Practical Signature-Free Asynchronous Common Subset in Constant Time

Sisi Duan
duansisi@tsinghua.edu.cn
Tsinghua University

Xin Wang
wangxin87@mail.tsinghua.edu.cn
Tsinghua University

Haibin Zhang
bchainzhang@aliyun.com
Beijing Institute of Technology

ABSTRACT

Asynchronous common subset (ACS) is a powerful paradigm enabling applications such as Byzantine fault-tolerance (BFT) and multi-party computation (MPC). The most efficient ACS framework in the information-theoretic (IT) setting is due to Ben-Or, Kelmer, and Rabin (BKR, 1994). The BKR ACS protocol has been both theoretically and practically impactful. However, the BKR protocol has an $O(\log n)$ running time (where n is the number of replicas) due to the usage of n parallel asynchronous binary agreement (ABA) instances, impacting both performance and scalability. Indeed, for a network of 16~64 replicas, the parallel ABA phase occupies about 95%~97% of the total runtime in BKR. A long-standing open problem is whether we can build an ACS framework with $O(1)$ time while not increasing the message or communication complexity of the BKR protocol.

In this paper, we resolve the open problem, presenting the first constant-time ACS protocol with $O(n^3)$ messages in the IT (and signature-free) settings. Moreover, as a key ingredient of our new ACS framework and an interesting primitive in its own right, we provide the first IT-secure multivalued validated Byzantine agreement (MVBA) protocol with $O(1)$ time and $O(n^3)$ messages. Both results can improve—asymptotically and concretely—various applications using ACS and MVBA in the IT, quantum-safe, or signature-free settings. As an example, we implement FIN, a BFT protocol instantiated using our framework. Via a 121-server deployment on Amazon EC2, we show FIN is significantly more efficient than PACE (CCS 2022), the state-of-the-art asynchronous BFT protocol of the same type. In particular, FIN reduces the overhead of the ABA phase to as low as 1.23% of the total runtime, and FIN achieves up to 3.41x the throughput of PACE.

1 INTRODUCTION

Overview. This paper is about resolving a long-standing open problem in fault-tolerant distributed computing and multi-party computation (MPC). We present the first practical $O(1)$-time and $O(n^3)$-message asynchronous common subset (ACS) protocol [7, 9], while prior constructions either have $O(\log n)$ time and $O(n^3)$ messages or fail to be practical due to $O(n^4)$ message complexity.

History. The concept of asynchronous common subset (ACS) is due to Ben-Or, Canetti, and Goldreich (BCG) in the context of asynchronous MPC—under a different name called *agreement on a core set* [7]. BCG proposed an ACS construction using two core building blocks in fault-tolerant distributed computing—reliable broadcast (RBC) and asynchronous binary agreement (ABA). Soon later, Ben-Or, Kelmer, and Rabin (BKR) presented a refined and practical ACS construction using n RBC and n ABA instances [9]. Meanwhile, BKR renamed "agreement on a core set" as "agreement on a common subset."

The ACS paradigm (BCG and BKR) has been enormously impactful, empowering numerous applications as well as implementations, such as MPC [6, 9, 10, 15, 19–22, 37], BFT [3, 10, 29, 36, 40, 52, 53], distributed key generation [25, 27, 34], and proactive secret sharing [31, 32, 49].

IT, signature-free, and quantum-safe settings. The ACS notion has been historically associated with the information-theoretic (IT) setting: as emphasized by Cachin et al. [12, Section 4], "the primitive of agreement on a core set (which) is used in the information-theoretic model." Indeed, ACS constructions typically rely on *ideal* building primitives in distributed computing such as RBC, ABA, and common coins, all of which can be realized in the IT-secure manner [11, 16].

A closely related setting is the *signature-free setting* focusing on protocols assuming the existence of common coins.[1] In particular, the line of work for signature-free ABA—which is at the core of practical ACS constructions—begins with the seminal work by Rabin [47] and is followed by [24, 39, 42, 43, 50] being implemented in various practical BFT and MPC systems [29, 36, 37, 40, 52, 53]. To summarize, these ACS protocols, by design, are IT-secure and signature-free if assuming the existence of IT-secure common coins.

Moreover, both settings yield ACS protocols achieving the desirable quantum safety property (but quantum liveness) as defined in [33], where the safety of the system is always achieved even in the presence of a quantum adversary.

The open problem. Almost all ACS implementations use the BKR ACS construction, which runs n parallel ABA instances, thereby achieving $O(n^3)$ messages and $O(\log n)$ time. The ABA phase involving n parallel ABA instances is the well-known performance and scalability bottleneck reported in various BFT implementations [29, 30, 50]. For example, the ABA phase in a network with 16~64 replicas occupies about 95%~97% of the total runtime in BKR [30]. Recently, Zhang and Duan proposed a refined PACE ACS framework [50]. In PACE ACS, the n ABA instances are made fully parallelizable to gain in improved performance, but the running time remains $O(\log n)$—remaining the critical bottleneck.

A long-standing open problem, ever since the 1990s, in ACS—or generally in fault-tolerant distributed computing and cryptography—is:

Can we build ACS—in the IT setting or signature-free setting—with $O(n^3)$ messages and $O(1)$ time?

Note the only known such ACS construction terminating in constant time is due to Ben-Or and El-Yaniv [8], but the construction

[1] Here we directly borrow the term from the line of work [42–44]; the setting may also be called *cryptography-free*.

protocol	messages	time	# (R)ABA
BKR variants [29, 36, 40]	$O(n^3)$	$O(\log n)$	n
PACE [50]	$O(n^3)$	$O(\log n)$	n
FIN (§7; this paper)	$O(n^3)$	$O(1)$	$O(1)$

Table 1: Comparison among ACS implementations relying on common coins.

uses n^2 RB instances with $O(n^4)$ messages (being prohibitively expensive).

Our results and our approach. In this work, we resolve the open problem, showing an ACS protocol with $O(n^3)$ messages and $O(1)$ time. At a high level, we begin with a new multivalued validated Byzantine agreement (MVBA) protocol [12] in the information-theoretical setting with $O(n^3)$ and $O(1)$ time. Then we reduce the ACS problem to MVBA and RBC.

Recall that the notion of MVBA is different from the conventional multivalued Byzantine agreement in that MVBA assumes the existence of a global predicate, and replicas only decide values satisfying the global predicate. To build an MVBA protocol towards our goal, we use RBC, common coins, and reproposable ABA (RABA)—a notion due to Zhang and Duan [50]. RABA, like ABA, can be readily built from common coins and authenticated channels and terminate in expected constant time. Our MVBA uses n parallel RBC instances and an expected constant number of RABA instances. Our MVBA protocol is also the first IT-secure and signature-free MVBA protocol—by directly assuming common coins—with $O(n^3)$ messages and $O(1)$ time.

We also show that one can use a weaker RBC primitive to realize a computation and bandwidth more efficient MVBA protocol while still achieving $O(n^3)$ messages and $O(1)$ time.

Our transformation from MVBA to ACS is simple and efficient, consisting of n parallel RBC instances and a single MVBA instance.

Both our MVBA and ACS constructions are efficient, inheriting the fast path in RABA enabling rapid termination.

Our contributions. We make the following contributions:

- We present the first IT-secure and signature-free ACS protocol with $O(1)$ time and $O(n^3)$ messages, assuming ideal building blocks only. In contrast to prior constructions, our ACS protocol requires only an expected constant number of binary agreement instances. Our protocol directly improves various ACS-enabled applications such as asynchronous MPC [9, 15, 22, 37].
- As a core ingredient of our ACS construction and a primitive that is interesting in its own right, we present the first signature-free MVBA protocols with $O(1)$ time and $O(n^3)$ messages while existing such MVBA protocols have $O(\log n)$ time and $O(n^3)$ messages. Moreover, our MVBA protocols lead to instantiations having lower communication than existing ones. We also show an efficient and tailored MVBA construction, optimizing both communication cost and computational efficiency.
- To demonstrate the efficiency of our ACS protocol, we instantiate it with a practical BFT system called FIN (see Table 1 for a comparison). We implement FIN and PACE, the state-of-the-art ACS protocol of the same kind. Via a 121-instance deployment on Amazon EC2, we show that FIN consistently and drastically outperforms PACE for $f > 5$. In particular, FIN significantly reduces the overhead of the ABA phase to only 1.23%-5.22% of the total runtime, in contrast to the 15.10%-83.66% overhead in PACE. Moreover, when $f = 40$, the peak throughput of FIN is 3.41x that of PACE.

2 RELATED WORK

Interactive consistency and vector consensus. The ACS problem can be viewed as an asynchronous version of the *interactive consistency* (IC) problem defined for synchronous systems by Pease, Shostak, and Lamport [46]. Replicas in aynchronous IC reach an agreement on a vector with the values proposed by *all* correct replicas.

In contrast, the ACS abstraction (also called asynchronous IC by Ben-Or and El-Yaniv [8]) naturally requires that the output of each correct replica contains $n-f$ values such that at least $n-2f$ elements are proposed by correct replicas. Namely, ACS requires only that the majority of the values were proposed by correct replicas. Indeed, it is impossible to guarantee that the vector has the values of all correct replicas in asynchronous settings.

ACS is also called vector consensus in some literature [23, 28, 41, 45]. Note that ACS is different from set agreement [18] that provides only an approximation of agreement.

ACS constructions (in IT and signature-free settings). BKR ACS reduces asynchronous BFT to RBC and ABA [9]. In BKR ACS, all replicas run an RBC phase to reliably broadcast their proposals. Then they run an ABA phase with n parallel ABA instances. The i-th ABA instance agrees on if the proposal of p_i has been delivered in the RBC phase. Upon RBC delivery of a proposal from p_j, the replica proposes 1 to the j-th ABA instance. If a correct replica p_j decides 1 for the i-th ABA instance, the proposal from p_i is delivered. BKR ACS requires if a replica has not received some proposals during the RBC phase, the replica abstains from proposing 0 until $n - f$ ABA instances terminate with 1. In PACE ACS, ABA instances are replaced using RABA instances, and thus all RABA instances can be run in a fully parallelizable manner [50]. Both BKR and PACE ACS approaches require running n (R)ABA instances terminating in expected constant rounds, leading to expected $O(\log n)$ time in total.

The ACS construction by Ben-Or and El-Yaniv [8] terminates in expected constant time and uses n^2 RBC instances with $O(n^4)$ messages, which is prohibitively expensive.

Another line of ACS constructions reduces the ACS problem to RBC and multivalued Byzantine agreement (MBA) [23, 41]. The construction requires running $O(f)$ sequential multivalued Byzantine agreement (MBA) instances, resulting in $O(n)$ running time.

Our ACS approach is fundamentally different from existing ones, reducing the ACS problem to IT-secure MVBA and then to RBC and a constant number of RABA instances.

RBC. We use RBC (aka BRB) [11] to build our MVBA construction and our ACS construction. For both our MVBA and ACS, RBC dominates their communication. One could use any RBC constructions to instantiate them. In this paper, we discuss constructions using CT RBC [14] (using hashes), EFBRB [4] (IT-secure), and CCBRB [4] (using hashes and online error correction coding [7]).

For our ACS implementation, we use CT RBC [14]. Internally in our MVBA implementation, we show that one can use a tailored, weaker RBC construction (using collision-resistant hashes only) to build a concretely more efficient protocol.

RABA. The notion of reproposable ABA (RABA) is due to Zhang and Duan [50]. RABA was originally proposed to solve a BKR bottleneck to allow all ABA instances to run in parallel and remove the two-subphase bottleneck. Later, such a primitive was used to develop a quantum secure and adaptively secure asynchronous BFT protocol without trusted setup [52], and to build an asynchronous distributed key generation protocol without random oracles or PKI [51].

Zhang and Duan have argued that RABA could be useful as a general and "first-class distributed computing primitive" [50]. Our results bolster this point of view.

However, the way we use RABA in this paper is fundamentally different from all these works. Indeed, existing protocols use RABA in the BKR framework (style) and need to run n parallel RABA instances; in contrast, this paper only needs to run an expected constant number of RABA instances, thereby developing the technique of using RABA.

MVBA. The notion of MVBA was introduced by Cachin, Kursawe, Petzold, and Shoup [12].

In the computational model (assuming—in addition to common coins—cryptographic tools such as threshold signatures), Abraham, Malkhi, and Spiegelman proposed an MVBA protocol [2] with $O(Ln^2 + \kappa n^2)$ communication, optimal word complexity, and the quality property, where L is the length of the input from each replica. Lu et al. [38] reduced the communication from $O(Ln^2+\kappa n^2)$ to $O(Ln + \kappa n^2)$ by additionally using constant-size vector commitments [17].

In the IT and signature-free setting, the asynchronous distributed key generation protocol by Das et al. [27] implies an MVBA, as demonstrated in a recent work [25] that recasts the agreement phase in [27] using the language of MVBA. The MVBA protocols [25, 27] are IT-secure (assuming IT-secure common coins) and signature-free if using IT-secure common coins, terminating in $O(\log n)$ time. The MVBA proposed in this paper has the same message complexity as those in [25, 27], but terminates in $O(1)$ time. Also, our MVBA protocol can lead to instantiations having lower communication than those from [25, 27].

Our MVBA protocol may also be used—possibly in a non-trivial manner—in certain (but not all) protocols in [31, 32, 49] without increasing the communication or making stronger assumptions.

MPC. Our results on ACS and MVBA directly improve a large number of asynchronous MPC protocols (with fairness and guaranteed output delivery) [6, 9, 10, 15, 19–22, 37].

From atomic broadcast to ACS. There are asynchronous atomic broadcast protocols without using ACS or MVBA (e.g., [33]). It is, however, unclear how to *efficiently* transform an atomic broadcast protocol to ACS which has direct and practical applications beyond just BFT state machine replication (e.g., MPC, ADKG).

3 SYSTEM MODEL AND PROBLEM STATEMENT

We consider a system with n replicas, $\{p_1, \cdots, p_n\}$, where f out of them may fail arbitrarily (Byzantine failures). A non-Byzantine replica is called a correct replica. All protocols we consider assume that $f \leq \lfloor \frac{n-1}{3} \rfloor$, which is optimal. A (Byzantine) *quorum* is a set of $\lceil \frac{n+f+1}{2} \rceil$ replicas. Without loss of generality, this paper may assume $n = 3f + 1$ and a quorum size of $2f + 1$. We assume the existence of point-to-point authenticated channels between each pair of replicas. We consider asynchronous networks making no timing assumptions on message processing or transmission delays.

We consider both adaptive corruption and static corruption. In adaptive corruption, the adversary 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. In contrast, in the static adversary model, the adversary is restricted to choosing its set of corrupted replicas at the beginning of the protocol. If all building blocks satisfy adaptive security, then our protocol achieves adaptive security. The ACS protocol implemented in this paper achieves static security, just as in prior protocols [29, 37, 40, 50]; but it can be made adaptively secure if using an adaptively secure common coin protocol [5, 35].

Each protocol instance is associated with a unique tag id. We may omit the identifiers in the pseudocode when no ambiguity arises. We may just use subscripts to denote the instance identifier; for instance, RBC_i denotes the RBC instance tagged with a unique identifier i and initiated by replica p_i.

Asynchronous Common Subset (ACS). In ACS [7, 9], each replica holds an input and correct replicas reach an agreement on a set of values. An ACS protocol is specified by *acs-propose* and *acs-decide* events. ACS should satisfy the following properties:

- **Validity**: If a correct replica *acs-decides* a set $\mathbf{v}$, then $|\mathbf{v}| \geq n - f$ and $\mathbf{v}$ contains values *acs-proposed* by at least $n - 2f$ correct replicas.
- **Agreement**: If a correct replica *acs-decides* $\mathbf{v}$, then every correct replicas *acs-decides* $\mathbf{v}$.
- **Termination**: If all correct replicas *acs-propose*, then all correct replicas *acs-decide.*

In this paper, we use the conventional validity notion. Note that as argued in [50], for the validity property, the size of $\mathbf{v}$ (denoted $|\mathbf{v}|$) can be relaxed such that $|\mathbf{v}| \geq f + 1$; namely, in many applications, it suffices to ensure a weaker validity notion by requiring that $\mathbf{v}$ contains values from at least one correct replica.

Multivalued validated Byzantine agreement (MVBA). MVBA allows each replica that has an input to agree on a value that satisfies a predicate Q known by all replicas [12]. An MVBA protocol satisfies the following properties:

- **External validity**: Every correct replica that terminates *mvba-decides* v such that $Q(v)$ holds.
- **Agreement**: If a correct replica *mvba-decides* v, then any correct replica that terminates *mvba-decides* v.
- **Integrity**: If all replicas follow the protocol, and if a correct replica *mvba-decides* v such that $Q(v)$ holds, then some replica *mvba-proposed* v such that $Q(v)$ holds.
- **Termination**: If all correct replicas are activated and all messages sent among correct replicas have been delivered, then all correct replicas *mvba-decide.*

The quality property was introduced by Abraham, Malkhi, and Spiegelman to bound the probability that the decided value was proposed by a correct replica [2]:

- **Quality**: The probability of *mvba-deciding* a value that was proposed by a correct replica is at least 1/2.

In this paper, we first present an MVBA protocol without the quality property and then show how to modify it to achieve the quality property additionally. For our ACS construction, we only need an MVBA protocol without the quality property though.

The way we present our MVBA protocol follows that of [25]. In particular, the MVBA formalization in [25] requires that the predicate additionally uses some variable depending on the state of each node, a property needed in [25] and our MVBA protocols.

Byzantine fault tolerance (BFT). In a BFT protocol, clients *submit* transactions (requests) and replicas *deliver* them. The client obtains a final response to the submitted transaction from the replica responses. In a BFT system with n replicas, it tolerates $f \leq \lfloor \frac{n-1}{3} \rfloor$ Byzantine failures. The correctness of a BFT protocol is specified as follows:

- **Safety**: If a correct replica *delivers* a transaction tx before *delivering* tx', then no correct replica *delivers* a transaction tx' without first *delivering* tx.
- **Liveness**: If a transaction tx is *submitted* to all correct replicas, then all correct replicas eventually *deliver* tx.

4 BUILDING BLOCKS

We review the building blocks for our systems. To help understand RABA [50], we first review the notion of ABA.

Asynchronous binary Byzantine agreement (ABA). An ABA abstraction is specified by *aba-propose* and *aba-decide*. Each replica proposes a binary value (aka a vote) and correct replicas will decide on some value. ABA should satisfy the following properties:

- **Validity**: If all correct replicas *aba-propose* v, then any correct replica that terminates *aba-decides* v.
- **Agreement**: If a correct replica *aba-decides* v, then any correct replica that terminates *aba-decides* v.
- **Termination**: Every correct replica eventually *aba-decides* some value.
- **Integrity**: No correct replica *aba-decides* twice.

Reproposable Asynchronous Binary Agreement (RABA). RABA is a new primitive introduced by Zhang and Duan [50]. In contrast to conventional ABA protocols, where replicas can vote once only, RABA allows replicas to change their votes and vote twice. A RABA protocol is specified by *raba-propose*, *raba-repropose*, and *raba-decide*. For our purpose, RABA is "biased towards 1." 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. RABA (biased toward 1) satisfies the following properties:

- **Validity**: If all correct replicas *raba-propose* v and never *raba-repropose* $\bar{v}$, then any correct replica that terminates *decides* v.
- **Unanimous termination**: If all correct replicas *raba-propose* v and never *raba-repropose* $\bar{v}$, then all correct replicas eventually terminate.
- **Agreement**: If a correct replica *raba-decides* v, then any correct replica that terminates *raba-decides* v.
- **Biased validity**: If $f + 1$ correct replicas *raba-propose* 1, then any correct replica that terminates *raba-decides* 1.
- **Biased termination**: Let Q be the set of correct replicas. Let Q_1 be the set of correct replicas that *raba-propose* 1 and never *raba-repropose* 0. Let Q_2 be correct replicas that *raba-propose* 0 and later *raba-repropose* 1. If $Q_2 \neq \emptyset$ and $Q = Q_1 \cup Q_2$, then each correct replica eventually terminates.
- **Integrity**: No correct replica *raba-decides* twice.

We explain some differences between ABA and RABA. Validity in RABA is slightly different from that for ABA, as we need to modify it to accommodate the RABA syntax. Integrity in RABA is used to ensure that RABA decides only once (even though we have an additional *raba-repropose* event).

Unanimous termination and biased termination are defined to ensure RABA termination in two different scenarios.

Biased validity in RABA requires that if $f + 1$ replicas, not all correct replicas, propose 1, then a correct replica that terminates decides 1. We emphasize that the biased validity property was initially defined to guarantee the PACE ACS framework [50] to have sufficient transactions delivered (the ACS validity property); however, in this paper, biased validity is essentially to ensure constant-time termination for our ACS construction.

For our implementation, we use the Pisa RABA protocol due to Zhang and Duan [50] assuming common coins and authenticated channels.

Byzantine reliable broadcast (RBC). The RBC abstraction allows a sender to reliably broadcast a message to the replicas. A RBC protocol is specified by two events *r-broadcast* and *r-deliver* such that the following properties hold:

- **Validity**: If a correct replica p *r-broadcasts* a message m, then p eventually *r-delivers* m.
- **Agreement**: If some correct replica *r-delivers* a message m, then every correct replica eventually *r-delivers* m.
- **Integrity**: For any message m, every correct replica *r-delivers* m at most once. Moreover, if a replica *r-delivers* a message m with sender p_s, then m was previously broadcast by replica p_s.

For our ACS implementation, we use CT RBC due to Cachin and Tessaro [14] that uses hash functions (with output length κ) and has a communication of $O(n|m| + \kappa n^2 \log n)$.

Common coins. Following prior works [9, 24, 42, 43, 50], we assume our protocols are supplied by a common coin, an object that is introduced by Rabin [47] which delivers the same sequence of random coins to replicas. We use the common coin protocol for the underlying *random leader election* protocol (denoted by $\mathsf{Election}()$) and used it in the underlying RABA protocol.

For our implementation, we use the threshold PRF scheme by Cachin, Kursawe, and Shoup [13].

5 OUR MVBA APPROACH

5.1 Overview

We present our signature-free MVBA protocol with $O(1)$ time and $O(n^3)$ messages. In particular, we reduce MVBA to RBC, random leader election (via common coins), and RABA.

protocol	communication (threshold PRF)	time	assumption
CKPS [12]	$O(Ln^2 + \kappa n^2 + n^3)$	$O(1)$	threshold sig
AMS [2]	$O(Ln^2 + \kappa n^2)$	$O(1)$	threshold sig
Dumbo-MVBA [38]	$O(Ln + \kappa n^2)$	$O(1)$	threshold sig; vc
DYX+ MVBA [27]	$O(Ln^2 + \kappa n^3 \log n)$	$O(\log n)$	hash
DXKR MVBA [25]	$O(Ln^2 + \kappa n^3 \log n)$	$O(\log n)$	hash
Our MVBA (Sec. 5.2) + CT RBC [14]	$O(Ln^2 + \kappa n^3 \log n)$	$O(1)$	hash
Our MVBA (Sec. 5.2) + EFBRB [4]	$O(Ln^2 + \kappa n^2 + n^3 \log n)$	$O(1)$	none
Our MVBA (Sec. 5.2)+ CCBRB [4]	$O(Ln^2 + \kappa n^3)$	$O(1)$	hash
Our tailored MVBA (Sec. 5.4)	$O(Ln^2 + \kappa n^3)$	$O(1)$	hash

Table 2: Comparison of the MVBA protocols using common coins. Here we examine the communication cost of protocols by using threshold PRF [13] to generate common coins. L is the length of the input from each replica and κ is a security parameter. The "assumption" column means the additional assumption besides common coins and authenticated channels. "vc" stands for a constant-size vector commitment which typically requires trusted setup and pairing assumptions. Our protocols lead to the first $O(1)$-time MVBA protocols without threshold signatures. In addition, instantiating our MVBA (Sec. 5.2) with EFBRB [4] leads to an MVBA protocol achieving lower communication than any instantiations from [25, 27], as our MVBA uses only $O(1)$ RABA and $O(1)$ common coin instances—$O(\kappa n^2)$ bits using threshold PRF.

```
MVBA
Input: value $v_i$ such that a global predicate $Q(v_i)$ holds
Output: value $v_k$ (proposed by $p_k$)
Initialization: $r \leftarrow 0$
-----------------------------------------------------------
01 upon event mvba-propose($v_i$)
02   r-broadcast $v_i$ for RBC$_i$                {▷ RBC phase}
     {every replica verifies whether $Q(v_i)$ holds upon receiving $v_i$
   before participating in RBC$_i$ }
03   wait for $n - f$ RBC instances to complete   {▷ Iteration phase}
04   repeat
05     $k \leftarrow$ Election()
06     if some value is r-delivered in RBC$_k$
07       raba-propose 1 for RABA$_r$
08     else
09       raba-propose 0 for RABA$_r$
10       if later some value is r-delivered in RBC$_k$
11         raba-repropose 1 for RABA$_r$
12     if RABA$_r$ outputs 1
13       wait for RBC$_k$ to r-deliver value $v_k$
14       terminate the protocol and mvba-decide($v_k$)
15     $r \leftarrow r + 1$
```

Figure 1: Our MVBA protocol with a predicate Q. Code for p_i.

At a high level, our MVBA protocol works as follows. First, each replica p_i runs RBC$_i$ to disseminate its proposal, resulting in n parallel RBC instances. We aim to have replicas agree on the value *r-delivered* by *exactly one* of the n RBC instances. A crucial observation is that if $n - f$ correct replicas complete $n - f$ RBC instances, then at least $f + 1$ correct replicas have *r-delivered* some values for $f + 1$ RBC instances. If we know this set I of the $f + 1$ RBC instances, then we are almost done. In particular, all we need to do is to pick any of the RBC instances in I, say, RBC$_k$, and correspondingly, replicas run a RABA instance by proposing 1 once they *r-delivered* some value for the RBC$_k$ instance. To see why this intuitively works, first note that the agreement property in RBC and the biased termination property in RABA together guarantee termination, and meanwhile, the biased validity property ensures that even if only $f + 1$ correct replicas, not all correct replicas, propose 1, correct replicas that terminate will decide 1. Observing $|I| \geq f + 1$, if we take a random guess among all $3f + 1$ replicas, then with a probability of at least 1/3, we can hit a good $k \in I$. Thus, our MVBA protocol will terminate in expected constant time. Throughout the process, we use ideal building blocks—RBC, RABA, and common coins—no cryptographic primitives such as threshold signatures.

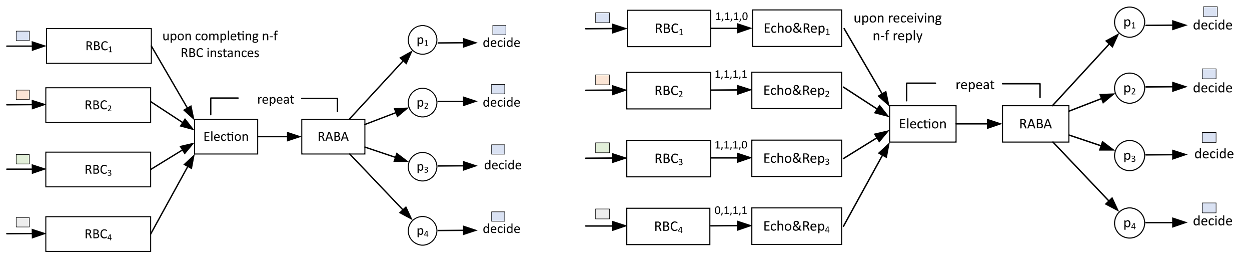
We comment that external validity can be trivially enforced, as long as the predicate is publicly verifiable. Namely, we have treated it as a general predicate. Note, however, that if removing external validity, our MVBA protocol does not directly lead to a multivalue Byzantine agreement (MBA) protocol (e.g., [44]); this is because it does not satisfy the validity property in MBA.[2]

In this section, we first build MVBA without the quality property, satisfying all the security properties defined in CKPS [12]. We will show this MVBA protocol suffices to build our ACS protocol with $O(1)$ time and $O(n^3)$ messages. Then we show that by including two additional communication rounds (using a variant of HotStuff technique [2, 48]), we can build MVBA with the quality property. Last, we demonstrate a highly efficient MVBA protocol that optimizes concrete communication cost and computational efficiency. Such an MVBA protocol benefits from a core observation that we may not necessarily need a fully-fledged RBC to construct MVBA.

5.2 Our MVBA Protocol

We present the psuedocode of our MVBA protocol in Figure 1. As illustrated in Figure 2a, the protocol has two phases: an RBC phase

[2]Validity in MBA requires that if *all* correct replicas propose 1, then all replicas that terminate decide 1.



(a) MVBA without the quality property. **(b) MVBA with the quality property.**

Figure 2: Our MVBA protocols.

with n parallel RBC instances; an iteration phase with only one RABA instance for each iteration.

RBC phase (lines 01-02). In the RBC phase, each replica p_i holds an input v_i that it proposes for the MVBA protocol such that $Q(v_i)$ holds. Upon the event *mvba-propose*(v_i), replica p_i *r-broadcasts* v_i for an RBC instance RBC_i. Upon receiving value v_i *r-broadcast* by p_i in RBC_i, every replica waits until $Q(v_i)$ holds before participating in RBC_i.

Note that there are up to n parallel RBC instances, and every correct replica p_i verifies the predicate for all RBC instances running. As usual, we require that the predicate Q is verifiable across all correct replicas. Meanwhile, in certain applications, we may also require that the predicate Q depends on the internal state *st* of a particular replica. Namely, it is possible that $Q(v, st)$ for some v and some replica fails to hold at the beginning, but $Q(v, st)$ will hold at some point, all depending on *st* of the replica.

Iteration phase (lines 03-15). Each replica then waits until it *r-delivers* $n - f$ RBC instances before it enters the iteration phase. In each iteration, replicas run the Election() function and a RABA instance until a RABA instance outputs 1.

Concretely, in each iteration r, replicas query the Election() function and obtain a random k such that $1 \leq k \leq n$ (line 05). At lines 06-07, if a replica has previously *r-delivered* some value in RBC_k, it *raba-proposes* 1 for the RABA instance denoted RABA_r[3] in iteration r. Otherwise, at lines 08-09, the replica *raba-proposes* 0. If a replica originally *raba-proposes* 0 but later *r-delivers* some value in RBC_k, p_i *raba-reproposes* 1 (at lines 10-11), ensuring protocol termination.

After each replica provides some input to RABA_r, it waits for the output of RABA_r. If p_i *raba-decides* 0, the replica continues to the next iteration $r + 1$ (line 15). If p_i *raba-decides* 1, it waits for the output of RBC_k denoted as v_k (line 13). Then it *mvba-decides* v_k and terminates the protocol (line 14).

Complexity and instantiations. To understand the time complexity of our MVBA protocol, we first observe that at the end of the RBC phase, for at least $f + 1$ instances, at least $f + 1$ correct replicas have *r-delivered* some values (see Lemma 5.4 in this section). Second, the biased validity property of RABA guarantees that for any of these $f + 1$ instances, replicas will *raba-decide* 1. As the Election() function selects any of the $f + 1$ instances with at least a probability of 1/3, the protocol terminates in expected $O(1)$ time.

[3]Note that if we assume the Election() function is a random permutation instead of a random function, then we can also use RABA_k to uniquely and unambiguously denote the RABA instance.

Unlike prior signature-free MVBA constructions [25, 27] that terminate in expected $O(\log n)$ time (summarized in Table 2), our MVBA protocols have expected $O(1)$ time.

Both the message complexity and the communication complexity of our MVBA are bounded by the n parallel RBC instances. As each RBC instance has $O(n^2)$ messages, our MVBA construction has $O(n^3)$ messages.

Table 2 summarizes the communication cost of our protocols assuming a threshold PRF [13]. For instance, if we use EFBRB [4] (an IT-secure RBC), we obtain an MVBA protocol assuming common coins only. If we use CCBRB [4], we obtain a protocol using hashes and common coins and having lower communication than instantiations from [25, 27]. In particular, our MVBA protocol with CCBRB results in a protocol with $O(Ln^2 + \kappa n^2 + n^3 \log n)$ communication (the term $O(\kappa n^2)$ is due to the cost of common coins in $O(1)$ RABA and Election() instances); in contrast, [25, 27] with CCBRB lead to MVBA protocols with $O(Ln^2 + \kappa n^3 + n^3 \log n)$ communication.

We comment that if the underlying RBC and common coin protocols are instantiated using IT-secure protocols, then our MVBA protocols are also IT-secure.

Proof of our MVBA. We show that the MVBA protocol presented in Sec. 5.2 achieves external validity, agreement, integrity, and termination.

Theorem 5.1 (External validity). *Every correct replica that terminates mvba-decides v such that $Q(v)$ holds.*

Proof. If any correct replica *mvba-decides* v, it has *raba-decided* 1. Hence, according to the validity property of RABA, at least one correct replica has *raba-proposed* 1 or *raba-reproposed* 1. If the correct replica *raba-proposes* 1 or *raba-reproposes* 1, it has *r-delivered* v in the corresponding RBC instance. In the RBC instance, according to our specification of MVBA, every replica verifies whether $Q(v)$ holds upon receiving any value from another replica in the RBC phase. If a correct replica *r-delivered* v in an RBC instance, then at least one correct replica has previously received v and verified that $Q(v)$ holds. Hence, if one correct replica *mvba-decides* v, then at least one correct replica has verified that $Q(v)$ holds. As $Q(v)$ is verifiable across all correct replicas, every correct replica that *mvba-decides* v must have that $Q(v)$ holds. □

Theorem 5.2 (Agreement). *If a correct replica mvba-decides v, then any correct replica that terminates mvba-decides v.*

Proof. Assume a correct replica p_i *mvba-decides* v. Then p_i must have *raba-decided* 1 in some RABA$_r$ for iteration $r > 0$, and for any iteration $\hat{r} < r$, it holds that RABA$_{\hat{r}}$ outputs 0.

Now we assume another correct p_j *mvba-decides* v'. We now prove by contradiction that $v' = v$. We distinguish two cases for p_j: p_j *mvba-decides* in iteration $r' = r$; p_j *mvba-decides* in iteration $r' \neq r$.

Case 1: p_j *mvba-decides* v' in round r. In this case, we assume that p_j obtains k' from the Election() function and RBC$_{k'}$ outputs v'. As Election() outputs a *common* coin for the same input r, it must hold that $k = k'$. Hence, if $v \neq v'$, the agreement property of RBC would be violated.

Case 2: If $r' > r$, then according to our protocol, p_j *raba-decides* 1 in RABA$_{r'}$ and *raba-decides* 0 for any lower iteration r', including r. This violates the agreement property of RABA$_r$. Similarly, the argument holds for the case $r' < r$. □

Theorem 5.3 (Integrity). *If all replicas follow the protocol, and if a correct replica mvba-decides v such that $Q(v)$ holds, then some replica mvba-proposed v such that $Q(v)$ holds.*

Proof. If a correct replica p_i *mvba-decides* v, it *raba-decides* 1 in some RABA$_r$ and *r-delivers* v in RBC$_k$ where k is the corresponding common coin. According to the integrity property of RBC, v was previously broadcast by replica p_k. □

Lemma 5.4. *If all correct replicas enter the iteration phase, then for at least $f + 1$ RBC instances, at least $f + 1$ correct replicas have r-delivered some values.*

Proof. Instead of *directly* bounding the number of correct replicas that have *r-delivered* some values, we bound the number of instances where fewer than $f + 1$ correct replicas have *r-delivered* some values. First, we observe that all correct replicas *r-deliver* some values for $(2f + 1)(2f + 1)$ RBC instances in total. As there are at most $(3f+1)(2f+1)$ instances for correct replicas, the total number of instances where correct replicas do not *r-deliver* some values are upper bounded by $(3f+1)(2f+1)-(2f+1)(2f+1) = 2f^2+f$. Hence, the number of RBC instances where fewer than $f+1$ correct replicas have *r-delivered* some values is bounded by $\frac{2f^2+f}{f+1} < \frac{2f^2+2f}{f+1} = 2f$. That is, the number of RBC instances where at least $f + 1$ correct replicas have *r-delivered* some values is at least $f + 1$. □

We comment that PACE observed a similar claim that works in a context with a different goal.

Theorem 5.5 (Termination). *If all correct replicas are activated and all messages sent among correct replicas have been delivered, then all correct replicas mvba-decide.*

Proof. If all correct replicas start the protocol, then according to the validity property of RBC, every correct replica completes at least $n - f$ RBC instances. During the iteration phase, we first show that any iteration r will complete and then show that eventually, some RABA$_r$ will output 1. For each iteration r, we assume that k is returned by the Election() function in iteration r.

We first show that every iteration r will complete. For each iteration r, we distinguish three cases: 1) all correct replicas have *r-delivered* some value in RBC$_k$; 2) at least one correct replica has *r-delivered* some value in RBC$_k$, and at least one correct replica has not *r-delivered* any value in RBC$_k$; 3) none of the correct replicas have *r-delivered* any value in RBC$_k$.

Case 1: Due to the unanimous termination property, it holds that RABA$_r$ terminates.

Case 2: If at least one correct replica has *r-delivered* some value in RBC$_k$, then from the agreement property of RBC, any correct replica eventually *r-delivers* some value. According to our protocol, any correct replica that provides 0 as RABA input (in which case it has not *r-delivered* any value in RBC$_k$ when the iteration begins) will eventually *raba-repropose* 1. Thus, the biased termination condition of RABA will eventually be satisfied. Hence, RABA$_r$ will terminate, and iteration r will eventually complete.

Case 3: If none of the correct replicas *r-deliver* any value in RBC$_k$, iteration r will complete due to the unanimous termination property of RABA. Otherwise, if at least one correct replica later *r-delivers* some value in RBC$_k$, then according to case 2, iteration r will complete (due to the biased termination of RABA).

We now prove that eventually, in some iteration r, RABA$_r$ outputs 1, so the protocol will terminate. From Lemma 5.4, for at least $f + 1$ RBC instances, at least $f + 1$ correct replicas have *r-delivered* some value after they enter the iteration phase. Let I be the set of the $f + 1$ RBC instances. Due to the biased validity property of RABA, RABA$_r$ outputs 1. As Election() outputs a uniformly random coin for each iteration, we have that with probability $\frac{f+1}{3f+1} \approx \frac{1}{3}$, it holds that $k \in I$.

After RABA$_r$ outputs 1, every correct replica waits for the output of RBC$_k$. If RABA$_r$ outputs 1, at least one correct replica has *raba-proposed* 1 or *raba-reproposed* 1. (Otherwise, the unanimous termination property of RABA would be violated.) Therefore, at least one correct replica has *r-delivered* some value in RBC$_k$. From the agreement property of RBC, every correct replica eventually *r-delivers* some value in RBC$_k$ and then *mvba-decides.* □

Theorem 5.6. *Our MVBA protocol has expected $O(1)$ running time.*

Proof. From Lemma 5.4, for at least $f+1$ RBC instances, at least $f + 1$ correct replicas have *r-delivered* some value after they enter the iteration phase. Due to the biased validity property of RABA, RABA$_r$ outputs 1. Since Election() outputs a uniformly distributed random coin for each iteration, it holds that with probability $\frac{f+1}{3f+1} \approx \frac{1}{3}$, we have $k \in I$. Therefore, the protocol has expected $O(1)$ running time. □

5.3 MVBA with the Quality Property

We show that by adding two additional communication steps, we can build MVBA with the quality property. As illustrated in Figure 2b, we introduce an echo and reply procedure between the RBC phase and the iteration phase. In pseudocode, all we need to do is to replace line 03 in Figure 1 using the lines of code in Figure 3. Each replica p_i now additionally maintains one vector W_i used to track the set of completed RBC instances. After p_i completes

```
MVBA with quality property
  Initialization: $W_i \leftarrow [0]^n$
---------------------------------------------------------------
replace line 03 in Figure 1 using the following lines:
01  upon r-delivering $v_j$ for RBC$_j$
02    $W_i[j] \leftarrow 1$
03  wait for $n - f$ RBC instances to complete
04    send (ECHO, $W_i$) to all replicas
05  upon (ECHO, $W_j$) from $p_j$ such that there are $n - f$ 1's in $W_j$
06    if for any $W_j[l] = 1$, RBC$_l$ outputs some value
07      send (REP, $i$) to $p_j$
08  wait for $n - f$ (REP) messages
```

Figure 3: Our MVBA protocol with the quality property. The code for p_i.

RBC$_j$, it sets $W_i[j]$ as 1 (lines 01-02). When p_i completes $n - f$ RBC instances, it broadcasts the W_i vector to all replicas (03-04). It then expects to receive $n - f$ (REP) messages from the replicas, representing that $n - f$ replicas have also *r-delivered* some values for the same $n - f$ RBC instances. To achieve this goal, for each replica p_i, upon receiving W_j from p_j, p_i first checks whether the vector contains $n - f$ 1's. Then for each $W_j[l] = 1$, p_i waits until some value is *r-delivered* in RBC$_l$ (lines 06-07). After that, p_i sends a (REP) message to p_j (line 08). Upon receiving $n - f$ (REP) messages, p_i enters the iteration phase.

The protocol achieves quality, mainly because upon receiving a vector W_j, each replica verifies whether it has *r-delivered* some value in each RBC$_l$ instance for $W_j[l] = 1$. Hence, if a correct replica p_j receives $n - f$ (REP) messages, at least $n - f$ replicas must have *r-delivered* the same values in the same set of $n - f$ RBC instances due to the agreement property of RBC. Hence, with probability $\frac{2f+1}{3f+1} \approx \frac{2}{3}$, replicas *mvba-decide* a value from a correct replica. The protocol thus achieves quality.

The way of achieving quality can be viewed as using the technique from [2, 48] and using the agreement property in RBC. We show the correctness of the MVBA in this subsection in Appendix B.

5.4 Tailored MVBA from Weak RBC

While we can use EFBRB and CCBRB for low communication cost in our MVBA protocol, both of them rely on online error correcting (OEC) code and may suffer some degraded performance during failures (due to the "trial-and-error" OEC pattern). Additionally, EFBRB has significantly more steps than the classic RBC protocols [11, 14].

In this section, we provide a more practical MVBA construction that achieves $O(Ln^2 + \kappa n^3)$ communication. While its communication is the same as that of using CCBRB, our protocol in this subsection outperforms the CCBRB instantiation (in Sec. 5.2) in terms of both concrete communication cost and computational efficiency. First, the construction in this subsection does not use erasure coding or online error correcting code. Hence, the hidden constant in the bulk data term Ln^2 is 1 (namely $1Ln^2$) instead of 3 (at least $3Ln^2$ or more if using erasure coding or error correcting code). Namely, for a large L, the communication cost of this construction is about 1/3 of that of CCBRB-based MVBA. Second, as the construction in this subsection uses hashes only and does not use online error correction, it is computationally more efficient in both gracious and uncivil executions.

```
A Practical MVBA Construction
  Input: Value $v_i$ such that a global predicate $Q(v_i)$ holds
  Output: Value $v_k$ proposed by $p_k$
  Initialization: $r \leftarrow 0, T_i \leftarrow [\perp]^n$
---------------------------------------------------------------
01 upon event mvba-propose($v_i$)
02   wr-broadcast($v_i$) for WRBC$_i$                  {▷ WRBC phase}
03   wait for $n - f$ WRBC instances to complete  {▷ Iteration phase}
04   repeat
05     $k \leftarrow$ Election()
06     if some value is wr-delivered in WRBC$_k$
07       raba-propose 1 for RABA$_r$
08     else
09       raba-propose 0 for RABA$_r$
10       if later some value is r-delivered in RBC$_k$
11         raba-repropose 1 for RABA$_r$
12     if RABA$_r$ outputs 1
13       wait for WRBC$_k$ to wr-deliver value $h_k$
14       if $T_i[k] \neq \perp$
15         broadcast (VALUE, $T_i[k]$)
16       else
17         wait for (VALUE, $v_k$) such that $Hash(v_k) = h_k$
18         $T_i[k] \leftarrow v_k$
19       terminate the protocol and mvba-decide($T_i[k]$)
20     $r \leftarrow r + 1$
---------------------------------------------------------------
21 upon event wr-broadcast($v_j$) for instance WRBC$_j$
22   {replica $p_j$ broadcasts (SEND, $v_j$)}
23   upon receiving (SEND, $v_j$) from $p_j$
24     if $Q(v_j)$ holds
25       $T_i[j] \leftarrow v$
26       broadcast (ECHO, $Hash(v_j)$)
28     else
29       store the message until $Q(v_j)$ holds
30   upon receiving $n - f$ matching (ECHO, $h$)
31     broadcast (READY, $h$)
32   upon receiving $f + 1$ matching (READY, $h$) and (READY) message
   has not been sent yet
33     broadcast (READY, $h$)
34   upon receiving $n - f$ matching (READY, $h$)
35     if $Hash(T_i[j]) \neq h$
36       $T_i[j] \leftarrow \perp$
37     wr-deliver $h$
```

Figure 4: A Practical MVBA protocol. Code is for p_i.

We show our tailored MVBA construction in Figure 4. The protocol relies on a weak RBC primitive, which we call WRBC. The workflow of WRBC (lines 21-37) is similar to the 3-phase Bracha's broadcast, but we only use hashes in the second and the third phases. (WRBC appears implicitly and partly used in, e.g., [26], and we claim no novelty about WCBC itself.) Concretely, at lines 21-37, the sender p_j in each WRBC$_j$ first broadcasts its input v_j in

(SEND) messages. Upon receiving the value v_j from p_j, each replica p_i verifies whether the predicate $Q(v_j)$ holds. If so, p_i updates $T_i[j]$ as v_j and then broadcasts a (ECHO, $Hash(v_j)$) message (lines 25-26). Otherwise, p_i stores the (SEND) message and processes it until $Q(v_j)$ holds (lines 28-29). Upon receiving $n - f$ (ECHO, h) messages with the same hash value h, each replica broadcasts a (READY, h) message (lines 30-31). If a replica p_i receives $f + 1$ (READY, h) messages but has not sent a (READY) message, p_i also broadcasts a (READY, h) message (lines 32-33). Upon receiving $2f + 1$ (READY, h) messages, p_i *wr-delivers* h (line 37). If some h is *wr-delivered* but the hash of $T_i[j]$ is not h, p_j sets $T_i[j]$ as $\perp$ (lines 35-36).

We now describe our tailored MVBA protocol. There are two major changes on top of our MVBA protocol in Sec. 5.2. First, in the RBC phase, we use WRBC instead of the standard RBC, where for each WRBC$_j$, the sender p_j *wr-broadcasts* a value v_j and correct replicas *wr-deliver* $h_j = Hash(v_j)$. Second, as each WRBC instance outputs the hash of the value instead of the original value broadcast by the sender, we need to retrieve the value after replicas reach an agreement in the iteration phase. In particular, in some iteration r where k is the output of the Election() function, after RABA$_r$ outputs 1, each replica first waits for WRBC$_k$ to output some value h_k and then starts the retrieval (lines 13-18). If p_i has some value in $T_i[k]$, it broadcasts a (VALUE, $T_i[k]$) to all replicas. If p_i does not hold a $T_i[k]$ value, it waits to receive a (VALUE, v_k) such that $Hash(v_k) = h_k$ and then sets $T_i[k]$ as v_k. After $Hash(T_i[k]) = h_k$, p_i terminates the protocol and *mvba-decides* $T_i[k]$ (line 19).

The communication bottleneck of this protocol is the WRBC phase, as the communication cost for other steps (including the retrieval step) is $O(\kappa n^2)$. For each WRBC instance, the sender broadcasts a message (length L), and replicas exchange hashes of values in the second phase and the third phase, so each instance has $O(n^2)$ messages and $O(Ln + \kappa n^2)$ communication. As there are n parallel WRBC instances, our practical MVBA construction has $O(Ln^2 + \kappa n^3)$ communication. The correctness of our protocol is similar to that of our MVBA in Sec. 5.2. In particular, each WRBC protocol WRBC$_j$ can guarantee that if a correct replica *wr-delivers* some value h, every correct replica eventually *wr-delivers* h. Furthermore, at least $f+1$ correct replicas must set their $T_i[j]$ as value v such that $Hash(v) = h$. Hence, the value v can be retrieved by any correct replicas. We prove the correctness of this tailored MVBA construction in Appendix C.

6 OUR ACS APPROACH

6.1 Overview

We now present our ACS protocol with $O(1)$ time and $O(n^3)$ messages. At the core of our ACS protocol is the reduction of ACS to our MVBA construction with a specific predicate. In particular, we use n parallel RBC instances for replicas to disseminate their *acs-proposed* values. Then each replica *mvba-proposes* a vector of $n - f$ bits, representing the $n - f$ completed RBC instances. Crucially, we define the global predicate of MVBA as the following: given a proposal with $n - f$ bits, each replica considers the proposal valid only if it has completed the same $n - f$ RBC instances. (As we commented earlier, the predicate depends on the state of a particular replica: it is possible that the predicate does not hold at the beginning, but will hold at some point.) In this way, we can guarantee that the output of ACS consists of at least $n - f$ *acs-proposed* values, satisfying the validity property of ACS. As our MVBA component completes in expected $O(1)$ time, our ACS protocol also terminates in expected constant time.

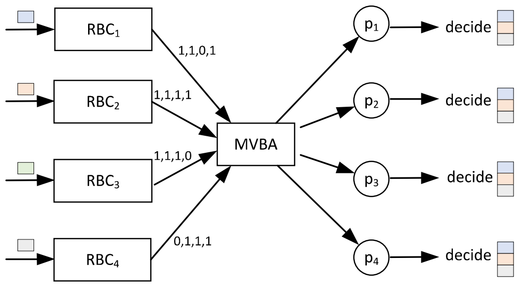


Figure 5: Our ACS protocol.

```
ACS
Input: value $v_i$
Output: $n - f$ values $\mathbf{v}$, among which at least $n - 2f$ are proposed
        by correct replicas
Initialization: $r \leftarrow 0, W \leftarrow [0]^n, \mathbf{v} \leftarrow \perp$
Let $Q$ be the following predicate for MVBA:
  Given value $W_j$ mvba-proposed by some replica $p_j$, $Q(W_j, W) \equiv$
  $(|W_j| = n$ and for at least $n - f$ $l$ such that $1 \le l \le n$, $W_j[l] = 1$
  and $W_j \subseteq W)$
---------------------------------------------------------------------
01 upon event acs-propose($v_i$)
02   r-broadcast $v_i$ for RBC$_i$                        {▷ RBC phase}
03   upon r-delivering $v_j$ for RBC$_j$
04     $W[j] \leftarrow 1$
05   wait for $n - f$ RBC instances to complete          {▷ MVBA phase}
06   $W_i \leftarrow W$
07   mvba-propose($W_i$) with predicate $Q(W_j, W)$
08   upon event mvba-decide($W_k$)
09     for every $W_k[j] = 1$
10        wait for RBC$_j$ to r-deliver value $v_j$
11        $\mathbf{v} \leftarrow \mathbf{v} \cup v_j$
12     terminate the protocol and acs-decide($\mathbf{v}$)
```

Figure 6: The ACS protocol. Code is for p_i.

6.2 The ACS Protocol

We describe our ACS protocol in Figure 5 and the pseudocode in Figure 6. Our ACS protocol has two phases: an RBC phase and an MVBA phase.

RBC phase (lines 02-04). Each replica p_i holds an input v_i. Upon the event *acs-propose*(v_i), p_i *r-broadcasts* v_i. Upon completing an RBC instance RBC$_j$, p_i sets $W[j]$ as 1 (lines 03-04), where W is a global map used to track the status of the RBC instances.

MVBA phase (lines 05-12). Replica p_i enters the MVBA phase after completing $n - f$ RBC instances. Each replica p_i sets W_i as W and uses W_i as input for MVBA with a predicate $Q(W_j, W)$ (lines

06-07). Note that the 1's in W for each p_i continue to grow as more RBC instances complete. The value *mvba-proposed* by each p_i is a snapshot of W_i with at least $n - f$ RBC instances completed.

We define a global predicate $Q(W_j, W)$ for each value W_j *mvba-proposed* by p_j as follows. First, each W_j is a n-bit vector. Second, W_j consists of at least $n-f$ 1's, representing the $n-f$ RBC instances that replica p_j has completed. Third, $W_j \subseteq W$. Namely, for each $W_j[l] = 1$, replica p_i must wait until it has *r-delivered* some value in RBC$_l$. Hence, in MVBA, every replica may need to wait until the global predicate is satisfied for each *mvba-proposed* message. As discussed previously in Sec. 5, the predicate depends on the internal state W of each replica p_i; it is possible that $Q(W_j, W)$ fails to hold when p_i receives the *mvba-proposed* value by p_j, but $Q(W_j, W)$ will hold at some point as p_i *r-delivers* more RBC instances.

After p_i provides some input to MVBA, it waits for the output of MVBA (line 08). According to the integrity property of MVBA, the output of MVBA W_k must be proposed by some replica such that $Q(W_k, W)$ is satisfied. At lines 09-11, for each $W_k[j] = 1$, p_i waits for the output of RBC$_j$, v_j. Then v_j is added to a set $\mathbf{v}$. After all such RBC instances complete, p_i *acs-decides* $\mathbf{v}$ (line 12).

Complexity and discussion. Our ACS protocol terminates in expected constant time, as the underlying MVBA protocol runs in expected $O(1)$ time. Moreover, our ACS protocol clearly has $O(n^3)$ messages. The communication cost of our ACS protocol depends on the underlying RBC protocol, as the input length to MVBA is only n. For instance, our ACS protocol with CT RBC has $O(Ln^2 + \kappa n^3 \log n)$ communication.

In contrast to BKR ACS and the state-of-the-art PACE ACS that terminate in $O(\log n)$ time and need n (R)ABA instances, our ACS protocol has $O(1)$ time and uses only $O(1)$ (R)ABA instances.

Note that as the input of MVBA indicates that at least $n - 2f$ *acs-proposed* values from correct replicas will eventually be *acs-decided*, we do not need the quality property for the underlying MVBA protocol.

We prove the correctness of our ACS protocol in Appendix D.

7 A PRACTICAL ACS INSTANTIATION

We use CT RBC as the underlying RBC protocol in ACS. We use our tailored MVBA protocol in Sec. 5.4 as our MVBA protocol. Our tailored MVBA protocol internally uses the hash-based WRBC protocol in Sec. 5.4 and Pisa RABA protocol [50]. We call the resulting instantiation FIN.

8 IMPLEMENTATION AND EVALUATION

We implemented FIN in Golang. To make a fair comparison, we implemented PACE [50] in the same library, the most efficient ACS construction of the same type. Our implementation involves around 9,000 LOC for the protocols and about 1,000 LOC for evaluation.

We use gRPC as the communication library. We use HMAC to realize the authenticated channel and use SHA256 as the underlying hash function. We implement threshold PRF [13] to realize common coins for RABA and the random leader election protocol. We use a Golang-based erasure coding library [1] to implement CT RBC.

We evaluate the performance of our protocols on Amazon EC2 using up to 121 virtual machines (VMs). We use *m5.xlarge* instances for our evaluation. The m5.xlarge instance has four virtual CPUs and 16GB memory. We deploy our protocols in the WAN setting, where replicas are evenly distributed across different regions: us-west-2 (Oregon, US), us-east-2 (Ohio, US), ap-southeast-1 (Singapore), and eu-west-1 (Ireland).

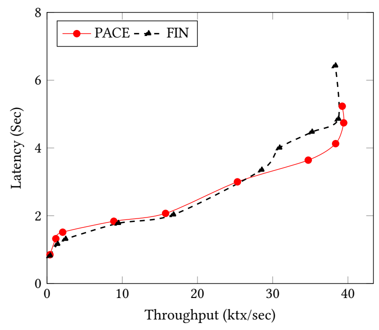
We conduct the experiments under different network sizes and batch sizes. We use f to denote the network size; in each experiment, we use $3f + 1$ replicas in total. We use b to denote the batch size, where each replica proposes b transactions in each epoch (i.e., one ACS instance). For each experiment, we run five epochs and report the average performance (for both throughput and latency). The default transaction size is 250 bytes. We also additionally evaluate the performance using a transaction size of 100 bytes and report the performance in Appendix A.

We summarize our main evaluation results as follows.

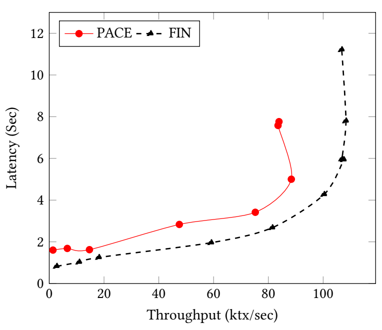
- We evaluate throughput vs. latency, and the peak throughput varying f from 1 to 40. Throughput is the number of transactions processed in each second. Latency is the *consensus* latency at the replica side, from the time when a replica *acs-proposes* to the time it *acs-decides*. Our results show that the performance of FIN is marginally lower than PACE for $f \leq 5$. When f is larger than 5, FIN consistently outperforms PACE, achieving higher throughput and lower latency. The performance difference between the two protocols drastically increases as f grows. For instance, when $f = 40$, the peak throughput of FIN is 3.41x that of PACE.
- The latency breakdown experiments that we carefully designed explain well why FIN outperforms PACE. In particular, we show that the RABA phase for FIN (with constant RABA instances) occupies only 1.23%-5.22% of the overall latency, in sharp contrast to PACE, where its RABA phase (with n parallel RABA instances) occupies 15.10%-83.66% of the total runtime.
- We evaluate the performance of the protocols under different failure scenarios. Our results show that FIN is highly robust against various failures.
- We additionally evaluate the performance with a smaller transaction size of 100 bytes, where we find that FIN outperforms PACE in a more significant manner. For instance, when $f = 20$, the throughput of FIN with 100-byte transactions is 2.14x that of PACE.

Throughput vs. Latency, peak throughput, and scalability. We report the throughput vs. latency results for $f = 1, 10, 20, 30, 40$ in Figure 7a-7e. For $f = 1$, FIN has only marginally higher latency and slightly lower throughput than PACE; in particular, the peak throughput of FIN is only 1.3% lower than PACE for $f = 1$. The reason why PACE is slightly more efficient than FIN for $f = 1$ is that FIN involves two phases of RBC (one for ACS and one inside MVBA), while the $n = 4$ parallel RABA instances do not bottleneck the performance in PACE for such a small f. When $f > 5$, FIN consistently outperforms PACE in terms of latency, peak throughput, and latency vs. throughput. The performance gain for FIN is clearly due to the constant-time termination and the constant number of RABA instances in FIN.

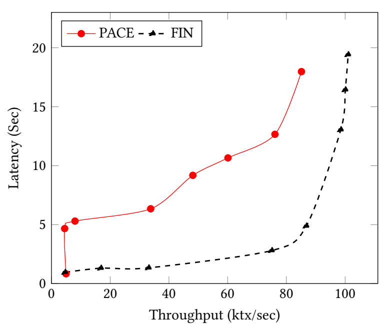
We report the peak throughput of PACE and FIN in Figure 7f. As shown in Figure 7f, FIN consistently outperforms PACE for $f \geq 10$. The performance difference between FIN and PACE becomes increasingly significant as f increases. For instance, when $f = 30$,



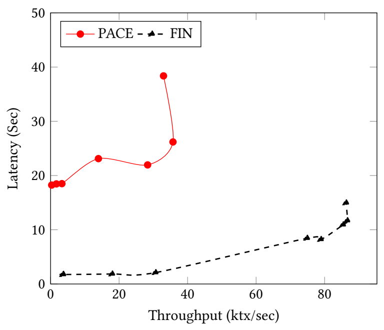
(a) Latency vs. throughput for $f = 1$.



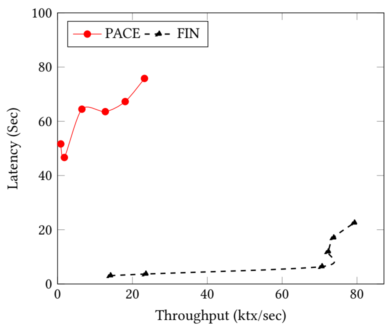
(b) Latency vs. throughput for $f = 10$.



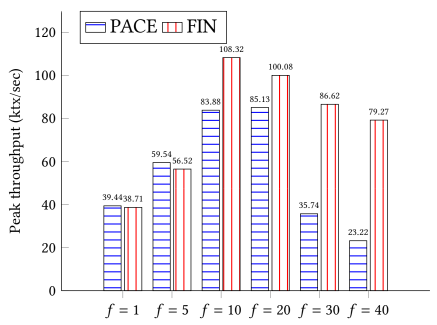
(c) Latency vs. throughput for $f = 20$.



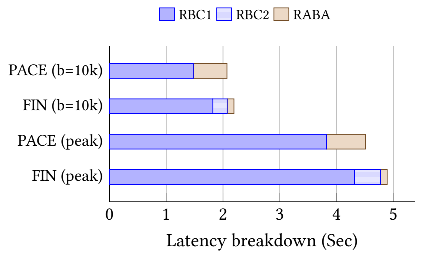
(d) Latency vs. throughput for $f = 30$.



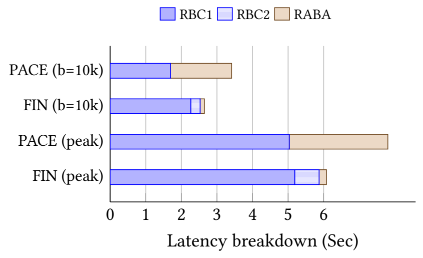
(e) Latency vs. throughput for $f = 40$.



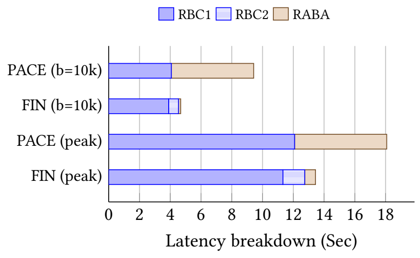
(f) Peak throughput as f grows.



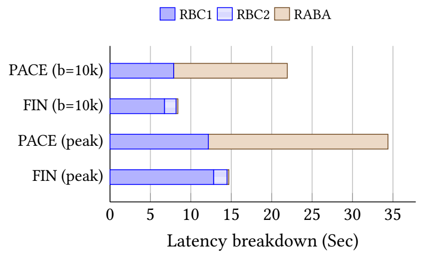
(g) Latency breakdown for $f = 1$.



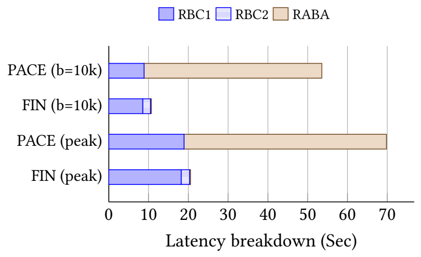
(h) Latency breakdown for $f = 10$.



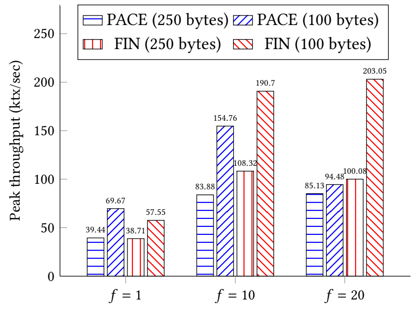
(i) Latency breakdown for $f = 20$.



(j) Latency breakdown for $f = 30$.



(k) Latency breakdown for $f = 40$.



(l) Performance with different transaction sizes.

Figure 7: Evaluation results of FIN and PACE.

the peak throughput of FIN is 2.42x that of PACE. When $f = 40$, the peak throughput of FIN is 3.41x that of PACE. We also observe that as f increases, the peak throughput of both PACE and FIN increases and decreases. Indeed, as f increases, the number of transactions delivered for both protocols increases, but when f further increases, the network bandwidth consumption dominates the performance. In our experiments, FIN achieves its highest throughput when $f = 10$, while PACE achieves its peak when $f = 20$.

Latency breakdown. To help understand why FIN outperforms PACE, we report the latency breakdown for the experiments. In FIN, there are three phases: the RBC phase for ACS (denoted as RBC1), the RBC phase inside MVBA (denoted as RBC2), and the iteration phase with a random leader election and one RABA instance at a time (denoted as RABA). In contrast, PACE has an RBC phase (the same as that in FIN, denoted as RBC1) and a RABA phase with n parallel RABA instances (denoted as RABA). Here the latency of RBC phase is measured from the beginning of the first RBC instance to the completion of the $n-f$-th RBC instance. Moreover, the latency of the RABA phase for PACE is measured from the beginning of the first RABA instance to the time ACS completes. Finally, the latency of the RABA phase for FIN is measured from the beginning of the iteration phase to the time when ACS completes. The latency breakdown experiments can explain why FIN outperforms PACE, help identify the bottleneck of the two protocols, and assist in understanding the scalability results.

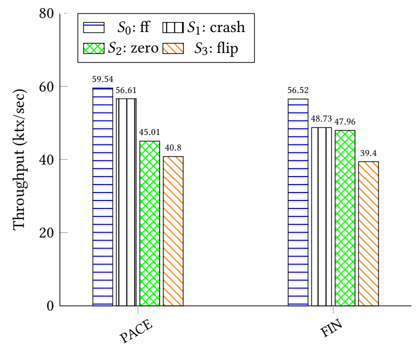
Figure 7g-7k reports the latency breakdown for FIN and PACE. We test two settings: a fixed one with b = 10,000, and the smallest batch size where both protocols achieve their peak throughput. We first observe that the RBC1 phases in the two protocols share almost the same latency. This is not surprising, as the RBC1 phase is the only phase that carries bulk data for both protocols, and we use CT RBC for both of them. Additionally, the latency percentage for the RBC2 phase in FIN is comparatively very low. This is because the RBC2 phase does not have any bulk data, and its input size is small (n bits). Hence, the RBC2 phase in FIN does not incur much overhead to the protocol.

In all cases, the RABA latency in PACE is much higher than that for FIN. For f =1 to 40, the latency of the RABA phase in PACE is 15.10%-83.66% of the overall latency. In contrast, the RABA phase in FIN occupies only 1.23%-5.22% of the total runtime.

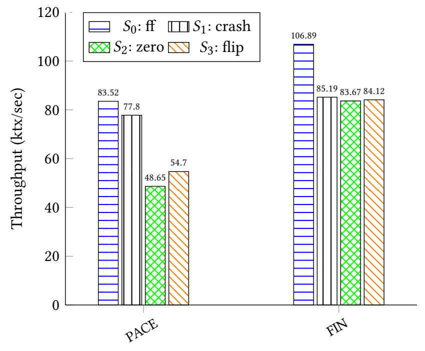
Moreover, the latency percentage of the RABA phase within the overall PACE latency becomes increasingly larger as f increases, but the RABA latency percentage in FIN remains steady despite an increasing f. Indeed, when f = 40, the RABA phase occupies 83.66% of the overall consensus latency for PACE. In contrast, the latency of the RABA phase in FIN is only 1.23% of the overall latency. In fact, even if we consider the latency caused by RBC2 and RABA (i.e., MVBA) in FIN, it only occupies 11.45% of the overall latency. This observation explains well why the performance difference between FIN and PACE becomes increasingly larger as f increases. Indeed, FIN only needs an expected constant number of RABA instances.

Performance under failures. To assess the robustness of FIN, we report the performance of FIN and PACE under various failure scenarios. Following prior works [50, 52], we consider the following scenarios, where the f faulty replicas are evenly distributed in the EC2 regions we use.

- S_0**: (failure-free)** In this scenario, all replicas are correct.
- S_1**: (crash)** In this scenario, we let f replicas crash by not participating in the protocols.
- S_2**: (Byzantine; keep voting 0)** In this scenario, we fail f replicas and let them keep voting for 0 in each step of RABA.
- S_3**: (Byzantine; flipping the RABA input)** In this scenario, we fail f replicas and ask them to always vote for a flipped value in RABA.



(a) f = 5.



(b) f = 10.

Figure 8: Throughput of the protocols in failure scenarios for b = 30,000.

We fix b to 30,000 and present the throughput for f = 5 in Figure 8a and f = 10 in Figure 8b. We choose these two settings because they are the settings where the two protocols share similar performance. First, for the crash failure scenario, the performance of both protocols degrades only slightly. For both Byzantine scenarios, the percentage for the performance degradation of FIN is lower than that of PACE. The performance of FIN in Byzantine scenarios degrades by 15.1%-30.2% compared to the failure-free scenario. In contrast, the performance of PACE degrades by 24.39%-41.74%. Notably, for f = 10, FIN under all failure scenarios outperforms PACE in its failure-free scenario.

Performance with different transaction sizes. We additionally evaluate the performance with a smaller transaction size of 100 bytes as shown in Figure 7l. In this setting, FIN outperforms PACE in a more significant manner. For instance, when f = 20, the throughput of FIN with 100 bytes transactions is 2.14x that of PACE, while for 250-bytes transactions, the throughput of FIN is only 17.56% higher. We also find that the throughput of FIN with 100 bytes transactions is roughly 2x that with 250-bytes transactions.

9 CONCLUSION

We present the first IT-secure and signature-free ACS protocol with $O(1)$ time and $O(n^3)$ messages, resolving a long-standing open problem in fault-tolerant distributed computing and cryptography. As a core ingredient in our ACS construction and a primitive of independent interests, we present the first signature-free MVBA protocols with $O(1)$ time and $O(n^3)$ messages. In contrast, existing signature-free MVBA protocols have $O(\log n)$ time and $O(n^3)$ messages. From the practical side, we implement a practical ACS protocol called FIN, and we demonstrate that FIN significantly outperforms the state-of-the-art BFT protocol of the same kind—PACE.

REFERENCES

[1] 2021. Reed-Solomon library. https://github.com/klauspost/reedsolomon. (2021).

[2] Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. 2019. Asymptotically Optimal Validated Asynchronous Byzantine Agreement. In *PODC*. ACM, 337–346.

[3] Andreea B. Alexandru, Erica Blum, Jonathan Katz, and Julian Loss. 2022. State Machine Replication under Changing Network Conditions. In *Asiacrypt*.

[4] Nicolas Alhaddad, Sourav Das, Sisi Duan, Ling Ren, Zhuolun Xiang Mayank Varia, and Haibin Zhang. 2022. Balanced Byzantine Reliable Broadcast with Near-Optimal Communication and Improved Computation. *PODC*.

[5] Renas Bacho and Julian Loss. 2022. On the Adaptive Security of the Threshold BLS Signature Scheme. In *ACM CCS*.

[6] Michael Backes, Fabian Bendun, Ashish Choudhury, and Aniket Kate. 2014. Asynchronous MPC with a strict honest majority using non-equivocation. In *Proceedings of the 2014 ACM symposium on Principles of distributed computing*. 10–19.

[7] Michael Ben-Or, Ran Canetti, and Oded Goldreich. 1993. Asynchronous secure computation. In *STOC*. ACM, 52–61.

[8] Michael Ben-Or and Ran El-Yaniv. 2003. Resilient-Optimal Interactive Consistency in Constant Time. *Distrib. Comput.* (2003).

[9] Michael Ben-Or, Boaz Kelmer, and Tal Rabin. 1994. Asynchronous secure computations with optimal resilience. In *Proceedings of the 13th annual symposium on Principles of distributed computing*. ACM, 183–192.

[10] Erica Blum, Chen-Da Liu-Zhang, and Julian Loss. 2020. Always have a backup plan: fully secure synchronous MPC with asynchronous fallback. In *Annual International Cryptology Conference*. Springer, 707–731.

[11] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Information and Computation* 75, 2 (1987), 130–143.

[12] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*. Springer, 524–541.

[13] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography. *Journal of Cryptology* 18, 3 (2005), 219–246.

[14] Christian Cachin and Stefano Tessaro. 2005. Asynchronous verifiable information dispersal. In *SRDS*. IEEE, 191–201.

[15] Ran Canetti. 1996. Studies in secure multiparty computation and applications. *Scientific Council of The Weizmann Institute of Science* (1996).

[16] Ran Canetti and Tal Rabin. 1993. Fast asynchronous Byzantine agreement with optimal resilience. In *STOC*, Vol. 93. Citeseer, 42–51.

[17] Dario Catalano and Dario Fiore. 2013. Vector commitments and their applications. In *International Workshop on Public Key Cryptography*. Springer, 55–72.

[18] Soma Chaudhuri. 1993. More choices allow more faults: Set consensus problems in totally asynchronous systems. *Information and Computation* 105, 1 (1993), 132–158.

[19] Annick Chopard, Martin Hirt, and Chen-Da Liu-Zhang. 2021. On communication-efficient asynchronous MPC with adaptive security. In *Theory of Cryptography Conference*. Springer, 35–65.

[20] Ashish Choudhury and Nikhil Pappu. 2020. Perfectly-Secure Asynchronous MPC for General Adversaries. In *International Conference on Cryptology in India*. Springer, 786–809.

[21] Ashish Choudhury and Arpita Patra. 2015. Optimally resilient asynchronous MPC with linear communication complexity. In *Proceedings of the 2015 International Conference on Distributed Computing and Networking*. 1–10.

[22] Ashish Choudhury and Arpita Patra. 2017. An Efficient Framework for Unconditionally Secure Multiparty Computation. *IEEE Transactions on Information Theory* 63, 1 (2017), 428–468.

[23] Miguel Correia, Nuno Ferreira Neves, and Paulo Veríssimo. 2006. From Consensus to Atomic Broadcast: Time-Free Byzantine-Resistant Protocols without Signatures. *Comput. J.* 49, 1 (2006), 82–96.

[24] Tyler Crain. 2020. Two More Algorithms for Randomized Signature-Free Asynchronous Binary Byzantine Consensus with t<n/3 and O(n^2) Messages and O(1) Round Expected Termination. *CoRR* abs/2002.08765 (2020). arXiv:2002.08765 https://arxiv.org/abs/2002.08765

[25] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical Asynchronous High-threshold Distributed Key Generation and Distributed Polynomial Sampling. *Cryptology ePrint Archive* (2022).

[26] Sourav Das, Zhuolun Xiang, and Ling Ren. 2021. Asynchronous Data Dissemination and Its Applications. In *CCS*.

[27] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew K. Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical Asynchronous Distributed Key Generation. In *IEEE Symposium on Security and Privacy*. IEEE, 2518–2534.

[28] Assia Doudou and André Schiper. 1998. Muteness detectors for consensus with Byzantine processes. In *Proceedings of the seventeenth annual ACM symposium on Principles of distributed computing*. 315.

[29] Sisi Duan, Michael K Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT made practical. In *CCS*. ACM, 2028–2041.

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

[31] Christoph U. Günther, Sourav Das, and Lefteris Kokoris-Kogias. 2022. Practical Asynchronous Proactive Secret Sharing and Key Refresh. Cryptology ePrint Archive, Paper 2022/1586. (2022).

[32] Bin Hu, Zongyang Zhang, Han Chen, You Zhou, Huazu Jiang, and Jianwei Liu. 2022. DyCAPS: Asynchronous Proactive Secret Sharing for Dynamic Committees. Cryptology ePrint Archive, Paper 2022/1169. (2022).

[33] Idit Keidar, Eleftherios Kokoris-Kogias, Oded Naor, and Alexander Spiegelman. 2021. All You Need is DAG. In *PODC*. ACM, 165–175.

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

[35] Benoît Libert, Marc Joye, and Moti Yung. 2016. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. *Theoretical Computer Science* 645 (2016), 1–24.

[36] Chao Liu, Sisi Duan, and Haibin Zhang. 2020. EPIC: Efficient Asynchronous BFT with Adaptive Security.. In *DSN*.

[37] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. 2019. HoneyBadgerMPC and AsynchroMix: Practical Asynchronous MPC and Its Application to Anonymous Communication. In *ACM CCS*.

[38] Y. Lu, Z. Lu, Q. Tang, and G. Wang. 2020. Dumbo-MVBA: Optimal Multi-Valued Validated Asynchronous Byzantine Agreement, Revisited. In *PODC*.

[39] Ethan MacBrough. 2018. Cobalt: BFT governance in open networks. *arXiv preprint arXiv:1802.07240* (2018).

[40] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *ACM CCS*. 31–42.

[41] Henrique Moniz, Nuno Ferreria Neves, Miguel Correia, and Paulo Verissimo. 2008. RITAS: Services for randomized intrusion tolerance. *IEEE transactions on dependable and secure computing* 8, 1 (2008), 122–136.

[42] Achour Mostefaoui, Hamouma Moumen, and Michel Raynal. 2014. Signature-free asynchronous byzantine consensus with t< n/3 and o (n 2) messages. In *Proceedings of the ACM symposium on Principles of distributed computing*. ACM, 2–9.

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

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

[45] Nuno Ferreira Neves, Miguel Correia, and Paulo Verissimo. 2005. Solving vector consensus with a wormhole. *IEEE Transactions on Parallel and Distributed Systems* 16, 12 (2005), 1120–1131.

[46] M. Pease, R. Shostak, and L. Lamport. 1980. Reaching Agreement in the Presence of Faults. *J. ACM* 27, 2 (1980), 228–234.

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

[48] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. HotStuff: BFT consensus with linearity and responsiveness. In *38th ACM symposium on Principles of Distributed Computing (PODC)*.

[49] Thomas Yurek, Zhuolun Xiang, Yu Xia, and Andrew Miller. 2022. Long Live The Honey Badger: Robust Asynchronous DPSS and its Applications. Cryptology ePrint Archive, Paper 2022/971. (2022).

[50] Haibin Zhang and Sisi Duan. 2022. PACE: Fully Parallelizable BFT from Reproposable Byzantine Agreement. In *CCS*.

[51] Haibin Zhang, Sisi Duan, Chao Liu, Boxin Zhao, Xuanji Meng, Shengli Liu, Yong Yu, Fangguo Zhang, and Liehuang Zhu. 2022. Practical Asynchronous Distributed Key Generation: Improved Efficiency, Weaker Assumption, and Standard Model. Cryptology ePrint Archive, Paper 2022/1678. (2022).

[52] Haibin Zhang, Sisi Duan, Boxin Zhao, and Liehuang Zhu. 2022. WaterBear: Practical Asynchronous BFT Matching Security Guarantees of Partially Synchronous BFT. Cryptology ePrint Archive, Paper 2022/021. (2022).

[53] Haibin Zhang, Chao Liu, and Sisi Duan. 2022. How to achieve adaptive security for asynchronous BFT? *J. Parallel and Distrib. Comput.* 169 (2022), 252–268.

A ADDITIONAL EVALUATION RESULTS

In Figure 9, we present the throughput vs. latency results for the transaction size of 100 bytes for $f = 1, 10, 20$. While the trend in this setting is similar to that with the transaction size of 250 bytes, FIN outperforms PACE in a more drastic way. Despite the case for $f = 1$ where the peak throughput of FIN is 17.4% lower than that of PACE, FIN achieves 1.23x and 2.14x the peak throughput of PACE, for $f = 10$ and $f = 20$, respectively.

B PROOF OF OUR MVBA PROTOCOL WITH QUALITY

We show that the MVBA protocol presented in Sec. 5.3 additionally achieves the quality property. All the other properties except termination follow from our MVBA without the quality property. Therefore, we prove quality and termination in this section.

LEMMA B.1. *If a correct replica enters the iteration phase, then for at least* $2f + 1$ *RBC instances, at least* $f + 1$ *correct replicas have r-delivered some values.*

PROOF. If a correct replica p_i enters the iteration phase, it receives $n - f$ (REP) message. Prior to that, p_i has broadcast (ECHO, W_i) where W_i consists of at least $n-f$ 1's, i.e., p_i has completed $n-f$ RBC instances. Each replica p_j replies with a (REP) message only if for any $W_i[l] = 1$, p_j has also *r-delivered* some value in RBC$_l$. Hence, for any $W_i[l] = 1$, at least $f + 1$ correct replicas have *r-delivered* some value. □

THEOREM B.2 (QUALITY). *The probability of mvba-deciding a value that was proposed by a correct replica is at least 1/2.*

PROOF. According to Lemma B.1, for at least $2f+1$ RBC instances, at least $f + 1$ correct replicas have *r-delivered* some value. From the biased validity property of RABA, if any of the $2f + 1$ RBC instances is selected by the Election() function, RABA will output 1. So every correct replica then *mvba-decides*. Therefore, the probability that the decided value was proposed by an adversary is bounded by $\frac{f+1}{3f+1}$. As the probability of deciding a value proposed by a faulty replica is at most $\Sigma_{k=1}^{\infty}(1/3)^k = 1/2$, the probability of deciding a value that was proposed by a correct replica is at least $1/2$. □

THEOREM B.3 (TERMINATION). *If all correct replicas are activated and all messages sent among correct replicas have been delivered, then all correct replicas mvba-decide.*

PROOF. If all correct replicas are activated, each correct replica starts one RBC instance. According to the validity property of RBC, at least $n - f$ RBC instances started by the $n - f$ correct replicas will eventually complete. Then, each correct replica p_i sends a (ECHO, W_i) message, every replica replies only if it has *r-delivered* some value in RBC$_l$ for any $W_i[l] = 1$. According to the agreement property of RBC, we know that for the message sent by p_i, every correct replica eventually replies with a (REP) message. Accordingly, every correct replica eventually enters the iteration phase.

During the iteration phase, we first prove that every iteration r completes and then show that eventually some RABA$_r$ outputs 1. For each iteration r, k is returned by the Election() function.

The proof that every iteration r completes is similar to that for the protocol without quality. We include one additional echo-and-reply procedure, where every replica sends its W_i to all replicas and proceeds to the next phase if it receives $n - f$ replies. According to the agreement property of RBC, we know that every correct replicas eventually *r-delivers* some values in the same set of RBC instances in W_i. Thus, every correct replica completes each epoch.

We now prove that eventually, in some iteration r, RABA$_r$ outputs 1, so the protocol terminates. From Lemma B.1, for at least $2f + 1$ RBC instances, at least $f + 1$ correct replicas have *r-delivered* some value after they enter the iteration phase. Let I be the set of the $2f + 1$ RBC instances. Due to the biased validity property of RABA, RABA$_r$ outputs 1. So with probability $\frac{2f+1}{3f+1} \approx \frac{2}{3}$, it holds that $k \in I$.

After RABA$_r$ outputs 1, every correct replica waits for the output of RBC$_k$. Note that if RABA$_r$ outputs 1, at least one correct replica has *raba-proposeed* 1 or *raba-reproposed* 1. This is due to the unanimous termination property of RABA. Therefore, at least one correct replica has *r-delivered* some value in RBC$_k$. Due to the agreement property of RBC, every correct replica eventually *r-delivers* some value in RBC$_k$ and then *mvba-decides*. □

C PROOF OF OUR TAILORED MVBA

In this section, we prove the correctness of our tailored MVBA protocol. We first show a few lemmas about WRBC and then show the correctness of our tailored MVBA construction. As external validity is the same as the MVBA protocol presented in Sec. 5.2, we focus on agreement, integrity, and termination in this section.

LEMMA C.1. *If a correct replica wr-delivers h and another correct replica wr-delivers h′, then* $h = h'$.

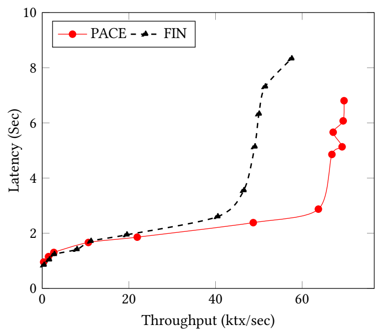
PROOF. If a correct replica p_i *wr-delivers* h, it receives $n - f$ (READY, h). If another correct replica p_j *wr-delivers* h', it receives $n - f$ (READY, h'). Therefore, at least one correct replica has sent both (READY, h) and (READY, h'), a contradiction to the fact that every correct replica only sends a (READY) message once. □

LEMMA C.2. *If a correct replica wr-broadcasts a value v, every correct replica eventually wr-delivers h such that* $h = Hash(v)$.

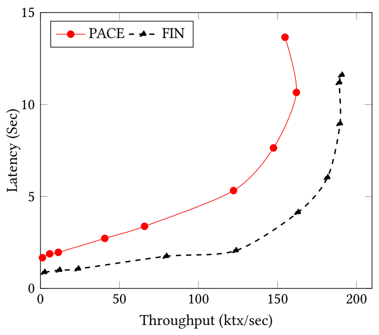
PROOF. For any WRBC instance, if the sender is correct, it is straightforward to see that every correct replica receives the same (SEND, v) message, broadcasts a (ECHO, m)essage, and receives $n - f$ (ECHO, $Hash(v)$) messages. Then every correct replica eventually sends (READY, $Hash(v)$) and will never receive $f+1$ (READY) messages with a value different from $Hash(v)$. Hence, every correct replica eventually *wr-delivers* $h = Hash(v)$. □

LEMMA C.3. *For any WRBC instance, if a correct replica wr-delivers some value h, any correct replica eventually wr-delivers some value.*

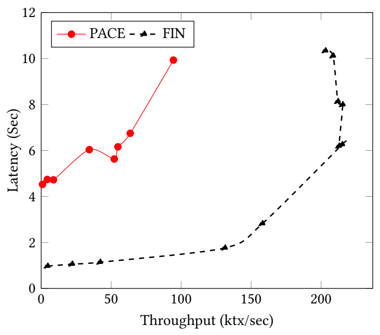
PROOF. If a correct replica *wr-delivers* h, it receives $n-f$ (READY, h) messages, among which are least $f + 1$ are sent by correct replicas. Thus, any correct replica that receives $f + 1$ (READY, h) messages but has not sent any (READY) message will also send a (READY, h). Therefore, every correct replica eventually receives $n - f$ (READY, h) messages and *wr-delivers*. □



(a) Latency vs. throughput for $f = 1$.



(b) Latency vs. throughput for $f = 10$.



(c) Latency vs. throughput for $f = 20$.

Figure 9: Evaluation results for transaction size of 100 bytes.

Lemma C.4. *If a correct replica wr-delivers some value h in WRBC$_j$, at least $f+1$ correct replica receives (*Send*, v) such that $Hash(v)$.*

Proof. If a correct replica *wr-delivers* h, it receives $n-f$ (Ready, h), among which are least $f+1$ are sent by correct replicas. Any of the correct replicas receive $n-f$ (Echo, h), among which at least $f+1$ are sent by correct replicas. The correct replicas must have received (Send, v) from p_j such that $Hash(v) = h$. □

For completeness, below, we provide self-contained proof for agreement, integrity, and termination. Note that the proof is very similar to that for our MVBA protocol in Sec. 5.2.

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

Proof. If a correct replica p_i *mvba-decides* v, we assume it has *raba-decided* 1 in RABA$_r$ for some iteration $r > 0$ and for any iteration $\hat{r} < r$, RABA$_{\hat{r}}$ outputs 0. Furthermore, if k is returned by the Election() function in iteration r, WRBC$_k$ outputs h and p_i receives some v (from p_k or from a (Value) message) such that $H(v) = h$.

We assume that another correct replica p_j *mvba-decides* $v' \neq v$ and prove the theorem by contradiction. We consider two cases: p_j *mvba-decides* in iteration r; p_j *mvba-decides* in iteration $r' \neq r$.

Case 1: If p_j *mvba-decides* v' in round r, it obtains k' from Election() function and WRBC$_{k'}$ outputs h'. Then p_j either receives value v' from $p_{k'}$ such that $Hash(v') = h'$ or receives v' from another replica in a (Value) message. As Election() outputs a common coin, it must hold that $k = k'$. Thus, if $Hash(v) \neq h'$, p_j *wr-delivers* $h' \neq h$, a violation of Lemma C.1. Furthermore, if $v \neq v'$, the collision resistance property of the hash function is violated.

Case 2: Without loss of generality, we assume $r' > r$. According to our protocol, p_j *raba-decides* 1 in RABA$_{r'}$ and *raba-decides* 0 for any iteration lower than r', including r. This would violate the agreement property of RABA. □

Theorem C.6 (Integrity). *If all replicas follow the protocol, and if a correct replica mvba-decides such that $Q(v)$ holds, then some replica mvba-proposed such that $Q(v)$ holds.*

Proof. If a correct replica p_i *mvba-decides* v, it *raba-decides* 1 in some RABA$_r$ and *wr-delivers* $h = H(v)$ in WRBC$_k$ where k is the output of the Election() function. According to our protocol, at least $n-f$ replicas have sent (Ready, h) messages, among which at least one is sent by a correct replica. The correct replica has received $n-f$ (Echo, h) messages, where at least $f+1$ message are sent by correct replicas. The correct replicas must have received a (Send, v) message from p_i such that $Hash(v) = h$. □

Lemma C.7. *If all correct replica enter the iteration phase, then for at least $f+1$ WRBC instances, at least $f+1$ correct replicas have wr-delivered some values.*

Proof. Correctness of the lemma is the same as that in Lemma 5.4, except that RBC is now WRBC. □

Theorem C.8 (Termination). *If all correct replicas are activated and all messages sent among correct replicas have been delivered, then all correct replicas mvba-decide.*

Proof. If all correct replicas start the protocol, each correct replica starts one WRBC instance. According to Lemma C.2, any correct replica completes a WRBC instance started by a correct replica. Thus, every correct replica completes at least $n-f$ WRBC instances.

During the iteration phase, we consider each iteration r where k is the corresponding output of the Election() function. We first show that every iteration r completes and then eventually some RABA$_r$ outputs 1.

We first show that every iteration r completes. For each iteration r, there are three cases: 1) all correct replicas have *wr-delivered* some value in WRBC$_k$; 2) at least one correct replica has *wr-delivered* some value in WRBC$_k$ and at least one correct replica has not *wr-delivered* any value in WRBC$_k$; 3) none of the correct replicas have *r-delivered* any value in WRBC$_k$.

Case 1: According to the unanimous termination property, RABA$_r$ terminates.

Case 2: If at least one correct replica has *wr-delivered* some value in WRBC$_k$, then according to Lemma C.3, any correct replica eventually *wr-delivers* some value. Note that any correct replica that provides 0 as the RABA input (it has not *wr-delivered* any value in WRBC$_k$ when the iteration begins) will eventually *raba-repropose* 1. Thus, the biased termination condition of RABA is satisfied. RABA$_r$ will terminate and iteration r will complete.

Case 3: If none of the correct replicas *wr-deliver* any value in WRBC_k, iteration r completes due to the unanimous termination property of RABA. Otherwise, if at least one correct replica later *wr-delivers* some value in WRBC_k, then according to the case 2, iteration r completes due to the biased termination property of RABA.

We now prove that eventually, in some iteration r, RABA_r outputs 1 so the protocol terminates. From Lemma C.7, for at least $f+1$ WRBC instances, at least $f+1$ correct replicas have *wr-delivered* some value after they enter the iteration phase. Let I be the $f+1$ WRBC instances. According to the biased validity property of RABA, RABA_r outputs 1. So with probability $\frac{f+1}{3f+1} \approx \frac{1}{3}$, we have $k \in I$.

After RABA_r outputs 1, every correct replica waits for the output of WRBC_k. We also know that if RABA_r outputs 1, then at least one correct replica has *raba-proposed* 1 or *raba-reproposed* 1. Otherwise, the unanimous termination of RABA is violated. Therefore, at least one correct replica has *wr-delivered* some value h in WRBC_k. From Lemma C.3 and Lemma C.1, every correct replica eventually *r-delivers* h in WRBC_k. From Lemma C.4, at least $f+1$ correct replicas receive $(\textsc{Send}, v)$ from p_i and set their $T_i[k]$ as v such that $Hash(v) = h$. The correct replicas will send $(\textsc{Value}, v)$ to all replicas. Therefore, any correct eventually receives v and then *mvba-decides.* □

D PROOF OF OUR ACS PROTOCOL

Theorem D.1 (Validity). *If a correct replica acs-decides a set* $\mathbf{v}$*, then* $|\mathbf{v}| \geq n-f$ *and* $\mathbf{v}$ *contains values acs-proposed by at least* $n-2f$ *correct replicas.*

Proof. According to the protocol, every correct replica p_i first *mvba-decide*(W_k) and then obtains $\mathbf{v}$. The predicate of $Q(W_k, W)$ specifies that there are at least $n-f$ 1's in the W_j vector. Then p_i waits for each RBC_j to output some v_j for each $W_k[j] = 1$ and includes v_j in $\mathbf{v}$. As there are $n-f$ values in $\mathbf{v}$, corresponding to $n-f$ RBC instances, it holds that $|\mathbf{v}| \geq n-f$. Since there are at most f faulty replicas, at least $n-2f$ values are *acs-proposed* by correct replicas. □

Theorem D.2 (Agreement). *If a correct replica acs-decides* $\mathbf{v}$*, then every correct replicas outputs* $\mathbf{v}$.

Proof. If a correct replica p_i *acs-decides* $\mathbf{v}$, it first *mvba-decides* W_k. Then for each $W_k[j] = 1$, v_j is *r-delivered* by RBC_j and v_j is included in $\mathbf{v}$. We assume that another correct replica p_j *acs-decides* $\mathbf{v}' \neq \mathbf{v}$ and then prove the theorem by contradiction.

If p_j *acs-decides* $\mathbf{v}'$, it *mvba-decides* W_k'. According to the agreement property of MVBA, it must hold that $W_k = W_k'$. Then for each $W_k'[j] = 1$, p_j obtains the output of RBC_j v_j' and includes v_j' in $\mathbf{v}'$. If for any j, $v_j \in \mathbf{v}$ and $v_j' \in \mathbf{v}'$ and $v_j \neq v_j'$, the agreement property of RBC is violated. Hence, we have $\mathbf{v} = \mathbf{v}'$. □

Theorem D.3 (Termination). *If all correct replicas acs-propose, then all correct replicas acs-decide.*

Proof. If all correct replicas *acs-propose*, every correct replica p_i starts RBC_i. Due to the validity property of RBC, every correct replica *r-delivers* in RBC_i. Therefore, every correct replica eventually completes $n-f$ RBC instances. Then each correct replica p_i *mvba-proposes* W_i. For each $W_i[j] = 1$, we know that p_i *r-delivers* some value in RBC_j. From the agreement property of RBC, every correct replica eventually *r-delivers* some value in RBC_j. Thus, the predicate $Q(W_i, W)$ eventually holds at every correct replica. Namely, for each $W_i[l] = 1$, each correct replica p_j eventually *r-delivers* some value in RBC_l, so $Q(W_i, W)$ eventually holds at p_j.

According to the termination property of MVBA, every correct replica eventually *mvba-decides* some value W_k. Furthermore, from the integrity property and external validity property of MVBA, $Q(W_k, W)$ eventually holds, so for each $W_k[j] = 1$, at least one correct replica *r-delivers* some value in RBC_j. Due to the agreement property of RBC, every correct replica eventually *r-delivers* some value v_j for each RBC_j. Thus, every correct replica includes each v_j in its output. □