Byzantine Reliable Broadcast with $O(nL + kn + n^2 \log n)$ Communication

Sisi Duan[1] and Haibin Zhang[2]

[1]Tsinghua University, duansisi@tsinghua.edu.cn
[2]Beijing Institute of Technology, haibin@bit.edu.cn

Abstract

Byzantine reliable broadcast (BRB) is one of the most fundamental primitives in fault-tolerant distributed computing. It is well-known that the best BRB protocol one can hope for has $O(nL + n^2)$ communication. It is unclear if this bound is achievable.

This paper provides a novel BRB protocol—BRB1, which achieves $O(nL + kn + n^2 \log n)$ communication, where n, L, and k are the number of replicas, the message length, and the security parameter, respectively. Our protocol is efficient, because the only building blocks we need are threshold signatures which have been used in various Byzantine fault-tolerant (BFT) protocols (e.g., SBFT, HoneyBadgerBFT, HotStuff). Our protocol is the first asynchronous BRB protocol that breaks the known $O(nL + kn^2)$ bound.

1 Introduction

Byzantine reliable broadcast (BRB) [12] is a fundamental tool in fault-tolerant distributed computing, ensuring that all replicas in a distributed system deliver the same message from a designated sender (even if some replicas, including the sender, are Byzantine). First, BRB itself is powerful enough to build killer applications such as online payment systems [16, 26]. More importantly, BRB is a popular building block for high-level protocols, such as Byzantine fault-tolerant state machine replication (BFT) protocols (e.g., HoneyBadgerBFT [38], BEAT [19], Dumbo [27], DAG-Rider [32], PACE [20], WaterBear [21], MiB [34]), and multi-party computation [35]. This paper introduces a novel BRB protocol reducing the communication complexity of BRB protocols, thereby improving upon all the above-mentioned protocols immediately.

A brief history of BRB. Bracha's broadcast [11, 12] is the first BRB protocol proposed and is one of the most classic protocols in fault-tolerant distributed computing. It has 3 steps, $O(n^2)$ messages, and achieves $O(nL^2)$ communication. Bracha's broadcast is information-theoretically secure (assuming authenticated channels only). Note the message complexity of $O(n^2)$ for Bracha's broadcast is optimal, and one cannot expect to achieve anything better. Hence, the epicenter of BRB research is to reduce the communication complexity of BRB protocols.

Assuming hash functions, Cachin, Kursawe, Petzold, and Shoup [14] describe a BRB protocol aiming to improve Bracha's BRB in an optimistic manner: if faulty replicas are not actively interfering with the protocol execution, the communication complexity of the protocol is $O(nL + kn^2)$, where k is the size of the hash function output; in the worst case, it has $O(n^2(L+k))$ communication (no better than Bracha's broadcast).

Still assuming hash functions, Cachin and Tessaro (CT) design an erasure-coded BRB protocol that achieves $O(nL + kn^2 \log n)$ communication. CT BRB is highly efficient, as it assumes hash functions and has been widely used in many practical asynchronous BFT protocols using BRB (e.g., HoneyBadgerBFT [38], BEAT [19], Dumbo [27]).

Assuming trusted setup, Nayak, Ren, Shi, Vaidya, and Xiang (NRSVX) propose an erasure-coded BRB protocol achieving a communication cost of $O(Ln + kn^2)$ [39]. Recently, Das, Xiang, and Ren (DXR) propose a new BRB protocol achieving the same communication, assuming hash functions only [17]. DXR BRB explores the idea of online error correction (OEC) due to Ben-Or, Canetti, and Goldreich [6]. Later, Alhaddad, Duan, Varia, and Zhang (ADVZ) have shown another BRB protocol using erasure coding proof system [3]. ADVZ BRB achieves the same communication as DXR BRB but assumes trusted setup. The benefit of this ADVZ BRB, however, is that it uses fewer steps than NRSVX but slightly less concrete communication than DXR BRB.

What we can hope for BRB? BRB requires validity and agreement. Validity means that if a correct replica p broadcasts a message M of length L, then all replicas eventually delivers M. Agreements means that if some correct replica delivers a message M, then every correct replica eventually delivers M. Validity implies the need of transmitting at least nL bits, as all replicas need to possess M. Agreement implies the need of broadcast (i.e., n^2 messages and at least n^2 bits needed), as a linear communication would not be possible to ensure agreement in the presence of failures in a constant number of rounds. Therefore, the best communication complexity can hope for a BRB protocol is $O(nL + n^2)$.

However, the above "best" bound is difficult, if not impossible, to achieve, as one would typically need to cryptography (e.g., hashes, commitments, signatures) or more advanced cryptographic building blocks to: 1) compress input, 2) authenticate data and ensure data transferability, or 3) ensure data consistency (if just sending the erasure-coded fragments instead of the input data itself). Doing so would seem to need agreeing on the underlying cryptographic tool, which at least incurs kn^2 communication, where k is the length of the underlying cryptographic primitive. So intuitively, what one could reasonably hope for a practical BRB protocol achieves $O(nL + kn^2)$ communication. Indeed, attaining $O(nL + kn^2)$ communication has already proven tricky: only very recently (after 2020), NRSVX BRB [39], DXR BRB [17], and ADVZ BRB [3] have achieved the goal, via different techniques (see Table 1).

This paper demonstrates a somewhat surprising result: we can construct BRB protocols with lower communication. In particular, our BRB protocol eliminates the cubic term which may easily dominate the communication with a large n or a small L.

1.1 Our Contributions

This paper provides a novel BRB protocol that achieves $O(nL + kn + n^2 \log n)$ communication, where n, L, and k are the number of replicas, the message length, and the security parameter, respectively. BRB1 breaks the symmetry in designing BRB protocols. Our protocol uses threshold signatures [9, 10] and authenticated channels. We begin with a half-baked construction satisfying validity but not agreement. Then we present BRB1 that tackles the problem *retroactively* in the sense that we simply allow the agreement problem to happen in the middle but fix the problem before the end of the protocol.

protocol	setup	asymptotic communication
Bracha's BRB [12]	none	$O(n^2L)$
CT BRB [15]	none	$O(nL + kn^2 \log n)$
NRSVX BRB [39]	trusted	$O(Ln + kn^2)$
DXR BRB [17]	none	$O(nL + kn^2)$
ADVZ BRB [3]	trusted	$O(nL + kn^2)$
BRB1 (this work)	trusted	$O(nL + kn + n^2 \log n)$

Table 1: Comparison of BRB protocols. Trusted setup means that a trusted party is responsible for generating public parameters for the system, say, the public key for threshold signatures. PKI means public-key infrastructure. L is the input length and k is the security parameter. Note our BRB protocol has better communication complexity than DXR and ADVZ BRB protocols, as $k + n \ll kn$ (for a given k, e.g., 128).

Our protocol has strictly lower communication than existing BRB protocols that have $O(nL + kn^2)$ communication, as long as one would not be "crazy" to use an exponentially large system with $n = O(2^k)$. In practice, for major BRB applications, such as online payment systems [16, 26] and asynchronous BFT systems [38, 19, 27, 32, 20], the message length is much longer than the security parameter (e.g., 128 bit), as these applications use message batching extensively for high system performance. (Meanwhile, it is possible that our BRB protocol is indeed optimal, as it is unclear if the $O(nL + n^2)$ bound is achievable.)

It is also trivial to replace threshold signatures using aggregate signatures, so the resulting protocol maintains the same complexity while working in the PKI model.

Discussion. This paper provides a new BRB protocol, BRB1 that uses the trusted setup model, relying on threshold signatures. In applications where trusted setup is permitted (for instance, all known asynchronous BFT protocols implemented using BRB), one can directly use our BRB protocol. In applications where trusted setup is not allowed, one may run a distributed key generation (DKG) algorithm to generate the needed public parameters: various DKG algorithms have been proposed in the synchronous setting [5, 23, 28], the partially synchronous setting [31], and the asynchronous setting (e.g., [17, 33, 1]).

Lastly, while our BRB protocol has better communication than DXR BRB, DXR BRB relies on hash functions only and does not assume trusted setup or PKI. So all those protocols are interesting and useful in their own regard.

One possible reaction to this work is to say: forget it, the improvement is *small.* Such a viewpoint underestimates the importance of the work. On the one hand, our protocol removes the cube term (i.e., kn^2) which is easily the bottleneck of the protocol when n is reasonably large *or* L is small. On the other hand, from the theoretical perspective, even just a "small" improvement would prove significant for BRB, because BRB is one of the most fundamental primitives in distributed computing: before this work, we do not even know if the bound of $O(Ln + kn^2)$ is the best we can hope for among BRB protocols using cryptography.

Versions of the paper. The early version of the paper contains a different BRB protocol that has more steps than BRB1. As there is no reason to favor that one, so we remove it.

2 System Model and Problem Statement

A Byzantine reliable broadcast (BRB) protocol consists of n replicas, where f out of them replicas may fail arbitrarily (Byzantine failures). We assume the existence of point-to-point authenticated channels between each pair of replicas. We consider asynchronous systems making no timing assumptions on message processing or transmission delays.

This paper considers adaptive corruption, where the adversary can choose its set of corrupted replicas at any moment during the execution of the protocol, based on the information it accumulated thus far. Note that the static corruption is weaker, as the adversary is restricted to choose its set of corrupted replicas at the start of the protocol and cannot change this set later on. All protocols we consider assume that f is a constant fraction of n with $f \leq \lfloor \frac{n-1}{3} \rfloor$, 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 $2f + 1$.

Byzantine reliable broadcast (BRB). We review the definition of Byzantine reliable broadcast (BRB). A BRB protocol is specified by two protocols *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.

Identifying protocol instances. We assume each protocol instance is associated with a unique tag *id*. For each step of the protocol, we provide a unique step name (e.g., CBC-SEND, DISPERSE) which will help readers better distinguish different steps.

3 Building Blocks

Reed-Solomon code. An (m, n) Reed-Solomon code consists of an *encode* algorithm and a *decode* algorithm. The *encode* algorithm takes as input m, n, and a data block M with m *data fragments* and uses those as m coefficients to produce a polynomial P of degree $m - 1$. Then the *encode* algorithm outputs n points (*coded fragments*) by evaluating P on n different points.

The *decode* algorithm takes as input m, a set T of coded fragments, and the number of incorrect coded fragments r, and outputs a degree $m - 1$ polynomial (coefficients as data fragments) by correcting up to r failures (incorrect fragments) in T. It is known that as long as $|T| \geq m + 2r$, *decode* can correct up to r failures in T and decode the original block [37, 44, 22].

Online error correcting (OEC) algorithm. Online error correcting algorithm due to Ben-Or, Canetti, and Goldreich [7] allows one to decode efficiently from a growing set T that keeps receiving coded fragments, where up to f out of $|T|$ coded fragments are incorrect.

For our purpose, fixing an $(f + 1, n)$ Reed-Solomon code where $n = 3f + 1$, let M be the original data block with m data fragments. There are at most n coded fragments, and up to f coded fragments may be replaced with arbitrary (incorrect) fragments. Suppose T be a set that keeps receiving these coded fragments, one after another. The OEC algorithm performs at most $f+1$ trials of the *decode* algorithm. In the r-th trial $(0 \leq r \leq f)$, the recipient waits until it receives fragments from $2f + r + 1$ replicas and then attempts to decode. In particular, the OEC algorithm has the first trial when $|T| \geq 2f + r + 1$ for $r = 0$. If not successful, then it keeps waiting for new fragments and then runs the *decode* algorithm for $1 \leq r \leq f$ until *decode* for some r is successful and outputs the data fragments. (For Reed-Solomon codes, if the reconstructed polynomial agree with $2f+1$ points in T, then *decode* outputs the coefficients of the reconstructed polynomial.) Note that the OEC algorithm must be successful when $r = f$, because the *decode* algorithm can correct f arbitrary failures among $|T| = 3f + 1$ coded fragments. The the *decode* algorithm may well output before the trial for $r = f$.

Asynchronous data dissemination. Das, Xiang, and Ren [17] propose asynchronous data dissemination (ADD) which allow $f + 1$ correct replicas to disseminate a message to all correct replicas in an asynchronous network, where f is the upper bound on the number of faulty replicas in the system. The ADD construction introduced by Das, Xiang, and Ren is as follows: in the first step, all replicas holding M send coded fragments to all replicas; in the second step, upon receiving $f + 1$ matching fragments d_i, a replica p_i fixes its fragment as d_i and broadcasts d_i. Then replicas wait to receive fragments and use OEC algorithm to decode the original block M. ADD is information-theoretical and does not use any cryptographic tools, and the communication cost of ADD is $O(nL + n^2 \log n)$. (Reed-Solomon code has a field size of at least n, so each symbol has at least $O(\log n)$ bits. Each data symbol is of size $O(\max(L/n, \log n))$, and therefore all-to-all sending the symbols incurs $O(n^2 \cdot \max(L/n, \log n)) = O(nL + n^2 \log n)$.)

While this paper uses ADD, we did not use it in a black-box manner. So in some sense, readers do not need to understand ADD to understand our protocols. In fact, knowing OEC is good enough for our constructions. We do, however, use ADD as an abstraction to motivate our constructions.

Signatures and threshold signatures. We use a conventional signature scheme consisting of three algorithms (*siggen*, *sigsign*, *sigverify*). *siggen* outputs a pair of public/secret keys (pk, sk). A signature signing algorithm *sigsign* takes as input a message M and a private key *sk* and outputs a signature σ. A signature verification algorithm *sigverify* takes as input *pk*, a message M, and a signature σ, and outputs a bit. We require the conventional unforgeability property for signatures.

A (t, n) threshold signature scheme [9, 42] consists of the following algorithms (*tgen*, *tsign*, *shareverify*, *tcombine*, *tverfiy*). *tgen* outputs a system public key known to anyone and a vector of n private keys. A partial signature signing algorithm *tsign* takes as input a message M and a private key sk_i and outputs a partial signature σ_i. A combining algorithm *tcombine* takes as input *pk*, a message M, and a set of t valid partial signatures, and outputs a signature σ. A signature verification algorithm *tverify* takes as input *pk*, a message M, and a signature σ, and outputs a bit. We require the conventional robustness and unforgeability properties for threshold signatures.

We simply omit the public keys, private keys, and key generation algorithms when no ambiguity arises. We may leave the verification of partial signatures and threshold signatures implicit when describing algorithms.

Consistent broadcast (CBC). We review the definition of consistent broadcast (CBC) [40, 14]. A CBC protocol is specified by *c-broadcast* and *c-deliver* such that the following properties hold:

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

A verifiable CBC (VCBC) protocol [14] is a CBC protocol with an additional verifiability

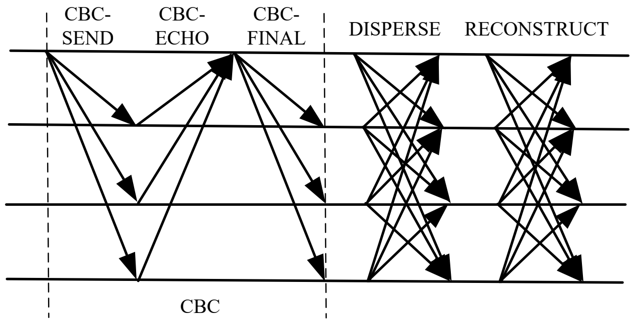

Figure 1: A half-baked idea.

property: when a correct replica has c-delivered a message M, then it can produce a single protocol message P that may be sent to other parties such that any other correct party will immediately deliver P once receiving P. VCBC can be used as a building block for high-level protocols [14].

Steps and phases. We use the standard notation of steps [13], where a step consists of receiving a message from some party, running a local computation (optional), and sending a message to some party. We also use the notation of phases, where a phase consists of a fixed number of steps. In our notation, a protocol has several phases, and each phase has several steps. Of course, it makes sense to directly count how many steps a protocol has.

4 Technical Paths

Review of DXR BRB. Das, Xiang, and Ren [17] use OEC to build DXR BRB protocol with $O(nL + kn^2)$ communication.[1] The idea is simple and efficient, following the three-step pattern of Bracha's broadcast. In the first step, the sender broadcasts the whole message m instead of sending individual coded fragments. In the second step, replicas echo replicas different coded fragments and a hash of m denoted as h. In the third step, upon receiving $2f + 1$ matching ECHO messages, replicas send READY messages with coded fragments and h; upon receiving $f + 1$ READY messages with the same h, replicas wait for $f + 1$ matching ECHO messages with the same h and then send READY messages (the "amplification" step). The replicas keep all READY messages with the same h in a set and run OEC algorithm until successfully reconstructing some data m'. The replica delivers m' only if it matches h. Apparently, the second step and the third step involves all-to-all broadcast of hashes, incurring $O(kn^2)$ communication.

A half-baked idea: breaking the symmetry for BRB design. Our first idea is to break the symmetry in designing BRB protocols. Indeed, when designing efficient BRB protocols, one typically follows a symmetric design approach: in the first step, the sender sends some data (either the whole input message M or a coded fragment) to every replica; in the following steps, replicas all broadcast fragments and/or short cryptographic proofs to each other in order to achieve agreement.

In our new design, we break BRB constructions into a linear communication phase and a broadcast phase. We use cryptography in the first linear communication phase, while the broadcast phase explicitly rules out cryptography.

[1]Note that the authors also provide a 5-step BRB protocol which may be viewed as a less efficient variant of the DXB BRB.

Our starting protocol works as follows. In the first linear communication phase, our goal is to disperse the input to ensure that $f+1$ correct replicas to have consistent data, a goal that consistent broadcast (CBC) [40, 14] or its information dispersal version may achieve; in the second broadcast phase, our idea is to "amplify" consistent data from $f + 1$ correct replicas to all replicas, a goal that asynchronous data dissemination (ADD) [17] may achieve. Such a construction is depicted in Figure 1. We use CBC and ADD in a black-box manner.

For the above construction, the first phase has $O(nL + kn)$ communication, while the second phase has $O(nL+n^2)$ communication. Adding them together, we have a construction with $O(nL+nk + n^2)$. Unfortunately, the construction only achieves validity but not agreement. Indeed, it is easy to show that some correct replicas *r-deliver* message M, while some other replicas do not *r-deliver* any message, violating agreement.

As an example, a faulty sender may make only one correct replica deliver the message in the CBC phase and enter the second phase. All f faulty replicas collude and disseminate correct fragments to $f + 1$ correct replicas. Together with the fragment from the correct replica, each of the $f + 1$ correct replicas receive $f + 1$ matching fragments, share their fragments, complete ADD, and deliver the corresponding message. The rest f correct replicas, however, cannot deliver the message, since they fail to start ADD.

Some trivial modifications of the above construction suffer from a similar problem. One such modification could be illustrated as follows: whether receiving a threshold signature from the sender to start ADD, replicas immediately broadcast fragments once receiving fragments. This approach does not work either, because it is still possible that some correct replicas *r-deliver* a message M, while other correct replicas do not deliver anything.

Below, we outline how we solve the agreement problem, by providing an approach to handling the issue *retroactively.*

BRB1 (Figure 2). In BRB1, our idea is simply to let the agreement issue occur and then fix it retroactively. We add one more READY step after the ADD phase and ask replicas to *r-deliver* a message M only if it receives enough READY messages. The most interesting part is that an amplification step is now introduced, *going back to the very first step of the broadcast phase*, instead of the beginning of the same step. To our knowledge, our novel amplification technique is in contrast to all other known amplification steps ever used in BRB and even fault-tolerant distributed computing. Strikingly, the READY step and the "unconventional" amplification step are all we need for a secure BRB construction.

5 BRB1

This section describes BRB1, a BRB protocol that has six steps. BRB1 uses a strategy that is different from BRB1. We show the workflow of BRB1 in Figure 2 and pseudocode in Algorithm 1. BRB1 consists of two phases: a linear CBC phase and a broadcast phase.

CBC phase (Algorithm 1: line 4-15). This phase runs a standard CBC. In particular, the sender p_s broadcasts an $(id,\text{CBC-SEND}, M)$ message. Upon receiving an $(id,\text{CBC-SEND}, M)$ message, each replica generates a partial signature σ_i and sends an $(id,\text{CBC-REP}, M, \sigma_i)$ message to p_s. If p_s receives $n - f$ partial signatures, it combines them into a threshold signature σ and broadcasts an $(id,\text{CBC-FINAL}, M, \sigma)$ message to all replicas. Upon receiving an $(id,\text{CBC-FINAL}, M, \sigma)$ message, each replica sets msg as M and $proof_1$ as σ and completes CBC.

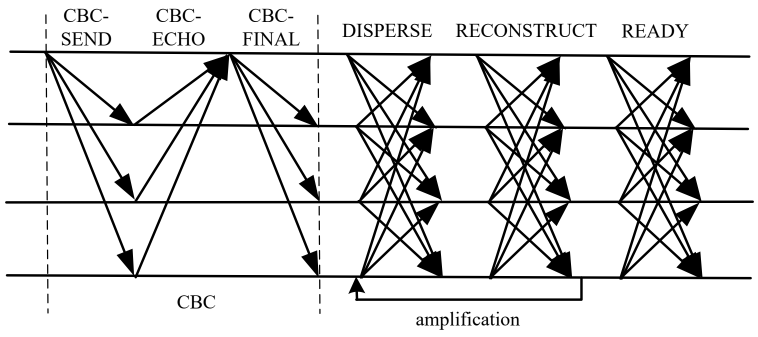

Figure 2: BRB1 workflow.

Broadcast phase (Algorithm 1: line 16-33). The broadcast phase consists of three steps: DISPERSE, RECONSTRUCT, and READY. The first two steps are similar to those in . In particular, upon the completion of CBC, each replica p_i encodes its msg into coded fragments $\boldsymbol{d}$. For each replica p_j, p_i sends it an $(id,\text{DISPERSE}, d_j)$ message. Upon receiving $f+1$ matching $(id,\text{DISPERSE}, d_i^*)$ messages, p_i fixes d_i^* and broadcasts $(id,\text{RECONSTRUCT}, d_i^*)$. Upon receiving at least $n-f$ RECONSTRUCT messages, each replica starts to decode the message using OEC. This process may continue until OEC outputs a message M. A local parameter *output* is then set as M.

But this is not the last step of the BRB1. When OEC outputs M, each replica broadcasts an (id,READY) message. Furthermore, if replica p_i previously has not sent a DISPERSE message, it disperses the coded fragments, i.e, p_i encodes M and sends p_j (for any $j \in \{0, \cdots n-1\}$) an $(id,\text{DISPERSE}, d_j)$ message. Each replica waits for $n-f$ (id,READY) messages and then delivers message *output* (message output by OEC).

Analysis. The crucial step for BRB1 to achieve agreement is the amplification step after message M is output by OEC. In particular, if the OEC outputs M and a replica has not previously sent a DISPERSE message, the replica encodes M and sends the coded fragments via a DISPERSE message to the replicas. If a replica *r-delivers* M, it has received $n-f$ (id,READY) messages and at least $f+1$ correct replicas have completed the OEC. These correct replicas will send their coded fragments via the DISPERSE message. Accordingly, it is guaranteed that all correct replicas eventually decode M, broadcast the READY messages, and *r-deliver* M.

The consistency property of CBC guarantees that no correct replicas will broadcast inconsistent coded fragments, as each replica only disperses the coded fragments upon the completion of CBC. An adversary cannot force any correct replicas to receive $f+1$ matching but incorrect fragments in the DISPERSE step. Therefore, OEC can correct the errors and ensure that all correct replicas *r-deliver* the same message M.

Let us analyze the communication complexity of BRB1. First, the first linear CBC phase has $O(Ln+kn)$ communication. The DISPERSE and RECONSTRUCT steps both have $O(Ln+n^2 \log n)$ communication. The READY phase does not carry bulk data and has $O(n^2)$ communication only. Therefore, the communication complexity for is $O(Ln+kn+n^2 \log n)$.

5.1 Proof of BRB1

Theorem 1. *Assuming a secure threshold signature and authenticated channels, BRB1 satisfies validity, agreement, and integrity.*

Algorithm 1 BRB1 with identifier id and sender p_s. Code shown for replica p_i.

```
1: Initialization
2:   $(pk, sk) \leftarrow tgen(1^k)$  {threshold signature key generation; $pk$ is the public key and $sk$ is a vector of $n$ private keys}
3:   $proof_1 \leftarrow \perp, msg \leftarrow \perp, output \leftarrow \perp, pset_1 \leftarrow \emptyset, T \leftarrow \emptyset$    {initialize the parameters}
4: upon $r\text{-}broadcast(M)$
5:   $msg \leftarrow M$
6:   broadcast $(id,\text{CBC-SEND}, M)$    {CBC-SEND step}
7: upon receiving $(id,\text{CBC-SEND}, M)$ from $p_s$ do    {CBC-REP step}
8:   $msg \leftarrow M$, $\sigma_i \leftarrow tsign(id, M)$
9:   send $(id,\text{CBC-REP}, M, \sigma_i)$ to $p_s$
10: upon receiving $(id,\text{CBC-REP}, M, \sigma_j)$ from $p_j$ and $p_i = p_s$ do    {CBC-FINAL step}
11:   if $shareverify((id, M), \sigma_j)$ and $M = msg$
12:     add $\sigma_j$ to $pset_1$
13:   if $|pset_1| \geq n - f$
14:     $\sigma \leftarrow tcombine((id, M), pset_1)$, broadcast $(id,\text{CBC-FINAL}, M, \sigma)$
15: upon receiving $(id,\text{CBC-FINAL}, M, \sigma)$ from $p_s$ do    {DISPERSE step}
16:   if $tverify((id, M), \sigma)$ and $M = msg$
17:     $proof_1 \leftarrow \sigma$, $\mathbf{d} \leftarrow encode(f + 1, n, msg)$
18:     for $j \in \{0, \cdots n - 1\}$
19:       send $(id,\text{DISPERSE}, d_j)$ to $p_j$
20: upon receiving $f + 1$ matching $(id,\text{DISPERSE}, d_i^*)$ do
21:   fix $d_i^*$, broadcast $(id,\text{RECONSTRUCT}, d_i^*)$    {RECONSTRUCT step}
22: upon receiving $(id,\text{RECONSTRUCT}, d_j)$ from $p_j$ do
23:   add $d_j$ to $T$
24:   for $0 \leq r \leq f$ do
25:     wait until $|T| \geq 2f + r + 1$
26:       if $decode(f + 1, T, r) = M$    {send (READY) and execute the amplification step}
27:         $output \leftarrow M$
28:         broadcast $(id,\text{READY})$
29:         if (DISPERSE) has not been sent
30:           $\mathbf{d} \leftarrow encode(f + 1, n, M)$
31:           for $j \in \{0, \cdots n - 1\}$, send $(id,\text{DISPERSE}, d_j)$ to $p_j$
32: upon receiving $n - f$ $(id,\text{READY})$ do    {READY step}
33:   if $output \neq \perp$, $r\text{-}deliver$ $output$
```

Proof. We first provide the following three lemmas.

Lemma 2. *If a correct replica sends a* DISPERSE *message with fragments encoded from message M, at least one correct replica has completed CBC and received M from the sender.*

Proof. Each correct replica sends a DISPERSE message with fragments encoded from M for two cases: 1) It completes CBC and receives M from the sender; 2) It completes the RECONSTRUCT step and obtains M. We show that in both cases, at least one correct replica has completed CBC and receives M from the sender. For the first case, trivial. For the second case, if a correct replica completes the RECONSTRUCT step, it must have received at least $n - f$ RECONSTRUCT messages, among which at least $f + 1$ are sent by correct replicas. For any of the correct replicas, it must have also received $f+1$ matching DISPERSE messages. As there are at most f faulty replicas, there must exist at least one correct replica that sends the DISPERSE message after it completes CBC and receives M. □

Lemma 3. *If a correct replica p_i sends an $(id,\textsc{reconstruct}, d_i^*)$ message and a correct replica p_j sends an $(id,\textsc{reconstruct}, d_j^*)$ message, d_i^* and d_j^* are both encoded from the same message M.*

Proof. Let d_i^* be a coded fragment encoded from message M. If a correct replica p_i sends an $(id,\textsc{reconstruct}, d_i^*)$ message, it must have received $f+1$ matching $(id,\textsc{disperse}, d_i^*)$ messages, among which at least one is sent by a correct replica. Among them, at least one correct replica has sent a DISPERSE message. According to Lemma 2, at least one correct replica completes CBC and receives message M from the sender.

Let d_j^* be a coded fragment encoded from message $\widetilde{M}$. If p_j sends an $(id,\textsc{reconstruct}, d_j^*)$ message, according to Lemma 2, at least one correct replica completes CBC and receives message $\widetilde{M}$. This violates the consistency property of CBC. Thus, it holds $M = \widetilde{M}$. □

Lemma 4. *If a correct replica r-delivers M, the sender p_s has previously broadcast M and at least one correct replica has received a valid threshold signature $proof_1 = \sigma$.*

Proof. If a correct replica p_i *r-delivers* M, it receives $n-f$ $(id,\textsc{ready})$ messages. Furthermore, in the RECONSTRUCT step, the OEC outputs a decoded message M. Accordingly, p_i must have received at least $2f+1$ $(id,\textsc{reconstruct}, d_j)$ messages, among which at least $f+1$ are sent by correct replicas.

According to Lemma 3, the $f+1$ correct replicas only send coded fragments encoded from the same message M'. Therefore, the reconstructed f-degree polynomial for M must agree with $f+1$ fragments from correct replicas. It must hold that $M = M'$.

If p_i *r-delivers* M, it has received at least $2f+1$ READY messages and also $2f+1$ RECONSTRUCT messages, among which at least $f+1$ are sent by correct replicas. According to Lemma 2, at least one correct replica has completed CBC and received a valid threshold signature $proof_1$ for M, and meanwhile the sender p_s has sent M. □

In the following, we prove that BRB1 satisfies validity, agreement, and integrity.

Validity. If a correct replica p_s *r-broadcasts* M, all correct replicas complete CBC, according to the validity property of CBC. Therefore, all correct replicas will send the DISPERSE messages for the same M, receive $f+1$ matching DISPERSE messages, and broadcast the RECONSTRUCT messages. No correct replica can receive $f+1$ matching $(id,\textsc{disperse}, d_i)$ for $M' \neq M$. Each correct replica will then send a $(id,\textsc{reconstruct}, d_i)$ messages such that d_i is encoded from M. Thus, all correct replicas will receive $2f+1$ RECONSTRUCT messages from all correct replicas, output some value and then send the $(id,\textsc{ready})$ messages. All correct replicas, including the sender, will then receive $n-f$ $(id,\textsc{ready})$ messages and *r-deliver* some message. Furthermore, as shown in Lemma 4, if any correct replica *r-delivers* $M' \neq M$, p_s has previously sent M', contradicting the fact that p_s is a correct replica and *r-broadcasts* M. Therefore, all correct replicas will *r-deliver* the original message M.

Agreement. The proof of agreement consists of two parts: 1) if a correct replica p_i *r-delivers* M and a correct replica p_j *r-delivers* M', then $M = M'$; 2) if a correct replica p_i *r-delivers* M, any correct replica will eventually *r-deliver* some message.

We prove the first part. If a correct replica p_i *r-delivers* M, according to Lemma 4, at least one correct replica has completed CBC and possessed a valid threshold signature for M. Furthermore, If p_j *r-delivers* M', at least one correct replica has completed CBC and possessed a valid threshold signature for M'. This violates the consistency property of CBC. Thus, $M = M'$.

Then we prove the second part. If a correct replica p_i *r-delivers* M, it has received $n - f$ READY messages, among which at least $f + 1$ are sent by correct replicas. The $f + 1$ correct replicas must all output M by OEC, as proved in the first part. Furthermore, for each of the $f + 1$ correct replicas, if it has not previously sent a DISPERSE message, it will encode message M and broadcast the DISPERSE messages according to our protocol. Therefore, each correct replica eventually receives at least $f + 1$ matching DISPERSE messages. Eventually, all correct replicas will receive at least $n - f$ RECONSTRUCT messages. According to Lemma 3, each correct replica sends a fragment $(id, \text{RECONSTRUCT}, d_i)$ such that d_i is encoded from the same message M. Therefore, each correct replica eventually outputs some message. Finally, each correct replica will eventually send an (id, READY) message, receive $n - f$ (id, READY) messages, and *r-deliver* the value.

Integrity. According to the protocol, a replica does not participate in the protocol after it *r-delivers*. Hence, each correct replica *r-delivers* at most once. Furthermore, as proved in Lemma 4, if a replica *r-delivers* a message M with sender p_s, M was previously broadcast by p_s. $\square$

6 Related Work

AVID vs. BRB. Asynchronous verifiable information dispersal (AVID), introduced by Cachin and Tessaro [15], is a primitive closely related to BRB. The original AVID constructions are both AVID and BRB protocols. The difference between these two distributed systems primitives is that BRB requires that replicas store a full copy of message, but AVID may only ask replicas to store erasure-coded fragments in a consistent manner.

Erasure codes and error correcting codes in fault-tolerant distributed systems. Erasure codes and error correcting codes are fundamental tools in fault-tolerant distributed computing protocols. Besides BRB, they have been used in various other primitives with Byzantine failures, such as asynchronous verifiable information dispersal (AVID) [15, 30], multi-valued validated Byzantine agreement (MVBA) [36], multi-valued Byzantine agreement [39], read/write storage [18, 4, 29], and BFT storage [19].

Definitions and constructions for reliable broadcast (in the crash failure model). The properties of reliable broadcast have been (somewhat informally) captured in SIFT [45]. Schneider, Gries, and Schlichting formally define reliable broadcast as well as implement a protocol in the crash failure model [41]. Crash fault-tolerant reliable broadcast becomes widely known through the ISIS system due to Birman and Joseph [8]. Note that the definition of BRB is identical to that in the crash failure model: the only difference lies in the failure assumption, where BRB considers Byzantine failures, and crash fault-tolerant reliable broadcast considers crash failures.

Byzantine consistent broadcast (CBC). Byzantine consistent broadcast may be viewed as BRB without the totality property. The notion has been implicitly discussed in [43, 12] and more formally by Reiter [40] and Cachin, Kursawe, Petzold, and Shoup (CKPS) [14]. The CBC construction is due to Cachin, Kursawe, Petzold, and Shoup [14].

BRB extensions. BRB has also been explored in some extended settings (e.g., probabilistic

BRB [25], BRB with dynamic membership [24]).

Good-case latency. Abraham, Nayak, Ren, and Xiang provide a complete categorization for Byzantine broadcast and BRB in terms of the *good-case latency* which measures the time one takes for all correct replicas to commit when the broadcaster is correct [2].

7 Conclusion

We propose a novel BRB protocol with $O(nL + kn + n^2 \log n)$ communication, where n, L, and k are the number of replicas, the message length, and the security parameter, respectively. Our protocol is the first asynchronous BRB protocol that breaks the known $O(nL + kn^2)$ bound.

Acknowledgement

We are indebted to Nicolas Alhaddad and Mayank Varia for the helpful discussion and insightful comments that lead to the paper. We thank the PODC reviewers for their insightful comments. Many thanks to Zhuolun Xiang for kindly clarifying the fact that the communication of ADD, strictly speaking, is $O(nL + n^2 \log n)$ instead of $O(nL + n^2)$.

References

[1] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. Reaching consensus for asynchronous distributed key generation. In *PODC '21: ACM Symposium on Principles of Distributed Computing, Virtual Event, Italy, July 26-30, 2021*, pages 363–373. ACM, 2021.

[2] I. Abraham, K. Nayak, L. Ren, and Z. Xiang. Good-case latency of byzantine broadcast: A complete categorization. PODC'21.

[3] N. Alhaddad, S. Duan, M. Varia, and H. Zhang. Succinct erasure coding proof systems. *Cryptology ePrint Archive*, 2021.

[4] E. Androulaki, C. Cachin, D. Dobre, and M. Vukolic. Erasure-coded byzantine storage with separate metadata. In *Principles of Distributed Systems - 18th International Conference*, 2014.

[5] M. J. B. Libert and M. Yung. Born and raised distributively: Fully distributed non-interactive adaptively-secure threshold signatures with short shares. In *Theoretical Computer Science*, 2016.

[6] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *STOC*, pages 52–61. ACM, 1993.

[7] M. Ben-Or, R. Canetti, and O. Goldreich. Asynchronous secure computation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 52–61, 1993.

[8] K. P. Birman and T. A. Joseph. Reliable communication in the presence of failures. *ACM Transactions on Computer Systems (TOCS)*, 5(1):47–76, 1987.

[9] A. Boldyreva. Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In *Public Key Cryptography — PKC 2003*, 2002.

[10] D. Boneh, B. Lynn, and H. Shacham. Short signatures from the weil pairing. *Journal of cryptology*, 17(4):297–319, 2004.

[11] G. Bracha. An asynchronous [(n-1)/3]-resilient consensus protocol. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 154–162. ACM, 1984.

[12] G. Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[13] C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to Reliable and Secure Distributed Programming.* 2nd edition, 2011.

[14] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*, pages 524–541. Springer, 2001.

[15] C. Cachin and S. Tessaro. Asynchronous verifiable information dispersal. In *SRDS*, pages 191–201. IEEE, 2005.

[16] D. Collins, R. Guerraoui, J. Komatovic, P. Kuznetsov, M. Monti, M. Pavlovic, Y.-A. Pignolet, D.-A. Seredinschi, A. Tonkikh, and A. Xygkis. Online payments by merely broadcasting messages. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 26–38. IEEE, 2020.

[17] S. Das, Z. Xiang, and L. Ren. Asynchronous data dissemination and its applications. CCS '21.

[18] D. Dobre, G. O. Karame, W. Li, M. Majuntke, N. Suri, and M. Vukolic. Proofs of writing for robust storage. *IEEE Trans. Parallel Distributed Syst.*, 30(11):2547–2566, 2019.

[19] S. Duan, M. K. Reiter, and H. Zhang. Beat: Asynchronous bft made practical. In *CCS*, pages 2028–2041. ACM, 2018.

[20] S. Duan and H. Zhang. Pace: Fully parallelizable bft from reproposable byzantine agreement. *Cryptology ePrint Archive*, 2022.

[21] S. Duan, H. Zhang, and B. Zhao. Waterbear: Information-theoretic asynchronous bft made practical. *Cryptology ePrint Archive*, 2022.

[22] S. Gao. A new algorithm for decoding reed-solomon codes. In *Communications, information and network security.* 2003.

[23] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[24] R. Guerraoui, J. Komatovic, P. Kuznetsov, Y.-A. Pignolet, D.-A. Seredinschi, and A. Tonkikh. Dynamic byzantine reliable broadcast. In *OPODIS 2020*, 2021.

[25] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovic, and D. Seredinschi. Scalable byzantine reliable broadcast. In *DISC 2019*, 2019.

[26] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovič, and D.-A. Seredinschi. The consensus number of a cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 307–316, 2019.

[27] B. Guo, Z. Lu, Q. Tang, J. Xu, and Z. Zhang. Dumbo: Faster asynchronous bft protocols. In *CCS*, 2020.

[28] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. Aggregatable distributed key generation. In A. Canteaut and F. Standaert, editors, *Advances in Cryptology - EUROCRYPT 2021*, volume 12696, pages 147–176. Springer, 2021.

[29] J. Hendricks, G. R. Ganger, and M. K. Reiter. Low-overhead byzantine fault-tolerant storage. In T. C. Bressoud and M. F. Kaashoek, editors, *SOSP*, pages 73–86, 2007.

[30] J. Hendricks, G. R. Ganger, and M. K. Reiter. Verifying distributed erasure-coded data. In *PODC*, 2007.

[31] A. Kate and I. Goldberg. Distributed key generation for the internet. In *2009 29th IEEE International Conference on Distributed Computing Systems*, pages 119–128. IEEE, 2009.

[32] I. Keidar, E. Kokoris-Kogias, O. Naor, and A. Spiegelman. All you need is DAG. In *PODC*, pages 165–175. ACM, 2021.

[33] E. Kokoris Kogias, D. Malkhi, and A. Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 1751–1767, 2020.

[34] C. Liu, S. Duan, and H. Zhang. Mib: Asynchronous bft with more replicas. *arXiv preprint arXiv:2108.04488*, 2021.

[35] D. Lu, T. Yurek, S. Kulshreshtha, R. Govind, A. Kate, and A. Miller. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *CCS '19*, 2019.

[36] Y. Lu, Z. Lu, Q. Tang, and G. Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *PODC*, 2020.

[37] F. J. MacWilliams and N. J. A. Sloane. *The theory of error correcting codes*, volume 16. Elsevier, 1977.

[38] A. Miller, Y. Xia, K. Croman, E. Shi, and D. Song. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 31–42. ACM, 2016.

[39] K. Nayak, L. Ren, E. Shi, N. H. Vaidya, and Z. Xiang. Improved extension protocols for byzantine broadcast and agreement. In H. Attiya, editor, *34th International Symposium on Distributed Computing, DISC 2020.*

[40] M. K. Reiter. Secure agreement protocols: Reliable and atomic group multicast in rampart. In *Proceedings of the 2nd ACM Conference on Computer and Communications Security*, pages 68–80, 1994.

[41] F. B. Schneider, D. Gries, and R. D. Schlichting. Fault-tolerant broadcasts. *Science of Computer Programming*, 4(1):1–15, 1984.

[42] V. Shoup. Practical threshold signatures. In *Advances in Cryptology — EUROCRYPT 2000*, 2000.

[43] S. Toueg. Randomized byzantine agreements. In *Proceedings of the third annual ACM symposium on Principles of distributed computing*, pages 163–178, 1984.

[44] L. R. Welch and E. R. Berlekamp. Error correction for algebraic block codes, 1986. US Patent 4,633,470.

[45] J. H. Wensley, L. Lamport, J. Goldberg, M. W. Green, K. N. Levitt, P. M. Melliar-Smith, R. E. Shostak, and C. B. Weinstock. Sift: Design and analysis of a fault-tolerant computer for aircraft control. *Proceedings of the IEEE*, 66(10):1240–1255, 1978.