Improved Extension Protocols for Byzantine Broadcast and Agreement

Kartik Nayak
Duke University, Durham, NC, USA
kartik@cs.duke.edu

Ling Ren
University of Illinois at Urbana-Champaign, Champaign, IL, USA
renling@illinois.edu

Elaine Shi
Cornell University, Ithaca, NY, USA
runting@gmail.com

Nitin H. Vaidya
Georgetown University, Washington, D.C., USA
nitin.vaidya@georgetown.edu

Zhuolun Xiang
University of Illinois at Urbana-Champaign, Champaign, IL, USA
xiangzl@illinois.edu

Abstract

Byzantine broadcast (BB) and Byzantine agreement (BA) are two most fundamental problems and essential building blocks in distributed computing, and improving their efficiency is of interest to both theoreticians and practitioners. In this paper, we study extension protocols of BB and BA, i.e., protocols that solve BB/BA with long inputs of l bits using lower costs than l single-bit instances. We present new protocols with improved communication complexity in almost all settings: authenticated BA/BB with $t < n/2$, authenticated BB with $t < (1-\epsilon)n$, unauthenticated BA/BB with $t < n/3$, and asynchronous reliable broadcast and BA with $t < n/3$. The new protocols are advantageous and significant in several aspects. First, they achieve the best-possible communication complexity of $\Theta(nl)$ for wider ranges of input sizes compared to prior results. Second, the authenticated extension protocols achieve optimal communication complexity given the current best available BB/BA protocols for short messages. Third, to the best of our knowledge, our asynchronous and authenticated protocols in the setting are the first extension protocols in that setting.

2012 ACM Subject Classification Theory of computation → Communication complexity; Theory of computation → Cryptographic protocols

Keywords and phrases Byzantine agreement, Byzantine broadcast, extension protocol, communication complexity

Digital Object Identifier 10.4230/LIPIcs.DISC.2020.28

Related Version A full version of the paper is available at `https://arxiv.org/abs/2002.11321`.

Funding Kartik Nayak and Ling Ren are supported in part by a VMware early career faculty grant.
Elaine Shi: Elaine Shi is supported in part by NSF award 1561209, and part of the work was done when the author was a long-terms visitor in Simons Institute for the “Proofs, Consensus, and Decentralizing Societ” program.
Nitin H. Vaidya: Nitin H. Vaidya is supported in part by NSF award 1849599.

Table 1 Cryptographically Secure Extension Protocols for Byzantine Agreement and Broadcast.

Threshold	Model	Problem	Communication Complexity	Input range l to reach optimality	Reference
$t < n/2$	sync.	agreement/broadcast	$O(nl + n\mathcal{B}(k) + kn^3)$ $O(nl + \mathcal{A}(k) + kn^2)$ [2]	$\Omega(n^3 + kn^2)$ $\Omega(n^2 + kn)$	[15, 16] **This paper**
$t < n$	sync.	broadcast	$O(nl + \mathcal{B}(nk) + n^2\mathcal{B}(n \log n))$	$\Omega(n^5 \log n + kn^4 \log n)$	[15, 16]
$t < (1-\varepsilon)n$	sync.	broadcast	$O(nl + \mathcal{B}(k) + kn^2 + n^3)$	$\Omega(n^2 + kn)$	**This paper**
$t < n/3$	async.	agreement reliable broadcast	$O(nl + \mathcal{A}(k) + kn^2)$ $O(nl + \mathcal{B}(k) + kn^2)$	$\Omega(kn)$ $\Omega(kn)$	**This paper** **This paper**

1 Introduction

This paper investigates extension protocols [15] for Byzantine broadcast (BB) and Byzantine agreement (BA). The goal of BB is for some designated party (sender) to send its message to all parties and let them output the same message, despite some malicious parties that may behave in a Byzantine fashion. The goal of BA is to let all parties each with an input message output the same message. We are interested in designing efficient BB/BA protocols with *long messages* since such protocols are widely used as building blocks for other distributed systems such as multi-party computation [32] and permissioned blockchain [24]. For example, practical blockchain systems typically achieve agreement on large blocks (e.g., 1MB).

A straightforward solution for BB/BA with l-bit long messages is to invoke the single-bit BB/BA oracle l times. This approach will incur at least $\Omega(n^2 l)$ communication complexity where n is the number of parties, because any deterministic single-bit BB/BA has cost $\Omega(n^2)$ due to a lower bound in [10]. Another tempting solution is to run BB/BA on the hash digest and let parties disseminate the actual message to each other. However, if a linear fraction of parties can be Byzantine (which is the typical assumption), they can each ask all honest parties for the long message, again forcing the communication complexity to be $\Omega(n^2 l)$.

It turns out that non-trivial techniques are needed to get better than $\Omega(n^2 l)$ or to achieve the optimal communication complexity of $O(nl)$. These are known in the literature as *extension protocols*, which construct BB/BA with long input messages using a small number of BB/BA primitives for short messages. In this paper, we focus on the authenticated setting where cryptographic techniques are used. Table 1 summarizes the most closely related works and our new results on authenticated extension protocols. (In the full version, we also present some improvements to unauthenticated extension protocols.) In Table 1, n is the number of parties, t is the maximum number of Byzantine parties, l is the length of the input, $\mathcal{A}(l)$ is the communication cost of l-bit BA oracle, and $\mathcal{B}(l)$ is the communication cost of l-bit BB or reliable broadcast oracle. Here we describe the related works in the table. Let k_h denote output size of the collision-resistant hash function. For both Byzantine broadcast and agreement in the synchronous setting under $t < n/2$, recent work proposes cryptographically secure extension protocols with communication cost $O(nl + n\mathcal{B}(k_h) + n^3 k_h)$ [15, 16]. For the case of $t < n$, the state-of-the-art cryptographically secure BB extension protocols have communication complexity $O(nl + \mathcal{B}(nk_h) + n^2\mathcal{B}(n \log n))$ [15, 16]. There exist information-theoretic authenticated protocols [14, 8] but they have worse communication complexity than cryptographic ones. To the best of our knowledge, there exist no extension protocols in the authenticated and asynchronous setting when the paper is written [1].

[1] A concurrent work [21] independently developed an extension protocol for validated Byzantine agreement in the authenticated and asynchronous setting.

[2] Our cryptographic BB extension protocol can achieve $O(nl + \mathcal{B}(k) + \mathcal{A}(1) + kn^2)$ in the full version.

Contributions. Table 1 also presents our improved protocols in the respective settings. Several cryptographic primitives have been employed in our work and prior works. To make the communication costs comparable, we assume that the output length of the involved cryptographic building blocks are on the same order, and are all represented by k. We will justify this decision in Section 3.

All our protocols achieve the optimal communication complexity $O(nl)$ for wider ranges of input sizes (see Table 1 above for authenticated protocols and the full version for unauthenticated protocols). In particular, our synchronous and authenticated protocols achieve $O(nl)$ communication complexity when the input size is at least $l = \Omega(n^2 + kn)$. For comparison, state-of-art protocol in the literature require a factor of n larger input size for the $t < n/2$ case, and a factor of $n^3 \log n$ larger input size for $n/2 \leq t < (1-\varepsilon)n$ where ε is a constant. But a limitation of our protocol is that it cannot achieve $O(nl)$ communication if $\varepsilon = o(1)$. As for the round complexity, all our extension protocols only adds $O(1)$ communication rounds, except the one for $t < n/2$ which adds $O(t)$ rounds. All our protocols only invoke the BB/BA oracle $O(1)$ times.

In addition to reaching optimality under smaller input size, our authenticated extension protocols have the following advantages.

- The communication complexity of our BA extension protocols is very close to the lower bound $\Omega(nl + \mathcal{A}(k) + n^2)$. In addition, under the current best BA primitives for short messages, they achieve best-possible communication complexity. In order to improve upon our extension protocols, one must invent BA primitives for short messages with cost $o(kn^2)$, which seems challenging as we discuss in Section 4.3.
- Our protocols can be easily adapted to the asynchronous setting. To the best of our knowledge, these are the first asynchronous authenticated extension protocols. [3]
- Their simplicity makes our protocols less error-prone and more appealing for practical adoption. On this note, in deriving our results, we discover a flaw in the prior best protocol [15, 16] and we provide a simple fix in the full version of this paper.

2 Related Work

Timing and setup assumptions. With different security assumptions on the adversary and timing assumptions, Byzantine broadcast and agreement can be solved for different thresholds of the Byzantine parties. For the timing assumptions, protocols under both synchrony and asynchrony have been studied. If a trusted setup like public-key infrastructure (PKI) exists, it is called the authenticated setting; otherwise, it is the unauthenticated setting. In the synchronous setting, BB/BA can be solved under $t < n/3$ without authentication [19]; with authentication, BA can be solved under $t < n/2$ and BB can be solved under $t < n$ [19, 11, 29]. In the asynchronous setting, BB is impossible; BA (randomized) and reliable broadcast can be solved under $t < n/3$ with or without authentication [5, 6].

Previous extension protocols. Table 1 summarizes the two most closely related works on authenticated extension protocols. Here, we mention several other ones. Cachin and Tessaro [7] adapt Bracha's broadcast [5] to handle l-bit long messages with communication cost $O(nl + k_h n^2 \log n)$. Their method partially inspired our work; but their method does

[3] Asynchronous unauthenticated protocol exist and they can be used in the authenticated setting, but the cost would be much higher than our new protocols (refer to the full version of this paper.)

not seem to apply to general protocols and hence does not yield an extension protocol. Related unauthenticated extension protocols are summarized in the full version. Liang and Vaidya [20] propose the first optimal error-free BB and BA with communication complexity $O(nl + (n^2\sqrt{l} + n^4)\mathcal{B}(1))$ for the synchronous case. Patra [28] improves the communication complexity to $O(nl + n^2\mathcal{B}(1))$ under synchrony and also extended the protocols to asynchrony with increased communication complexity.

State-of-the-art oracle schemes. To better interpret the improvements we obtained for extension protocols, we provide a summary of the state-of-the-art broadcast and agreement protocols that can be used as the oracle in our extension protocol. Since our extension protocols are all deterministic, we focus on *deterministic* solutions for the most part of the paper, except for asynchronous BA where randomization is necessary. The best deterministic solution to authenticated BB for $t < n$ is the classic Dolev-Strong [11] protocol. After applying multi-signatures, the communication complexity to broadcast k bits is $\mathcal{B}(k) = \Theta((k + k_s)n^2 + n^3)$ where k_s is the signature size. The Dolev-Strong protocol can also be modified to solve authenticated BA for the $t < n/2$ case (BA is impossible if $t \geq n/2$). Using an initial all-to-all round with multi-signature to simulate the sender, the communication complexity remains as $\mathcal{A}(k) = \Theta((k+k_s)n^2+n^3)$. In the unauthenticated setting, only $t < n/3$ Byzantine parties can be tolerated and Berman et al. [3] achieves $\mathcal{B}(1) = \mathcal{A}(1) = \Theta(n^2)$ (when $t = \Theta(n)$), matching the lower bound on communication complexity.

In the asynchronous setting, Bracha's reliable broadcast [5] is deterministic and has communication complexity $\mathcal{B}(1) = O(n^2)$. Randomization is necessary for asynchronous BA given the FLP impossibility [13]. State-of-art protocols rely on "common coins" to provide shared randomness but are deterministic otherwise. The most efficient unauthenticated asynchronous BA [25] achieves expected communication complexity $\mathcal{A}(1) = O(n^2)$ assuming a common coin oracle. The most efficient authenticated asynchronous BA [1] achieves expected communication complexity $\mathcal{A}(k) = O((k + k_s)n^2)$ and provides a construction for the common coin oracle.

Coding schemes in consensus systems. Several works have taken advantage of coding schemes in practical fault-tolerant consensus systems. HoneyBadgerBFT [24] and BEAT [12] use the reliable broadcast proposed by Cachin and Tessaro [7] as a component for broadcasting blocks efficiently. Recent works also apply erasure coding to crash-tolerant systems like Paxos [26] and Raft [31].

3 Preliminaries

We consider n parties $P_1, ..., P_n$ connected by a reliable, authenticated all-to-all network, where up to t parties may be corrupted by an adversary A and behave in a Byzantine fashion. We consider both the synchronous model, where there exists a known upper bound on the communication and computation delay, and the asynchronous model, where such an upper bound does not exist. We consider a *static* adversary which decides the set of corrupted parties at the beginning of the execution. We denote parties that are not corrupted by the adversary as honest parties. Two types of the adversary are considered: a computationally bounded adversary is considered in cryptographically secure protocols and a computationally unbounded adversary is considered in the error-free protocols. Our cryptographically secure protocols additionally assume a trusted setup for a public key infrastructure (PKI) and cryptographic accumulators (see Section 3.1). The *communication complexity* [33] of the protocol is measured by the worst-case or expected number of bits transmitted by the honest parties according to the protocol specification over all possible executions under any adversary strategy. Here, we provide the formal definition of Byzantine broadcast (BB) and Byzantine agreement (BA).

▶ **Definition 1** (Byzantine Broadcast). *A protocol for a set of parties $\mathcal{P} = \{P_1, ..., P_n\}$, where a distinguished party called the sender $P_s \in \mathcal{P}$ holds an initial l-bit input m, is a Byzantine broadcast protocol tolerating an adversary A, if the following properties hold*

- *Termination. Every honest party outputs a message.*
- *Agreement. All the honest parties output the same message.*
- *Validity. If the sender is honest, all honest parties output the message m.*

▶ **Definition 2** (Byzantine Agreement). *A protocol for a set of parties $\mathcal{P} = \{P_1, ..., P_n\}$, where each party $P_i \in \mathcal{P}$ holds an initial l-bit input m_i, is a Byzantine agreement protocol tolerating an adversary A, if the following properties hold*

- *Termination. Every honest party outputs a message.*
- *Agreement. All the honest parties output the same message.*
- *Validity. If every honest party P_i holds the same input message m, then all honest parties output the message m.*

For cryptographically secure protocols and randomized protocols, the above properties hold except for a negligible probability in the security parameter. For brevity, our theorem statements will not mention this explicitly.

3.1 Primitives

In this section, we define several primitives that will be used in our extension protocols. Our extension protocols use standard coding and cryptographic schemes from the literature, such as linear error correcting codes, muti-signature schemes and cryptographic accumulators.

Linear error correcting code [30]. We will use standard Reed-Solomon (RS) codes [30] in our protocols, which is a (n, b) RS code in Galois Field $\mathbb{F} = GF(2^a)$ with $n \leq 2^a - 1$. This code encodes b data symbols from $GF(2^a)$ into codewords of n symbols from $GF(2^a)$, and can decode the codewords to recover the original data.

- $\texttt{ENC}$. Given inputs $m_1, ..., m_b$, an encoding function $\texttt{ENC}$ computes $(s_1, ..., s_n) = \texttt{ENC}(m_1, ..., m_b)$, where $(s_1, ..., s_n)$ are codewords of length n. By the property of the RS code, knowledge of any b elements of the codeword uniquely determines the input message and the remaining of the codeword.
- $\texttt{DEC}$. The function $\texttt{DEC}$ computes $(m_1, ..., m_b) = \texttt{DEC}(s_1, ..., s_n)$, and is capable of tolerating up to c errors and d erasures in codewords $(s_1, ..., s_n)$, if and only if $n - b \geq 2c + d$. In our protocol, We will invoke $\texttt{DEC}$ with specific values of c, d satisfying the above relation, and $\texttt{DEC}$ will return correct output.

In our extension protocols, we will use the above RS codes with n equal the number of all parties, and b equal the number of honest parties, i.e., $b = n - t$.

Multi-signatures [4]. Multi-signature scheme can aggregate n signatures into one signature, therefore reduce the size of signatures. Given n signatures $\sigma_i = \texttt{Sign}(sk_i, m)$ on the same message m with corresponding public keys pk_i for $1 \leq i \leq n$, a multi-signature scheme can combine the n signatures above into one signature Σ where $|\Sigma| = |\sigma_i|$. The combined signature can be verified by anyone using a verification function $\texttt{Ver}(PK, \Sigma, m, \mathcal{L})$, where $\mathcal{L}$ is the list of signers and PK is the union of n public keys pk_i.

Cryptographic accumulators [2, 9]. We present the definition of *cryptographic accumulators* proposed by Barić and Pfitzmann [2]. Intuitively, the cryptographic accumulator constructs an accumulation value for a set of values and can produce a witness for each value in the set. Given the accumulation value and a witness, any party can verify if a value is indeed in the set. Formally, given a parameter k, and a set $\mathcal{D}$ of n values $d_1, ..., d_n$, an accumulator has the following components:

- $\texttt{Gen}(1^k, n)$: This algorithm takes a parameter k represented in unary form 1^k and an accumulation threshold n (an upper bound on the number of values that can be accumulated securely), returns an accumulator key ak. This step is run by a trusted dealer, so the accumulator key ak is known to all parties.
- $\texttt{Eval}(ak, \mathcal{D})$: This algorithm takes an accumulator key ak and a set $\mathcal{D}$ of values to be accumulated, returns an accumulation value z for the value set $\mathcal{D}$.
- $\texttt{CreateWit}(ak, z, d_i)$: This algorithm takes an accumulator key ak, an accumulation value z for $\mathcal{D}$ and a value d_i, returns $\perp$ if $d_i \notin \mathcal{D}$, and a witness w_i if $d_i \in \mathcal{D}$.
- $\texttt{Verify}(ak, z, w_i, d_i)$: This algorithm takes an accumulator key ak, an accumulation value z for $\mathcal{D}$, a witness w_i and a value d_i, returns *true* if w_i is the witness for $d_i \in \mathcal{D}$, and *false* otherwise.

For simplicity, our definition of the cryptographic accumulator above omits the auxiliary information aux that appears in the standard definition [2] because the bilinear accumulator we will use does not use aux. We also assume that the function `Eval` is *deterministic*, which is the case with the bilinear accumulator. We give the detailed description of the *bilinear accumulator* [27, 17] in the full version. The bilinear accumulator satisfies the following property.

▶ **Lemma 3** (Collision-free accumulator [27]). *The bilinear accumulator is collision-free. That is, for any set size n and any probabilistic polynomial-time adversary $\mathcal{A}$, there exists a negligible function* $\mathsf{negl}(\cdot)$*, such that*

$$\Pr\left[\begin{array}{l} ak \leftarrow \mathit{Gen}(1^k, n), (\{d_1, ..., d_n\}, d', w') \leftarrow \mathcal{A}(1^k, n, ak), z \leftarrow \mathit{Eval}(ak, \{d_1, ..., d_n\}) : \\ (d' \notin \{d_1, ..., d_n\}) \wedge (\mathit{Verify}(ak, z, w', d') = \mathit{true}) \end{array}\right] \leq \mathsf{negl}(k)$$

To better understand the definition of the cryptographic accumulator, it is helpful to note that the Merkle tree [23] is a cryptographic accumulator, where the accumulator key ak is the hash function, the accumulation value z is the Merkle tree root, and the witness w is the Merkle tree proof. We will use the *bilinear accumulator* [27, 17] instead of Merkle tree in our protocols, since the witness size of the Merkle tree is logarithmic in the number of values whereas the witness size of the bilinear accumulator is a constant. On the other hand, the bilinear accumulator requires a trusted dealer, which is a stronger trust assumption than public key infrastructure (PKI). The trusted dealer needs to know an upper bound on $|\mathcal{D}|$, i.e., the number of items accumulated (see the construction in the full version). In our protocols, $|\mathcal{D}|$ always equals the number of parties n. Hence, the trusted setup (both the PKI and the accumulator) can be reused across invocations if the parties participating in the extension protocol do not change. If a trusted dealer for accumulators cannot be assumed, our protocol can use Merkle tree as the accumulator; in that case, the $O(kn^2)$ term in the communication complexity becomes $O(kn^2 \log n)$ and our protocol still has an advantage (albeit smaller) over prior art.

Normalizing the length of cryptographic building blocks. Let λ denote the security parameter, $k_h = k_h(\lambda)$ denote the hash size, $k_s = k_s(\lambda)$ denote the (multi-)signature size, $k_a = k_a(\lambda)$ denote the size of the accumulation value and witness of the accumulator. Further let $k = \max(k_h, k_s, k_a)$; we assume $k = \Theta(k_h) = \Theta(k_s) = \Theta(k_a) = \Theta(\lambda)$ This assumption is reasonable since the signature scheme and accumulator scheme with the shortest output length are both based on pairing-friendly curves, which are believed to require $\Theta(\lambda)$ bits for λ-bit security given the state-of-the-art attack [18]. As for hash functions, it is common to model them as random oracles, in which case λ-bit security requires $\Theta(\lambda)$-bit hash size. Therefore, throughout the paper, we can use the same variable k to denote the hash size, signature size and accumulator size for convenience.

4 Cryptographically Secure Extension Protocols under $t < n/2$ Faults

In this section, we design cryptographically secure extension protocols with improved communication complexity for the synchronous and authenticated setting with $t < n/2$ faults. We start by presenting some building blocks that will be frequently used in all our authenticated protocols. Then, we give an extension protocol for synchronous BA with communication complexity $O(nl+\mathcal{A}(k)+kn^2)$. Under synchrony, this also implies a BB protocol with $t < n/2$ and the same communication complexity, by first having the sender send the message to all parties and then performing a Byzantine agreement [22]. In the full version, we show another extension protocol for $t < n/2$ BB with communication complexity $O(nl+\mathcal{B}(k)+\mathcal{A}(1)+kn^2)$. The protocols are adapted to the asynchronous case in Section 6. At the end of this section, we discuss the small gap between our BA protocol and a simple lower bound on BA with long messages.

4.1 Building Blocks: `Encode`, `Distribute` and `Reconstruct`

We first define three subprotocols `Encode`, `Distribute` and `Reconstruct` that will be used as building blocks for our cryptographically secure extension protocols, listed in Figure 1.

- `Encode` first divides a message m into b blocks, then compute n coded values $(s_1, ..., s_n)$ using RS codes (defined in Section 3), and attaches an index j for each value s_j. The purpose of `Encode` is to introduce resilience by encoding the message into fault-tolerant coded values – after applying `Encode` to a message m, even if $n - b$ coded values in $(s_1, ..., s_n)$ are erased, one can recover the message from the remaining coded values.
- `Distribute` computes a witness w_j for each indexed value $\langle j, s_j \rangle$ in the input set, and sends the j-th value with its witness to party j. The purpose of `Distribute` is to distribute the values in a robust yet efficient manner – if at least one honest party that has the correct message m (the accumulation value z of m is correct) invokes `Distribute`, then it is guaranteed that any honest party j receives and accepts the j-th value s_j of m, thanks to the witness w_j sent together with the value.
- `Reconstruct` first removes any invalid value s_j that cannot be verified by witness w_j and the accumulation value z, and then decode the message m using RS code (defined in Section 3) from the remaining values with at most d_0 values being removed. The purpose of `Reconstruct` is to recover the message, despite the presence of at most d_0 corruptions in the value, which will be detected by the accumulator scheme and thus erased.

Our extension protocols in Sections 4 and 5 will use `Encode` at the beginning of the protocol to encode the input message to coded values, use `Distribute` in the middle to let every party distribute their coded values with the witnesses, and use `Reconstruct` to reconstruct the original input message after receiving the coded values.

- $\texttt{Encode}(\boldsymbol{m}, \boldsymbol{b})$
  Input: a message m, a number b
  Output: n coded values $s_1, ..., s_n$
  Divide m into b blocks evenly, $m_1, ..., m_b$, each has l/b bits where l is the length of m. Compute $(s_1, ..., s_n) = \texttt{ENC}(m_1, ..., m_b)$ using RS codes, where ENC is defined in Section 3.1. Add an index to every value in $(s_1, ..., s_n)$, i.e., $\mathcal{D} = (\langle 1, s_1\rangle, ..., \langle n, s_n\rangle)$, and return $\mathcal{D}$.
- $\texttt{Distribute}(\boldsymbol{\mathcal{D}}, \boldsymbol{ak}, \boldsymbol{z})$
  Input: a set of indexed values $\mathcal{D} = (\langle 1, s_1\rangle, ..., \langle n, s_n\rangle)$, an accumulator key ak, an accumulation value z
  Compute $w_j = \texttt{CreateWit}(ak, z, \langle j, s_j\rangle)$ for every $\langle j, s_j\rangle \in \mathcal{D}$. Send (s_j, w_j) to party P_j for every $j \in [n]$.
- $\texttt{Reconstruct}(\boldsymbol{\mathcal{S}}, \boldsymbol{ak}, \boldsymbol{z}, \boldsymbol{d_0})$
  Input: $\mathcal{S} = ((\langle 1, s_1\rangle, w_1), ..., (\langle n, s_n\rangle, w_n))$ where each $(\langle i, s_i\rangle, w_i)$ is a pair of indexed value and witness, an accumulator key ak, an accumulation value z, a number d_0
  Output: a message m
  For every $j \in [n]$, if $\texttt{Verify}(ak, z', w_j, \langle j, s_j\rangle) =$ false, let $s_j = \perp$. Apply DEC on the codewords $(s_1, ..., s_n)$ with $c = 0$ and $d = d_0$, where DEC is defined in Section 3.1. Return $m = m_1|...|m_b$ where $m_1, ..., m_b$ are the data returned by DEC.

Figure 1 Building Blocks.

▶ **Lemma 4.** *For any message m, let $z = \texttt{Eval}(ak, \texttt{Encode}(m, b))$. The adversary cannot find $m' \neq m$ such that $z = \texttt{Eval}(ak, \texttt{Encode}(m', b))$ except for negligible probability in k.*

Proof. Let $\mathcal{D} = \texttt{Encode}(m, b)$ and $\mathcal{D}' = \texttt{Encode}(m', b)$. By the RS code, the same codewords correspond to the same message. Thus, if $m \neq m'$, we have $\mathcal{D} \neq \mathcal{D}'$, i.e., there exists $d' = \langle i, s_i\rangle$ such that $d \in \mathcal{D}$ and $d' \notin \mathcal{D}'$. However, under the accumulation value $z = \texttt{Eval}(ak, \mathcal{D}'))$, a witness for $d = \langle i, s_i\rangle \notin \mathcal{D}'$ exists. Due to Lemma 3, this happens with negligible in k probability. ◀

4.2 Byzantine Agreement under $< \frac{n}{2}$ faults

The protocol Synchronous Crypto. $\frac{n}{2}$-BA is presented in Figure 2. In the protocol, let t denote the maximum number of Byzantine parties, and let $b = n - t$. We briefly describe each step of the protocol. First, each party encodes its message using RS codes and computes the accumulation value for the set of coded values. With a deterministic Eval, any honest party with the same accumulator key and set will produce the same accumulation value. The RS codes can recover the message with up to t coded values being erased, and the accumulation value uniquely corresponds to the set of coded values and equivalently the original message (Lemma 4). Then every party runs an instance of k-bit Byzantine agreement with the accumulation value as the input. After the above agreement terminates, each party checks whether the agreement output matches its accumulation value, and inputs the result to an 1-bit Byzantine agreement instance. If the above agreement outputs 0, all parties output $\perp$ and abort. If the above agreement outputs 1, then at least one honest party has the accumulation value z_i matching with the agreement output z, and every honest party will agree on the message corresponding to z. Then in Distribute, all parties send the j-th coded value to party P_j. After that, each honest party P_j will send a valid j-th coded

Input of every party P_i: An l-bit message m_i
Primitives: Byzantine agreement oracle, cryptographic accumulator with `Eval`, `CreateWit`, `Verify`
Protocol for party P_i:

1. Compute $\mathcal{D}_i = (\langle 1, s_1\rangle, ..., \langle n, s_n\rangle) = \texttt{Encode}(m_i, b)$. Compute the accumulation value $z_i = \texttt{Eval}(ak, \mathcal{D}_i)$. Input z_i to an instance of k-bit BA oracle.
2. When the above BA outputs z, if $z = z_i$, set $happy_i = 1$, otherwise set $happy_i = 0$. Input $happy_i$ to an instance of 1-bit Byzantine agreement oracle.
3. - If the above BA outputs 0, output $o_i = \perp$ and abort.
   - If the above BA outputs 1 and $happy_i = 1$, invoke $\texttt{Distribute}(\mathcal{D}_i, ak, z)$.
4. For the set of pairs $\{(s_i, w_i)\}$ received from the previous step, if there exists a pair (s_i, w_i) such that $\texttt{Verify}(ak, z, w_i, \langle i, s_i\rangle) = true$, then send (s_i, w_i) to all other parties.
5. If $happy_i = 1$, set $o_i = m_i$. Otherwise, let (s_j, w_j) be the message received from party P_j from the previous step and $\mathcal{S}_i = ((\langle 1, s_1\rangle, w_1), ..., (\langle n, s_n\rangle, w_n))$, and set $o_i = \texttt{Reconstruct}(\mathcal{S}_i, ak, z, t)$.
6. Output o_i.

Figure 2 Protocol Synchronous Crypto. $\frac{n}{2}$-BA.

value to all other parties, from which the correct message can be obtained in `Reconstruct`. One nice property of our protocol is that, if at least one honest party with message m invokes `Distribute`, then all honest parties can obtain m from `Reconstruct` (see the proof of Lemma 6). We prove the validity and agreement properties and analyze the communication complexity below.

▶ **Lemma 5.** *If every honest party has the same input message $m_i = m$, all honest parties output the same message m.*

Proof. If all honest parties have the same input message $m_i = m$, they compute and input the same accumulation value z to the instance of Byzantine agreement in step 1. Then in step 2, the BA outputs z by the validity condition, and any honest party sets $happy_i = 1$. Therefore, every honest party P_i inputs 1 to the 1-bit Byzantine agreement oracle in step 2. By the validity of the Byzantine agreement oracle, the agreement will output 1. Then any honest party P_i sets $o_i = m$ in step 5 since $happy_i = 1$. Hence, all honest parties output m when the protocol terminates. ◀

▶ **Lemma 6.** *All honest parties output the same message.*

Proof. If the Byzantine agreement in step 3 outputs 0, then all honest parties output the same message $\perp$. If the agreement agreement in step 3 outputs 1, then by the validity of the Byzantine agreement, some honest party P_i must input 1 and thus has $z_i = z$. By Lemma 4, any honest party P_i with $happy_i = 1$ has the identical message m corresponding to z, and sets the output to be m at step 5. In step 3, any honest party P_i with $happy_i = 1$ invokes `Distribute` to compute witness w_j for each index value $\langle j, s_j\rangle$, and sends the valid (s_j, w_j) pair computed from message m to party P_j for every P_j. By Lemma 3, the Byzantine parties cannot generate a different pair (s'_j, w'_j) that can be verified. Therefore, in step 4, every honest party P_j receives at least one valid (s_j, w_j) pair, and forwards it to all other parties. Since there are at least $b = n - t$ honest parties, in step 5, each honest party will receive at least b valid coded values. In `Reconstruct`, using the accumulation value associated with the coded value, any party P_i can detect the corrupted values and remove them. By the property of RS codes, any honest party P_i with $happy_i = 0$ is able to recover the message m, and any honest party P_i with $happy_i = 1$ already has the message m. Therefore all honest parties outputs m. ◀

▶ **Theorem 7.** *Protocol Synchronous Crypto. $\frac{n}{2}$-BA satisfies Termination, Agreement and Validity, and has communication complexity $O(nl + \mathcal{A}(k) + kn^2)$.*

Proof. Termination is clearly satisfied. By Lemma 6, agreement is satisfied. By Lemma 5, validity is satisfied.

Step 1 has cost $\mathcal{A}(k)$, where k is the size of the cryptographic accumulator. Step 2 has cost $\mathcal{A}(1) \leq \mathcal{A}(k)$. Step 3 has cost $O(nl + kn^2)$, since each honest party invokes an instance of `Distribute`, which leads to an all-to-all communication with each message of size $O(l/b + k) = O(l/n + k)$. For step 4, it also has cost $O(nl + kn^2)$ similarly as step 3. Hence the total cost is $O(nl + \mathcal{A}(k) + kn^2)$. ◀

4.3 Lower Bound on BA for Long Messages

Let $\mathcal{A}(l)$ denote the communication complexity in bits of the best possible deterministic protocol for Byzantine agreement with l-bit inputs, n parties, and up to $t = \Theta(n)$ faulty parties. We show a straightforward lower bound that $\mathcal{A}(l) = \Omega(nl + \mathcal{A}(k) + n^2)$ for $l \geq k$ by combining several known lower bounds in the literature.

▶ **Theorem 8.** $\mathcal{A}(l) = \Omega(nl + \mathcal{A}(k) + n^2)$ *for $l \geq k$.*

Proof. The proof combines several simple lower bounds known in the literature.

First of all, $\mathcal{A}(l) = \Omega(nl)$ according to [14]. We briefly mention the proof idea from [14] for completeness. Let a set A of $n - t$ parties have input m and a set B of the rest t parties have input $m' \neq m$. In scenario 1, let parties in B be Byzantine but behave as if they are honest. Then by the validity condition, all parties in A will output m. In scenario 2, let parties in B be honest. To parties in A, the scenario 2 is indistinguishable from scenario 1, and thus they will output m. By the agreement condition, parties in B also need to output m. Therefore each party in B needs to learn the message m, which leads to a lower bound on the communication cost of $\Omega(tl) = \Omega(nl)$.

Secondly, since $\mathcal{A}(l)$ denotes the communication complexity of a BA oracle with l-bit inputs, it is clear that $\mathcal{A}(l) \geq \mathcal{A}(k)$ for $l \geq k$.

Finally, according to [10], $\Omega(n^2)$ is a lower bound on the communication complexity for any deterministic Byzantine agreement protocol tolerating $t = \Theta(n)$ faults (even for single-bit inputs). Thus, $\mathcal{A}(l) \geq \mathcal{A}(1) = \Omega(n^2)$.

The above lower bounds together imply a lower bound $\mathcal{A}(l) = \Omega(nl + \mathcal{A}(k) + n^2)$ for deterministic protocol that solves l-bit BA. ◀

By Theorem 7, our Protocol Synchronous Crypto. $\frac{n}{2}$-BA has cost $O(nl + \mathcal{A}(k) + kn^2)$, which is very close to the lower bound. Although it does not meet the lower bound, we remark that further improvements seem challenging. Notice that if $\mathcal{A}(k) = \Omega(kn^2)$, then a lower bound of $\Omega(nl + \mathcal{A}(k) + kn^2)$ follows, matching our upper bound. Thus, improving upon our upper bound requires a k-bit BA oracle whose communication complexity is $o(kn^2)$.

However, if we were to design an $o(kn^2)$ BA protocol, we have to follow a very particular paradigm. The $\Omega(n^2)$ lower bound from [10] is a lower bound on the number of messages. If every message is signed, then $\Omega(kn^2)$ communication must be incurred. Yet, we know

Input of the sender P_s: An l-bit message m_s
Primitives: Byzantine broadcast oracle, cryptographic accumulator with `Eval`, `CreateWit`, `Verify`
Protocol for party P_i:

1. The sender P_s initializes $o_s = m_s, happy_s = 1$, and other parties P_i initialize $o_i = \perp, happy_i = 0$. The sender computes $\mathcal{D}_s = \texttt{Encode}(m_s, b)$, the accumulator value $z_s = \texttt{Eval}(ak, \mathcal{D}_s)$, and broadcasts z_s by invoking an instance of k-bit Byzantine broadcast oracle. Let z_i denote the output of the Byzantine broadcast at party P_i.
2. For iterations $r = 1, ..., t+1$:
   Distribution step:
   If $happy_i = 1$, then sign the `HAPPY` message using the multi-signature scheme, send the multi-signature signed by r distinct parties to all other parties, invoke $\texttt{Distribute}(\mathcal{D}_i, ak, z_i)$, and skip the Distribution step in all future iterations.
   Sharing step:
   If a valid (s_i, w_i) pair is received from the Distribution step such that $\texttt{Verify}(ak, z_i, w_i, \langle i, s_i \rangle) = \text{true}$, then send (s_i, w_i) to all other parties and skip the Sharing step in all future iterations.
   Reconstruction step (no communication involved):
   Let (s_j, w_j) be the first message received from party P_j from the Sharing step (possibly from previous iterations). Let $\mathcal{S}_i = ((\langle 1, s_1 \rangle, w_1), ..., (\langle n, s_n \rangle, w_n))$. Compute $M_i = \texttt{Reconstruct}(\mathcal{S}_i, ak, z_i, t)$ and $\mathcal{D}_i = \texttt{Encode}(M_i, b)$. If $\texttt{Eval}(ak, \mathcal{D}_i) = z_i$ and a `HAPPY` message signed by r distinct parties excluding P_i was received in the Distribution step of this iteration, then set $happy_i = 1$, set $o_i = M_i$, and skip the Reconstruction step in all future iterations.
3. Output o_i.

Figure 3 Protocol Synchronous Crypto. $(1-\varepsilon)$-BB.

authentication is necessary for tolerating minority faults. Thus, such a protocol must use $\Omega(n^2)$ messages in total but only sign a small subset of them. We are not aware of any work exploring this direction, and closing this gap is an interesting open problem.

5 Cryptographically Secure Extension Protocol under $t < (1-\varepsilon)n$

In this section, we propose an extension protocol for synchronous and authenticated BB with communication complexity $O(nl + \mathcal{B}(k) + kn^2 + n^3)$ under $t < (1-\varepsilon)n$ where $\varepsilon > 0$ is some constant. The protocol still solves Byzantine broadcast under any $t < n$ faults by setting $b = n - t$, but the communication complexity increases by a factor of $1/\varepsilon$ if ε is not a constant (see Theorem 12). Thus, compared to state-of-art solutions [15, 16], our protocol is more efficient when ε is a constant but less efficient otherwise.

Protocol Synchronous Crypto. $(1-\varepsilon)n$-BB. The protocol is presented in Figure 3, and we briefly explain each step of the protocol. Again let t denote the maximum number of Byzantine parties and let $b = n - t$. First the sender encodes its message and computes the accumulation value using the coded values. Then the sender broadcasts the accumulation value via an instance of k-bit Byzantine broadcast oracle. By the agreement condition, all honest replicas output the same value for BB. The remaining of the protocol runs in iterations

$r = 1, 2, ..., t+1$. Each iteration consists 3 steps. The Distribution step, Sharing step and Reconstruction step are analogous to steps $3-5$ in Protocol Synchronous Crypto. $\frac{n}{2}$-BA in Figure 2, but here each step is examined in every iteration for execution, and is executed only once. The Distribution step aims to distribute the indexed coded values to other parties. The Sharing step forwards the correct coded value to other parties. The Reconstruction step aims to reconstruct the original message from the coded values received from other parties and set the output. Similar to Protocol Synchronous Crypto. $\frac{n}{2}$-BA, the above steps provide a nice guarantee that if at least one honest party with message m invokes `Distribute` in the Distribution step, then all honest parties can obtain m in the Reconstruction step (see the proof of Lemma 9).

Now we give a more detailed description. A party becomes happy (i.e., sets $happy_i = 1$) if it is ready to output a message that is not $\perp$. In the first iteration, only the sender is happy; it invokes `Distribute` and also signs and sends a message `HAPPY` of a constant size. The role of the message `HAPPY` is to be signed by the rest of the parties using multi-signatures to form a signature chain, similar to the Dolve-Strong Byzantine broadcast algorithm [11]. An honest party becomes happy at the end of iteration r, if it reconstructs the correct message (matching the agreed upon accumulation value) in the Reconstruction step of iteration r and has received a `HAPPY` message signed by r parties in the Distribution step of iteration r. When an honest party becomes happy, it will set its output to be the reconstructed message M_i; then, in the Distribution step of the next iteration (if there is one), it will also send its own signature of `HAPPY` to all other parties, and invoke `Distribute`. This way, if an honest party becomes happy in the last iteration $r = t + 1$, it can be assured that some honest party has invoked `Distribute`, so that all honest parties will be ready to output the correct message. We reiterate that each step is executed at most once in the entire protocol. Finally, after $t + 1$ iterations, every party outputs the message.

▶ **Lemma 9.** *If any honest party P_i invokes `Distribute` with message m, then every honest party P_j outputs $o_j = m$.*

Proof. By the agreement condition of the Byzantine broadcast, the output z_i of the BB at every honest party P_i is identical. If an honest party P_i invokes `Distribute` with message m, m satisfies $z_i = \texttt{Eval}(ak, \texttt{Encode}(m, b))$. If any other honest party P_j sets $o_j = m'$ after initialization, it must satisfy $\texttt{Eval}(ak, \texttt{Encode}(m', b)) = z_j = z_i$. By Lemma 4, $m = m'$. Thus, we only need to show that every other honest party P_j sets o_j.

Suppose that P_i invokes `Distribute` in some iteration r. According to the subprotocol `Distribute`, P_i computes a witness w_j for each indexed value $\langle j, s_j\rangle$ and sends the pair (s_j, w_j) to each party P_j. According to Lemma 3, the adversary cannot generate $d' \notin \mathcal{D}_i$ and a witness w' such that $\texttt{Verify}(ak, z_i, w', d') = true$. Then, in Sharing step of iteration r, every honest party P_j can identify and forward the valid pair (s_j, w_j) to all other parties, unless it has already done that in previous iterations. Since there are at least $n - t = b$ honest parties, in the Reconstruction step of iteration r, every honest party P_j receives at least $n - t = b$ correct coded values. In `Reconstruct`, using the witness associated with the indexed coded value, every party P_j can identify the corrupted values and remove them. The number of erased values is at most t. By the property of RS codes, P_j with $happy_j = 0$ is able to recover the message m.

Furthermore, we will show that each party receives a `HAPPY` message signed by r distinct parties in the Reconstruction step of iteration r. If $r = 1$, then $P_i = P_s$ and every P_j will receive a signature for `HAPPY`. If $r > 1$, then P_i has received a multi-signature of `HAPPY` signed by $r - 1$ distinct parties excluding P_i in the Reconstruction step of iteration $r - 1$; P_i adds its own signature of `HAPPY` in iteration r, so each honest P_j will receive a multi-signature of `HAPPY` signed by r distinct parties in the Reconstruction step of iteration r.

Therefore, if $happy_j = 0$ up till now, then an honest P_j will set $happy_j = 1$ and $o_j = m$ in the Reconstruction step of iteration r. If $happy_j = 1$, then P_j has already set $o_j = m$. Note that an honest sender does not set its output again in the Reconstruction step, since the `HAPPY` message always contains its signature. Once P_j sets o_j, it will skip the Reconstruction step in all future iterations, and o_j will not be changed. Therefore, all honest parties output m when they terminate. ◀

▶ **Lemma 10.** *If the sender is honest and has input m_s, every honest party outputs m_s.*

Proof. In iteration $r = 1$, the sender sends a signed `HAPPY` to all other parties and invokes `Distribute`. By Lemma 9, every honest parties output m_s. ◀

▶ **Lemma 11.** *Every honest party outputs the same message.*

Proof. If all honest parties output $\perp$, then the lemma is true. Otherwise, suppose some honest party P_i outputs $o_i = m$ where $m \neq \perp$. If P_i is the sender, then by Lemma 10, all honest parties output m. Now consider the case where P_i is not the sender. According to the protocol, if $P_i \neq P_s$ sets $o_i = m \neq \perp$ in the Reconstruction step of iteration $1 \leq r \leq t$, P_i will invoke `Distribute` with m in iteration $r + 1$. By Lemma 9, all honest parties output m. If the honest party P_i sets $o_i = m$ in iteration $r = t + 1$, according to the protocol, P_i receives a `HAPPY` signed by $t + 1$ distinct parties. Since there are at most t Byzantine parties, there exists at least one honest party $P_j \neq P_i$ that has signed `HAPPY` and invoked `Distribute` with $o_j = m'$ in a previous iteration $1 \leq r' \leq t$. Then, by Lemma 9, all honest parties including P_i output m'. Therefore, $m' = m$, and all honest parties output m. ◀

▶ **Theorem 12.** *Protocol Synchronous Crypto. $(1-\varepsilon)n$-BB satisfies Termination, Agreement and Validity. The protocol has communication complexity $O(nl/\varepsilon + \mathcal{B}(k) + kn^2 + n^3)$.*

Proof. Termination is clearly satisfied. By Lemma 11, agreement is satisfied. By Lemma 10, validity is satisfied.

Step 1 has cost $\mathcal{B}(k)$ for the k-bit BB oracle. The Distribution step has total communication cost $O(nl/\varepsilon + kn^2 + n^3)$, since each honest party executes the Distribution step at most once, where invoking `Distribute` has cost $O(n \cdot (l/b+k)) = O(\frac{n}{n-t}l + kn) = O(l/\varepsilon + kn)$, and sending the signed `HAPPY` message has cost $O((k + n)n)$ where the $(k + n)$ term is due to the signature size and the list of signers in the multi-signature scheme. The Sharing step is also performed at most once for every honest party, and has total cost $O(nl/\varepsilon + kn^2)$ since each honest party in the Sharing step sends a message of size $O(l/(n\varepsilon)+k)$ to all other parties. The Reconstruction step has no communication cost. Hence, the total communication complexity is $O(nl/\varepsilon + \mathcal{B}(k) + kn^2 + n^3)$. ◀

Optimality with the current best BB oracle. From Section 2, the classic Dolev-Strong [11] protocol remains the best deterministic solution for $t > n/2$ BB, with cost $\mathcal{B}(k) = \Theta((k + k_s)n^2 + n^3)$ for k-bit inputs where k_s is the signature size. Our protocol invokes Dolev-Strong with $k = k_a$ (the size of the accumulation value). Since $\Theta(k_s) = \Theta(k_a)$, our protocol achieves $\mathcal{B}(l) = O(nl + kn^2 + n^3)$.

As before, $\Omega(nl)$ is a trivial lower bound for l-bit BB [14] (intuitively, all parties need to receive the sender's message); in addition $\mathcal{B}(l) \geq \mathcal{B}(k)$ if $l \geq k$. Thus, the $\mathcal{B}(l) = O(nl+kn^2+n^3)$ communication complexity cannot be further improved unless a deterministic BB protocol better than Dolev-Strong is found.

Input of every party P_i: An l-bit message m_i
Primitives: asynchronous Byzantine agreement oracle, cryptographic accumulator with `Eval`, `CreateWit`, `Verify`
Protocol for party P_i:

1. Compute $\mathcal{D}_i = (\langle 1, s_1\rangle, ..., \langle n, s_n\rangle) = \texttt{Encode}(m_i, b)$. Compute the accumulation value $z_i = \texttt{Eval}(ak, \mathcal{D}_i)$. Input z_i to an instance of k-bit asynchronous Byzantine agreement oracle.
2. When the above ABA outputs z, if $z = z_i$, set $happy_i = 1$, otherwise set $happy_i = 0$. Input $happy_i$ to an instance of 1-bit asynchronous Byzantine agreement oracle.
3. - If the above ABA outputs 0, output $o_i = \perp$ and abort.
   - If the above ABA outputs 1 and $happy_i = 1$, invoke $\texttt{Distribute}(\mathcal{D}_i, ak, z)$.
4. Wait for a valid (s_i, w_i) pair such that $\texttt{Verify}(ak, z, w_i, \langle i, s_i\rangle) =$ true, then send (s_i, w_i) to all other parties.
5. If $happy_i = 1$, set $o_i = m_i$. Otherwise, perform the following. Wait for at least $n-t$ valid pairs $\{(s_j, w_j)\}$ from the previous step that satisfies $\texttt{Verify}(ak, z, w_j, \langle j, s_j\rangle) =$ true. Let $\mathcal{S}_i = ((\langle 1, s_1\rangle, w_1), ..., (\langle n, s_n\rangle, w_n))$, where (s_j, w_j) is the pair received from party P_j. Compute $o_i = \texttt{Reconstruct}(\mathcal{S}_i, ak, z, t)$.
6. Output o_i.

Figure 4 Protocol Asynchronous Crypto. $\frac{n}{3}$-BA.

6 Cryptographically Secure Extension Protocols Under Asynchrony

As mentioned, our cryptographically secure extension protocols can be extended to the asynchronous setting to solve BA and reliable broadcast (RB) under $< n/3$ faults. No extension protocol has been proposed for this case to the best of our knowledge. As before, let t denote the maximum number of Byzantine parties, and let $b = n - t$.

6.1 Asynchronous Byzantine Agreement

The protocol is presented in Figure 4, which consists steps analogous to the synchronous protocol. The main difference is that in the asynchronous extension protocol, Steps 4 and 5 are executed once enough messages are received. As a result, the proofs are also similar to the synchronous version and we omit them.

▶ **Theorem 13.** *Protocol Asynchronous Crypto. $\frac{n}{3}$-BA satisfies Termination, Agreement and Validity, and has communication complexity $O(nl + \mathcal{A}(k) + kn^2)$.*

6.2 Asynchronous Reliable Broadcast

Reliable broadcast relaxes the termination property of the broadcast definition (Definition 1): only when the sender is honest, all honest parties are required to output; otherwise, it is allowed that *either* all honest parties output *or* no honest party outputs. The agreement property is slightly modified accordingly.

▶ **Definition 14** (Reliable Broadcast). *A protocol for a set of parties $\mathcal{P} = \{P_1, ..., P_n\}$, where a distinguished party called the sender $P_s \in \mathcal{P}$ holds an initial l-bit input m, is a reliable broadcast protocol tolerating an adversary A, if the following properties hold*

- *Termination. If the sender is honest, then every honest party eventually outputs a message. Otherwise, if some honest party outputs a message, then every honest party eventually outputs a message.*

Input of the sender P_s: An l-bit message m_s
Primitive: asynchronous Byzantine agreement oracle, asynchronous reliable broadcast oracle, cryptographic accumulator with `Eval`, `CreateWit`, `Verify`
Protocol for party P_i:

1. If $i = s$, perform the following. Compute $\mathcal{D}_s = (\langle 1, s_1\rangle, ..., \langle n, s_n\rangle) = \texttt{Encode}(m_s, b)$. Compute the accumulation value $z_s = \texttt{Eval}(ak, \mathcal{D}_s)$. Send m_s to every party, and broadcast z_s by invoking a k-bit asynchronous reliable broadcast oracle.
2. When receiving the message m from the sender, and the reliable broadcast above outputs z, perform the following. Compute $\mathcal{D}_i = (\langle 1, s_1\rangle, ..., \langle n, s_n\rangle) = \texttt{Encode}(m, b)$. Compute the accumulation value $z_i = \texttt{Eval}(ak, \mathcal{D}_i)$. If $z_i = z$, set $happy_i = 1$, otherwise set $happy_i = 0$.
3. If $happy_i = 1$, invoke $\texttt{Distribute}(\mathcal{D}_i, ak, z)$.
4. Step 4 to 6 are identical to that of Protocol Asynchronous Crypto. $\frac{n}{3}$-BA in the Figure 4, except that the replica computes $\mathcal{D}'_i = \texttt{Encode}(o_i, b)$, and invokes $\texttt{Distribute}(\mathcal{D}'_i, ak, z)$ at the end of Step 5.

Figure 5 Protocol Asynchronous Crypto. $\frac{n}{3}$-RB.

- *Agreement. If some honest party outputs a message m', then every honest party eventually outputs m'.*
- *Validity. If the sender is honest, all honest parties eventually output the message m.*

The extension protocol for asynchronous reliable broadcast is presented in Figure 5.

▶ **Lemma 15.** *If an honest party P_i invokes* `Distribute` *with $\mathcal{D}_i = \texttt{Encode}(m, b)$, then any honest party P_j eventually output $o_j = m$.*

Proof. By the agreement condition of asynchronous reliable broadcast oracle used in step 1, if any honest party obtains z, then any honest party also eventually obtains z. Then at step 2, by Lemma 4, any honest party P_j with $happy_j = 1$ has the identical message m corresponding to z, and sets $o_j = m$ at step 5. For other honest parties, the honest party P_i with $happy_i = 1$ invokes `Distribute` to compute witness w_j for each indexed value $\langle j, s_j\rangle$, and sends the valid (s_j, w_j) pair computed from message m to party P_j for every P_j. By Lemma 3, the Byzantine parties cannot generate a different pair (s'_j, w'_j) that can be verified. Therefore, in step 4, every honest party P_j eventually receives at least one valid (s_j, w_j) pair, and forwards it to all other parties. Since there are at least $n - t$ honest parties, in step 5, each honest party will eventually receive at least $n - t$ valid coded values. In `Reconstruct`, using the accumulation value associated with the coded value, any party P_j can detect the corrupted values and remove them. By the property of RS codes, any honest party P_j with $happy_j = 0$ is able to recover the message m, and any honest party P_j with $happy_j = 1$ already has the message m. Therefore all honest parties output m. ◀

▶ **Lemma 16.** *If the sender is honest and has input m_s, all honest parties eventually output the same message m_s.*

▶ **Lemma 17.** *If some honest party outputs a message m, then every honest party eventually outputs m.*

▶ **Theorem 18.** *Protocol Asynchronous Crypto. $\frac{n}{3}$-RB satisfies Termination, Agreement and Validity. The protocol has communication complexity $O(nl + \mathcal{B}(k) + kn^2)$.*

The proofs of Lemma 16, 17 and Theorem 18 are in the full version of this paper.

7 Conclusion

We investigate and propose several extension protocols with improved communication complexity for solving Byzantine broadcast and agreement under various settings. We propose simple yet efficient authenticated extension protocols with improved communication complexity, for Byzantine agreement under $t < n/2$, and for Byzantine broadcast under $t < (1 - \varepsilon)n$ where $\varepsilon > 0$ is a constant. The above results can be extended to the asynchronous case to obtain authenticated extension protocols for Byzantine agreement and reliable broadcast.

References

1 Ittai Abraham, Dahlia Malkhi, and Alexander Spiegelman. Asymptotically optimal validated asynchronous byzantine agreement. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, pages 337–346, 2019.

2 Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 480–494. Springer, 1997.

3 Piotr Berman, Juan A Garay, and Kenneth J Perry. Bit optimal distributed consensus. In *Computer science*, pages 313–321. Springer, 1992.

4 Dan Boneh, Craig Gentry, Ben Lynn, and Hovav Shacham. Aggregate and verifiably encrypted signatures from bilinear maps. In *International Conference on the Theory and Applications of Cryptographic Techniques*, pages 416–432. Springer, 2003.

5 Gabriel Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

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

7 Christian Cachin and Stefano Tessaro. Asynchronous verifiable information dispersal. In *24th IEEE Symposium on Reliable Distributed Systems (SRDS'05)*, pages 191–201. IEEE, 2005.

8 Wutichai Chongchitmate and Rafail Ostrovsky. Information-theoretic broadcast with dishonest majority for long messages. In *Theory of Cryptography Conference*, pages 370–388. Springer, 2018.

9 David Derler, Christian Hanser, and Daniel Slamanig. Revisiting cryptographic accumulators, additional properties and relations to other primitives. In *Cryptographers' Track at the RSA Conference*, pages 127–144. Springer, 2015.

10 Danny Dolev and Ruediger Reischuk. Bounds on information exchange for byzantine agreement. In *Proceedings of the first ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 132–140. ACM, 1982.

11 Danny Dolev and H. Raymond Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

12 Sisi Duan, Michael K Reiter, and Haibin Zhang. Beat: Asynchronous bft made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 2028–2041, 2018.

13 Michael J Fischer, Nancy A Lynch, and Michael S Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

14 Matthias Fitzi and Martin Hirt. Optimally efficient multi-valued byzantine agreement. In *Proceedings of the twenty-fifth annual ACM symposium on Principles of distributed computing*, pages 163–168. ACM, 2006.

15 Chaya Ganesh and Arpita Patra. Broadcast extensions with optimal communication and round complexity. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 371–380. ACM, 2016.

16 Chaya Ganesh and Arpita Patra. Optimal extension protocols for byzantine broadcast and agreement. *Distributed Computing*, pages 1–19, 2020.

17 Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 177–194. Springer, 2010.

18 Taechan Kim and Razvan Barbulescu. Extended tower number field sieve: A new complexity for the medium prime case. In *Annual International Cryptology Conference*, pages 543–571. Springer, 2016.

19 Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, 1982.

20 Guanfeng Liang and Nitin Vaidya. Error-free multi-valued consensus with byzantine failures. In *Proceedings of the 30th annual ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 11–20. ACM, 2011.

21 Yuan Lu, Zhenliang Lu, Qiang Tang, and Guiling Wang. Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*, page 129–138, 2020.

22 Nancy A Lynch. *Distributed algorithms*. Elsevier, 1996.

23 Ralph C Merkle. A digital signature based on a conventional encryption function. In *Conference on the theory and application of cryptographic techniques*, pages 369–378. Springer, 1987.

24 Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn 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.

25 Achour Mostéfaoui, Hamouma Moumen, and Michel Raynal. Signature-free asynchronous binary byzantine consensus with $t < n/3$, $O(n^2)$ messages, and $O(1)$ expected time. *Journal of the ACM (JACM)*, 62(4):1–21, 2015.

26 Shuai Mu, Kang Chen, Yongwei Wu, and Weimin Zheng. When paxos meets erasure code: Reduce network and storage cost in state machine replication. In *Proceedings of the 23rd international symposium on High-performance parallel and distributed computing*, pages 61–72, 2014.

27 Lan Nguyen. Accumulators from bilinear pairings and applications. In *Cryptographers' Track at the RSA Conference*, pages 275–292. Springer, 2005.

28 Arpita Patra. Error-free multi-valued broadcast and byzantine agreement with optimal communication complexity. In *International Conference On Principles Of Distributed Systems*, pages 34–49. Springer, 2011.

29 Birgit Pfitzmann and Michael Waidner. Information-theoretic pseudosignatures and byzantine agreement for $t \geq n/3$. *Research report, IBM Research*, 1996.

30 Irving S Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics*, 8(2):300–304, 1960.

31 Zizhong Wang, Tongliang Li, Haixia Wang, Airan Shao, Yunren Bai, Shangming Cai, Zihan Xu, and Dongsheng Wang. Craft: An erasure-coding-supported version of raft for reducing storage cost and network cost. In *18th USENIX Conference on File and Storage Technologies*, pages 297–308, 2020.

32 Andrew C Yao. Protocols for secure computations. In *Proceedings of the 23rd Annual Symposium on Foundations of Computer Science*, pages 160–164. IEEE Computer Society, 1982.

33 Andrew Chi-Chih Yao. Some complexity questions related to distributive computing. In *Proceedings of the eleventh annual ACM symposium on Theory of computing*, pages 209–213. ACM, 1979.