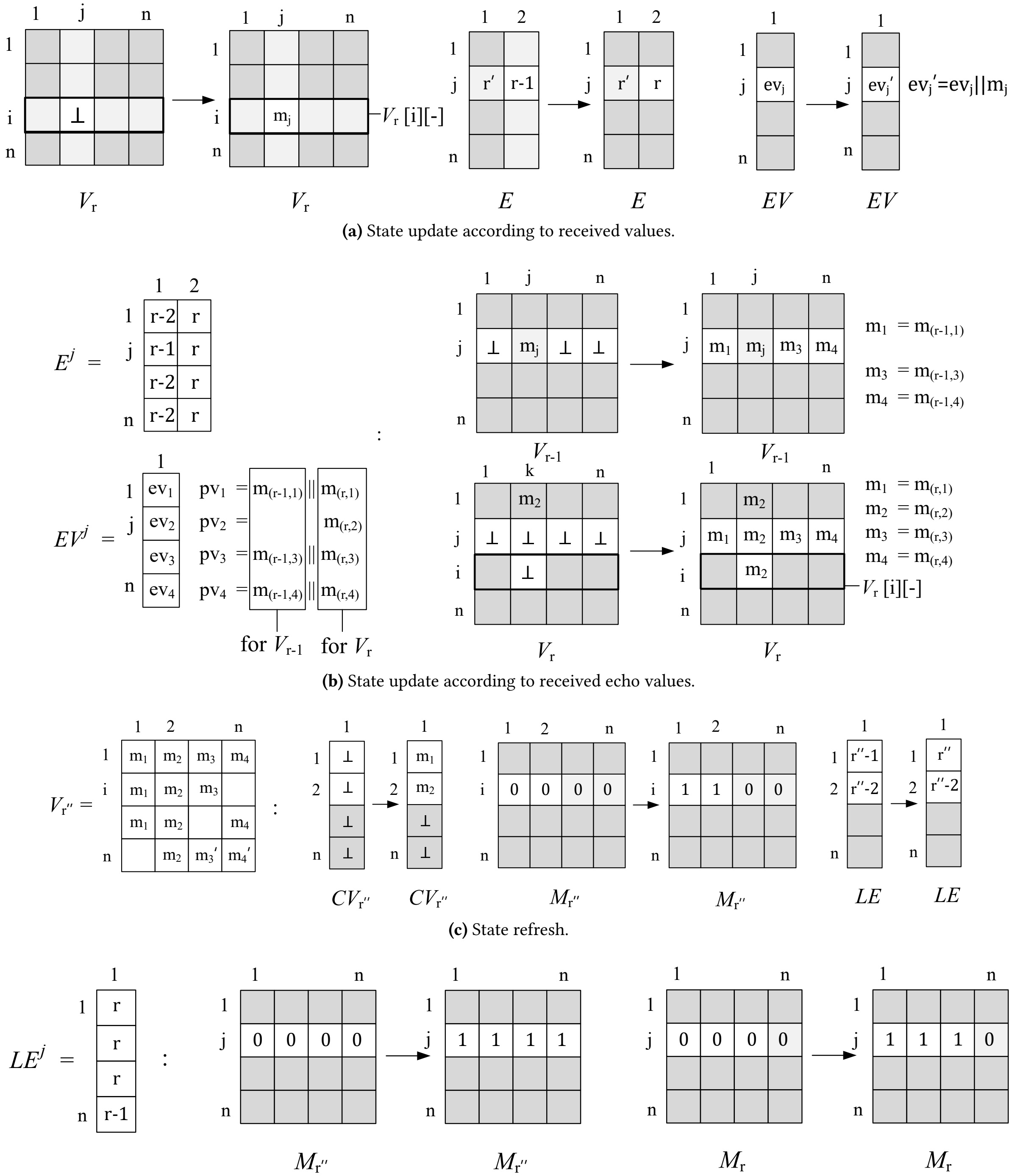
Figure 11. The PCBW$_r$ protocol at replica p_i. PCBW events are highlighted in blue.

C Implementation-Level Hash-based PCBW Construction

We present the pseudocode of the hash variant of PCBW in Figure 13. Here we highlight the changes from Figure 11 to Figure 13.

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,



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

Figure 12. The update procedure at replica p_i upon receiving a (PROPOSE, r, j, m_j, E^j, EV^j, LE^j) message from p_j. In this example, $j = 2$.

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

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*$(r, m_k, k) \equiv (V_r[k] = m_k \wedge m_k \neq \perp \wedge H_r[i][k] = h_r \wedge \text{Col\_Comp}(H_r, k, h_r) \geq 2f + 1)$

- **(Broadcast)** Upon *pcbw-broadcast*(m_i) in PCBW$_r$: Broadcast (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 (Propose, $r, j, m_j, E^j, EH^j, LE^j$) from p_j:
  Let PCBW$_{r'}$ be the instance such that every 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 (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 $p_k \in S$, $\text{Col\_Sum}(M_r, k) \geq 2f + 1$ 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 $\perp$.

Figure 13. Implementation of hash variant of PCBW$_r$ protocol at replica p_i. PCBW events are highlighted in blue.

speaking, echo hashes are hashes of the echo values EV used in SQ.

- 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, $H_r[j][k_r] = hash(m_k)$.