WaterBear: Practical Asynchronous BFT Matching Security Guarantees of Partially Synchronous BFT

Haibin Zhang  
bchainzhang@aliyun.com  
Beijing Institute of Technology

Sisi Duan*  
duansisi@tsinghua.edu.cn  
Tsinghua University, Zhongguancun Laboratory

Boxin Zhao*  
zhaobx@zgclab.edu.cn  
Zhongguancun Laboratory

Liehuang Zhu*  
liehuangz@bit.edu.cn  
Beijing Institute of Technology

Abstract

Asynchronous Byzantine fault-tolerant (BFT) protocols assuming no timing assumptions are inherently more robust than their partially synchronous counterparts, but typically have much weaker security guarantees.

We design and implement WaterBear, a family of new and efficient asynchronous BFT protocols matching all security guarantees of partially synchronous protocols. To achieve the goal, we have developed the local coin (flipping a coin locally and independently at each replica) based BFT approach—one long deemed as being inefficient—and designed more efficient asynchronous binary agreement (ABA) protocols and their reproposable ABA (RABA) versions from local coins. Our techniques on ABA and RABA are of independent interests and also allow us to build more efficient ABA protocols from common coins (distributively generating the same random coins for all replicas), helping improve various other protocols such as distributed key generation and BFT assuming trusted setup.

We implemented in total five BFT protocols in a new golang library, including four WaterBear protocols and BEAT. Via extensive evaluation, we show that our protocols are efficient under both failure-free and failure scenarios, achieving at least comparable or superior performance to BEAT with much weaker security guarantees. Specifically, the most efficient WaterBear protocol consistently outperforms BEAT in terms of all metrics. For instance, when the number of replicas is 16, the latency of our protocol is about 1/8 of that of BEAT and the throughput of our protocol is 1.23x that of BEAT.

Our work pushes the boundaries of asynchronous BFT, showing the strongest security levels that we know of and high performance can co-exist.

1 Introduction

Byzantine fault-tolerant state machine replication (BFT), a technique traditionally used to build mission-critical systems, has nowadays been the standard model for permissioned blockchains [9, 18, 35, 36, 60, 63, 64] and is used in various ways in hybrid blockchains. Due to their inherent robustness against performance and DoS attacks, asynchronous BFT protocols—relying on no timing assumptions—have been receiving significant attention [11]. While one line of works focuses on performance [28, 44, 45, 53], some other works aim at improving their "security." For instance, BEAT [33], PACE [66], and FIN [34] eliminated the less-established pairing assumption in these protocols; EPIC and HALE aimed at providing adaptive security [49, 69]; DAG-Rider strived to achieve quantum safety (though not quantum liveness) [47]; recent works studied how to avoid trusted setup [4, 30, 48, 67].

Table 1 summarizes the security levels that can be achieved for asynchronous BFT protocols implemented. The situation is in sharp contrast to their partially synchronous BFT counterparts (relying on timing assumptions): for example, the classic PBFT protocol [22]—based on authenticated channels only—easily achieves all the properties listed in the table. It is thus our goal to design and implement practical asynchronous BFT protocols achieving all these properties in Table 1—the same security guarantees as in partially synchronous BFT.

1.1 Background on Security Guarantees

Authenticated channels only vs. no PKC vs. quantum security. When designing practical fault-tolerant and cryptographic protocols, it is vital to use weak assumptions to reduce the attack surface and thus achieve strong guarantees. Arguably the "minimal" (and most frequently used) assumption is the point-to-point authenticated channel. A slightly stronger assumption is to use symmetric cryptography only—no public-key cryptography (PKC). Symmetric cryptography primitives, such as message authentication codes (MACs) and hash functions—with appropriately chosen parameters—are believed to defend against quantum adversaries that may leverage quantum mechanical phenomena to solve the problem intractable for conventional computers (see NIST reports [23]).

In the partial synchrony setting, the classic PBFT protocol [22] is the first one relying on no PKC—authenticated channels and hash functions only. (The protocol in Castro’s PhD thesis [21] and Cachin’s formulation for PBFT [15] as-
Table 1: Comparison of efficient asynchronous BFT systems.

<table>
<thead>
<tr>
<th></th>
<th>authenticated channel only</th>
<th>no pkc</th>
<th>quantum secure</th>
<th>no trusted setup</th>
<th>adaptive security</th>
<th>high WAN throughput</th>
</tr>
</thead>
<tbody>
<tr>
<td>SINTRA [16]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RITAS [54]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HoneyBadger [53], BEAT [33]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Dumbo family [40,44,45]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>EPIC [49]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Turk [28], Bullshark [42]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PACE [66]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>SodsBC [31]</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WaterBear-QS (this work)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>WaterBear (this work)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

sume authenticated channels and hash functions too.) They can be modified to assume authenticated channels only, as commented in PBFT "It is also possible to modify the algorithm (PBFT) not to use a cryptographic hash function by replacing the hash of a message by the value of the message." Recently, Stern and Abraham proposed a variant of HotStuff relying on authenticated channels only and achieving improved message complexity [62]. All these partially synchronous BFT protocols achieve quantum security.

In contrast, we do not know how to build asynchronous BFT with quantum security, asynchronous BFT without PKC, or asynchronous BFT with authenticated channels only.

Adaptive security vs. static security. Depending on how the adversary decides to corrupt replicas, there are two types of corruptions: static corruptions and adaptive corruptions. A static adversary is restricted to choose its set of corrupted replicas at the start of the protocol. An adaptive adversary can choose its set of corrupted replicas at any moment during the execution of the protocol, based on the information it has accumulated. There is a strong separation result that statically secure protocols are not necessarily adaptively secure [19, 27].

Classic partially synchronous protocols such as PBFT naturally achieve adaptive security, while all asynchronous BFT protocols implemented (except RITAS [54] and EPIC [49]) achieve static security and even for RITAS and EPIC, they achieve adaptive security with significant performance penalty.

1.2 Why Matching Security Guarantees of Partially Synchronous BFT Hard?

Unlike partially synchronous BFT protocols, asynchronous BFT protocols must be randomized to achieve liveness (due to the celebrated FLP impossibility result [39]). Existing asynchronous BFT protocols rely critically on cryptographic common coin protocols (a distributed object that generates the same random coins to all replicas) which are inefficient if no trusted setup is assumed (see Sec. 2 for detailed discussion). Even if relaxing to quantum security, we currently lack efficient common coin instantiations from quantum secure primitives (e.g., lattices). Note while SodsBC is a quantum-secure BFT protocol, it directly relies on trusted setup to generate the common coins and thus bypasses the core problem of generating common coins efficiently [31].

Meanwhile, achieving stronger security in asynchronous BFT typically comes with a higher cost. For instance, adding adaptive security in [49] makes the original BFT system much slower. We must guarantee that ensuring these properties altogether would not incur too much overhead.

1.3 Our Approach

Reducing the problem to local coin RABA then to ABA.

Instead of using common coins, we revisit the local coin based BFT approach that has been long viewed as being inefficient. In the local coin based approach, replicas need to independently and locally flip coins and existing protocols terminate in exponential expected rounds and fail to scale [25, 54].

We thus take a detour and develop the PACE BFT framework [66] using authenticated channels only and achieving quantum security. (Recall existing instantiations in PACE use trusted setup and threshold cryptography, achieving static security only.) PACE devises a variant of asynchronous binary Byzantine agreement (ABA) called reproposable ABA (i.e., RABA). Compared to ABA, RABA has additional properties, allowing all RABA instances to run in parallel and hence improving system throughput. It is also shown that some ABA protocols can be efficiently converted to RABA protocols. Crucially, RABA in PACE enables a fast path for consensus. We observe that while PACE was designed with common coin based RABA, the protocol, even with local coin based RABA, can—on average—terminate within a single RABA round with high probability. As most RABA instances will terminate in one round, the exponential rounds in local coin based ABA is no longer a major efficiency obstacle. Now the per-round complexity of RABA protocols becomes critical. Our strategy is to reduce asynchronous BFT to RABA with local coins and then to ABA with local coins and then improve the per-round complexity of them.

Improving the per-round complexity of (R)ABA. As reported in almost all asynchronous BFT systems [33, 45, 66], ABA is their major performance bottleneck. It is shown that

Jumping ahead, we experimentally demonstrate through latency-breakdown and robustness tests to validate the claim.
WaterBear-QS-C have ABA and Quadratic-ABA, respectively. Unlike prior transformations following a generic approach in [66], we identify and tackle a subtle liveness problem when transforming ABA from weak common coins and perfect common coins. Here weak common coins mean all correct replicas output 0 and 1, both with probability 1/d, where d is a constant and d ≥ 2. If d = 2, weak common coins become perfect common coins. In both cases, CC-ABA compares favorably with existing protocols. CC-ABA with weak coins can be used to improve the distributed key generation protocol [4] and VABA protocols [41, 51], while CC-ABA with perfect coins can be used to improve various BFT protocols such as PACE and Dumbo [45], and the recent distributed key generation protocol requiring the good-case-coin-free property [30].

1.4 Practical Contributions

The WaterBear family of BFT protocols. Table 2 summarizes the characteristics of WaterBear protocols. We use Cubic-RABA to build WaterBear-C and Quadratic-RABA to build WaterBear-Q. WaterBear has all desirable properties a BFT protocol one could think of, being optimally resilient, using authenticated channels only, achieving quantum security and adaptive security, and not relying on trusted setup—matching the security guarantees of the classic PBFT protocol. We comment that WaterBear does not attain information-theoretic (IT) security, as we use HMAC that is not IT-secure.

We also build WaterBear-QS using authenticated channels and hash functions and achieving quantum security for both safety and liveness properties. Similar to WaterBear, WaterBear-QS family also consists of two protocols: WaterBear-QS-C and WaterBear-QS-Q, quantum secure versions of WaterBear-C and WaterBear-Q, respectively.

A new BFT platform. Starting from HoneyBadger, existing asynchronous BFT protocols, including BEAT, Dumbo, and EPIC, use the HoneyBadger programming framework using Python 2.7 (end of life and end of support on January 1, 2020). We instead build a new platform using Go language that is more modular and developer-friendly than existing ones. Our platform currently supports WaterBear-C, WaterBear-Q, WaterBear-QS-C, WaterBear-QS-Q, and BEAT (one of the most efficient open-source asynchronous BFT) [1,33]. Due to

### Table 2: WaterBear-QS and WaterBear instantiations. As in PACE, both WaterBear-QS and WaterBear have a fast path allowing them to terminate in \(O(\log n)\) time. As shown in PACE, the probability of triggering fast paths is high. WaterBear-C and WaterBear-QS-C have \(O(n^3)\) messages on average due to the usage of Cubic-RABA, while WaterBear-Q and WaterBear-QS-Q have \(O(n^5)\) messages on average due to the usage of Quadratic-RABA—matching those of HoneyBadger, BEAT, and PACE.

<table>
<thead>
<tr>
<th>protocol</th>
<th>reference implementation</th>
<th>RBC</th>
<th>RABA</th>
<th>building blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>WaterBear-C</td>
<td>Bracha’s RBC [14]</td>
<td>Cubic-RABA</td>
<td>MAC and hash</td>
<td></td>
</tr>
<tr>
<td>WaterBear-Q</td>
<td>Bracha’s RBC [14]</td>
<td>Quadratic-RABA</td>
<td>MAC and hash</td>
<td></td>
</tr>
<tr>
<td>WaterBear-QS-C</td>
<td>CT RBC [17]</td>
<td>Cubic-RABA</td>
<td>MAC</td>
<td></td>
</tr>
<tr>
<td>WaterBear-QS-Q</td>
<td>C’T RBC [17]</td>
<td>Quadratic-RABA</td>
<td>MAC</td>
<td></td>
</tr>
</tbody>
</table>

### Table 3: Local coin based ABA protocols with optimal resilience. We consider the messages and steps in each round. Messages/round and steps/round denote number of messages and steps among all replicas per round.

<table>
<thead>
<tr>
<th>ABA (local coins)</th>
<th>messages/round</th>
<th>steps/round</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bracha’s ABA [14]</td>
<td>(n^3)</td>
<td>9 to 12</td>
</tr>
<tr>
<td>Cubic-ABA (this work)</td>
<td>(n^3)</td>
<td>5 to 7</td>
</tr>
<tr>
<td>Quadratic-ABA (this work)</td>
<td>(n^3)</td>
<td>4 or 5</td>
</tr>
</tbody>
</table>

The techniques we introduce for Quadratic-ABA are independent interests, allowing us to obtain CC-ABA that works for both weak common coins and perfect common coins. Here weak common coins mean all correct replicas output 0 and 1, both with probability 1/d, where d is a constant and d ≥ 2. If d = 2, weak common coins become perfect common coins. In both cases, CC-ABA compares favorably with existing protocols. CC-ABA with weak coins can be used to improve the distributed key generation protocol [4] and VABA protocols [41, 51], while CC-ABA with perfect coins can be used to improve various BFT protocols such as PACE and Dumbo [45], and the recent distributed key generation protocol requiring the good-case-coin-free property [30].

1.4.2 Practical Contributions

The WaterBear family of BFT protocols. Table 2 summarizes the characteristics of WaterBear protocols. We use Cubic-RABA to build WaterBear-C and Quadratic-RABA to build WaterBear-Q. WaterBear has all desirable properties a BFT protocol one could think of, being optimally resilient, using authenticated channels only, achieving quantum security and adaptive security, and not relying on trusted setup—matching the security guarantees of the classic PBFT protocol. We comment that WaterBear does not attain information-theoretic (IT) security, as we use HMAC that is not IT-secure.

We also build WaterBear-QS using authenticated channels and hash functions and achieving quantum security for both safety and liveness properties. Similar to WaterBear, WaterBear-QS family also consists of two protocols: WaterBear-QS-C and WaterBear-QS-Q, quantum secure versions of WaterBear-C and WaterBear-Q, respectively.

A new BFT platform. Starting from HoneyBadger, existing asynchronous BFT protocols, including BEAT, Dumbo, and EPIC, use the HoneyBadger programming framework using Python 2.7 (end of life and end of support on January 1, 2020). We instead build a new platform using Go that is more modular and developer-friendly than existing ones. Our platform currently supports WaterBear-C, WaterBear-Q, WaterBear-QS-C, WaterBear-QS-Q, and BEAT (one of the most efficient open-source asynchronous BFT) [1,33]. Due to

---

**Efficient local coin based ABA.** Table 3 shows two novel local coin based ABA protocols that we introduce in the paper: Cubic-ABA and Quadratic-ABA. Cubic-ABA is easy to understand and implement, and can be viewed as an optimized version of Bracha’s ABA. Cubic-ABA has 7 steps per round in the worst case, while Bracha’s ABA uses 12 steps, almost doubling the number of steps of Cubic-ABA. In contrast, Quadratic-ABA adopts a novel design, having 4 or 5 steps per round only and being the first local coin based ABA protocol to terminate in a single step in the optimistic mode.

**Tackling a subtle liveness issue for RABA.** We go on to design Cubic-RABA and Quadratic-RABA based on Cubic-ABA and Quadratic-ABA, respectively. Unlike prior transformations following a generic approach in [66], we identify and tackle a subtle liveness problem when transforming Quadratic-ABA to Quadratic-RABA. The issue that we identify demonstrates the subtlety of transforming ABA to RABA, and once again underlines the importance of a full proof when designing Byzantine-resilient protocols.

**ABA from weak common coins and perfect common coins.**

The techniques we introduce for Quadratic-ABA are independent interests, allowing us to obtain CC-ABA that works for both weak common coins and perfect common coins. Here weak common coins mean all correct replicas output 0 and 1, both with probability 1/d, where d is a constant and d ≥ 2. If d = 2, weak common coins become perfect common coins. In both cases, CC-ABA compares favorably with existing protocols. CC-ABA with weak coins can be used to improve the distributed key generation protocol [4] and VABA protocols [41, 51], while CC-ABA with perfect coins can be used to improve various BFT protocols such as PACE and Dumbo [45], and the recent distributed key generation protocol requiring the good-case-coin-free property [30].
its modularity, the library only contains about 11,000 LOC. Large-scale experiments and robustness evaluation. With a 61-instance deployment on Amazon EC2, we show our protocols offer comparable performance as the state-of-the-art asynchronous BFT protocols, while achieving much stronger security. We also design and evaluate various failure and attack scenarios, showing all our protocols are highly robust during failures and attacks. Specifically, one of our protocols, WaterBear-QS-Q, consistently outperforms BEAT (with much weaker security guarantees); for instance, when \( n = 16 \), the latency of WaterBear-QS-Q is about 1/8 that of BEAT and the throughput of WaterBear-QS-Q is 1.23x that of BEAT. The peak throughput of WaterBear-QS-Q (as \( n \) grows larger) is about 1.47x that of BEAT.

1.5 Paper Organization

In what follows, we first discuss related work (Sec. 2) and describe the system model and definitions (Sec. 3). Then we present Cubic-ABA and Quadratic-ABA from local coins (Sec. 4), their RABA counterparts—Cubic-RABA and Quadratic-RABA (Sec. 5), and WaterBear BFT protocols (Sec. 6). Last, we present our evaluation results (Sec. 7) before concluding the paper (Sec. 8).

1.6 The Proceeding Version

The paper is the full paper of our proceeding version [68], appearing at Usenix Security 2023. The full paper contains many new theoretical and experimental results, including ADA protocols from perfect common coins and weak common coins.

2 Related Work

ADKG. A line of recent works studied how to eliminate trusted setup by using asynchronous distributed key generation (ADKG) [4, 30, 48, 67]. Even if using ADKG in existing asynchronous BFT protocols, they would neither achieve quantum security nor security with no PKC.

Adaptive vs. static security for BFT. Most asynchronous BFT protocols implemented, including SINTRA, HoneyBadgerBFT, BEAT, and Dumbo, defend against static adversary only. These protocols rely critically on efficient but statically secure threshold cryptography. EPIC is an asynchronous BFT that uses adaptively secure threshold pseudorandom function (PRF) to achieve adaptive security but is not as efficient as its statically secure counterparts. RTAS [54] contains an adaptively secure BFT protocol, but due to inefficient local coin based ABA, it is less efficient than other protocols in large-size networks. The situation for asynchronous environments is in sharp contrast to that of partially synchronous protocols, most of which attain adaptive security [8, 22, 24, 32, 43, 46, 61].

Quantum safety (but no quantum liveness). A BFT protocol is quantum secure if its safety is quantum resistant (quantum safety) and its liveness is quantum resistant (quantum liveness) [47]. DAG-Rider [47] achieves quantum safety. Moreover, the BKR protocol and their descendants (e.g., HoneyBadger [53], MiB [50], PACE [66]) achieve quantum safety if using techniques from EPIC [49]. All these protocols, however, do not achieve quantum liveness. Tusk [28] and Bullshark [42] are variants of DAG-Rider; they extensively use signatures and hashes and achieve neither quantum safety nor quantum liveness.

Byzantine agreement and common coins. Byzantine agreement (BA) is a central tool for both distributed computing and cryptography. The condition \( n \geq 3f+1 \) is both necessary and sufficient for both synchronous and asynchronous BA protocols [58]. The celebrated impossibility result of Fischer, Lynch, and Paterson [39] implies that a randomized BA protocol must have non-terminating executions. BA protocol may thus be either \((1-\epsilon)\)-terminating, where correct replicas terminate the protocol with an overwhelming probability, or almost-surely terminating, where replicas terminate with probability 1. For both types, we review BA protocols assuming authenticated channels only. Note there is no need to consider BA with authenticated channels and hash functions, as the input to BA is a binary value and we do not need to use hash functions to compress the input.

For almost-surely BA terminating with probability 1, Ben-Or’s BA requires \( n \geq 5f + 1 \) [11], while Bracha’s BA [13] achieves optimal resilience. The two protocols use local coins and require an exponential expected running time. Feldman and Micali propose a BA protocol having a constant expected running time in synchronous environments and extend it to build a polynomial-time BA protocol requiring \( n \geq 4f + 1 \) [38]. Abraham, Dolev, and Halpern [3] provide the first almost-surely BA with polynomial efficiency (concretely, expected \( O(n^2) \) time) and optimal resilience. Bangalore, Choudhury, and Patra [10] improve the expected running time of [3] by a factor of \( n \).

For \((1-\epsilon)\)-terminating BA, Canetti and Rabin [20] build an expected constant-round BA protocol with optimal resilience. Patra, Choudhury, and Rangan [57] build a more efficient construction in terms of communication complexity. Both types of BA protocols follow the classic framework of Feldman and Micali [38] that reduces BA to asynchronous verifiable secret sharing (AVSS). The framework uses AVSS to build common coins. (The original idea of using common coin for BA is due to Rabin [59].) Unfortunately, the framework of using AVSS for common coins is prohibitively expensive. For instance, to build AVSS, the approach of Canetti and Rabin [20] needs to begin with an information checking protocol, then asynchronous recoverable sharing, then asynchronous weak secret sharing, and finally AVSS. The improved approach of Patra, Choudhury, and Rangan [57] remains complex, following the route of information-checking protocol, then asynchronous weak commitment, and then AVSS. Moreover, the transformation from AVSS to BA is equally expensive, requiring running \( n^2 \) AVSS instances to generate a single (weak) coin. Patra, Choudhury, and Ran-
gan [57] also propose an approach for sharing multiple secrets simultaneously. While such an approach is useful for building more efficient multi-valued BA (MBA), it is unknown if it would yield more efficient ABA protocols. While, for instance, the CNV framework [25] does use MBA, it may run \( O(n) \) consecutive MBA instances (which is inefficient).

3 System Model and Definitions

3.1 System and Threat Model

This section describes the system model for distributed protocols in the paper, where \( f \) out of \( n \) replicas may fail arbitrarily (Byzantine failures). We assume point-to-point authenticated channels between each pair of replicas; some of our protocols additionally assume hash functions. The WaterBear BFT protocols (and subprotocols we use or we invent) have the following properties:

- **Optimal resilience**: The protocols in this work assume \( f \leq \left\lfloor \frac{n-1}{3} \right\rfloor \), which is optimal. A (Byzantine) *quorum* is a set of \( \lceil \frac{n+f+1}{2} \rceil \) replicas. For simplicity, we may assume \( n = 3f + 1 \) and a quorum size of \( 2f + 1 \).

- **Asynchronous network**: We consider completely asynchronous systems making no timing assumptions on message processing or transmission delays. In contrast, partially synchronous systems assume that there exist an upper bound on message processing and transmission delays but the bound may be unknown to anyone [37].

Designing asynchronous systems is challenging, because in asynchronous environments it is impossible to distinguish Byzantine faulty replicas from "slow" replicas. In particular, one cannot use timers or timeout to assist in the design of asynchronous systems.

- **No dealer/trusted setup**: We do not assume the existence of a trusted dealer or trusted setup. Neither do we assume there exists an interactive protocol for any public keys, reference strings, or public parameters.

- **Adaptive corruptions**: We consider adaptive adversary that can choose its set of corrupted replicas at any moment during the execution of the protocol, based on the information it has accumulated thus far (i.e., the messages observed and the states of previously corrupted replicas).

We may associate a protocol instance with a unique identifier \( id \), tagging each message in the instance with \( id \). If no ambiguity arises, we may omit the identifiers.

3.2 Definitions and Preliminaries

**BFT.** In a BFT protocol, a replica *a-delivers* (atomically deliver) *transactions*, each *submitted* by some client. The client computes a final response to its submitted transaction from the responses it receives from replicas. We consider the following properties:

- **Agreement**: If any correct replica *a-delivers* a transaction \( tx \), then every correct replica *a-delivers* \( tx \).

- **Total order**: If a correct replica *a-delivers* a transaction \( tx \) before *a-delivering* \( tx' \), then no correct replica *a-delivers* a transaction \( tx' \) without first *a-delivering* \( tx \).

- **Liveness**: If a transaction \( tx \) is *submitted* to all correct replicas, then all correct replicas eventually *a-deliver* \( tx \).

Below, we first introduce ABA and then its variant—RABA. Then we review the PACE framework using RBC and RABA.

**Asynchronous binary Byzantine agreement (ABA).** An ABA protocol is specified by *propose* and *decide*. Each replica proposes an initial binary value (called *vote*) for consensus and replicas will decide on some value. ABA should satisfy the following properties:

- **Validity**: If all correct replicas *propose* \( v \), then any correct replica that terminates *decides* \( v \).

- **Agreement**: If a correct replica *decides* \( v \), then any correct replica that terminates *decides* \( v \).

- **Termination**: Every correct replica eventually *decides* some value.

- **Integrity**: No correct replica *decides* twice.

**RABA.** Reprosposable ABA (RABA) is a new distributed computing primitive introduced in PACE [66]. In contrast to conventional ABA protocols, where replicas can vote once only, RABA allows replicas to change their votes. Formally, a RABA protocol tagged with a unique identifier \( id \) is specified by *propose*(\( id, \cdot \)), *repropose*(\( id, \cdot \)), and *decide*(\( id, \cdot \)), with the input domain being \( \{0, 1\} \). For our purpose, RABA is “biased towards 1.” Each replica can propose a vote \( v \) at the beginning of the protocol. Each replica can propose a vote only once. A correct replica that proposed 0 is allowed to change its mind and repropose 1. A replica that proposed 1 is not allowed to repropose 0. If a replica reproposes 1, it does so at most once. A replica terminates the protocol identified by \( id \) by generating a decide message. RABA (biased towards 1) satisfies the following properties:

- **Validity**: If all correct replicas *propose* \( v \) and never *repropose* \( \bar{v} \), then any correct replica that terminates *decides* \( v \).

- **Unanimous termination**: If all correct replicas *propose* \( v \) and never *repropose* \( \bar{v} \), then all correct replicas eventually terminate.

- **Agreement**: If a correct replica *decides* \( v \), then any correct replica that terminates *decides* \( v \).

- **Biased validity**: If \( f + 1 \) correct replicas *propose* 1, then any correct replica that terminates *decides* 1.

- **Biased termination**: Let \( Q \) be the set of correct replicas. Let \( Q_1 \) be the set of correct replicas that propose 1 and never repropose 0. Let \( Q_2 \) be correct replicas that propose 0 and later repropose 1. If \( Q_2 \neq \emptyset \) and \( Q = Q_1 \cup Q_2 \), then each correct replica eventually terminates.

- **Integrity**: No correct replica *decides* twice.

Validity is slightly different from those for ABA. They are modified to accommodate the RABA syntax. Integrity is defined to ensure RABA decides once and once only.

Unanimous termination and biased termination are carefully introduced to help achieve RABA termination in certain
scenarios. External operations would have to force the protocol to meet these termination conditions.

Biased validity in RABA requires that if \( f + 1 \) correct replicas, not simply all correct replicas, propose 1, then a correct replica that terminates decides 1. The property guarantees the PACE framework has sufficient transactions delivered.

This paper introduces new RABA protocols from local coins.

**RBC.** In a Byzantine reliable broadcast (RBC) protocol [5–7, 14, 17, 29], a replica \( p \) first starts the protocol by executing \( r\text{-broadcast} \) with messages \( m \), and all replicas terminate the protocol by executing \( r\text{-deliver} \) with message \( m \). We consider the following properties:

- **Validity:** If a correct replica \( p \) \( r\text{-broadcasts} \) a message \( m \), then \( p \) eventually \( r\text{-delivers} \) \( m \).
- **Agreement:** If some correct replica \( r\text{-delivers} \) a message \( m \), then every correct replica eventually \( r\text{-delivers} \) \( m \).
- **Integrity:** For any message \( m \), every correct replica \( r\text{-delivers} \) \( m \) at most once. Moreover, if the sender is correct, then \( m \) was previously \( r\text{-broadcast} \) by the sender.

This paper uses Bracha’s broadcast [13] that assumes authenticated channels only and has a bandwidth of \( O(n^2|m|) \), and uses CT RBC due to Cachin and Tessaro [17] that additionally uses hash functions (with output length \( \lambda \)) to reduce the bandwidth to \( O(n|m| + \lambda^2 \log n) \).

**PACE framework.** PACE uses RBC and RABA in a black-box manner to construct efficient asynchronous BFT. The framework allows all RABA instances to run in parallel, removing a well-known bottleneck in the original framework of Ben-Or, Kelmer, and Rabin [12]. Concretely, PACE has a RBC phase and a RABA phase; correct replicas can run the RABA phase in parallel once \( n - f \) RBC instances have completed. PACE also provides a fast path for consensus, allowing the protocol to terminate using a single RABA round.

**Steps, phases, and rounds.** In asynchronous environments, the network delay is unbounded. To measure the latency of asynchronous protocols, we use the standard notion of asynchronous steps [20], where a protocol runs in \( x \) asynchronous steps if its running time is at most \( x \) times the maximum message delay between honest replicas during the execution.

We also use the notion of phases for ease of description, where a phase in a protocol consists of a fixed number of steps. When describing some of our protocols, we may divide a protocol into several phases, each of which has several steps.

In this paper, the notion of rounds is restricted to ABA protocols: an ABA protocol proceeds in rounds, where an ABA round consists of a fixed number of steps. For instance, local coin ABA protocols terminate in expected exponential rounds, while ABA assuming common coins (including CC-ABA we introduce in this paper) terminates in expected constant rounds. An ABA round may consist of several phases and each phase consists of several steps. In asynchronous ABA systems, replicas proceed in rounds and they might not be always in the same round, but each correct replica eventually terminates every round that it has participated in.

## ABA from Local Coins

**Summary of our results.** The state-of-the-art local coin based ABA protocol, Bracha’s ABA [13], has \( O(n^3) \) messages and 12 steps in each round. We design two new ABA protocols from local coins, Cubic-ABA and Quadratic-ABA, with two goals in mind—being more efficient than Bracha’s ABA and being compatible with RABA.

We begin with the simpler one, Cubic-ABA, that achieves the same message complexity as Bracha’s ABA but has only 7 steps in each round. Cubic-ABA admits a clean and intuitive proof of correctness. We go on to suggest Quadratic-ABA on top of Cubic-ABA. Compared to Cubic-ABA, Quadratic-ABA reduces the messages from \( O(n^3) \) to \( O(n^2) \) and reduces the number of steps to 5 in each round. The improvement is significant, allowing WaterBear to attain the same average message complexity in normal cases as PACE—\( O(n^3) \).

Equally important, both ABA protocols can be modified for efficient RABA protocols.

As an important by-product, extending the idea of Quadratic-ABA and assuming the existence of (weak or perfect) common coins, we can present ABA protocols that have expected constant rounds and outperform the state-of-the-art protocols, as shown in Appendix G.

### 4.1 Cubic-ABA

**Overview.** Our motivation for Cubic-ABA is to reduce the number of parallel RBCs in Bracha’s ABA from three to two. In particular, in each round, Bracha’s ABA has three phases, where in each phase, replicas run \( n \) parallel RBCs. In total, Bracha’s ABA has 12 steps and \( O(n^2) \) messages per round. (We recall Bracha’s ABA in Appendix A.) In Cubic-ABA, we replace the first two RBC phases with one or two steps of all-to-all broadcast so Cubic-ABA only has 5 to 7 steps.

**The protocol.** Figure 1 describes the pseudocode of Cubic-ABA. In particular, local coin ABA protocols terminate in expected exponential rounds, while ABA assuming common coins (including CC-ABA we introduce in this paper) terminates in expected constant rounds. An ABA round may consist of several phases and each phase consists of several steps. In asynchronous ABA systems, replicas proceed in rounds and they might not be always in the same round, but each correct replica eventually terminates every round that it has participated in.
Upon receiving pre-vote, (v) from f + 1 replicas, if pre-vote, (v) has not been sent, broadcast pre-vote, (v) and wait until bset, ≠ ∅.

Upon receiving pre-vote, (v) from 2f + 1 replicas (phase 2):

- If there are n − f main-vote, (v) that for each received main-vote, (b), b ∈ bset, (phase 3):
- If there are n − f − 1 final-vote, (v) then ivr + 1 ← v, decide v.
- Else if there are f + 1 final-vote, (v) then ivr + 1 ← v.

In the third phase, we need to ensure that if a correct replica receives n − f final-vote, (v) and decide, any correct replica will propose v and eventually decide v. Note for the case where f + 1 correct replicas vote for v and f correct replicas vote for * we need to guarantee that if a correct replica receives n − f − 1 final-vote, (v) any correct replica will receive at least f + 1 final-vote, (v) and therefore vote for v in the following round. Thus, we rely on RBC, ensuring all correct replicas eventually receive consistent values, even in the presence of Byzantine replicas. As we show in the proof, this is crucial for the agreement property.

Note that our protocol presented is asynchronous and replicas are not always in the same round. If a replica pr is in round r and receives a message from pr tagged by r′ > r (e.g., pre-vote, r, (v)), pr can simply store the message and will process it after pr enters round r′. As shown in the termination property (Theorem 7), each correct replica will proceed to the next round before it decides a value and all replicas will eventually decide in the same round.

As the number of n parallel RBC instances is 1 instead of 3, the number of steps is reduced from 12 to 7.

### 4.2 Quadratic-ABA

**Overview.** In Quadratic-ABA, we replace the only parallel RBC phase used in Cubic-ABA using a novel two-step all-to-all broadcast. The goal is to ensure that at the end of each round, if a correct replica receives n − f matching votes for a value v, any correct replica will receive either n − f votes for v or at least f + 1 matching v. This will guarantee that all correct replicas will vote for v in the following round. By eliminating parallel RBC instances causing $O(n^3)$ messages and $O(n^3)$ communication, Quadratic-ABA now achieves $O(n^2)$ messages and $O(n^2)$ communication per round.

**The protocol.** The pseudocode of Quadratic-ABA is shown in Figure 3. The Quadratic-ABA protocol is round-based, starting from round 0. In each round, there are four phases—pre-vote, (), vote, (), main-vote, () and.
Figure 3: The Quadratic-ABA protocol. The code for $p_i$.

every correct replica accepts a main-vote, message only if $bset_r = \{0, 1\}$. Upon receiving $n - f$ main-vote, messages, one of the following two conditions holds.

- **Ln 20-21:** If $p_i$ receives $n - f$ main-vote, messages, it broadcasts a final-vote, message.
- **Ln 22:** Otherwise, $p_i$ broadcasts final-vote, message.

Every correct replica accepts a final-vote, message only if it has received $f + 1$ main-vote, messages. Every correct replica accepts a final-vote, message only if $bset_r = \{0, 1\}$. Upon receiving $n - f$ final-vote, messages, there are three cases:

- **Ln 24-25:** If $p_i$ receives $n - f$ final-vote, messages, it decides $v$ and also sets $iv_{r+1}$ as $v$. It participates in the protocol for one more round and terminates the protocol.
- **Ln 26-27:** If $p_i$ receives $n - f$ final-vote, messages that carry only value $v$ and $\bar{v}$, it uses $v$ for round $r + 1$.
- **Ln 28-29:** Otherwise, $p_i$ generates a local random coin and uses it as input for the next round.

**Analysis.** Quadratic-ABA guarantees that if a correct replica receives $n - f$ final-vote, messages, any correct replica will set $iv_{r+1}$ as $v$ (Lemma 15). First, if a correct replica sends main-vote, no correct replica will send main-vote, message. In particular, if a correct replica sends main-vote, message, it must have received $n - f$ vote, messages. If another correct replica sends main-vote, message, at least $n - f$ replicas have sent vote, messages. Therefore, at least one correct replica has sent both vote, and vote, which contradicts the fact that each correct replica only sends a single vote, message in each round. Furthermore, if a correct replica sends final-vote, message, no correct replica will send final-vote, message or even accept final-vote, message from other replicas. This is because if a correct replica accepts final-vote, message, at least one correct replica has sent main-vote, message. Meanwhile, if a correct replica accepts final-vote, message, at least one correct replica has sent main-vote, message, contradicting Lemma 13. Hence, at the end of each round, if a correct replica receives only final-vote, message, another correct replica receives only final-vote, message, and final-vote, message, it holds that $v_1 = v_2$. This result is crucial for agreement and termination.

Furthermore, if a correct replica decides $v$ in round $r$, it must have received $n - f$ final-vote, messages. Among them, at least $f + 1$ correct replicas have sent final-vote, messages. With $3f + 1$ replicas in total, there are at most $2f$ final-vote, message or final-vote, message. Hence, every correct replica receives at least one final-vote, message. As no correct replica will ac-
cept final-vote, every correct replica will only have final-vote,1(v) and final-vote,2(v). Thus, every correct replica either decides in round r, or enters round r + 1 and sets ivr+1 to v. Doing so ensures agreement.

5 RABA from Local Coins

As shown in PACE [66], the PACE framework with RABA significantly outperforms the conventional BKR diagram and enables a fast path for termination. Our goal here is to use local coin based ABA to design RABA without trusted setup. We use Cubic-ABA and Quadratic-ABA to build Cubic-RABA and Quadratic-RABA, respectively. Here, we focus on Quadratic-RABA and present Cubic-RABA in Appendix B.

Subtlety of building Quadratic-RABA. PACE introduced a general approach to converting ABA to RABA [66]. Following their approach, we present Quadratic-RABA in Figure 5. Quadratic-RABA is identical to Quadratic-ABA except for the first round (round 0), where we make the following changes. First, we use a propose() event and a repropose() event (ln 03-07). Upon propose(v), a replica pi starts round 0 and executes the broadcast-vote(v) function. Upon repropose(1) event, regardless of which round a replica is in, pi still executes the broadcast-vote(v) function. The propose() and repropose() events are crucial for biased termination. So if a quorum of correct replicas either propose 1 or repropose 1, the protocol will eventually terminate.

Second, in the broadcast-vote(v) function, replica pi broadcasts a pre-vote0(v) message (ln 09). At ln 10-14, if v = 1, pi immediately adds 1 to bset0, and broadcasts vote0(1), main-vote0(1), and final-vote0(1).

Third, the coin value in round 0 is set to 1 (ln 38). The second and the third modifications guarantee both biased validity property and a fast path allowing terminating the protocol in one step only. Namely, if f + 1 correct replicas propose 1, no correct replica will receive 2f + 1 final-vote0(0). As we will show in the proof, every correct replica will either directly decide 1 or set iv1 as 1, so all correct replicas decide within two rounds. Furthermore, our protocol has a fast path: if all correct replicas propose 1, they will directly send vote0(1), main-vote0(1), final-vote0(1), allowing correct replicas to decide in one step.

The above modifications largely follow the generic transformation. We find that these modifications are sufficient for a secure Cubic-ABA, just as all known ABA protocols that can be transformed into their secure RABA counterparts (shown in [66]). Unexpectedly, we find for Quadratic-ABA, however, there is a subtle liveness issue for round 0. Suppose f correct replicas propose 1 and f + 1 correct replicas propose 0. The f replicas directly broadcast vote0(1), main-vote0(1), and final-vote0(1). Even if the f + 1 correct replicas that proposed 0 may later repropose 1, they may have already sent vote0(0), main-vote0(0), and final-vote0(0). In this case, no correct replica will accept final-vote0(1), as they do not receive f + 1 main-vote0(1). The issue is, in essence, caused by the fact that each correct replica accepts a main-vote,1(v) message only if it has previously received f + 1 vote,1(v) messages, and each correct replica accepts a final-vote,1(v) message only if it has previously received f + 1 main-vote,1(v) messages. We visualize the example in Appendix C.

To resolve the above issue, we introduce another change to round 0 of the protocol. In particular, we relax the conditions for round 0: for each main-vote,1(v) (ln 28) and final-vote,1(v)
01 upon selecting $m_i$ for $p_i$ using the technique of EPIC
02 $r$-broadcast($[e, i], m_i$) for RBC$_i$
03 upon $r$-deliver($[e, j], m_j$) for RBC$_j$
04 if RABA$_j$ has not been started
05 propose($[e, j], 1$) for RABA$_j$
06 else
07 repropose($[e, j], 1$) for RABA$_j$
08 upon delivery of $n - f$ RBC instances
09 for RABA instances that have not been started
10 propose($[e, j], 0$)
11 upon decide($[e, j], v$) for any value $v$ for all RABA instances
12 let $S$ be set of indexes for RABA instances that decide 1
13 wait until $r$-deliver($[e, j], m_j$) for all RABA$_j$ where $j \in S$
14 a-deliver($\bigcup_{j \in S} m_j$)

Figure 6: The WaterBear family. The code for replica $p_i$ in epoch $e$. WaterBear uses the EPIC technique to select transactions.

(In 32), a correct replica accepts it as long as $v \in bset_0$. With this modification, the set of $f + 1$ correct replicas that proposed 0 will repropose 1, so every correct replica will eventually add 1 in $bset_0$. Hence, every correct replica will eventually accept main-vote$_0(1)$ and final-vote$_0(1)$. Our result underlines the subtlety of constructing RABA from ABA and the importance of a full proof for a new protocol (proof in Appendix I).

6 The WaterBear Family

This section describes our asynchronous BFT protocols—WaterBear (WaterBear-C and WaterBear-Q), and WaterBear-QS (WaterBear-QS-C, and WaterBear-QS-Q). All the protocols are quantum secure, and WaterBear-C and WaterBear-Q rely on authenticated channels only.

6.1 The WaterBear Protocols

WaterBear follows the PACE paradigm but uses the trick in EPIC [49] to avoid the usage of threshold encryption (needed for achieving adaptive security). In particular, WaterBear uses $r$-broadcast and $r$-deliver primitives of Bracha’s broadcast, and propose, repropose and decide primitives of WaterBear RABA. Figure 6 depicts the pseudocode of WaterBear. Earlier asynchronous BFT protocols such as HoneyBadger and BEAT use threshold encryption to allow parallel, random transaction selection while achieving censorship resilience (liveness); however, no efficient adaptively secure threshold encryption was known. EPIC thus proposes a new crypto-free transaction selection strategy: replicas select random transactions in plaintext for most epochs, and to achieve liveness, they periodically switch to the first-in, first-out (FIFO) selection, where replicas maintain a log of transactions according to the order that transactions are received and select as input the first transaction group in the buffer. In WaterBear, we use this trick to avoid threshold encryption.

Following the PACE paradigm, for each epoch $e$, WaterBear consists of $n$ parallel RBC instances and $n$ parallel RABA instances. In the RBC phase, each replica $p_i$ $r$-broadcasts a proposal $m_i$ for RBC$_i$. If $p_i$ $r$-delivers a proposal from RBC$_j$, it proposes 1 for RABA$_j$. Upon delivery of $n - f$ RBC instances, instead of waiting for $n - f$ RABA instances to terminate, $p_i$ proposes 0 for all RABA instances that have not been started. If $p_i$ later delivers a proposal from some RBC$_j$, it has proposed 0 for RABA$_j$, and has not terminated RABA$_j$, it reproposes 1 for RABA$_j$.

Let $S$ be the set of indexes where RABA$_j$ decides 1. When all RABA instances terminate and all RBC$_i$ ($i \in S$) instances are delivered, $p_i$ $a$-delivers $\bigcup_{j \in S} m_j$. The security of WaterBear directly follows from that of the PACE paradigm. As we propose two RABA protocols Cubic-RABA and Quadratic-RABA. We use Cubic-RABA to build WaterBear-C and Quadratic-RABA to build WaterBear-Q.

6.2 The WaterBear-QS Protocols

We now describe WaterBear-QS, also consisting of two asynchronous BFT protocols. We use Cubic-RABA to build WaterBear-QS-C and Quadratic-RABA to build WaterBear-QS-Q. The difference between WaterBear and WaterBear-QS is that WaterBear-QS additionally uses hash functions (used in CT RBC [17]) to reduce the communication complexity of RBC. Jumping ahead, we show the modification leads to a dramatic performance improvement.

7 Implementation and Evaluation

Implementation. We implemented WaterBear-C, WaterBear-Q, WaterBear-QS-C, and WaterBear-QS-Q in a new Golang library. For comparison, we choose to implement BEAT2 [33]
We use gRPC as the communication library. In the LAN setting, the replicas are run in the same region of machines (VMs). We consider both LAN and WAN settings.

All the protocols use authenticated channels and WaterBear-QS additionally uses hash functions. We use HMAC to realize authenticated channels. We use SHA256 as the hash function. We use gRPC as the communication library.

All our implemented asynchronous protocols use RBC in their RBC phases, and WaterBear-C and WaterBear-QS-C additionally use RBC in the ABA phase. For WaterBear-C and WaterBear-Q, we use Bracha’s broadcast in the RBC phase. For WaterBear-QS-C and WaterBear-QS-Q, we use CT RBC [17] (using erasure coding and hash functions) in the RBC phase. In ABA phases of WaterBear-C and WaterBear-QS-C, we directly use Bracha’s broadcast because there is no bulk data (and no need to use erasure coding). To implement CT RBC, we use a Golang Reed-Solomon code library [2].

There are several reasons we chose BEAT as the baseline protocol. First, BEAT is one of the most efficient open-source asynchronous BFT implementations. As shown in PACE [66], BEAT is more efficient than Dumbo [45] for \( n \leq 46 \). Second, all WaterBear protocols achieve adaptive security, and EPIC is the only known adaptively secure asynchronous BFT protocol implemented. It is shown that BEAT significantly outperforms EPIC in both LAN and WAN settings [49]. Hence, if we show the performance difference between BEAT and our protocols, we can argue which is the most efficient adaptively secure asynchronous BFT protocol among EPIC and WaterBear protocols. We do not attempt to compare our protocols with other BFT protocols in Table 1, as those protocols neither achieve adaptive nor quantum security, relying on PKC and trusted setup. Indeed, our goal is not to claim WaterBear protocols are the most efficient asynchronous BFT protocols, but we aim at refuting the conventional wisdom that asynchronous BFT protocols cannot match the security guarantees of partially synchronous protocols while preserving performance.

Overview of evaluation. We evaluate the performance of our protocols on Amazon EC2 utilizing up to 61 virtual machines (VMs). We consider both LAN and WAN settings. In the LAN setting, the replicas are run in the same region in our library. We implemented a new version of BEAT, replacing MMR ABA with Cobalt-ABA, as Cobalt ABA addressed the liveness issue of MMR. Our implementation has been made publicly available\(^3\) and involves more than 11,000 LOC for the protocol implementations and about 1,000 LOC for evaluation.

All the protocols use authenticated channels and WaterBear-QS additionally uses hash functions. We use HMAC to realize authenticated channels. We use SHA256 as the hash function. We use gRPC as the communication library.

All our implemented asynchronous protocols use RBC in their RBC phases, and WaterBear-C and WaterBear-QS-C additionally use RBC in the ABA phase. For WaterBear-C and WaterBear-Q, we use Bracha’s broadcast in the RBC phase. For WaterBear-QS-C and WaterBear-QS-Q, we use CT RBC [17] (using erasure coding and hash functions) in the RBC phase. In ABA phases of WaterBear-C and WaterBear-QS-C, we directly use Bracha’s broadcast because there is no bulk data (and no need to use erasure coding). To implement CT RBC, we use a Golang Reed-Solomon code library [2].

\(^3\)https://github.com/fififish/beat

\(^3\)https://github.com/fififish/waterbear
of EC2—US Virginia. In the WAN setting, the replicas are evenly distributed in four different regions: us-west-2 (Oregon, US), us-east-2 (Ohio, US), ap-southeast-1 (Singapore), and eu-west-1 (Ireland). The lowest one-way latency (resp., the highest latency) is 24.5 ms for Ohio-Oregon (resp., 90 ms for Ireland-Singapore). We use both t2.medium and m3.xlarge instances for our evaluation. The t2.medium type has two virtual CPUs and 4GB memory and the m3.xlarge has four virtual CPUs and 16GB memory. Unless otherwise mentioned, we use m3.xlarge instances by default. We conduct the experiments under different network sizes and contention levels (batch size). We use $f$ to denote the network size; in each experiment, we use $3f + 1$ replicas in total. We let $b$ denote the contention level. In particular, each replica proposes $b$ transactions in each epoch. For each experiment, we vary the batch size $b$ from 1 to 25,000. For each experiment, we run the tests 10 times and compute the average (for both throughput and latency). We evaluate the performance of the protocols with two different transaction sizes—100 bytes by default and also 250 bytes.

We assess the protocols under failure-free and failure scenarios. While our failure-case evaluation is not the first such evaluation for asynchronous BFT protocols, the testbed we built aims to be comprehensive, encompassing realistic failure and attack scenarios we can envision. We roughly summarize our main results in the following:

- The WaterBear protocols using Quadratic-RABA (WaterBear-QS-Q and WaterBear-Q) are much more efficient than the protocols using Cubic-RABA (WaterBear-QS-C and WaterBear-C), as Quadratic-RABA has $O(n^2)$ messages and much fewer steps than Cubic-RABA (with $O(n^3)$ messages). The result justifies the importance of designing Quadratic-RABA and Quadratic-ABA.

- The quantum secure WaterBear protocols (WaterBear-QS-C and WaterBear-QS-Q) drastically outperform their counterparts assuming authenticated channels only (WaterBear-C and WaterBear-Q), as the RBC used for WaterBear-QS-C and WaterBear-QS-Q is more bandwidth-efficient than that for WaterBear-C and WaterBear-Q. The finding highlights the cost of achieving security with authenticated channels only from quantum security for our protocols.

- Regarding latency, all WaterBear protocols have lower latency (under no contention) than BEAT. Regarding throughput, all our protocols, except WaterBear-QS-Q, share similar performance as BEAT.

- WaterBear-QS-Q consistently and significantly outpaces BEAT. For instance, when $n = 16$, WaterBear-QS-Q has about 1/8 the latency of that of BEAT and 1.23x the throughput of BEAT. As $n$ grows larger, the peak throughput of WaterBear-QS-Q is about 1.47x that of BEAT.

- All four protocols we propose are highly robust against various crash and Byzantine failures, just as BEAT.

### 7.1 Performance in Failure-Free Cases

**Latency.** We report the latency of the asynchronous protocols in both LAN and WAN settings for $f = 1, 2$, and 5 in Figure 7 with for $b = 1$ and 100. All WaterBear protocols consistently achieve lower latency than BEAT in both LAN and WAN environments, mainly because our protocols have a coin-free fast path. Among the protocols, WaterBear-QS-Q has consistently lower latency than all other protocols, as WaterBear-QS-Q has the lowest communication complexity among the WaterBear protocols. As $f$ increases, the latency difference between WaterBear-QS-Q and other protocols becomes more visible. For instance, when $f = 5$ in the WAN setting, BEAT achieves 3.47x latency of that for WaterBear-QS-Q; in the LAN setting, the latency for BEAT is 8.78x of that for WaterBear-QS-Q.

Let us explain the latency results in more detail. First, take WaterBear-QS-Q in WANs for an example. In the optimistic mode (with no failures and synchronous networks), it takes four steps (3 for RBC and 1 for RABA due to the fast path in Quadratic-RABA) to terminate. When $f = 1$, the latency reported for WaterBear-QS-Q is 310 ms, which is consistent with the ping latency mentioned above (24.5 ms~90 ms). In contrast, WaterBear-QS-C in its optimistic mode has 6 steps, which justifies its latency of 570 ms.

**Throughput and scalability.** We report throughput and throughput vs. latency of all our implemented protocols in Figure 8 by varying the network size $f$ from 1 to 20.

Our results show that the throughput of WaterBear-C and WaterBear-Q are consistently lower than the other protocols. As WaterBear-C (resp. WaterBear-Q) and WaterBear-QS-C (resp. WaterBear-QS-Q) differ in RBC only, RBC is clearly one performance bottleneck. The result highlights the over-
head of achieving security with authenticated channels only.

We assess the throughput of all the protocols for $f = 1$ in WAN as depicted in Figure 8b: the peak throughput of WaterBear-QS-Q is higher than all other WaterBear-QS-C. The peak throughput of WaterBear-QS-C is 65.7 ktx/sec in LANs and 37.8 ktx/sec in WANs.

We also conduct a separate experiment in LANs, as shown in Figure 8a. Unlike the results in WANs, the throughput of BEAT in LANs is marginally higher than WaterBear-QS-C, and the throughput of WaterBear-QS-Q is marginally higher than BEAT: the peak throughput of BEAT is 2.3% higher than WaterBear-QS-C, and the peak throughput of WaterBear-QS-Q is 3.9% higher than BEAT. The peak throughput of WaterBear-QS-C is 65.7 ktx/sec in LANs and 38.4 ktx/sec in WANs. The peak throughput of WaterBear-QS-Q is 69.9 ktx/sec in LANs and 38.4 ktx/sec in WANs.

When $f$ increases, in general, WaterBear-QS-Q and WaterBear-QS-C outpace all other protocols. The peak throughput of WaterBear-QS-C is higher than BEAT when $f = 5$ and $f = 10$ but lower when $f = 20$ only. Meanwhile, WaterBear-QS-Q is consistently more efficient than all other asynchronous protocols (higher throughput and lower latency). For instance, when $f = 10$, the peak throughput of WaterBear-QS-Q is 47.4% higher than BEAT. The reason is WaterBear-QS-Q uses a more communication-efficient RABA protocol.

We report the latency breakdown of the WaterBear protocols for $b = 1$ and 5000 in Figure 8g-8j. These experiments justify the design of our most efficient protocol—WaterBear-QS-Q (with CT RBC that is communication-efficient and Quadratic-RABA with asymptotically reduced per-round message complexity and concretely reduced number of steps). Indeed, as $f$ increases, RABA dominates the latency and Quadratic-RABA performs indeed much better than CubicRABA; when $b$ increases, RBC becomes the latency bottleneck and CT RBC outperforms Bracha’s RBC significantly.

Additional evaluation results. We show in Appendix D additional evaluation results, including the performance comparison with HotStuff [65], the performance on different types of VMs, the performance with different transaction sizes, and the memory and the CPU usage for the protocols implemented.

7.2 Performance under Failures

To assess the protocol performance under failures and attacks, we carefully design various experiments as follows. We focus on the five asynchronous protocols.

- $S_0$: (failure-free) All replicas are correct. $S_0$ is the baseline scenario used to compare with failure scenarios.
- $S_1$: (crash) $f$ replicas crash by not participating in the protocols.
- $S_2$: (Byzantine; keep voting 0) We control all $f$ faulty replicas to keep voting 0 in each step of (R)ABA. For all protocols, doing so would intuitively make fewer (R)ABA instances to decide 1 and would likely decrease the throughput of the protocols. We aim to observe the throughput reduction compared to failure-free scenarios.
- $S_3$: (Byzantine; flipping the (R)ABA input) We let $f$ replicas exhibit Byzantine behavior in the (R)ABA phase. The strategy is to vote for a flipped value in (R)ABA. In other words, in each (R)ABA step, each Byzantine replica inputs $b$ when it should have input $b$. Doing so could potentially force each (R)ABA instance to experience more steps to terminate for all five protocols. For WaterBear and WaterBear-QS, the strategy would, at first glance, likely be more fruitful. For both protocols, a RABA instance may terminate in round 0, thanks to the biased validity property of RABA. The flipping strategy illustrated above may make them not decide in round 0 and force them to enter the second round of RABA, where the two protocols start to query the local coins.

We assess the performance for $f = 1$ (Figure 9a), $f = 2$ (Figure 9b), and $f = 5$ (Figure 9c).

Performance under crash failures ($S_1$). The throughput of all five protocols implemented under crash failures is higher than that in the failure-free case, except for $f = 1$, where all protocols share similar performance between the two scenarios. Our result echoes those of previous works. The reason is that under crash failures, the network bandwidth consumption is much lower (about 33% lower) than in the failure-free case. Note that when $f = 1$, the network bandwidth consumption is not as dominating as in other cases; hence, the performance difference among the protocols for $f = 1$ is less visible.

Performance under Byzantine failures ($S_2$ and $S_3$). The performance of all the protocols under Byzantine failures is slightly lower than that in the failure-free scenario and the crash failure scenario. WaterBear-QS-C and WaterBear-QS-Q suffer from slightly higher performance degradation under Byzantine failures compared to BEAT. The higher performance degradation is due to the use of local coins. As replicas start to use local coins in round $r > 0$, the RABA protocol may decide in more rounds. In all cases, WaterBear-QS-C and WaterBear-QS-Q remain more efficient than BEAT.

The difference between $S_2$ and $S_3$ is that faulty replicas broadcast 0 in $S_2$ but broadcast the flipped value in $S_3$. For BEAT, the performance in $S_3$ is higher for $f = 1$ and $f = 5$ but lower for $f = 2$; the difference in all the cases is not significant though. In contrast, for WaterBear-QS-C and WaterBear-QS-Q, the performance in $S_3$ is consistently lower than $S_2$, showing that the flipping strategy in $S_3$ works slightly better than that in $S_2$.

8 Conclusion

This paper designs and implements a family of practical asynchronous BFT protocols matching the security guarantees of their partially synchronous counterparts. Our experiments demonstrate that our protocols are efficient in both failure and failure-free scenarios. In particular, one of our protocols, WaterBear-QC-Q, consistently outperforms the state-of-the-art asynchronous protocols with much weaker security guarantees. We also build in different settings more efficient
ABA and RABA protocols that can be used to improve various high-level Byzantine-resilient protocols. Our work, for the first time, shows that the strongest security models and high performance can co-exist for asynchronous BFT.

Acknowledgment

This work was supported in part by the National Key R&D Program of China under 2022YFB2701700 and 2022YFB2701500, the National Natural Science Foundation of China under 62272043 and 92267203, Beijing Natural Science Foundation under M23015, Major Program of Shandong Provincial Natural Science Foundation for the Fundamental Research under ZR2022ZD03, and the Piloting Fundamental Research Program for the Integration of Scientific Research, Education and Industry of Qilu University of Technology (Shandong Academy of Sciences) 2022XD001.

References


[47] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. All you need is dag. In PODC, 2021.


A Bracha’s ABA

We present Bracha’s ABA [13]. The pseudocode is shown in Figure 10. Bracha’s ABA has three phases. In each phase, each replica broadcasts its value via a RBC instance, i.e., there are $n$ parallel RBC instances in each of the three phases. As the underlying RBC has $O(n^2)$ messages and 4 steps, Bracha’s ABA has $O(n^3)$ messages and 12 steps in each round.

In Bracha’s ABA, every replica maintains a set $vset$ containing valid values. In each phase, every replica only accepts messages that carry valid values. The valid values $vset$ must be congruent with the values each replica receives from the previous phase (or the last phase of the previous round). In the first phase of round 0, both 0 and 1 are considered valid. In the second and third phases, a value is added to $vset$ only if the replica receives the value from enough replicas.

In the first phase, every replica $p_i$ $r$-broadcasts a $pre-vote_r(iv_r)$ message (ln 08), where $iv_r$ is the input value of $p_i$ for round $r$.

In the second phase, $p_i$ waits for $n-f$ $pre-vote_r(.)$ messages such that for each $pre-vote_r(v), v \in vset$. There are two cases:

- Ln 10-13: If $p_i$ has received $n-f$ $pre-vote_r(v)$ for some $v \in \{0,1\}$, $p_i$ decides $v$ and sets both $vset$ and $iv_{r+1}$ as $v$. Replica $p_i$ continues for one more round and terminates the protocol (up to either ln 10 or ln 25 before $p_i$ decides some value again).
- Ln 14-15: Otherwise, $p_i$ sets $v$ as the majority value in the set of $pre-vote_r(.)$ messages it receives. The set $vset$ is not changed, i.e., $vset = \{0,1\}$.

In both cases, $p_i$ $r$-broadcasts a main-$vote_r(.)$ message (ln 16).
Figure 10: The Bracha’s ABA protocol [13]. The code for $p_i$.

In the third phase, every replica $p_i$ waits for $n - f$ valid main-vote($r$) messages (Line 17). There are two cases:

- Ln 18-19: If $p_i$ receives at least $n/2$ main-vote($r$), it sets vset as $\{v\}$.
- Ln 20-22: Otherwise, $p_i$ sets $v$ as $\ast$ and vset as $\{0, 1\}$.

In both cases, $p_i$ $r$-broadcasts a final-vote($r$) message (Line 23). Then every replica waits for $n – f$ valid final-vote($r$) messages (Line 24). Note that final-vote($r$) is considered valid only if vset = $\{0, 1\}$. There are three cases:

- Ln 25-28: If $p_i$ receives at least $2f + 1$ final-vote($r$), it decides $v$ and sets iv$_{r+1}$ as $v$. Replica $p_i$ continues for one more round (up to either Line 10 or Line 25) and terminates the protocol.
- Ln 29-31: If $p_i$ receives at least $f + 1$ final-vote($r$), it sets iv$_{r+1}$ as $v$ and vset as $\{v\}$.

Figure 11: Cubic-RABA. The code for $p_i$.

- Ln 32-34: Otherwise, $p_i$ uses the local coin value as iv$_{r+1}$ and vset as $\{0, 1\}$, i.e., $p_i$ accepts both 0 and 1 in the first phase of the following round.

B Cubic-RABA

The pseudocode of Cubic-RABA protocol is shown in Figure 11. Cubic-RABA is identical to Cubic-ABA, except for round 0 (the first round). We have made the following changes for round 0. First, both propose() and repropose() events are allowed. Upon the propose(v) event (Line 03), a replica $p_i$ executes the broadcast-vote(v) function and starts round 0. Upon the repropose(v) function (Line 06), $p_i$ executes broadcast-vote(v). Note that upon a repropose() event, $p_i$ must have already started the protocol and may even proceed to a round greater than 0. In this case, regardless of which round the replica is in, it executes the broadcast-vote(v) function and
We show that the changes we have made on top of Cubic-ABA can transform Cubic-ABA into a RABA protocol. The proof of Cubic-RABA is shown in Appendix H.

Analysis. The proof of Cubic-RABA is shown in Appendix H. We show that the changes we have made on top of Cubic-ABA can transform Cubic-ABA into a RABA protocol. The first change can ensure the biased termination property. In particular, it guarantees that if a quorum of correct replicas either directly propose 1 or propose 0 and later on repropose 1, the protocol will terminate. The second and third changes ensure the biased validity property. If \( f + 1 \) correct replicas propose 1, they will directly add 1 to \( bset_0 \), broadcast pre-vote\(_0(1) \), main-vote\(_0(1) \), and \( r\)-broadcast final-vote\(_0(1) \). Namely, no correct replica can receive \( n - f \) main-vote\(_0(0) \) or \( r\)-broadcast final-vote\(_0(0) \). Furthermore, no correct replica can receive \( n - f \) final-vote\(_0(0) \) or \( f + 1 \) final-vote\(_0(0) \). For the case where a correct replica uses the local coin to enter the next round, the coin value is also 1. Accordingly, Cubic-RABA achieves biased validity. Other properties of Cubic-RABA follow from Cubic-ABA, as we only modify round 0 of the protocol.

C Liveness Challenge of Building Quadratic-RABA

We illustrate an example in Figure 12 to show the subtle liveness challenge when converting Quadratic-ABA to Quadratic-RABA. Votes for 1 are marked in dashed lines.

![Figure 12: Subtle challenge when converting Quadratic-ABA to Quadratic-RABA. Votes for 1 are marked in dashed lines.](image)

Table 4: One-way latency (ms) between any two regions on Amazon EC2. The regions we use to evaluate the protocols are Ohio (us-east-2), Oregon (us-west-2), Singapore (ap-southeast-1), and Ireland (eu-west-1).

<table>
<thead>
<tr>
<th>Region</th>
<th>Ohio</th>
<th>Oregon</th>
<th>Singapore</th>
<th>Ireland</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ohio</td>
<td>-</td>
<td>24.5</td>
<td>55</td>
<td>39</td>
</tr>
<tr>
<td>Oregon</td>
<td>24.5</td>
<td>-</td>
<td>81</td>
<td>59</td>
</tr>
<tr>
<td>Singapore</td>
<td>55</td>
<td>81</td>
<td>-</td>
<td>90</td>
</tr>
<tr>
<td>Ireland</td>
<td>39</td>
<td>59</td>
<td>90</td>
<td>-</td>
</tr>
</tbody>
</table>

We show in Table 4 one-way latency between any two regions on Amazon EC2. As shown in the table, the lowest one-way latency (resp., the highest latency) is 24.5 ms for Ohio-Oregon (resp., 90 ms for Ireland-Singapore).

Let us explain why our latency result makes sense in such a setting. WaterBear-QC-Q, for example, has a RBC phase and a RABA phase. The RBC phase has 3 steps; the RABA phase in the optimistic mode just takes 1 step, as Quadratic RABA has a (coin-free) fast path that allows terminating in 1 step only. Namely, in the optimistic mode, WaterBear-QC-Q has 4 steps (i.e., about 4x average one-way latency).

The latency we reported for Water-QC-Q is 310 ms when \( f = 1 \) and \( n = 4 \), reasonably matching the one-way latency in Table 4: in practice, a replica delivers a transaction once receiving (fastest) \( 2f + 1 \) replicas instead of all \( 3f + 1 \) replicas.

Consider another example for WaterBear-QS-C. WaterBear-QS-C in its optimistic mode has 6 steps. The number of steps for WaterBear-QS-C justifies its latency of 570 ms for \( f = 1 \) and \( n = 4 \).

D Setup Detail and Additional Evaluation Results

D.1 Detailed Evaluation Setup

We show in Table 4 one-way latency between any two regions on Amazon EC2. As shown in the table, the lowest one-way latency (resp., the highest latency) is 24.5 ms for Ohio-Oregon (resp., 90 ms for Ireland-Singapore).

Let us explain why our latency result makes sense in such a setting. WaterBear-QC-Q, for example, has a RBC phase and a RABA phase. The RBC phase has 3 steps; the RABA phase in the optimistic mode just takes 1 step, as Quadratic RABA has a (coin-free) fast path that allows terminating in 1 step only. Namely, in the optimistic mode, WaterBear-QC-Q has 4 steps (i.e., about 4x average one-way latency).

The latency we reported for Water-QC-Q is 310 ms when \( f = 1 \) and \( n = 4 \), reasonably matching the one-way latency in Table 4: in practice, a replica delivers a transaction once receiving (fastest) \( 2f + 1 \) replicas instead of all \( 3f + 1 \) replicas.

Consider another example for WaterBear-QS-C. WaterBear-QS-C in its optimistic mode has 6 steps. The number of steps for WaterBear-QS-C justifies its latency of 570 ms for \( f = 1 \) and \( n = 4 \).

D.2 Additional Evaluation Results

We provide additional evaluation results, including the performance comparison with HotStuff [65], the performance on different types of VMs, the performance with different transaction sizes, and the memory and the CPU usage for the asynchronous BFT protocols implemented.

<table>
<thead>
<tr>
<th>Region</th>
<th>Ohio</th>
<th>Oregon</th>
<th>Singapore</th>
<th>Ireland</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ohio</td>
<td>-</td>
<td>24.5</td>
<td>55</td>
<td>39</td>
</tr>
<tr>
<td>Oregon</td>
<td>24.5</td>
<td>-</td>
<td>81</td>
<td>59</td>
</tr>
<tr>
<td>Singapore</td>
<td>55</td>
<td>81</td>
<td>-</td>
<td>90</td>
</tr>
<tr>
<td>Ireland</td>
<td>39</td>
<td>59</td>
<td>90</td>
<td>-</td>
</tr>
</tbody>
</table>
WaterBear-QS-Q achieves 1.65x throughput of HotStuff. The latency result is expected, because the message complexity of HotStuff is \( \mathcal{O} \left( \frac{n^3}{f} \right) \), whereas the figures) and \( \mathcal{O} \left( \frac{n}{f} \right) \) for WaterBear protocols using both prior protocols (HoneyBadger, BEAT, Dumbo, EPIC) that all achieve higher throughput. For instance, when \( b = 15,000 \) for \( f = 1 \), \( f = 2 \), and \( f = 5 \). The results are shown in Figure 14a. For all the protocols, the peak throughput on m5.xlarge instances is about 2× that on t2.medium.

**Performance with different transaction sizes.** We also report the throughput of the protocols by fixing \( b \) to 15,000 but using different sizes of transactions (100 bytes and 250 bytes), the results of which are shown in Figure 14b. For all five protocols, the performance using transaction size of 100 bytes is consistently higher, being at least twice as efficient as that with 250 bytes. The finding highlights the main bottleneck for the protocols for large transaction sizes is RBC.

**Memory and CPU usage.** We present in Figure 15 memory and CPU usage for \( n = 16 \) and varying batch sizes. The results are obtained by using the top Linux monitoring tool. For the memory usage, all protocols consume higher memory when \( b \) increases. This is expected, since replicas need to process more transactions as the batch size grows. Meanwhile, WaterBear-C and WaterBear-Q consistently consume slightly higher memory than the other protocols, because the RBC for both protocols requires more bandwidth. For the CPU usage, WaterBear-QS-C and WaterBear-QS-Q have lower CPU usage than other protocols, as these two protocols are PKC-free and use symmetric cryptography only.

**E Proof of Cubic-ABA**

We show that Cubic-ABA achieves validity, agreement, termination, and integrity.
Lemma 1. If all correct replicas propose \( iv_r = v \) in round \( r \), then any correct replica that enters round \( r + 1 \) sets \( iv_{r+1} \) as \( v \).

Proof. If all correct replicas propose \( v \) in round \( r \), every correct replica broadcasts \( \text{pre}-\text{vote}_r(\bar{v}) \). No correct replica will forward \( \text{pre-vote}_r(\bar{v}) \), as there are no more than \( f + 1 \) \( \text{pre-vote}_r(\bar{v}) \) messages. Hence, no correct replica will add \( \bar{v} \) to \( \text{bset}_r \). Furthermore, all correct replicas will eventually send \( \text{main-vote}_r(v) \) and \( r\text{-broadcast} \) \( \text{final-vote}_r(v) \). No correct replica accepts \( \text{final-vote}_r(\bar{v}) \) or \( \text{final-vote}_r(*) \), since they only have \( v \) in their \( \text{bset}_r \). Hence, any correct replica that enters round \( r + 1 \) sets \( iv_{r+1} \) as \( v \).

Note that the lemma above holds for the case where a correct replica decides \( v \) in round \( r \).

Lemma 2. If all correct replicas propose \( v \) in round \( r \), then for any \( r' > r \), any correct replica that enters round \( r' \) sets \( iv_r \) as \( v \).

Proof. The proof is by induction on the round number. The base case holds for \( r \) according to Lemma 1. For the induction step, we show that the lemma holds for round \( r' + 1 \). In other words, if all correct replicas propose \( iv_r = v \) in round \( r' \), then in round \( r' + 1 \), any correct replica sets \( iv_{r+1} \) as \( v \).

In round \( r' \), as no correct replica sends \( \text{pre-vote}_{r'}(\bar{v}) \), no correct replica can receive \( f + 1 \) \( \text{pre-vote}_{r'}(\bar{v}) \) messages. In other words, no correct replica will forward \( \text{pre-vote}_{r'}(\bar{v}) \). Meanwhile, no correct replica will accept \( \text{final-vote}_{r'}(\bar{v}) \) or \( \text{final-vote}_{r'}(*) \) since correct replicas only have \( v \) in their \( \text{bset}_{r'} \). Furthermore, every correct replica will \( r\text{-broadcast} \) \( \text{final-vote}_r(v) \). Therefore, any correct replica that enters round \( r' + 1 \) sets \( iv_{r+1} \) as \( v \).

Theorem 3 (Validity). If all correct replicas propose \( v \), then any correct replica that terminates decides \( v \).

Proof. We assume that a correct replica \( p_1 \) terminates and decides \( \bar{v} \) and prove the correctness by contradiction.

If \( p_1 \) terminates and decides \( \bar{v} \) in round \( 0 \), it will enter round \( 1 \) with \( iv_1 = \bar{v} \). This is a contradiction with Lemma 1, as if all correct replicas propose \( v \), any correct replica that enters round \( 1 \) sets \( iv_1 \) as \( v \). If \( p_1 \) terminates and decides \( \bar{v} \) in round \( r > 0 \), it \( r\text{-delivers} n - f \) \text{final-vote}_r(\bar{v}) \). Similarly, it has put \( \bar{v} \) in its \( \text{bset}_r \). Therefore, at least one correct replicas has set \( iv_r = \bar{v} \) and broadcast \( \text{pre-vote}_r(\bar{v}) \). This is a contradiction with Lemma 2 since any correct replica that enters round \( r \) sets \( iv_r \) as \( v \). This completes the proof of the theorem.

Lemma 4. If a correct replica \( p_i \) decides \( v \) in round \( r \), any correct replica that enters round \( r + 1 \) sets \( iv_{r+1} \) as \( v \).

Proof. If \( p_i \) decides \( v \) in round \( r \), it \( r\text{-delivers} n - f \) \text{final-vote}_r(\bar{v}) \). In other words, at least \( f + 1 \) correct replicas \( r\text{-broadcast} \text{final-vote}_r(\bar{v}) \). We assume that a correct replica \( p_k \) enters round \( r + 1 \) using value \( iv_{r+1} = \bar{v} \) and prove the lemma by contradiction. If \( p_k \) sets \( iv_{r+1} \) as \( \bar{v} \), there are three conditions: A) \( p_k \) \( r\text{-delivers} \) at least \( n - f \) \text{final-vote}_r(\bar{v}) \); B) \( p_k \)
delivers \( f + 1 \) final-vote, (\( \bar{v} \)); C) none of the conditions holds. In other words, \( p_k \) has received fewer than \( f + 1 \) final-vote, (\( \bar{v} \)) and fewer than \( f + 1 \) final-vote, (\( \bar{v} \)). We now show that none of the three conditions is possible.

Condition A): Replica \( p_k \) r-delivers \( n - f \) final-vote, (\( \bar{v} \)). We already know that at least \( n - f \) replicas r-broadcast final-vote, (\( \bar{v} \)). Therefore, at least one correct replica r-broadcasts both final-vote, (\( \bar{v} \)) and final-vote, (\( \bar{v} \)), a contradiction.

Condition B): Replica \( p_k \) r-delivers \( f + 1 \) final-vote, (\( \bar{v} \)). We already know that \( p_i \) r-delivers \( n - f \) final-vote, (\( v \)). Therefore, at least one replica (correct or Byzantine) r-broadcast both final-vote, (\( \bar{v} \)) and final-vote, (\( v \)) such that \( p_i \) r-delivers final-vote, (\( \bar{v} \)) and \( p_i \) r-delivers final-vote, (\( v \)). This is a violation of the agreement property or RBC.

Condition C): Replica \( p_k \) r-delivers \( n - f \) final-vote, (\( i \)) messages (let the set of replicas be \( S_1 \)). Among the messages from \( S_1 \), fewer than \( f + 1 \) are final-vote, (\( \bar{v} \)) and fewer than \( f + 1 \) are final-vote, (\( v \)). Other messages can only be final-vote, (\( * \)). We already know that \( p_i \) r-delivers \( n - f \) final-vote, (\( v \)) (let the set of replicas be \( S_2 \)). \( S_1 \) and \( S_2 \) have at least \( n - 2f \geq 1 \) replicas in common. Therefore, at least one replica r-broadcasts both \( v \) and * (or \( \bar{v} \)) such that \( p_i \) r-delivers final-vote, (\( v \)) and \( p_k \) has r-delivers final-vote, (\( * \)) (or final-vote, (\( \bar{v} \))), a violation of agreement property of RBC.

**Theorem 5** (Agreement). If a correct replica decides \( v \), then any correct replica that terminates decides \( v \).

**Proof.** We assume that a correct replica \( p_i \) decides \( v \) and another correct replica \( p_j \) decides \( \bar{v} \) and prove the theorem by contradiction. There are two cases: 1) \( p_i \) and \( p_j \) decide in the same round; 2) \( p_i \) and \( p_j \) decide in different rounds.

We first prove case 1). If replica \( p_i \) decides \( v \) in round \( r \), it r-delivers \( n - f \) final-vote, (\( \bar{v} \)). If \( p_j \) decides \( \bar{v} \), it r-delivers \( n - f \) final-vote, (\( \bar{v} \)). The two sets of \( n - f \) replicas have at least \( f + 1 \) replicas in common. Among the \( f + 1 \) replicas, at least one is correct. Therefore, at least one correct replica must have r-broadcast both final-vote, (\( v \)) and final-vote, (\( \bar{v} \)), a contradiction.

We now prove case 2) by assuming that \( p_i \) decides value \( v \) in round \( r \) and \( p_j \) decides \( \bar{v} \) in round \( r' \) where \( r' > r \).

According to Lemma 4, any correct replica enters round \( r + 1 \) sets \( iv_{r+1} \) as \( v \). Furthermore, according to Lemma 2, for any round \( r'' \geq r + 1 \), any correct replica sets round \( r'' \) sets \( iv_{r''} \) as \( v \). If replica \( p_j \) decides value \( \bar{v} \) in round \( r' \), at least one correct replica has set \( iv_{r''} \) as \( \bar{v} \) and sent pre-vote, (\( \bar{v} \)), a contradiction with Lemma 2.

**Lemma 6.** Let \( v_1 \in \{0, 1\} \) and \( v_2 \in \{0, 1\} \). If a correct replica \( p_i \) r-delivers \( f + 1 \) final-vote, (\( v_1 \)) and enters round \( r + 1 \), another correct replica \( p_j \) r-delivers \( f + 1 \) final-vote, (\( v_2 \)) and enters round \( r + 1 \), then it holds that \( v_1 = v_2 \).

**Proof.** If \( p_i \) r-delivers \( f + 1 \) final-vote, (\( v_1 \)), at least one correct replica r-broadcasts final-vote, (\( v_1 \)). According to the protocol, the correct replica has received \( n - f \) main-vote, (\( v_1 \)). Therefore, for any other correct replicas, among the \( n - f \) main-vote, (\( v_1 \)) messages, at least one must be main-vote, (\( v_1 \)). They either receive \( n - f \) main-vote, (\( v_1 \)) and r-broadcast final-vote, (\( v_1 \)), or receive both main-vote, (\( v_1 \)) and main-vote, (\( v_1 \)) and r-broadcast final-vote, (\( v_1 \)). No correct replica will r-broadcast final-vote, (\( v_1 \)). For replica \( p_j \), if it r-delivers \( f + 1 \) final-vote, (\( v_2 \)), at least one correct replica r-broadcasts final-vote, (\( v_2 \)). Therefore, it must hold that \( v_1 = v_2 \).

**Theorem 7.** (Termination). Every correct replica eventually decides some value.

**Proof.** The proof consists of two parts. First, in each round \( r \), correct replicas will enter the next round. Second, the value \( iv_r \) used by any correct replica cannot be manipulated by the adversary.

We first show that in round \( r \), correct replicas will enter the next round. In each round, every replica sets \( iv_r \) as either 0 or 1 in Cubic-ABA. Accordingly, at least \( f + 1 \) correct replicas have the same \( iv_r = v \). Therefore, all correct replicas will eventually receive \( 2f + 1 \) pre-vote, (\( v \)) for some \( v \) and send main-vote, (\( v \)) message. Correct replicas will have at least \( v \) in their \( hset \) and r-broadcast either final-vote, (\( v \)) for some \( v \) or final-vote, (\( * \)). Similarly, any correct replica will eventually r-deliver \( n - f \) final-vote, (\( i \)) messages and enter the next round.

We then show that if a correct replica \( p_i \) does not decide in round \( r \), the value \( iv_{r+1} = v \) cannot be manipulated by a malicious network scheduler such that correct replicas always enter the next round with inconsistent values. If \( p_i \) does not decide in round \( r \), there are two conditions: A) \( p_i \) r-delivers \( f + 1 \) final-vote, (\( v \)); B) \( p_i \) r-delivers \( n - f \) final-vote, (\( i \)) messages. In the final-vote, (\( i \)) messages, fewer than \( f + 1 \) are final-vote, (\( v \)) and fewer than \( f + 1 \) are final-vote, (\( v \)). For condition B, a correct replica enters the next round with its local coin. The value of the local coin is independent with the value chosen by any correct replica. We now prove that the value \( v \) in condition A cannot be manipulated.

According to Lemma 6, if a correct replica receives \( f + 1 \) final-vote, (\( v_1 \)) and another correct replica receives \( f + 1 \) final-vote, (\( v_2 \)), then it holds that \( v_1 = v_2 \). If correct replicas use local coins to enter the next round, with a probability of \( \frac{1}{2f+1} \), replicas will enter the next round with the same value. The protocol will reach a state where agreement can be reached in \( 2n/f \) expected rounds. After that, it takes another round for each replica to terminate, i.e., the protocol terminates in \( 2n/f + 1 \) expected rounds.

**Theorem 8.** (Integrity). No correct replica decides twice.

**Proof.** According to the protocol, after a correct replica decides some value, it participates in one more round of the protocol. However, it terminates the protocol after it r-broadcasts a final-vote, (\( i \)) message. Thus, the replica does not decide
again in the following round. This completes the proof of the theorem.

\section*{F Proof of Quadratic-ABA}

We show that Quadratic-ABA achieves validity, agreement, termination, and integrity.

\textbf{Lemma 9.} If all correct replicas propose \( iv_r = v \) in round \( r \), then any correct replica that enters round \( r + 1 \) sets \( iv_{r+1} \) as \( v \).

\textit{Proof.} If all correct replicas propose \( iv_r = v \) in round \( r \), every correct replica broadcasts \( \text{pre-vote}_r(v) \). No correct replica will receive more than \( f + 1 \) \( \text{pre-vote}_r(v) \) messages. Hence, no correct replica will add \( v \) to \( \text{bset}_r \). Furthermore, all correct replicas will eventually send \( \text{vote}_r(v) \) and receive \( n - f \) \( \text{vote}_r(v) \). As no correct replica ever has \( v \) in \( \text{bset}_r \), all correct replica will not accept \( \text{vote}_r(v) \). Therefore, all correct replicas will send \( \text{main-vote}_r(v) \). No correct replica will accept \( \text{main-vote}_r(v) \) or \( \text{main-vote}_r(*) \) as \( \bar{v} \notin \text{bset}_r \), and it cannot receive more than \( f + 1 \) \( \text{vote}_r(v) \). Accordingly, every correct replica will send \( \text{final-vote}_r(v) \) and receive \( n - f \) \( \text{final-vote}_r(v) \). No correct replica accepts \( \text{final-vote}_r(\bar{v}) \) as they only have \( v \) in their \( \text{bset}_r \). Hence, any correct replica that enters round \( r + 1 \) sets \( iv_{r+1} \) as \( v \). \hfill \blacksquare

Note that the lemma above holds for the case where a correct replica decides \( v \) in round \( r \).

\textbf{Lemma 10.} If all correct replicas propose \( iv_r = v \) in round \( r \), then for any \( r' > r \), any correct replica that enters round \( r' \) sets \( iv_{r'} = v \).

\textit{Proof.} The proof is by induction on the round number. The base case holds for \( r \) according to Lemma 9. For the induction step, we show that the lemma holds for round \( r' + 1 \). In other words, if all correct replicas propose \( iv_{r'} = v \) in round \( r' \), then in round \( r' + 1 \), any correct replica sets \( iv_{r'+1} \) as \( v \).

In round \( r' \), as no correct replica sends \( \text{pre-vote}_{r'}(\bar{v}) \), no correct replica can receive \( f + 1 \) \( \text{pre-vote}_{r'}(\bar{v}) \) messages. In other words, no correct replica will put \( \bar{v} \) to \( \text{bset}_{r'} \). Therefore, all correct replicas will send \( \text{vote}_{r'}(v) \) and no correct replicas will receive \( f + 1 \) \( \text{vote}_{r'}(v) \). Accordingly, all correct replicas will send \( \text{main-vote}_{r'}(v) \) and will not accept \( \text{main-vote}_{r'}(\bar{v}) \). Any correct replica then only sends \( \text{final-vote}_{r'}(v) \). Meanwhile, no correct replica will accept \( \text{final-vote}_{r'}(v) \) or \( \text{final-vote}_{r'}(*) \) since correct replicas only have \( v \) in their \( \text{bset}_{r'} \) and no correct replica can receive \( f + 1 \) \( \text{main-vote}_{r'}(v) \). Furthermore, every correct replica will receive \( n - f \) \( \text{final-vote}_{r'}(v) \). It is now clear that any correct replica that enters round \( r' + 1 \) sets \( iv_{r'+1} \) as \( v \). \hfill \blacksquare

\textbf{Lemma 11.} If a correct replica \( p_i \) sends final-vote\(_r(v)\), at least one correct replica has proposed \( iv_r = \bar{v} \) and broadcast pre-vote\(_r(\bar{v})\).

\textit{Proof.} If \( p_i \) sends final-vote\(_r(v)\), it must have received \( n - f \) main-vote\(_0(\bar{v})\). Among the replicas that sent main-vote\(_0(\bar{v})\), at least \( f + 1 \) are correct. The correct replicas must have sent vote\(_0(\bar{v})\) and put \( \bar{v} \) to \( \text{bset}_r \). Each replica puts \( \bar{v} \) to \( \text{bset}_r \) only if it receives \( n - f \) pre-vote\(_r(\bar{v})\). Therefore, at least one correct replica has proposed \( iv_r = \bar{v} \) and broadcast pre-vote\(_r(\bar{v})\). \hfill \blacksquare

\textbf{Theorem 12} (Validity). If all correct replicas propose \( v \), then any correct replica that terminates decides \( v \).

\textit{Proof.} We assume that a correct replica \( p_i \) terminates and decides \( v \) and prove the correctness by contradiction.

If \( p_i \) terminates and decides \( v \) in round \( 0 \), it will enter round \( 1 \) with \( iv_1 = v \). This is a contradiction with Lemma 9. If \( p_i \) terminates and decides \( v \) in round \( r > 0 \), it receives \( n - f \) final-vote\(_r(\bar{v})\). Among the replicas that sent final-vote\(_r(\bar{v})\), at least \( f + 1 \) are correct. According to Lemma 11, at least one correct replica has broadcast pre-vote\(_r(\bar{v})\). This is a contradiction with Lemma 10 since any correct replica that enters round \( r \) sets \( iv_r \) as \( v \). \hfill \blacksquare

\textbf{Lemma 13.} If a correct replica \( p_i \) sends main-vote\(_r(v)\), any correct replica \( p_{ij} \) only sends main-vote\(_r(v)\) or main-vote\(_r(*)\).

\textit{Proof.} If \( p_i \) sends main-vote\(_r(v)\), it has received \( n - f \) vote\(_r(v)\). We assume that \( p_{ij} \) sends main-vote\(_r(v)\) and prove the lemma by contradiction. If \( p_{ij} \) sends main-vote\(_r(v)\), it has received \( n - f \) vote\(_r(v)\). According to the protocol, every correct replica only sends vote\(_r(*)\) message once and each replica only sends either vote\(_r(v)\) or vote\(_r(\bar{v})\). Therefore, at least one correct replica has sent vote\(_r(v)\) to \( p_i \) and sent vote\(_r(\bar{v})\) to \( p_{ij} \), a contradiction. \hfill \blacksquare

\textbf{Lemma 14.} If a correct replica \( p_i \) sends final-vote\(_r(v)\), any correct replica \( p_{ij} \) only sends final-vote\(_r(v)\) or final-vote\(_r(*)\).

\textit{Proof.} If \( p_i \) sends final-vote\(_r(v)\), it has received \( n - f \) main-vote\(_r(v)\). We assume that \( p_{ij} \) sends final-vote\(_r(v)\) and prove the lemma by contradiction. If \( p_{ij} \) sends final-vote\(_r(v)\), it has received \( n - f \) main-vote\(_r(v)\). According to the protocol, every correct replica only sends main-vote\(_r(*)\) message once. Therefore, at least one correct replica has sent main-vote\(_r(v)\) to \( p_i \) and sent main-vote\(_r(\bar{v})\) to \( p_{ij} \), a contradiction. \hfill \blacksquare

\textbf{Lemma 15.} If a correct replica \( p_i \) decides \( v \) in round \( r \), any correct replica that enters round \( r + 1 \) sets \( iv_{r+1} \) as \( v \).

\textit{Proof.} If \( p_i \) decides \( v \) in round \( r \), it receives \( n - f \) final-vote\(_r(v)\). In other words, at least \( f + 1 \) correct replicas have broadcast final-vote\(_r(v)\). We assume that a correct replica \( p_k \) enters round \( r + 1 \) sets \( iv_{r+1} = \bar{v} \) and prove the lemma by contradiction. If \( p_k \) sets \( iv_{r+1} \) as \( \bar{v} \), there are three conditions: A) \( p_k \) receives at least \( n - f \) final-vote\(_r(\bar{v})\); B) \( p_k \) only receives final-vote\(_r(\bar{v})\) and final-vote\(_r(*)\); C) none of the above holds. In other words, \( p_k \) receives only final-vote\(_r(*)\)
or receives both final-vote, (v) and final-vote, (v). We now show that none of the three conditions is possible.

Condition A): Replica pk receives n − f final-vote, (v).
We already know that at least n − f replicas have sent final-vote, (v) as pk receives n − f final-vote, (v). Therefore, at least one correct replica has sent both final-vote, (v) and final-vote, (v), a contradiction.

Condition B): Replica pk receives n − f final-vote, (v) and final-vote, (v) and has not received final-vote, (v). We already know that pk receives n − f final-vote, (v). Therefore, at least one correct replica has sent final-vote, (v) to pk and either final-vote, (v) or final-vote, (v) to pk, a contradiction.

Condition C): Replica pk receives only final-vote, (v) or receives both final-vote, (v) and final-vote, (v). We know that pk receives n − f final-vote, (v). Therefore, at least f + 1 correct replicas have sent final-vote, (v). If pk receives n − f final-vote, (v) messages, at least one of them must be final-vote, (v). In this case, if pk enters round r + 1 with iv, +1 as v, pk must have received at least one final-vote, (v), as if pk only receives final-vote, (v) and final-vote, (v), it will set iv, +1 as v. If pk accepts final-vote, (v), it has received f + 1 main-vote, (v), among which at least one is sent by a correct replica. If pk accepts final-vote, (v), it has received f + 1 main-vote, (v), among which at least one is sent by a correct replica. This is a contradiction with Lemma 13.

Theorem 16 (Agreement). If a correct replica decides v, then any correct replica that terminates decides v.

Proof. We assume that a correct replica pk decides v and another correct replica pj decides v and prove the theorem by contradiction. There are two cases: 1) pk and pj decide in the same round r; 2) pk and pj decide in different rounds.

We first prove case 1). If replica pk decides v in round r, it receives n − f final-vote, (v). If pj decides v, it receives n − f final-vote, (v). The two sets of n − f replicas have at least f + 1 replicas in common. Among the f + 1 replicas, at least one is correct. Therefore, at least one correct replica must have sent both final-vote, (v) and final-vote, (v), a contradiction.

We now prove case 2) by assuming that pk decides value v in round r and pj decides v in round r′ where r′ > r.

According to Lemma 15, if pk decides v, any correct replica enters round r + 1 sets iv, +1 as v. Furthermore, according to Lemma 10, for any round r′′ ≥ r + 1, any correct replica that enters round r′′ sets iv, r′′ as v. If replica pk decides value v in round r′, it has received n − f final-vote, (v) so at least f + 1 correct replicas have sent final-vote, (v).

According to Lemma 11, at least one correct replica has set iv, r′ as v and sent pre-vote, (v), a contradiction with Lemma 10.

Lemma 17. If a correct replica pk sends vote, (v) for v ∈ {0, 1}, any correct replica eventually accepts vote, (v).

Proof. If pk sends vote, (v) message, it has received n − f pre-vote, (v), among which at least f + 1 are sent by correct replicas. Accordingly to the protocol, any correct replica that has not sent pre-vote, (v) will also send pre-vote, (v) upon receiving f + 1 pre-vote, (v). Therefore, every correct replica eventually sends pre-vote, (v), receives n − f pre-vote, (v), and then adds v to bset,. Hence, every correct replica eventually accepts vote, (v).

Lemma 18. If a correct replica pk broadcasts a main-vote, (v) or a main-vote, (v) message given that v ∈ {0, 1}, any correct replica accepts the main-vote, (v) message from pk.

Proof. If pk sends a main-vote, (v) message, it has received and accepted n − f vote, (v), among which at least f + 1 are sent by correct replicas. Accordingly to Lemma 17, every correct replica eventually accepts vote, (v). After each correct replica receives f + 1 vote, (v), it accepts main-vote, (v).

If pk sends a main-vote, (v) message, it must have received and accepted both vote, (v) and vote, (v), or it has received at least one vote, (v). In any of the cases, pk has put both 0 and 1 to bset,. If pk puts v to bset, it has received 2f + 1 pre-vote, (v), among which at least f + 1 are sent by correct replicas. Then any correct replica eventually receives f + 1 pre-vote, (v) and sends pre-vote, (v). Every correct replica eventually receives n − f pre-vote, (v) and adds v to bset,. Therefore, every correct replica eventually accepts main-vote, (v).

Lemma 19. If a correct replica pk broadcasts a final-vote, (v) or a final-vote, (v) message given that v ∈ {0, 1}, any correct replica accepts the final-vote, (v) message.

Proof. The lemma can be proved similarly as in Lemma 18.

Lemma 20. Let v1 ∈ {0, 1} and v2 ∈ {0, 1}. If a correct replica pk receives only n − f final-vote, (v1) and final-vote, (v1) messages, another correct replica pk only receives n − f final-vote, (v2) and final-vote, (v2) messages, v1 = v2.

Proof. If pk accepts final-vote, (v1), it has previously received f + 1 main-vote, (v1), among which at least one is sent by a correct replica. If pk accepts final-vote, (v1), it has previously received f + 1 main-vote, (v2), among which at least one is sent by a correct replica. According to Lemma 13, it holds that v1 = v2.

Theorem 21 (Termination). Every correct replica eventually decides some value.

Proof. The proof consists of two parts. First, in each round r, correct replicas will enter the next round. Second, the value iv, used by any correct replica cannot be manipulated by the adversary.

We first show that in round r, correct replicas will enter the next round. In each round, every replica sets iv, to either 0 or 1 in Quadratic-ABA. Accordingly, at least f + 1 cor-
correct replicas have the same $iv_r = v$. All correct replicas will eventually receive $2f + 1$ pre-vote$_r(v)$ for some $v$ and send a vote$_r(\cdot)$ message. Correct replicas will send either vote$_r(0)$ or vote$_r(1)$ and receive at least $n-f$ main-vote$_r(\cdot)$ messages. For any correct replica, if it sends vote$_r(v)$ for $v \in \{0, 1\}$, any correct replica will eventually accept vote$_r(v)$, according to Lemma 17. All correct replicas will then send either main-vote$_r(v)$ for $v \in \{0, 1\}$ or main-vote$_r(\ast)$. According to Lemma 18, every correct replica eventually accepts any main-vote$_r(\cdot)$ message sent by a correct replica. Then every correct replica either sends final-vote$_r(v)$ or final-vote$_r(\ast)$. According to Lemma 19, every correct replica accepts any final-vote$_r(\cdot)$ message from a correct replica. Therefore, any correct replica will eventually receive $n-f$ final-vote$_r(\cdot)$ messages and enter the next round.

We then show that if a correct replica $p_i$ does not decide in round $r$, the value $iv_{r+1} \equiv v$ cannot be manipulated by a malicious network scheduler such that correct replicas always enter the next round with inconsistent values. If $p_i$ does not decide in round $r$, there are two conditions: A) $p_i$ receives $n-f$ final-vote$_r(v)$ and final-vote$_r(\ast)$; B) $p_i$ receives both final-vote$_r(v)$ and final-vote$_r(\overline{v})$ messages, or receives $n-f$ final-vote$_r(\ast)$. For condition B, a correct replica enters the next round with its local coin. The value of the local coin is independent with the value chosen by any correct replica. We now prove that the value $v$ in condition A cannot be manipulated.

According to Lemma 20, if $p_i$ receives $n-f$ final-vote$_r(v_1)$ and final-vote$_r(\ast)$ and $p_j$ receives $n-f$ final-vote$_r(v_1)$ and final-vote$_r(\ast)$, $v_1 = v_2$. In other words, the value $v$ used by any correct replica cannot be manipulated by the network scheduler.

If correct replicas use local coins to enter the next round, with a probability of $\frac{1}{2^{2f+1}}$, replicas will enter the next round with the same value. Replicas will reach a state where agreement with a probability of $\frac{1}{2}$.

Theorem 22 (Integrity). No correct replica decides twice.

Proof. According to the protocol, after a correct replica decides some value, it participates in one more round of the protocol. However, it terminates the protocol after it receives a final-vote$_r(\cdot)$ message. Hence, the replica does not decide again in the following round.

G CC-ABA

Both Cubic-ABA and Quadratic-ABA can be transformed to ABA from weak common coins [20, 56] and perfect common coins. Here by weak common coins, we mean that all correct replicas output 0 with probability $1/d$ and output 1 with probability $1/d$ where $d$ is a constant and $d \geq 2$, and the probability that correct replicas obtain different values is $(d-2)/d$. By perfect common coins, we mean that all correct replicas always output the same random coin. Note perfect coins are a special case of weak coins (by setting $d = 2$).

As Quadratic-ABA is more efficient, we here focus on Quadratic-ABA. Our main result is that by replacing local coins of Quadratic-ABA with weak (or perfect) common coins, we immediately obtain CC-ABA terminating in $O(1)$ time. As shown in Figure 16, we only need to replace one line of code in Figure 3 to obtain CC-ABA.

Figure 16: The CC-ABA protocol from weak coin or common coin.

CC-ABA reduces the expected number of steps of prior constructions, as shown in Table 5 and Table 6. Note that ABA is the major bottleneck in asynchronous BFT protocols as reported in [33, 44, 45]. The improvement is significant and has practical implications, as the recent work of PACE has shown that even a single step improvement can lead to a drastic performance improvement (for instance, easily 2x) in BFT protocols [66].

<table>
<thead>
<tr>
<th>ABA (weak common coins)</th>
<th>steps/round</th>
<th>rounds</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMR15 [56, 2nd alg]</td>
<td>9 to 13</td>
<td>$d + 1$</td>
</tr>
<tr>
<td>Cobalt [52]</td>
<td>3 or 4</td>
<td>4</td>
</tr>
<tr>
<td>Crain [56, 1st alg]</td>
<td>5 to 7</td>
<td>yes</td>
</tr>
<tr>
<td>Crain [56, 2nd alg]</td>
<td>2 or 3</td>
<td>yes</td>
</tr>
<tr>
<td>Pillar [66]</td>
<td>2 or 3</td>
<td>no</td>
</tr>
<tr>
<td>CC-ABA (this work)</td>
<td>4 or 5</td>
<td>$d + 1$</td>
</tr>
</tbody>
</table>

Table 5: ABA protocols using weak common coins. Rounds denote the expected number of rounds. The total number of steps is a product of steps/round and rounds.

<table>
<thead>
<tr>
<th>ABA (common coins)</th>
<th>steps/round</th>
<th>rounds</th>
<th>good-case-coin-free</th>
</tr>
</thead>
<tbody>
<tr>
<td>MMR15 [56, 2nd alg]</td>
<td>9 to 13</td>
<td>3</td>
<td>yes</td>
</tr>
<tr>
<td>Cobalt [52]</td>
<td>3 or 4</td>
<td>4</td>
<td>no</td>
</tr>
<tr>
<td>Crain [56, 1st alg]</td>
<td>5 to 7</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Crain [56, 2nd alg]</td>
<td>2 or 3</td>
<td>yes</td>
<td></td>
</tr>
<tr>
<td>Pillar [66]</td>
<td>2 or 3</td>
<td>4</td>
<td>no</td>
</tr>
<tr>
<td>CC-ABA (this work)</td>
<td>4 or 5</td>
<td>3</td>
<td>yes</td>
</tr>
</tbody>
</table>

Table 6: ABA protocols using perfect common coins. 1The second algorithm of Crain relies high threshold common coins and is less efficient than Pillar. Compared to Pillar, CC-ABA has the good-case-coin-free property that is vital for the asynchronous distributed key generation protocol [30].

G.1 Proof of CC-ABA

We prove the correctness of CC-ABA that simply replaces the local coins of Quadratic-ABA with weak common coins or perfect common coins. We comment that our proof presented in the subsection applies to both cases.
Theorem 23. CC-ABA achieves validity, agreement, and integrity.

Proof. The three properties follow that of Quadratic-ABA.

Lemma 24. If a correct replica receives and accepts both final-vote, (v_1) and final-vote, (v_2) such that v_1, v_2 ∈ {0, 1}, v_1 = v_2.

Proof. If a correct replica accepts final-vote, (v_1), it has previously received at least f + 1 main-vote, (v_1). If the replica accepts final-vote, (v_2), it has previously received at least f + 1 main-vote, (v_2). Therefore, at least one correct replica has sent main-vote, (v_1) and at least one correct replica has sent main-vote, (v_2). According to Lemma 13, if a correct replica sends main-vote, (v_1), any correct replicas will only send main-vote, (v_1) or main-vote, (∗). Therefore, we conclude that v_1 = v_2.

Theorem 25 (Termination). Every correct replica eventually decides some value.

Proof. The proof consists of two parts. First, in each round r, correct replicas will enter the next round. Second, the value iv, used by any correct replica cannot be manipulated by the adversary.

We first show that in round r, correct replicas will enter the next round. In each round, every replica sets iv, as either 0 or 1. Thus, at least f + 1 correct replicas have the same iv, v. All correct replicas will eventually receive 2f + 1 pre-vote, (v) for some v and send vote, (v) message. Correct replicas will send either vote, (0) or vote, (1) and receive at least n − f main-vote, (v) messages. For any correct replica, according to Lemma 17, if it sends vote, (v) such that v ∈ {0, 1}, any correct replica will eventually accept vote, (v). All correct replicas will then send either main-vote, (v) (v ∈ {0, 1}) or main-vote, (∗). According to Lemma 18, every correct replica eventually accepts any main-vote, (v) message sent by a correct replica. Then every correct replica either sends final-vote, (v) or final-vote, (∗). Due to Lemma 19, every correct replica accepts any final-vote, (v) message from a correct replica. Therefore, any correct replica will eventually receives n − f final-vote, (v) messages and enter the next round.

We then show that if a correct replica p_i does not decide in round r, the value iv, +1 = v cannot be manipulated by an malicious network scheduler such that correct replicas always enter the next round with inconsistent values. If p_i does not decide in round r, there are two conditions: A) p_i receives n − f final-vote, (v) messages with only final-vote, (∗) and final-vote, (v) messages; B) p_i receives both final-vote, (v) and final-vote, (∗) messages, or receives n − f final-vote, (∗).

If condition A applies to at least two correct replicas, then according to Lemma 20, if p_i receives n − f final-vote, (v_1) and final-vote, (∗) and p_j receives n − f final-vote, (v_1) and final-vote, (∗), v_1 = v_2. In other words, the value v used by any correct replica cannot be manipulated by an adversary.

If condition B applies to at least two correct replicas, the correct replicas enter the next round with the weak common coin. With a probability of 2/d, all correct replicas will have the same iv, +1 value. This value cannot be manipulated by an adversary.

We now show that if condition A applies to a correct replica p_i and condition B applies to a correct replica p_j, the values cannot be manipulated by an adversary. If p_j sets iv, +1 as the weak common coin value, it has either received n − f final-vote, (v) or both final-vote, (v) and final-vote, (∗). According to Lemma 24, the latter case is impossible. Therefore, p_j receives n − f final-vote, (v). Hence, at least f + 1 correct replicas have sent final-vote, (v). From Lemma 13, the correct replicas have previously sent either main-vote, (v) or main-vote, (∗) for some v ∈ {0, 1}. No correct replica will send main-vote, (v). If condition A applies to p_j and p_j sets iv, +1 as v_1 (v_1 ∈ {0, 1}), p_j has received at least f + 1 main-vote, (v_1). Since at least one correct replica has sent main-vote, (v_1) and no correct replica will send main-vote, (v), this value v_1 can only be v. Namely, the value iv, +1 cannot be manipulated by an adversary.

CC-ABA uses weak or perfect common coins. If correct replicas begin the protocol with different input values, replicas will reach a state where decisions can be made in expected 1 − Σ_{r=1} d (1 − 1/d)^r−1 = d rounds. Therefore, any correct replica that enters round r vote, v cannot be manipulated by an adversary.

H Proof of Cubic-RABA

We now show that Cubic-RABA achieves validity, unanimous termination, agreement, biased validity, biased termination, and integrity.

Lemma 26. If all correct replicas propose v in round 0 and never repropose v̅, then any correct replica enters the round 1 sets iv, v as v.

Proof. In round 0, all replicas send pre-vote, (v). No correct replica will receive f + 1 pre-vote, (v) and send pre-vote, (v). Similarly, all correct replicas will send main-vote, (v) and will never accept main-vote, (v). All correct replicas will re-broadcast final-vote, (v) and will never accept final-vote, (v). Therefore, any correct replica that enters round 1 sets iv, v as v.

Theorem 27 (Validity). If all correct replicas propose v and never repropose v̅, then any correct replica that terminates decides v.

Proof. We assume that a correct replica p_i terminates and decides v and prove the correctness by contradiction. If p_i
terminates and decides \( v \) in round 0, correctness follows from Lemma 26. We now prove the case where \( p_i \) decides in round \( r > 0 \).

Since Cubic-RABA follows Cubic-ABA starting from round 1, Lemma 2 holds for \( r > 0 \). If \( p_i \) terminates and decides \( v \) in round \( r > 0 \), it \( r\)-delivers \( n - f \) final-vote(_\( v \)). Additionally, \( p_i \) has added \( v \) to its \( \text{bsset}_i \). Therefore, at least one correct replica has set \( iv \) as \( v \) and broadcast pre-vote(_\( v \)). This is a contradiction with Lemma 2 since any correct replica that enters round \( r \) sets \( iv \) as \( v \). This completes the proof of the theorem.

\[ \text{Theorem 28 (Unanimous termination). If all correct replicas propose \( v \) and never repropose \( \bar{v} \), then all correct replicas eventually terminate.} \]

**Proof.** If all correct replicas propose \( v \) and never repropose \( \bar{v} \), all correct replicas only send pre-vote(_\( v \)). No correct replica will add \( \bar{v} \) to \( \text{bsset}_0 \). Furthermore, no correct replica will accept main-vote(_0\( v \)) or final-vote(_\( v \)). Eventually all correct replicas will receive \( 2f + 1 \) pre-vote(_\( v \)), add \( v \) to \( \text{bsset}_i \), and broadcast main-vote(_\( v \)). Similarly, all correct replicas will eventually receive \( n - f \) main-vote(_\( v \)) and \( r\)-broadcast final-vote(_\( v \)). All correct replicas will \( r\)-deliver \( n - f \) final-vote(_\( v \)). In other words, all correct replicas will terminate and decide \( v \).

\[ \text{Lemma 29. If \( p_i \) decides \( v \) in round 0, any correct replica that enters round 1 sets \( iv_1 \) as \( v \).} \]

**Proof.** If \( p_i \) decides \( v \) in round 1, it \( r\)-delivers \( n - f \) final-vote(_\( v \)), among which at least \( f + 1 \) replicas are correct. We assume that a correct replica \( p_k \) enters round 1 with \( iv_1 = \bar{v} \) and prove the correctness by contradiction. If \( p_k \) enters round \( r + 1 \) and sets \( iv_1 = \bar{v} \), there are three conditions: A) \( p_k \) \( r\)-delivers at least \( n - f \) final-vote(_\( v \)); B) \( p_k \) \( r\)-delivers \( f + 1 \) final-vote(_\( v \)); C) \( p_k \) has not received more than \( f + 1 \) final-vote(_\( v \)) and \( p_k \) has not received more than \( f + 1 \) final-vote(_\( v \)). We now show that none of the three conditions is possible.

Condition A): Replica \( p_i \) \( r\)-delivers \( n - f \) final-vote(_\( v \)). We already know that at least \( f + 1 \) correct replicas \( r\)-broadcast final-vote(_\( v \)). Therefore, at least one correct replica \( r\)-broadcasts both final-vote(_\( v \)) and final-vote(_\( v \)), a contradiction.

Condition B): Replica \( p_k \) \( r\)-delivers \( f + 1 \) final-vote(_\( v \)). We already know that \( p_i \) \( r\)-delivers \( n - f \) final-vote(_\( v \)). Therefore, at least one replica (correct or Byzantine) \( r\)-broadcasts both final-vote(_\( v \)) and final-vote(_\( v \)) such that \( p_k \) \( r\)-delivers final-vote(_\( v \)) and \( p_i \) \( r\)-delivers final-vote(_\( v \)), a violation of the agreement property of RBC.

Condition C): Replica \( p_k \) \( r\)-delivers \( n - f \) final-vote(_\( v \)) messages (let the set of replicas be \( S_2 \)). In the messages, fewer than \( f + 1 \) are final-vote(_\( v \)) and fewer than \( f + 1 \) are final-vote(_\( v \)). Other messages must be final-vote(_\( v \)). We already know that \( p_i \) \( r\)-delivers \( n - f \) final-vote(_\( v \)) (let the set of replicas be \( S_2 \)). \( S_1 \) and \( S_2 \) have at least \( 2n - 2f - n = n - 2f \geq f + 1 \) replicas in common. In other words, at least one replica \( r\)-broadcasts a final-vote(_\( v \)) message such that \( p_i \) \( r\)-delivers final-vote(_\( v \)) and \( p_k \) \( r\)-delivers final-vote(_\( v \)) (or final-vote(_\( v \))). a violation of the agreement property of RBC.

\[ \text{Theorem 30 (Agreement). If a correct replica decides \( v \), then any correct replica that terminates decides \( v \).} \]

**Proof.** We assume that a correct replica \( p_i \) decides \( v \) and a correct replica \( p_j \) decides \( \bar{v} \) and prove the theorem by contradiction. Since Cubic-RABA follows Cubic-ABA starting from round \( r > 0 \), if both \( p_i \) and \( p_j \) decide in round \( r > 0 \), correctness follows from the agreement property of Cubic-ABA.

We now show the correctness in the following cases: 1) both \( p_i \) and \( p_j \) decide in round 0; 2) \( p_i \) decides in round 0 and \( p_j \) decides in round \( r > 0 \).

Case 1): If \( p_i \) decides \( v \), it \( r\)-delivers \( n - f \) final-vote(_\( v \)). If \( p_j \) decides \( \bar{v} \), it \( r\)-delivers \( - \bar{v} \) final-vote(_\( v \)). The two quorum of replicas have at least \( n - 2f \) replicas in common. Among the \( n - 2f \) replicas, at least one is correct since \( n - 2f \geq f + 1 \). Therefore, at least one correct replica \( r\)-broadcasts both final-vote(_\( v \)) and final-vote(_\( v \)), a contradiction since each replica only \( r\)-broadcasts a final-vote(_\( v \)) message once in each round. 

Case 2): If \( p_j \) decides \( v \) in round \( r = 1 \), it has received at least \( 2f + 1 \) pre-vote(_\( v \)), where at least one correct replica has sent pre-vote(_\( v \)), a contradiction with Lemma 29. Starting from round 1, Cubic-RABA follows Cubic-ABA so that Lemma 2 holds. If \( p_j \) decides \( v \) in round \( r > 1 \), at least one correct replica must have sent pre-vote(_\( v \)), a contradiction with Lemma 2 since any correct replica sets \( iv \) as \( v \). This completes the proof of the theorem.

\[ \text{Lemma 31. If \( f + 1 \) correct replicas propose \( 1 \) in round 0, every replica either directly decides \( 1 \) in round 0 or/and enters round 1 with \( iv_1 = 1 \).} \]

**Proof.** If a correct replica \( p_i \) enters round 1, there are three conditions: A) \( p_i \) \( r\)-delivers \( n - f \) final-vote(_\( v \)) with the same \( v \); B) \( p_i \) \( r\)-delivers at least \( f + 1 \) final-vote(_\( v \)) for some \( v \); C) none of condition A or B holds. We show that \( v = 1 \) for all three conditions and replicas will set \( iv \) as \( v = 1 \).

For condition A, we already know that at least \( f + 1 \) correct replicas have broadcast final-vote(_\( v \)). Therefore, \( p_i \) must have received \( n - f \) final-vote(_\( v \)). This is because if \( p_i \) receives \( n - f \) final-vote(_\( v \)), at least one correct replica \( r\)-broadcasts both final-vote(_\( v \)) and final-vote(_\( v \)), a contradiction. In other words, \( p_i \) decides 1.

For condition B, we assume \( p_i \) \( r\)-delivers \( f + 1 \) final-vote(_\( v \)) and prove the correctness by contradiction. If \( p_i \) \( r\)-delivers \( f + 1 \) final-vote(_\( v \)), at least one correct replica \( r\)-broadcasts final-vote(_\( v \)). If the correct replica \( r\)-broadcasts final-vote(_\( v \)), the replica must have received \( n - f \) final-vote(_\( v \)). We already know that at least \( f + 1 \)
correct replicas have sent main-vote_0(1). Any correct replica broadcasts main-vote_0(1) message once. In other words, at least one correct replica has broadcast both main-vote_0(0) and main-vote_0(1), a contradiction. Therefore, in this condition, \( p_i \) must have \( r \)-delivered \( f + 1 \) final-vote_0(1). It is now clear that any correct replica uses \( iv_1 = 1 \) to enter round 1.

For condition C, any correct replica will use 1 as \( iv_1 \) since the local coin value is set as 1 in round 0. This completes the proof of the lemma. □

**Theorem 32** (Biased validity). If \( f + 1 \) correct replicas propose 1, then any correct replica that terminates decides 1.

**Proof.** We assume that a correct replica \( p_i \) decides 0 and prove the correctness by contradiction. If \( p_i \) decides in round 0, correctness follows from Lemma 31. If \( p_i \) decides 0 in round \( r > 0 \), at least one correct replica has set \( iv_r \) as 0 and broadcast pre-vote_0(0). Since Cubic-RABA follows Cubic-ABA starting from round 1, Lemma 2 holds. Therefore, the claim that at least one correct replica has set \( iv_0 \) as 0 is a contradiction with Lemma 2. This completes the proof of the theorem. □

**Theorem 33** (Biased termination). Let \( Q \) be the set of correct replicas. Let \( Q_1 \) be the set of correct replicas that propose 1 and never repropose 0. Let \( Q_2 \) be correct replicas that propose 0 and later repropose 1. If \( Q_2 \neq \emptyset \) and \( Q = Q_1 \cup Q_2 \), then each correct replica eventually terminates.

**Proof.** The proof consists of two parts. First, every replica correct eventually enters the next round. Second, if a correct replica enters the next round with input \( v \), \( v \) cannot be manipulated by the adversary.

We first prove that every replica eventually enters the next round. Since Cubic-RABA follows Cubic-ABA starting from round 1, this part follows from termination of Cubic-ABA. We only need to prove that every correct replica eventually enters round 1. For replicas in \( Q_1 \), they broadcast pre-vote_0(1) and add 1 to \( bset_0 \). For replicas in \( Q_2 \), they broadcast pre-vote_0(0) upon the propose(0) function, broadcast pre-vote_0(1) upon the repropose(1) function, and eventually add 1 to \( bset_0 \).

There are two cases: 1) the size of \( Q_1 \) is greater than \( f + 1 \); 2) the size of \( Q_1 \) is smaller than \( f + 1 \).

For the first case, at least \( f + 1 \) replicas in \( Q_1 \) will directly broadcast main-vote_0(1) and \( r \)-broadcast final-vote_0(1). For any correct replica \( p_i \) in \( Q_2 \), it may main-vote_0(1) or main-vote_0(0). There are two sub-cases: none of the correct replicas send main-vote_0(0); at least one correct replica has sent main-vote_0(0). For the first sub-case, it is clear that every correct replica eventually receives and accepts \( n - f \) main-vote_0(1), as every correct replica has 1 in its \( bset_0 \). Similarly, every correct replica will \( r \)-broadcast final-vote_0(1) and accept \( n - f \) final-vote_0(1). For the second sub-case, if a correct replica \( p_i \) sends main-vote_0(0), it receives \( 2f + 1 \) pre-vote_0(0), among which at least \( f + 1 \) are sent by correct replicas. Therefore, every correct replica will eventually receive \( f + 1 \) pre-vote_0(0) and broadcast pre-vote_0(0).

Every replica eventually adds 0 to \( bset_0 \). Since every correct replica has both 1 and 0 in \( bset_0 \), every correct replica accepts both main-vote_0(0) and main-vote_0(1). Similarly, every correct replica accepts both final-vote_0(0) and final-vote_0(1). In other words, every correct replica eventually enters the next round.

For the second case, replicas in \( Q_2 \) will send pre-vote_0(0) upon propose(0). They will send pre-vote_0(1) upon repropose(1) and add 1 to \( bset_0 \). Since the size of \( Q_2 \) is greater than \( f + 1 \) (the size of \( Q_1 \) is smaller than \( f + 1 \) and \( Q = Q_1 \cup Q_2 \)), every replica will receive \( f + 1 \) pre-vote_0(0), send pre-vote_0(0), and add 0 to \( bset_0 \). Furthermore, every correct replica in \( Q_2 \) broadcasts pre-vote_0(1) upon repropose(1). Since the size of \( Q_2 \) is greater than \( f + 1 \), it holds that every correct replica eventually adds 1 to \( bset_0 \). Therefore, every replica will accept main-vote_0(0) and main-vote_0(1), final-vote_0(0), and final-vote_0(1). In other words, every correct replica eventually enters the next round.

We now prove the second part that the value \( iv \) used by any correct replica cannot be manipulated by the adversary. Since Cubic-RABA follows Cubic-ABA starting from round 1, correctness follows from Lemma 6 and termination of Cubic-ABA. □

**Theorem 34** (Integrity). No correct replica decides twice.

**Proof.** In each round, every replica only sends a main-vote_r(\( v \)) message and a final-vote_r(\( v \)) message once. Hence, only one value will be decided and integrity thus follows. □

I Proof of Quadratic-RABA

We now show that Quadratic-RABA achieves validity, unanimous termination, agreement, biased validity, biased termination, and integrity.

**Lemma 35.** If all correct replicas propose \( v \) in round 0 and never repropose \( \bar{v} \), then any correct replica enters the round 1 sets \( iv_1 = v \).

**Proof.** If all correct replicas propose \( v \) in round 0, every correct replica broadcasts pre-vote_0(v). No correct replica will receive more than \( f + 1 \) pre-vote_0(\( \bar{v} \)) messages. Hence, no correct replica will add \( \bar{v} \) to \( bset_0 \). Furthermore, all correct replicas will eventually send vote_0(\( v \)) and receive \( n - f \) vote_0(\( v \)). As no correct replica ever has \( \bar{v} \) in \( bset_0 \), no correct replica will accept vote_0(\( \bar{v} \)). Therefore, all correct replicas will send main-vote_0(\( v \)). No correct replica will accept main-vote_0(\( v \)) or main-vote_0(*) as \( \bar{v} \notin bset_0 \) and no replicas will never receive more than \( f + 1 \) vote_0(\( \bar{v} \)). Accordingly, every correct replicas will send final-vote_0(\( v \)) and receive \( n - f \) final-vote_0(\( v \)). No correct replica accepts final-vote_0(\( \bar{v} \)) as they only have \( v \) in their \( bset \), and no correct replica can receive more than \( f + 1 \) final-vote_0(\( \bar{v} \)). Hence, any correct replica that enters round \( r + 1 \) sets \( iv_{r+1} = v \). □
Theorem 36 (Validity). If all correct replicas propose \( v \) and never repropose \( \bar{v} \), then any correct replica that terminates decides \( v \).

Proof. We assume that a correct replica \( p_i \) terminates and decides \( \bar{v} \) and prove the correctness by contradiction. If \( p_i \) terminates and decides \( \bar{v} \) in round 0, correctness follows from Lemma 35. We now prove the case where \( p_i \) decides in round \( r > 0 \).

Since Quadratic-RABA follows Quadratic-ABA starting from round 1, Lemma 10 holds for \( r > 0 \). If \( p_i \) terminates and decides \( \bar{v} \) in round \( r > 0 \), it receives \( n - f \) final-vote\(_i\)(\( \bar{v} \)). Among the replicas that sent final-vote\(_i\)(\( \bar{v} \)), at least \( f + 1 \) are correct. According to Lemma 11, at least one correct replica has broadcast pre-vote\(_i\)(\( \bar{v} \)). This is a contradiction with Lemma 10 since any correct replica that enters round \( r \) sets iv\(_r\) as \( v \). This completes the proof of the theorem.

Theorem 37 (Unanimous termination). If all correct replicas propose \( v \) and never repropose \( \bar{v} \), then all correct replicas eventually terminate.

Proof. If all correct replicas propose \( v \) and never repropose \( \bar{v} \), all correct replicas only send pre-vote\(_0\)(\( v \)). No correct replica will add \( \bar{v} \) to bset\(_0\). Furthermore, no correct replica will accept vote\(_0\)(\( \bar{v} \)), main-vote\(_0\)(\( \bar{v} \)), or final-vote\(_0\)(\( \bar{v} \)). Eventually all correct replicas will receive \( n - f \) pre-vote\(_0\)(\( v \)), add \( v \) to bset\(_0\), and broadcast vote\(_0\)(\( v \)). Similarly, all correct replicas will eventually receive \( n - f \) main-vote\(_0\)(\( v \)) and broadcast main-vote\(_0\)(\( v \)). All correct replicas will receive \( n - f \) main-vote\(_0\)(\( v \)) and broadcast final-vote\(_0\)(\( v \)). In other words, all correct replicas will eventually receive \( n - f \) final-vote\(_0\)(\( v \)) and decide \( v \).

Lemma 38. If \( p_i \) decides \( v \) in round 0, any correct replica that enters round 1 sets iv\(_1\) as \( v \).

Proof. If \( p_i \) decides \( v \) in round 1, it receives \( n - f \) final-vote\(_0\)(\( v \)), among which at least \( f + 1 \) are sent by correct replicas. We assume that a correct replica \( p_k \) enters round 1 with iv\(_1\) = \( v \) and prove the correctness by contradiction. If \( p_k \) enters round \( r + 1 \) and sets iv\(_1\) as \( \bar{v} \), there are three conditions: A) \( p_k \) receives at least \( n - f \) final-vote\(_i\)(\( \bar{v} \)); B) \( p_k \) receives only final-vote\(_0\)(\( \bar{v} \)) and final-vote\(_0\)(\( \ast \)); C) none of the above applies. In case C, \( p_j \) will set the common coin value 1 as iv\(_1\), the case is impossible. We now show that none of the first two conditions is possible.

Condition A): Replica \( p_i \) receives \( n - f \) final-vote\(_0\)(\( \bar{v} \)). We already know that at least \( f + 1 \) correct replicas have sent final-vote\(_0\)(\( v \)). Therefore, at least one correct replica sends both final-vote\(_0\)(\( v \)) and final-vote\(_0\)(\( \bar{v} \)), a contradiction.

Condition B): Replica \( p_k \) receives final-vote\(_0\)(\( \bar{v} \)) and final-vote\(_0\)(\( \ast \)). We already know that \( p_i \) receives \( n - f \) final-vote\(_0\)(\( v \)). Therefore, at least one replica has sent final-vote\(_0\)(\( v \)) to \( p_i \) and a final-vote\(_0\)(\( \bar{v} \)) (or final-vote\(_0\)(\( \ast \)) message) to \( p_j \), a contradiction.

Theorem 39 (Agreement). If a correct replica decides \( v \), then any correct replica that terminates decides \( v \).

Proof. We assume that a correct replica \( p_i \) decides \( v \) and a correct replica \( p_j \) decides \( \bar{v} \) and prove the theorem by contradiction. Since Quadratic-RABA follows Quadratic-ABA starting from round \( r > 0 \), if both \( p_i \) and \( p_j \) decide in round \( r > 0 \), correctness follows from the agreement property of Quadratic-ABA. We now show the correctness in the following cases: 1) both \( p_i \) and \( p_j \) decide in round 0; 2) \( p_i \) decides in round 0 and \( p_j \) decides in round \( r > 0 \).

Case 1): If \( p_j \) decides \( v \), it receives \( n - f \) final-vote\(_0\)(\( v \)). If \( p_j \) decides \( \bar{v} \), it receives \( n - f \) final-vote\(_0\)(\( \bar{v} \)). The two quorum of replicas have at least \( n - 2f \) replicas in common. Among the \( n - 2f \) replicas, at least one is correct since \( n - 2f \geq f + 1 \). Therefore, at least one correct replica sends both final-vote\(_0\)(\( v \)) and final-vote\(_0\)(\( \bar{v} \)), a contradiction since each replica only sends a final-vote\(_i\)(\( v \)) message once in each round.

Case 2): If \( p_j \) decides \( \bar{v} \) in round \( r = 1 \), it has received at least \( n - f \) pre-vote\(_1\)(\( \bar{v} \)), where at least one correct replica has sent pre-vote\(_1\)(\( \bar{v} \)), a contradiction with Lemma 38. Starting from round 1, Quadratic-RABA follows Quadratic-ABA so that Lemma 10 holds. If \( p_j \) decides \( \bar{v} \) in round \( r > 1 \), at least one correct replica must have sent pre-vote\(_i\)(\( \bar{v} \)), a contradiction with Lemma 10 since any correct replica sets iv\(_r\) as \( v \).

Lemma 40. If \( f + 1 \) correct replicas propose \( 1 \) in round 0, every replica either directly decides \( 1 \) in round 0 or/and enters round 1 with iv\(_1\) = 1.

Proof. If a correct replica \( p_i \) enters round 1, there are three conditions: A) \( p_i \) receives \( n - f \) final-vote\(_0\)(\( v \)) with the same \( v \); B) \( p_i \) receives at least a final-vote\(_0\)(\( v \)) message for some \( v \); C) none of condition A or B holds. We show that \( v = 1 \) for all three conditions and replicas will set iv\(_1\) as \( v = 1 \).

For condition A, we already know that at least \( f + 1 \) correct replicas have broadcast final-vote\(_0\)(1). If \( p_i \) receives \( n - f \) final-vote\(_0\)(0), at least one correct replica has sent both final-vote\(_0\)(1) and final-vote\(_0\)(0), a contradiction. In other words, in this condition \( p_i \) decides 1.

For condition B, we \( p_i \) receives only final-vote\(_0\)(0) and final-vote\(_0\)(\( \ast \)). We already know that at least \( f + 1 \) correct replicas have sent final-vote\(_0\)(1). Therefore, at least one correct replica must have sent both final-vote\(_0\)(1) and final-vote\(_0\)(0) (or final-vote\(_0\)(\( \ast \))), a contradiction.

For condition C, any correct replica will use 1 as input for round 1 since the local coin value is set as 1 in round 0.

Theorem 41 (Biased validity). If \( f + 1 \) correct replicas propose \( 1 \), then any correct replica that terminates decides 1.

Proof. If \( p_i \) decides in round 0, correctness follows from Lemma 40. If \( p_i \) decides 0 in round \( r > 0 \), at least one
correct replica has set \( iv_r \) as 0 and broadcast pre-vote, \((0)\). Since Quadratic-RABA follows Quadratic-ABA starting from round 1, Lemma 10 holds. Therefore, the claim that at least one correct replica has set \( iv_r \) as 0 is a contradiction with Lemma 10. This completes the proof of the theorem. ■

**Lemma 42.** If \( f + 1 \) correct replicas propose 1 in round 0, every correct replica eventually accepts final-vote, \((0)\).

**Proof.** If \( f + 1 \) correct replicas propose 1, they will directly broadcast pre-vote, \((0)\), main-vote, \((0)\), and final-vote, \((0)\). Every correct replica will eventually receive \( f + 1 \) pre-vote, \((0)\). For those correct replicas that have not sent pre-vote, \((0)\), they will also broadcast pre-vote, \((0)\). Therefore, every correct replica eventually adds 1 to bset, \(0\). As \( f + 1 \) correct replicas broadcast main-vote, \((0)\), every correct replica eventually accepts main-vote, \((0)\) message. Similarly, as \( f + 1 \) correct replicas broadcast main-vote, \((0)\), every correct replica eventually accepts final-vote, \((0)\). ■

**Lemma 43.** If a correct replica \( p_i \) sends final-vote, \((0)\) or final-vote, \((*)\), every correct replica eventually accepts the final-vote, \((*)\) message sent by \( p_i \).

**Proof.** Case 1: If a correct replica \( p_i \) sends final-vote, \((0)\), it has received \( n - f \) main-vote, \((0)\), among which at least \( f + 1 \) are sent by correct replicas. Furthermore, the correct replica has put 0 in its bset, \(0\), so it receives \( n - f \) pre-vote, \((0)\). As \( f + 1 \) correct replicas have sent pre-vote, \((0)\), every correct replica eventually receives \( f + 1 \) pre-vote, \((0)\) and send pre-vote, \((0)\). Accordingly, every correct replica puts 0 in bset, \(0\). There are two cases: \( r = 0 \) and \( r > 0 \). If \( r = 0 \), every correct replica will accept main-vote, \((0)\) and final-vote, \((0)\). If \( r > 0 \), every correct replica will accept final-vote, \((0)\) as 0 \( \in \) bset, \(0\). Additionally, since \( f + 1 \) correct replicas have sent main-vote, \((0)\), they all have received \( n - f \) vote, \((0)\), among which at least \( f + 1 \) are sent by correct replicas. As every correct replica receives \( f + 1 \) vote, \((0)\), they will accept main-vote, \((0)\). As every correct replica eventually receive \( f + 1 \) main-vote, \((0)\), every correct replica will accept final-vote, \((0)\).

Case 2: If a correct replica sends final-vote, \((*)\), its bset, \(0\), is \( \{0, 1\} \), i.e., it has received both \( n - f \) pre-vote, \((0)\) and \( n - f \) pre-vote, \((1)\). Following the prior case, every correct replica eventually has bset, \(0\) = \( \{0, 1\} \), so every correct replica accepts final-vote, \((*)\).

**Theorem 44 (Biased termination).** Let \( Q \) be the set of correct replicas. Let \( Q_1 \) be the set of correct replicas that propose 1 and never repropose 0. Let \( Q_2 \) be correct replicas that propose 0 and later repropose 1. If \( Q_2 \neq \emptyset \) and \( Q = Q_1 \cup Q_2 \), then each correct replica eventually terminates.

**Proof.** The proof consists of two parts. First, every replica eventually enters the next round. Second, if a correct replica enters the next round with input \( v \), \( v \) cannot be manipulated by the adversary.

We first prove that every replica eventually enters the next round. Since Quadratic-RABA follows Quadratic-ABA starting from round 1, termination for \( r > 0 \) follows from termination of Cubic-ABA. We only need to prove that every correct replica eventually enters round 1. For replicas in \( Q_1 \), they broadcast pre-vote, \((1)\) and add 1 to bset, \(0\). For replicas in \( Q_2 \), they broadcast pre-vote, \((0)\) upon the propose, \((0)\) event, broadcast pre-vote, \((0)\) upon the repropose, \((1)\) event, and eventually add 1 to bset, \(0\). There are two cases: 1) the size of \( Q_1 \) is greater than \( f + 1 \); 2) the size of \( Q_1 \) is smaller than \( f + 1 \).

For the first case, at least \( f + 1 \) replicas in \( Q_1 \) will directly broadcast vote, \((1)\), main-vote, \((1)\), and final-vote, \((1)\). For any correct replica \( p_i \) in \( Q_2 \), it may send vote, \((1)\) or vote, \((0)\). There are two sub-cases: none of the correct replicas send vote, \((0)\); at least one correct replica has sent vote, \((0)\).

For the first sub-case, it is straightforward to see that every correct replica eventually receives and accepts \( n - f \) vote, \((0)\), as every correct replica has 1 in its bset, \(0\). Similarly, every correct replica will send main-vote, \((0)\) and accept \( n - f \) main-vote, \((0)\). Similarly, every correct replica will send final-vote, \((0)\). According to Lemma 42, every correct replica eventually accepts final-vote, \((0)\) so correct replicas will enter the next round. For the second sub-case, if a correct replica \( p_i \) sends vote, \((0)\), it receives \( n - f \) pre-vote, \((0)\), among which at least \( f + 1 \) are sent by correct replicas.

Therefore, every correct replica will eventually receive \( f + 1 \) pre-vote, \((0)\) and broadcast pre-vote, \((0)\). Every replica eventually adds 0 to bset, \(0\). Since every correct replica has both 1 \( \in \) bset, \(0\), every correct replica accepts both vote, \((0)\) and vote, \((1)\). Similarly, every correct replica accepts both main-vote, \((0)\) and main-vote, \((1)\). In this case, every correct replica will eventually send a final-vote, \((0)\) message. According to Lemma 42, every correct replica eventually accepts final-vote, \((0)\). According to Lemma 43, any correct replica accepts the final-vote, \((0)\) message sent by any correct replica. Therefore, every correct replica eventually enters the next round.

For the second case, replicas in \( Q_2 \) will send pre-vote, \((0)\) upon propose, \((0)\). They will send pre-vote, \((1)\) upon repropose, \((1)\) and add 1 to bset, \(0\). Since the size of \( Q_2 \) is greater than \( f + 1 \) (the size of \( Q_1 \) is smaller than \( f + 1 \) and \( Q = Q_1 \cup Q_2 \)), every replica will receive \( f + 1 \) pre-vote, \((0)\), send pre-vote, \((0)\), and add 0 to bset, \(0\). Furthermore, every correct replica in \( Q_2 \) broadcasts pre-vote, \((1)\) upon repropose, \((1)\). Since the size of \( Q_2 \) is greater than \( f + 1 \), every correct replica eventually adds 1 to bset, \(0\). According to the protocol, in round 0, every correct replica accepts main-vote, \((0)\) and final-vote, \((0)\) if \( v \) is added to bset, \(0\). Therefore, every replica will accept both vote, \((0)\) and vote, \((1)\), and main-vote, \((0)\) and final-vote, \((0)\) with any value. Accordingly, every correct replica eventually enters the next round.

We now prove that the value \( iv \) used by any correct replica cannot be manipulated by the adversary. Since Quadratic-RABA follows Quadratic-ABA starting from round 1, correct-
ness follows from Lemma 20 and termination of Quadratic-ABA.

Theorem 45 (Integrity). No correct replica decides twice.

Proof. In each round, every replica only sends a final-vote,() message once. Hence, only one value will be decided and integrity thus follows.

■