Asynchronous Data Dissemination and its Applications

Sourav Das
University of Illinois at Urbana-Champaign
souravd2@illinois.edu

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

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

ABSTRACT

In this paper, we introduce the problem of Asynchronous Data Dissemination (ADD). Intuitively, an ADD protocol disseminates a message to all honest nodes in an asynchronous network, given that at least $t+1$ honest nodes initially hold the message where t is the maximum number of malicious nodes. We design a simple and efficient ADD protocol for n parties that is information-theoretically secure, tolerates up to one-third malicious nodes, and has a communication cost of $O(n|M|+n^2)$ for disseminating a message M.

We then use our ADD protocol to improve many important primitives in cryptography and distributed computing. For asynchronous reliable broadcast (RBC), assuming collision-resistant hash functions, we give a RBC protocol with communication cost $O(n|M|+\kappa n^2)$ where κ is the size of the hash function output. This improves over the prior best scheme with communication cost $O(n|M|+\kappa n^2 \log n)$ under the same setting. Our improved RBC protocol immediately improves the communication cost of asynchronous atomic broadcast and Asynchronous Distributed Key Generation (ADKG) protocols. We also use our improved RBC protocol along with additional new techniques to improve the communication cost of Asynchronous Verifiable Secret Sharing (AVSS), Asynchronous Complete Secret Sharing (ACSS), and dual-threshold ACSS from $O(\kappa n^2 \log n)$ to $O(\kappa n^2)$ without using any trusted setup.

KEYWORDS

Data Dissemination; Asynchronous Networks; Reliable Broadcast; Verifiable Secret Sharing; Distributed Key Generation; Communication Complexity.

1 INTRODUCTION

In this paper, we propose and study a new problem which we call *Asynchronous Data Dissemination* (ADD). Briefly, the goal is to disseminate a data blob from a subset of honest nodes to *all* honest nodes, despite the presence of some malicious nodes. More precisely, in a network of n nodes where up to t nodes could be malicious, a subset of at least $t+1$ honest nodes start with a common message blob M, and the goal is to have all honest nodes output M at the end of the protocol.

We present a solution to the ADD problem with $n \geq 3t+1$, where at least $t+1$ honest nodes start with the message M. For a message M of size $|M|$, our protocol has a total communication cost of $O(n|M|+n^2)$. Moreover, our solution is *information theoretically* secure, i.e., it does not rely on any cryptographic assumption. Additionally, if we use a collision-resistant hash function with output size κ, we can extend our solution to any $n > 2t$ with a total communication cost of $O(n^2|M|/(n-2t)+\kappa n^2)$. For example, if $t = (1/2-\epsilon)n$ for some $\epsilon > 0$, our ADD protocol incurs a communication cost of $O(n|M|/\epsilon + \kappa n^2)$.

We then observe that ADD can help improve many important problems, including asynchronous reliable broadcast (RBC) [14, 16], asynchronous verifiable secret sharing (AVSS) [4, 15], asynchronous complete secret sharing (ACSS) [52], dual-threshold ACSS [3, 34], and asynchronous distributed key generation (ADKG) [2, 34], as illustrated in Figure 1.

Overview of our results. To solve ADD, our protocol first leverages error-correcting code to encode the input message M into n symbols, and then the i-th node is responsible for dispersing the i-th symbol. For any node i that has input $\perp$, it learns the i-th symbol by receiving it from $t+1$ nodes (the set of honest nodes that have the input message M). After each node i disperses the i-th symbol, honest nodes repeatedly try to reconstruct the message M after receiving at least $2t+1$ symbols. The node outputs the reconstructed message if it matches at least $2t+1$ symbols. This protocol is information-theoretic, has a total communication cost of $O(n|M|+n^2)$, and tolerates $t < n/3$ faults.

We then use the above ADD protocol to improve a variety of other problems. Table 1 summarizes our results and compares with existing works.

The first primitive we improve is the asynchronous reliable broadcast (RBC) [14, 16], a fundamental primitive in distributed computing. Briefly, an asynchronous reliable broadcast protocol implements a broadcast channel in an asynchronous network and ensures that all honest nodes in the network deliver the same message if any honest node delivers or if the broadcaster is honest. RBC has been used to construct many higher-level protocols such as atomic broadcast [1, 25, 27, 30, 37, 40], asynchronous multi-party computation [36, 52], and asynchronous distributed key generation [2, 34]. All of these involve RBC for long messages, typically of

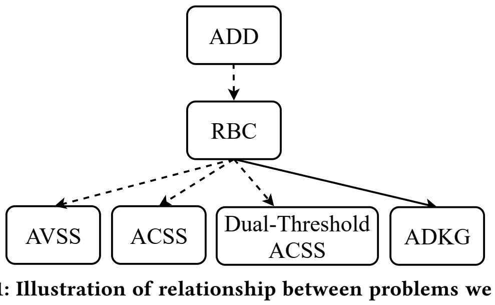

Figure 1: Illustration of relationship between problems we discuss in this paper. We use Asynchronous Data Dissemination (ADD) to solve Asynchronous Reliable Broadcast (RBC) for long messages. Our improved RBC directly improves the communication cost of Asynchronous Distributed Key Generation (ADKG). We also use our improved RBC along with additional techniques (illustrated using dotted arrows) to improve the communication cost of Asynchronous Verifiable Secret Sharing (AVSS), Asynchronous Complete Secret Sharing (ACSS), and Dual-threshold ACSS.

Table 1: Comparison of protocols proposed in this paper with prior best known protocols realizing different primitives under different setup and cryptographic assumption. Here n is the number of nodes in the system and κ is the security parameter.

Scheme	Communication Cost (total)	Cryptographic Assumption	Setup	Reference
RBC for long message	$O(n^2\|M\|)$	None	None	[14]
	$O(n\|M\|+\kappa n^2 \log n)$	Hash	None	[16]
	$O(n\|M\|+\kappa n^2)$	q-SDH+DBDH	Trusted	[41]
	$O(n\|M\|+\kappa n^2)$	Hash	None	**this work**
AVSS and ACSS	$O(\kappa n^2 \log n)$	DL+RO	PKI	[52]
	$O(\kappa n^2 \log n)$	DL+RO	None	[3]*
	$O(\kappa n^2)$	q-SDH + Hash	Trusted	[4]
	$O(\kappa n^2)$	DL+Hash	None†	**this work**
Dual Threshold ACSS*	$O(\kappa n^2 \log n)$	DL+RO	None	[3]
	$O(\kappa n^2)$	q-SDH + Hash	Trusted	[3]
	$O(\kappa n^2)$	DDH+RO†	PKI	**this work**
ADKG	$O(\kappa n^3 \log n)$	DL+RO	PKI	[2]
	$O(\kappa n^3)$	DL+RO	PKI	**this work**

† Our AVSS does not require PKI, but our ACSS does. Also, If we assume the DBDH assumption, then we do not need RO in our dual-threshold ACSS.

* As presented, the ACSS scheme of [3] only supports uniform random secrets. All the dual-threshold ACSS scheme also only support uniform random secrets. But they can be extended to arbitrary secrets using techniques from [48].

size $\Omega(\kappa n)$ where n is the total number of nodes in the protocol and κ is the cryptographic security parameter. The classic RBC protocol by Bracha [14] solves 1-bit RBC with communication cost $O(n^2)$. It thus needs $O(n^2|M|)$ cost for broadcasting a message M. Prior to our work, the most closely related work was the asynchronous information dispersal (AVID) protocol of Cachin and Tessaro [16]. Their protocol uses error-correcting codes and Merkle path proofs and has a communication cost of $O(n|M|+\kappa n^2 \log n)$; it can be modified to solve RBC with the same communication cost.

We show that we can combine Bracha's classic RBC protocol [14] with our ADD protocol to obtain an improved solution to RBC for long messages. The key observation is that running the Bracha's RBC protocol on the hash digest of M can establish exactly the initial condition for ADD, i.e., at least $t + 1$ honest nodes start with M and no honest node starts with any other message. Then, after running our ADD protocol, every honest node unanimously outputs the message M. Compared to the Cachin-Tessaro protocol [16], we remove the $\log n$ term in the communication cost as we remove the Merkle path proofs.

Our improved RBC immediately improves asynchronous atomic broadcast [27, 37, 40] and asynchronous distributed key generation [2]. It can also help improve the communication costs of Asynchronous Verifiable Secret Sharing (AVSS) [4, 15] and its variants such as Asynchronous Complete Secret Sharing (ACSS) and Dual-Threshold ACSS [3, 34]. But for these primitives, it is not as simple as plugging in our improved RBC as a black-box because state-of-the-art solutions for these primitives have other bottleneck steps in terms of communication. For example, the best known AVSS protocol without any trusted setup [3] has a communication cost of $O(\kappa n^2 \log n)$ due to two steps: a RBC on an $O(\kappa n)$-sized message, and an all-to-all gossip of $O(\kappa \log n)$-sized polynomial evaluation proofs. Thus, we need to open the black-box of the RBC protocol and incorporate additional techniques to improve these primitives.

In summary, we make the following contributions.

- We introduce the Asynchronous Data Dissemination (ADD) problem.
- We design an information-theoretically secure ADD protocol that tolerates up to 1/3 malicious nodes and has communication cost $O(n|M|+n^2)$ for a message M of size $|M|$. Assuming a collision-resistant hash function, we can extend this ADD protocol to tolerate up to 1/2 malicious nodes.
- We use ADD to design an improved asynchronous reliable broadcast protocol for a message M with a communication cost of $O(n|M|+\kappa n^2)$ where κ is the size of the hash.
- Finally, we use our improved RBC along with additional techniques to design asynchronous verifiable secret sharing (AVSS), asynchronous complete secret sharing (ACSS), dual-threshold ACSS, asynchronous distributed key generation, all with improved communication cost or weaker assumptions comparing to the state-of-the-art solutions.

Paper organization. The rest of the paper is organized as follows. We describe our system model, introduce notations, and provides some necessary background §2. In §3, we formally introduce the problem of Asynchronous Data Dissemination (ADD) and describe our solutions to ADD. Next, in §4 we describe how we use ADD to implement reliable broadcast for long messages. We then provide improved constructions of AVSS ACSS and dual-threshold ACSS in §5. In §6 we present lower bound results and show our protocols are near-optimal. We describe related work in §7 and conclude after a discussion in §8.

2 SYSTEM MODEL AND PRELIMINARIES

Cryptographic assumptions. Let $\mathbb{G}$ be a group of order q where q is a prime number. Let $\mathbb{Z}_q$ be the field with integer operations modulo q. Throughout the paper, we use $\mathsf{hash}(\cdot)$ to denote a collision resistant hash function. We will use κ to denote the size of cryptographic objects, e.g., the length of the hash function output, the size of a ciphertext of a CPA-secure encryption, or the size of an element in the group $\mathbb{G}$. These objects may slightly differ in size in practice but they are on the same order; or one can interpret κ as the largest among them.

Network and adversarial assumptions. We consider an asynchronous network of $n \geq 3t + 1$ nodes where a malicious adversary can corrupt up to t nodes in the network. The corrupted nodes can deviate arbitrarily from the protocol. The remaining nodes are honest and strictly adhere to the protocol. We assume every pair of honest nodes have access to pairwise reliable and authenticated channel. The network is asynchronous, so the adversary can arbitrarily delay or reorder messages between honest nodes, but must eventually deliver every message.

Error correcting code. Our ADD protocol uses error correcting codes. For concreteness, we will use the standard Reed-Solomon (RS) codes [46]. A (m, k) RS code in Galois Field $\mathbb{F} = \mathrm{GF}(2^a)$ with $m \leq 2^a - 1$, encodes k data symbols from $\mathrm{GF}(2^a)$ into a codeword of m symbols from $\mathrm{GF}(2^a)$. Let $\mathsf{RSEnc}(M, m, k)$ be the encoding algorithm. Briefly, the RSEnc takes as input a message M consisting of k symbols, treats it as a polynomial of degree $k - 1$ and outputs m evaluations of the corresponding polynomial.

Let RSDec(k, r, T) be the RS decoding procedure. RSDec takes as input a set of symbols T (some of which may be incorrect), and outputs a degree $k - 1$ polynomial, i.e., k symbols, by correcting up to r errors (incorrect symbols) in T. It is well-known that RSDec can correct up to r errors in T and output the original message provided that $|T| \geq k + 2r$ [38]. Concrete instantiations of RS codes include the Berlekamp-Welch algorithm [50] and the Gao algorithm [28].

3 ASYNCHRONOUS DATA DISSEMINATION

In this section, we formally define the problem of Data Dissemination. We then provide our solution to the data dissemination problem in *asynchronous* network, which we refer to as *Asynchronous* Data Dissemination (ADD). Finally, we analyze its correctness and efficiency.

3.1 Problem Statement

Definition 3.1 (Data Dissemination). Given a network of n nodes, of which up to t could be malicious, let M be a data blob that is the input of at least $t + 1$ honest nodes. The remaining honest nodes have input $\perp$. A protocol solves Data Dissemination if it ensures that every honest node eventually outputs M.

Here on, we refer to the honest nodes that start with the message M as the *sender* nodes and the honest nodes that start with $\perp$ as the *recipient* nodes.

A simple but important observation is that to solve data dissemination in a network with t malicious nodes, the number of honest sender nodes must be at least $t + 1$. Otherwise, to any honest recipient node, the set of t honest senders starting with M are indistinguishable from the set of t malicious nodes claiming to start with M' and behaving honestly otherwise. This justifies the initial condition of at least $t + 1$ honest sender nodes.

The simplest data dissemination protocol just has each honest sender send its input to all other nodes. A recipient node, upon receiving $t + 1$ matching copies of a message M, outputs M. Since we begin with $t + 1$ honest senders, every recipient node will eventually receive $t + 1$ identical messages. Furthermore, there will not be $t + 1$ nodes sending a different message M' since there are only t malicious nodes. The issue with this approach is that it is not communication efficient. Specifically, this approach has a total communication cost of $O(n^2|M|)$.

An alternate solution is to have each recipient node request M from the sender nodes, to which an honest sender replies by sending M. Again, the issue is that malicious nodes may redundantly request M from all honest senders. Since there are $t = \Theta(n)$ malicious nodes and each malicious node requests M from all honest senders, this approach also has a total communication cost of $O(n^2|M|)$.

3.2 Our Approach

As mentioned, in this paper, we focus on data dissemination in an asynchronous network, which we refer to as the *Asynchronous Data Dissemination* (ADD) problem. We present two variants of our ADD protocol. The first variant requires $t < n/3$ (or equivalently $n \geq 3t + 1$) and is information-theoretically secure. The second variant requires $t < n/2$ but needs a collision resistant hash function.

Algorithm 1 Pseudocode for node i in ADD for $n = 3t + 1$

```
1: // encoding phase.
2: input $M_i$: either $M_i = M$ or $M_i = \perp$
3: if $M_i \neq \perp$ then
4:     Let $M' := [m_1, m_2, \dots, m_n] :=$ RSEnc$(M_i, n, t+1)$
5: // dispersal phase
6: if $M_i \neq \perp$ then
7:     Let $m_i^* := m_i$
8:     send $\langle$DISPERSE$, m_j\rangle$ to node $j$ for every $j = 1, 2, \dots, n$
9: else
10:    upon receiving $t+1$ identical $\langle$DISPERSE$, m_i\rangle$ do
11:        Let $m_i^* := m_i$
12: // reconstruction phase
13: send $\langle$RECONSTRUCT$, m_i^*\rangle$ to all nodes
14: if $M_i \neq \perp$ then
15:     output $M$ and return;
16: Let $T := \{\}$
17: For every $\langle$RECONSTRUCT$, m_j^*\rangle$ received from node $j$, add $(j, m_j^*)$ to $T$
18: for $0 \leq r \leq t$ do        // online Error Correction
19:     Wait till $|T| \geq 2t + r + 1$
20:     Let $p_r(\cdot) :=$ RSDec$(t+1, r, T)$
21:     if $2t+1$ elements $(j, a) \in T$ satisfy $p_r(j) = a$ then
22:         output coefficients of $p_r(\cdot)$ as $M$ and return
```

Since all the applications we consider in this paper require $t < n/3$ (due to reasons orthogonal to ADD), we will focus on the $t < n/3$ variant of our ADD protocol. We present our ADD protocol for $t < n/2$ in Appendix B.

Our ADD protocol has three phases: *Encoding, Dispersal,* and *Reconstruction.* Figure 2 illustrates our protocol in a network of $n = 4$ nodes with $t = 1$. We provide the pseudocode of the protocol for node i in Algorithm 1.

Encoding phase. In the encoding phase, every sender (who holds $M \neq \perp$) encodes M using a $(n, t + 1)$ Reed-Solomon code, i.e., using RSEnc$(\cdot)$ described in §2 (Line 2 – 4 in Algorithm 1). The encoded message $M' =$ RSEnc$(M, n, t + 1)$ can be written as a vector $M' = [m_1, m_2, \dots, m_n]$ of n symbols. Here each symbol m_i is of size approximately $|m_i| = |M|/(t + 1)$ because, after encoding,

$$|M'| = \frac{n|M|}{t+1} \Rightarrow \frac{|M'|}{n} = \frac{|M|}{t+1} \quad (1)$$

Dispersal phase. After encoding M into M', the senders start the dispersal phase (Line 5 – 10 in Algorithm 1). During the dispersal phase, every sender sends the message $\langle$DISPERSE$, m_j\rangle$ to node j. A recipient node j, upon receiving $t + 1$ matching DISPERSE messages for a symbol m'_j sets m'_j as its reconstruction symbol m_j^*. Each honest sender node i sets m_i as its reconstruction symbol m_i^*.

Reconstruction phase. During the reconstruction phase, every node i sends $\langle$RECONSTRUCT$, i, m_i^*\rangle$ to all other nodes. Then, every recipient node j, upon receiving enough symbols, uses the standard *Online Error Correcting* (OEC) algorithm from [6] (line 16 – 22 in Algorithm 1). Briefly, the OEC algorithm [6] performs up to t trials of reconstruction. In the r-th trial, the recipient waits until it receives RECONSTRUCT messages from $2t + r + 1$ nodes and tries to decode. If the reconstructed polynomial agrees with $2t + 1$ points in the RECONSTRUCT messages received so far, the recipient outputs the

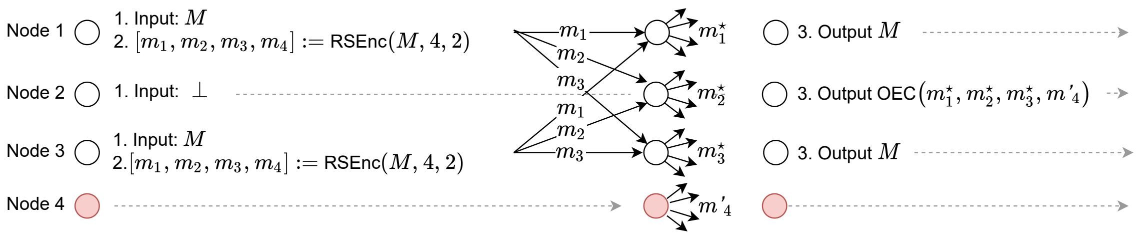

Figure 2: A example execution of our solution to ADD with 4 nodes $\{1, 2, 3, 4\}$ among which node 4 is malicious and node 2 does not start with M. As described, after the dispersal phase, node 2 will receive $2 = t + 1$ identical copies of m_2 from node 1 and 4 and hence will set m_2 as m_2^*. As a result, after the reconstruction phase, node 2 will receive 3 correct symbols of M' which is sufficient for reconstructing M.

decoded message; otherwise, it waits for one more RECONSTRUCT message and tries again.

3.3 Analysis

We will prove that Algorithm 1 solves the ADD problem. Towards this end, we will first prove that every honest node will hold the correct reconstruction symbol at the end of the dispersal phase. Next, we will argue that every honest node successfully reconstructs the message M. Recall that we refer to the nodes that start with the message M as the *sender* nodes and the honest nodes that start with $\perp$ as the *recipient* nodes.

Lemma 3.2. *After the dispersal phase each honest node j holds m_j^* where m_j^* is the j^{th} coordinate of $M' = \mathsf{RSEnc}(M, n, t + 1)$.*

Proof. Recall that RSEnc encoding is deterministic. If node j is a sender, then it trivially holds m_j^*. Thus, we focus on a recipient node j. No honest node will send DISPERSE for any $m_j \neq m_j^*$. Since node j holds m_j only if it receives $t+1$ identical DISPERSE messages for m_j, it will not hold $m_j \neq m_j^*$. All $t+1$ honest senders send m_j^* to node j. So node j will eventually receive $t + 1$ DISPERSE message for m_j^* and will hold m_j^* at the end of the dispersal phase. □

Next we will argue that each honest recipient will eventually recover the message at the end of the reconstruction phase.

Lemma 3.3. *At the end of the reconstruction phase of Algorithm 1, every honest node outputs M.*

Proof. Clearly, every honest sender outputs M. Thus, we again focus on honest recipient nodes.

We will first argue that an honest recipient node will never output a wrong message. Suppose node i outputs a polynomial $p_r(\cdot)$ as the message in the r^{th} iteration in the reconstruction phase. Let T_r be the set of symbols node i used in iteration r. Then, $p_r(\cdot)$ is consistent with $2t+1$ points from T_r, of which at least $t+1$ are from honest nodes. All these points belong on both polynomial $p(\cdot)$ and $p_r(\cdot)$. Since both $p_r(\cdot)$ and $p(\cdot)$ are degree-t polynomials and agree on $t + 1$ points, $p_r(\cdot)$ and $p(\cdot)$ are the same polynomial.

Next, we argue that every honest recipient node will eventually output a message. From Lemma 3.2, at the end of the dispersal phase, every honest node holds the correct symbol of $M' = \mathsf{RSEnc}(M, n, t + 1)$ and will send it to all other nodes in its RECONSTRUCT messages. Thus, every recipient node will eventually receive $2t + 1$ correct symbols from all honest nodes, and at most t incorrect symbols from malicious nodes. Hence, the RSDec algorithm will correct all the errors and will return the correct message. □

We will next argue about the total communication cost of our ADD protocol.

Lemma 3.4. *The total communication cost of our ADD protocol is $O(n|M|+n^2)$.*

Proof. Recall that $|M'|= n|M|/(t + 1) = O(|M|)$. During the dispersal phase, each sender node sends a message of size $|M'|/n+O(1)$ to every other node. During the reconstruction phase, each node sends a message of size $|M'|/n+O(1)$ to every other node. Therefore, the total communication cost of the protocol is $n|M'|+O(n^2)$, which is the same as $O(n|M|+n^2)$. □

Using Lemma 3.2, 3.3 and 3.4, we get the following theorem.

Theorem 3.5. *In an asynchronous network of $n = 3t + 1$ nodes, of which up to t nodes can be malicous, Algorithm 1 solves ADD with a communication complexity of $O(n|M|+n^2)$.*

Remark. One subtle point to note here is that the Reed-Solomon code requires the field size to be larger than n. Hence, each symbol in the encoded codeword is of size $O(\log n)$. As a result, the size of the encoded message is at least $O(n \log n)$. Thus, ADD is more useful when the message size is $\Omega(n \log n)$ which is the case for our RBCand AVSS applications. We will give a lower bound in § 6 showing that the communication complexity of Algorithm 1 is optimal for $|M|\geq \Theta(n \log n)$.

Now let us analyze the computation cost of each node in Algorithm 1. The encoding step at every sender during the dispersal phase involves $O(n^2)$ operations (additions and multiplications) in $\mathbb{Z}_q$ (the recipient nodes do nothing). During the reconstruction phase, each recipient node needs to invoke the RSDec algorithm anywhere between one to t times, and each invocation of RSDec requires $O(n\,\text{polylog}(n))$ computation if we use the decoding algorithm of Gao [28].

4 RELIABLE BROADCAST

Reliable broadcast (RBC) was introduced by Bracha [14]. In the same paper, Bracha provided a RBC protocol for a single bit with a communication complexity of $O(n^2)$. In this paper, we present two RBC protocols for long messages. Both protocols are built upon Bracha's RBC. The first protocol is conceptually simpler and uses Bracha's RBC and ADD in a modular way. However, it requires

Algorithm 2 Bracha's RBC [14]

```
// only broadcaster node
input $M$
send $\langle \texttt{PROPOSE}, M \rangle$ to all
// all nodes
input $P(\cdot)$ // predicate $P(\cdot)$ returns true unless otherwise specified.
upon receiving $\langle \texttt{PROPOSE}, M \rangle$ from the broadcaster do
    if $P(M)$ then
        send $\langle \texttt{ECHO}, M \rangle$ to all
upon receiving $2t + 1$ $\langle \texttt{ECHO}, M \rangle$ messages and not having sent a READY message do
    send $\langle \texttt{READY}, M \rangle$ to all
upon receiving $t + 1$ $\langle \texttt{READY}, M \rangle$ messages and not having sent a READY message do
    send $\langle \texttt{READY}, M \rangle$ to all
upon receiving $2t + 1$ $\langle \texttt{READY}, M \rangle$ messages do
    output $M$
```

two additional rounds of communication than Bracha's RBC. Our second protocol merges the steps of Bracha's RBC and ADD and matches the round complexity of Bracha's RBC.

Definition 4.1 (Reliable Broadcast). A protocol for a set of nodes $\{1, ..., n\}$, where a distinguished node called the broadcaster holds an initial input M of size $|M|$, is a reliable broadcast protocol, if the following properties hold

- **Agreement.** If an honest node outputs a message M' and another honest node outputs a message M'', then $M' = M''$.
- **Validity.** If the broadcaster is honest, all honest nodes eventually output the message M.
- **Totality.** If an honest node outputs a message, then every honest node eventually outputs a message.

Since our RBC protocols rely upon Bracha's RBC protocol [14], we first describe Bracha's RBC protocol in Algorithm 2. The main idea of Bracha's RBC is to use quorum intersection (of ECHO messages) for agreement and use vote amplification (of READY messages) for totality. However, the protocol needs to attach the input M in every message, leading to a high communication cost of $O(n^2|M|)$.

We modify the RBC protocol interface to add an external check before sending the ECHO message, denote it as a predicate $P(\cdot)$. We do so to make our AVSS and ACSS protocols use RBC in a modular way. For standard RBC, this step can be skipped, i.e., $P(\cdot)$ always returns true.

We give our first RBC protocol for long messages in Algorithm 3 where we highlight the changes from Bracha's RBC in blue. We then prove it achieves all the required properties of RBC. We give our second RBC protocol in Algorithm 4 and defer its proofs to Appendix A.

The core idea in our first RBC protocol is to run the Bracha's RBC only on the hash of the message M, and then disseminate the message M using ADD. The previous best RBC protocol for long messages due to Cachin and Tessaro [16] also runs Bracha's RBC on the hash digest. However, their protocol requires attaching Merkle path proofs in the messages, which inevitably incurs a cost of $O(n|M|+\kappa n^2 \log n)$. Our ADD protocol removes these Merkle path proofs and leads to a total communication cost of $O(n|M|+\kappa n^2)$ for RBC.

More specifically, the broadcaster in our ADD-based RBC protocol first sends M to all other nodes. Every honest node, upon receiving the message from the broadcaster, first participates in a Bracha's RBC on h, the hash digest of M. Once Bracha's RBC terminates, i.e., a node receives $2t + 1$ READY messages for some h, the node inputs M to the ADD protocol if $h = \text{hash}(M)$. Otherwise (if $h \neq \text{hash}(M)$ or if the node has not received a PROPOSE message from the broadcaster), the node inputs $\perp$ to ADD. Recall that in

Algorithm 3 ADD-based RBC for long messages

```
// only broadcaster node
input $M$
send $\langle \texttt{PROPOSE}, M \rangle$ to all
// all nodes
input $P(\cdot)$ // predicate $P(\cdot)$ returns true unless otherwise specified.
upon receiving $\langle \texttt{PROPOSE}, M \rangle$ from the broadcaster do
    if $P(M)$ then
        let $h := \text{hash}(M)$
        send $\langle \texttt{ECHO}, h \rangle$ to all
upon receiving $2t + 1$ $\langle \texttt{ECHO}, h \rangle$ messages and not having sent a READY message do
    send $\langle \texttt{READY}, h \rangle$ to all
upon receiving $t + 1$ $\langle \texttt{READY}, h \rangle$ messages and not having sent a READY message do
    send $\langle \texttt{READY}, h \rangle$ to all
upon receiving $2t + 1$ $\langle \texttt{READY}, h \rangle$ messages do
    if received $\langle \texttt{PROPOSE}, M \rangle$ and $h = \text{hash}(M)$ then
        ADD($M$)
    else
        ADD($\perp$)
```

Algorithm 4 Four-round RBC protocol for long messages

```
// only broadcaster node
input $M$
send $\langle \texttt{PROPOSE}, M \rangle$ to all
// all nodes
input $P(\cdot)$ // predicate $P(\cdot)$ returns true unless otherwise specified.
upon receiving $\langle \texttt{PROPOSE}, M \rangle$ from the broadcaster do
    if $P(M)$ then
        Let $h := \text{hash}(M)$
        Let $M' := [m_1, m_2, \dots, m_n] := \text{RSEnc}(M_i, n, t + 1)$
        send $\langle \texttt{ECHO}, m_j, h \rangle$ to node $j$ for $j = 1, 2, \dots, n$
upon receiving $2t + 1$ $\langle \texttt{ECHO}, m_i, h \rangle$ matching messages and not having sent a READY message do
    send $\langle \texttt{READY}, m_i, h \rangle$ to all
upon receiving $t + 1$ $\langle \texttt{READY}, *, h \rangle$ messages and not having sent a READY message do
    Wait for $t + 1$ matching $\langle \texttt{ECHO}, m'_i, h \rangle$
    send $\langle \texttt{READY}, m'_i, h \rangle$ to all
For the first $\langle \texttt{READY}, m^*_j, h \rangle$ received from node $j$, add $(j, m^*_j)$ to $T_h$ // $T_h$ initialized as $\{\}$
for $0 \le r \le t$ do        // Error Correction
    upon $|T_h| \ge 2t + r + 1$ do
        Let $M'$ be coefficients of $\text{RSDec}(t + 1, r, T)$
        if $\text{hash}(M') = h$ then
            output $M'$ and return
```

Bracha's RBC, a node outputs M upon receiving $2t + 1$ READY messages for M, which implies at least $t + 1$ honest nodes have received M from the broadcaster before sending ECHO for M. Similarly, in our RBC protocol, when a node receives $2t + 1$ READY messages for some hash h, at least $t + 1$ honest nodes have received the message M such that $\mathsf{hash}(M) = h$ Moreover, the agreement property of the Bracha's RBC guarantees that no two honest nodes will agree on different hashes, and thus any honest node that receives $M' \neq M$ from the broadcaster will input $\perp$ to the ADD. Therefore, the initial condition of ADD is met. Hence, the guarantees of our ADD protocol ensure the desired properties of the RBC protocol.

The four-round RBC protocol (Algorithm 4) saves two rounds by merging our ADD into Bracha's RBC, thanks to the similarity in their message patterns. More specifically, the ECHO and READY messages in Bracha's RBC now also attach the symbols of the message which were sent separately in ADD before.

4.1 Analysis of Algorithm 3

First, we show that running Bracha's RBC on the hash h sets up the initial condition for ADD.

Lemma 4.2. *If any honest node executes* $\mathsf{ADD}(M)$*, then there are at least* $2t + 1$ *nodes, among which at least* $t + 1$ *are honest nodes, that receive* M *from the broadcaster and send* $\langle \mathsf{ECHO}, h\rangle$ *to all other nodes where* $h = \mathsf{hash}(M)$.

Proof. An honest node i executes $\mathsf{ADD}(M)$ only upon receiving $\langle \mathsf{READY}, h = \mathsf{hash}(M)\rangle$ messages from a quorum Q_2 of $2t + 1$ nodes. Since there are at most t malicious nodes in the system, at least $t + 1$ of the nodes in Q_2 are honest. This implies that at least one honest node receives $\langle \mathsf{ECHO}, h\rangle$ from a quorum Q_1 of $2t + 1$ nodes. Again, at least $t + 1$ nodes of Q_1 are honest. These honest nodes receive M from the broadcaster and send $\langle \mathsf{ECHO}, h\rangle$ to all nodes. □

Lemma 4.3. *If any honest node executes* $\mathsf{ADD}(M)$*, then at least* $t + 1$ *honest nodes execute* $\mathsf{ADD}(M)$ *and the rest of the honest nodes execute* $\mathsf{ADD}(\perp)$.

Proof. Let node i execute $\mathsf{ADD}(M)$. By Lemma 4.2, there exits $2t+1$ nodes who sent $\langle \mathsf{ECHO}, h\rangle$ messages where $h = \mathsf{hash}(M)$. Let us first show that no honest executes $\mathsf{ADD}(M')$ for some $M' \neq M$ and $M' \neq \perp$. For the sake of contradiction, assume that an honest node j executes $\mathsf{ADD}(M')$ for such an M'. By Lemma 4.2, there exist $2t + 1$ nodes that sent $\langle \mathsf{ECHO}, h'\rangle$ where $h' = \mathsf{hash}(M')$. This is impossible because by quorum intersection, $2(2t + 1) - n = t + 1$ nodes echoed both messages but there are at most t malicious nodes.

Since at least $t + 1$ honest nodes send $\langle \mathsf{READY}, h\rangle$, these messages eventually reach all honest nodes, and according to the protocol all honest nodes will send $\langle \mathsf{READY}, h\rangle$ to all nodes. Hence, eventually all honest nodes receive at least $2t + 1$ $\langle \mathsf{READY}, h\rangle$ messages and execute $\mathsf{ADD}(\cdot)$. Since they cannot execute $\mathsf{ADD}(\cdot)$ with any other message, they execute either $\mathsf{ADD}(M)$ or $\mathsf{ADD}(\perp)$. By Lemma 4.2, at least $t + 1$ honest nodes receive M from the broadcaster. According to the protocol, these nodes will execute $\mathsf{ADD}(M)$. □

Now, we can prove that the properties of RBC are satisfied.

Lemma 4.4 (Agreement and Totality). *If an honest node outputs* M*, then no honest node will output* $M' \neq M$*, and every honest node will eventually output* M.

Proof. If an honest node outputs M, we show some honest node must execute $\mathsf{ADD}(M)$. If any honest node execute $\mathsf{ADD}(M')$ where $M' \neq M, M' \neq \perp$, then by Lemma 4.3 the initial condition of ADD for M' is met, and all honest nodes output M', contradiction. If no honest node execute $\mathsf{ADD}(M)$, i.e., honest nodes either execute $\mathsf{ADD}(\perp)$ or nothing, then in ADD, there are at most t DISPERSE messages (from malicious nodes), and no honest node can output, contradiction. Hence, some honest node must execute $\mathsf{ADD}(M)$. By Lemma 4.3, all honest nodes will execute $\mathsf{ADD}(M)$ or $\mathsf{ADD}(\perp)$, among which at least $t + 1$ honest nodes execute $\mathsf{ADD}(M)$. Hence, the initial condition for ADD holds, and by Lemma 3.3, every honest node will output M. □

Lemma 4.5 (Validity). *If the broadcaster is honest and has an input* M*, then every honest will eventually output* M.

Proof. When the broadcaster is honest, it will send $\langle \mathsf{PROPOSE}, M\rangle$ to all nodes. As a result, all $2t + 1$ honest nodes will send $\langle \mathsf{ECHO}, h\rangle$ for $h = \mathsf{hash}(M)$ to all nodes. Then, all $2t+1$ honest nodes will eventually send $\langle \mathsf{READY}, h\rangle$, and no honest node will send $\langle \mathsf{READY}, h'\rangle$ for $h' \neq h$ since there are at most t malicious nodes. Thus, all honest nodes eventually start ADD after receiving $2t + 1$ READY messages, with inputs either M or $\perp$. When an honest node starts ADD, there are at least $t + 1$ honest nodes that hold M, since the first honest node that sends READY for h receives $2t + 1$ ECHO messages for h. Those $t+1$ honest nodes eventually invoke $\mathsf{ADD}(M)$, thus the initial condition of ADD is met, and ADD guarantees that every honest node will eventually output M. □

Next, we analyze the communication complexity of the protocol.

Lemma 4.6. *Assuming existence of collision resistant hash functions, Algorithm 3 solves RBC with communication complexity of* $O(n|M|+\kappa n^2)$ *where* κ *is the output length of the hash function.*

Proof. The proposal step for M has a communication cost of $O(n|M|)$. ECHO and READY messages consume a communication cost of $O(n^2\kappa)$. From Lemma 3.4, the communication cost of ADD is $O(n|M|+n^2)$. Combining all these costs, we get that the total communication of Algorithm 2 is $O(n|M|+\kappa n^2)$. □

Combining the above lemmas, we get the following theorem.

Theorem 4.7. *In an asynchronous network of* n *nodes where* $t < n/3$*, assuming existence of collision resistant hash functions, Algorithm 3 solves RBC with communication complexity* $O(n|M|+\kappa n^2)$ *where* κ *is the output length of the hash function.*

4.2 Applications of Reliable Broadcast

Reliable broadcast for long message has been used as a crucial primitive in many protocols ranging from atomic broadcast [25, 27, 30, 37, 40], state machine replication [20, 51], asynchronous verifiable secret sharing [3, 4, 16, 52], asynchronous multi-party computation [36, 52], asynchronous distributed key generation [2, 34], etc.. Almost all these protocols involve reliable broadcast of messages of size at least linear in the number of participants in the network.

Direct improvements. For some of the above applications, our improvements to RBC directly improve them. For example, state-of-art asynchronous BFT protocols such as HoneyBadger [40], Dumbo [30] and Aleph [27] use $O(n)$ reliable broadcast instances of $O(\kappa n)$-sized messages. Hence, our RBC protocol immediately improves their total communication cost from $O(\kappa n^3 \log n)$ to $O(\kappa n^3)$.

Recently, Abraham et al. [2] presents a ADKG protocol in which each node reliably broadcasts a $O(\kappa n)$-sized message. Thus, our new RBC protocol lowers their communication cost from $O(\kappa n^3 \log n)$ to $O(\kappa n^3)$. A concurrent work [29] observes that the ADKG protocol of [2] implicitly solves the asynchronous Byzantine agreement (ABA) problem without any trusted setup. This implies that using our RBC, we can also get a ABA protocol without any trusted setup with a communication cost of $O(\kappa n^3)$.

Other applications. However, for some other primitives, simply replacing the RBC protocol with our improved one does not immediately reduce the asymptotic communication cost of the overall protocol because there exist other bottleneck steps in the protocol. For example, hbACSS [52] is an ACSS scheme with $O(\kappa n^2 \log n)$ communication cost. But the cost arises from an RBC of message of size $O(\kappa n \log n)$, so simply plugging in our improved RBC does not improve the communication cost. As another example, Haven [3] is a dual-threshold ACSS protocol for uniform secrets and it has communication cost $O(\kappa n^2 \log n)$ from two sources: a reliable broadcast of $O(\kappa n)$ size message and a step where nodes gossip zero-knowledge proofs of size $O(\kappa \log n)$. The next section introduces additional techniques to obtain protocols for these primitives with communication complexity of $O(\kappa n^2)$.

5 ASYNCHRONOUS (DUAL THRESHOLD) VERIFIABLE SECRET SHARING

In this section, we present protocols for Asynchronous Verifiable Secret Sharing (AVSS) that do not require any trusted setup and have a communication complexity of $O(\kappa n^2)$. Prior to our work, the best-known protocol for AVSS without a trusted setup had a communication cost of $O(\kappa n^2 \log n)$ [16] and $O(\kappa n^2)$ has only been achieved with a trusted setup [4]. Before we dive into our solutions, we formally define several variants of the AVSS problem following the definitions from [42].

5.1 Definitions and Preliminaries

For all these primitives, we assume a finite field $\mathbb{F}$ of order q. Let κ be the security parameter and $\mathrm{negl}(\kappa)$ be a negligible function in κ.

Definition 5.1 (Asynchronous Verifiable Secret Sharing). Let $(\mathsf{Sh}, \mathsf{Rec})$ be a pair of protocols in which a dealer L shares a secret s using Sh and other nodes use Rec to recover the shared secret. We say that $(\mathsf{Sh}, \mathsf{Rec})$ is a t-resilient AVSS scheme if the following properties hold with probability $1 - \mathrm{negl}(\kappa)$:

- **Termination:**
  (1) If the dealer L is honest, then each honest node will eventually terminate the Sh protocol.
  (2) If an honest node terminates the Sh protocol, then each honest node will eventually terminate Sh.
  (3) If all honest node start Rec, then each honest node will eventually terminate Rec.
- **Correctness:**
  (1) If L is honest, then each honest node upon terminating Rec, outputs the shared secret s.
  (2) If some honest node terminates Sh, then there exists a fixed secret $s' \in \mathbb{F}$, such that each honest node upon terminating Rec, will output s'.
- **Secrecy:** If L is honest and no honest node has started Rec, then an adversary that corrupts up to t nodes has no information about s.

Note that, the above definition of the Termination property of AVSS allow a situation where despite finishing the sharing phase and entering the reconstruction phase, an honest node does not receive its share from the dealer. Such a situation usually arises when the dealer is malicious and only sends valid shares to a subset of honest nodes, yet the corrupted nodes also claim to have the shares so that all honest nodes terminate the sharing phase.

To avoid this scenario, a stronger primitive called *Asynchronous Complete Secret Sharing* (ACSS) is defined. An ACSS is an AVSS scheme that additionally ensures that at the end of sharing phase, every honest node receives its share of a consistent secret. This holds even if the dealer is malicious. We call this property "completeness" following Choudhury [22], though it was called Ultimate Secret Sharing in the earlier work of Ben-Or et al. [7].

Definition 5.2 (Asynchronous Complete Secret Sharing). An Asynchronous Complete Secret Sharing protocol is an AVSS protocol that additionally satisfies the following completeness property.

- **Completeness:** If some honest node terminates Sh, then there exists a degree-t polynomial $p(\cdot)$ over $\mathbb{F}$ such that $p(0) = s'$ and each honest node i will eventually hold a share $s_i = p(i)$. Moreover, when L is honest, $s' = s$.

We are also interested in dual-threshold ACSS, also referred to as high-threshold ACSS, as defined as follows.

Definition 5.3 (Dual-threshold ACSS). A (n, ℓ, t) dual-threshold ACSS a t-resilient ACSS scheme for n nodes with the additional property that the reconstruction threshold ℓ can be any value in the range $[t, n - t)$, i.e., for any given ℓ, the secret should remain hidden from an adversary that corrupts up to ℓ nodes in the system. Moreover, any set of $\ell + 1$ or nodes should be able to recover the shared secret.

The advantage of dual-threshold ACSS with $\ell > t$ is that, the protocol can ensure secrecy of the secret even when the adversary corrupts more than t and up to ℓ nodes in the system. However, to ensure termination, $\ell < n - t$ is needed.

5.2 Asynchronous Verifiable Secret Sharing

Our construction of AVSS has a total communication cost of $O(\kappa n^2)$ and does not require any trusted setup. Briefly, our AVSS protocol is inspired by the VSS scheme of Pedersen [44]: the dealer uses our RBC protocol to broadcast Pedersen's polynomial commitment and at least a large fraction of honest nodes receive valid shares. The AVSS algorithm is given in Algorithm 5.

We describe the interface of Pedersen's polynomial commitment below and give more details in Appendix D. Let pp be the public parameters.

- $\mathsf{PedPolyCommit}(pp, p(\cdot), t, n) \rightarrow \boldsymbol{v}, \boldsymbol{s}, \boldsymbol{r}$: The $\mathsf{PedPolyCommit}(\cdot)$ algorithm takes as input a degree-t polynomial $p(\cdot)$ and outputs

Algorithm 5 AVSS

PUBLIC PARAMETER: pp of Pedersen's polynomial commitment

```
SHARING PHASE:
    // As dealer L with input s:
101: Sample a $t$-degree random polynomial $p(\cdot)$ such that $p(0) = s$
102: $\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{r} \leftarrow$ PedPolyCommit($pp, p(\cdot), t, n$)
103: for $i = 1, 2, ..., n$ do
104:     send $\langle$SHARE, $\boldsymbol{s}[i]$, $\boldsymbol{r}[i]\rangle$ to node $i$
105: RBC($\boldsymbol{v}$) with predicate $P(\cdot)$ described below.
    // predicate $P(\cdot)$ for node i during RBC
106: procedure $P(\boldsymbol{v})$
107:     upon receiving $\langle$SHARE, $s_i, r_i\rangle$ from dealer $L$ do
108:         return PedEvalVerify($pp, \boldsymbol{v}, i, s_i, r_i$)

RECONSTRUCTION PHASE:
    // every node i after finishing the sharing phase
201: if PedEvalVerify($pp, \boldsymbol{v}, s_i, r_i$) then
202:     send $\langle$RECONSTRUCT, $s_i, r_i\rangle$ to all
203: upon receiving $\langle$RECONSTRUCT, $s_j, r_j\rangle$ from node $j$ do
204:     if PedEvalVerify($pp, \boldsymbol{v}, s_j, r_j$) then
205:         $T = T \cup \{s_j\}$
206:         if $|T| \geq t + 1$ then
207:             output $s$ using Lagrange interpolation and return
```

three vectors each consisting of n elements. The vector $\boldsymbol{v}$ is the commitment to the polynomial and vectors $\boldsymbol{s}$ and $\boldsymbol{r}$ consist of shares for the n nodes.

- PedEvalVerify(pp, v, i, s_i, r_i) $\rightarrow 0/1$: The PedEvalVerify($\cdot$) algorithm takes as input a commitment $\boldsymbol{v}$ to a polynomial $p(\cdot)$ and a tuple (i, s_i, r_i), and it outputs 1 if $p(i) = s_i$ and 0 otherwise.

At any given node i, the predicate $P(\cdot)$ in Algorithm 5 returns true if the dealer's RBC proposal contains a valid Pedersen's polynomial commitment and the share received by node i is a valid share corresponding to the committed polynomial.

During the sharing phase, the dealer L with the secret s first samples a random degree-t polynomial $p(\cdot)$ such that $p(0) = s$. The dealer then invokes PedPolyCommit($\cdot$) to obtain a commitment vector $\boldsymbol{v}$ as well as share vectors $\boldsymbol{s}, \boldsymbol{r}$. Each vector contains n elements each of size $O(n\kappa)$. Then, L sends node j its shares (i.e., $\boldsymbol{s}[j]$ and $\boldsymbol{r}[j]$) using private messages and reliably broadcasts the commitment $\boldsymbol{v}$. The predicate $P(\cdot)$ (line 106 – 108) in the RBC requires each node to check whether the message received from the dealer is a valid share or not.

During the reconstruction phase, each node that received a valid share from L during the sharing phase sends $\langle$RECONSTRUCT, $i, s_i, r_i\rangle$ to all other nodes. Each node upon receiving $\langle$RECONSTRUCT, $j, s_j, r_j\rangle$ from node j, checks whether (j, s_j, r_j) is valid using PedEvalVerify($\cdot$). Upon receiving $t + 1$ valid shares, a node reconstructs the secret using Lagrange interpolation.

We prove our AVSS protocol guarantees termination, correctness, and secrecy against all computationally bounded adversaries.

Lemma 5.4 (Termination). *Assuming a collision resistant hash function, the AVSS scheme in Algorithm 5 guarantees Termination.*

Proof. From Lemma 4.6 and 3.3, whenever an honest node terminates the sharing phase, each honest node will eventually terminate the sharing phase. Furthermore, from Lemma 4.4 and the fact that nodes only send ECHO messages if their PedEvalVerify($\cdot$) is successful (line 108), we get that if the sharing phase terminates at one honest node, then PedEvalVerify($\cdot$) was successful for at least $t+1$ honest node. Let S be this set of honest nodes. Then, Lemma D.1 implies that the shares of nodes in S are sufficient to reconstruct the secret s. During the reconstruction phase, each honest node will eventually receive RECONSTRUCT messages from all nodes in S, so each honest node will eventually output a secret. Hence, the AVSS scheme ensures termination. □

Lemma 5.5 (Correctness). *Assuming a collision resistant hash function, the AVSS scheme in Algorithm 5 guarantees Correctness.*

Proof. When the dealer is honest, the secret reconstructed using the shares of only honest nodes is clearly equal to s. Additionally, note that during the reconstruction phase, an honest node only accepts shares for which PedEvalVerify($\cdot$) is successful. Hence, the uniqueness property (Theorem D.3) of the Pedersen's VSS protocol implies that whenever an honest node outputs a secret s', $s' = s$.

Similar to the proof of termination, when the sharing phase terminates at an honest node (possibly under a malicious dealer), PedEvalVerify($\cdot$) associated with $\boldsymbol{v}$ was successful for at least $t + 1$ honest node. By Lemma D.1, these honest nodes can recover an appropriate secret. Thus, using the same argument as above, every honest node reconstructs the same secret. □

Observe that the view of an adversary in our AVSS scheme is identical to the view of the adversary in Pedersen's VSS [44]. Hence, the Secrecy of our AVSS protocol follows from Theorem D.4.

5.3 Asynchronous Complete Secret Sharing

We can also extend our AVSS scheme to ensure the completeness guarantees described in definition 5.2 using the *encrypt-then-disperse* technique from [36, 52]. We describe this technique briefly below and refer readers to [52].

During the sharing phase, the dealer additionally computes a ciphertext vector $\boldsymbol{c}$ that consists of encryptions of shares (s_i, r_i) of each node i.

$$\boldsymbol{c} = \{c_1, \ldots, c_n\} = \{\mathsf{Enc}_{pk_1}(s_1, r_1), \ldots, \mathsf{Enc}_{pk_n}(s_n, r_n)\} \qquad (2)$$

Here, pk_i is the public key of node i, $\mathsf{Enc}_{pk_i}(x)$ denotes a CPA secure public key encryption of x using public key pk_i.

The dealer then sends $\boldsymbol{v}||\boldsymbol{c}$ using our RBC protocol. Note that since the dealer is now sending $\boldsymbol{c}$ through RBC, it no longer needs to send the plaintext shares using private channels. Upon receiving $\boldsymbol{c}$ as a part of a PROPOSE message, each node decrypts its plaintext share using its secret key and validates it using PedEvalVerify($\cdot$). As in [52], if a node i receives an invalid, the node reveals its secret key by sending $\langle$IMPLICATE, $i, sk_i\rangle$ to all other nodes. Upon receiving an IMPLICATE message, an honest node repeats the decryption and verification procedure to confirm that the dealer indeed dispersed an invalid share.

If any honest node detects that some node received an invalid share and other nodes have already output, it starts the *recovery* protocol. During the recovery protocol, nodes with valid shares are presented with evidence that the dealer is faulty. If convinced, these nodes divulge their keys required to decrypt their shares. In particular, each node i who received a valid share during the sharing phase sends a $\langle \mathsf{RECOVER}, sk_i \rangle$ to all other nodes. Recall from Lemma 4.2 that if any honest node outputs during the ACSS protocol, then at least $t+1$ honest node received valid shares. Therefore, at least $t+1$ honest nodes will send valid RECOVER messages to all nodes. Upon receiving a $\langle \mathsf{RECOVER}, sk_i' \rangle$ message from node i, each node validates that sk_i' corresponds to the public key pk_i and the decrypted share is valid. Note that this validation check will be successful for all honest nodes. Thus, each honest node would receive at least $t+1$ valid RECOVER messages and can use Lagrange interpolation to reconstruct the polynomial $p'(\cdot)$ and recover their individual share $p'(i)$.

The communication costs of both dealer implication and share recovery protocol are $O(\kappa n^2)$ because the secret key has size $O(\kappa)$. Yurek et al. [52] also discusses an alternative approach that obviates the need for revealing secret keys. This is important if the secret keys are for long-term usage. In the case of long-term keys, instead of revealing the secret keys, each node sends a $O(\kappa)$-sized non-interactive zero-knowledge (NIZK) proofs to all other nodes. Hence, this approach also has communication cost $O(\kappa n^2)$.

Now we give a proof sketch to this ACSS protocol. Termination and Correctness are not affected. For Secrecy, the key facts to note are that ACSS does not seek to ensure Secrecy when the dealer is malicious, and that when the dealer is honest, the share recovery phase is not invoked. Thus, when the dealer is honest, before any honest node starts reconstruction, the encryption hides the shares, and the view of the adversary in the ACSS scheme is computationally indistinguishable to the view of the adversary in AVSS. Hence, secrecy holds.

Lastly, Completeness is clear when the dealer is honest. When the dealer is malicious, Completeness follows from the fact that a single genuine blame is sufficient to initiate the recovery protocol. The recovery protocol ensures that every honest node recovers the committed polynomial and outputs its share by evaluating the polynomial at the appropriate index.

5.4 Dual-threshold ACSS

We will first describe a dual-threshold ACSS that allows a dealer to share uniformly random secrets. For any given $n \geq 3t+1$, our dual-threshold ACSS scheme can tolerate any reconstruction threshold ℓ in the range $t \leq \ell < n-t$. In particular, given a group $\mathbb{G}$ of a prime order q with a randomly chosen generator g_1, this scheme allows the dealer to share a secret of the form g_1^s where s is chosen uniformly at random from $\mathbb{Z}_q$. Then, using techniques from [48], we can extend the scheme to allow the dealer to share an arbitrary secret in $\mathbb{Z}_q$.

Before providing details of our dual-threshold ACSS, we want to discuss why we cannot extend our ACSS scheme from §5.3 to a higher reconstruction threshold. More concretely, what would go wrong if $p(\cdot)$ is a polynomial of degree ℓ for $\ell > t$? The issue is that, in our ACSS protocol, it is possible that the sharing phase terminates, but due to a malicious dealer, only $t+1$ honest nodes received their shares. As a result, during the recovery protocol honest nodes are only guaranteed to receive $t+1$ evaluation points

Algorithm 6 Dual-threshold ACSS for uniform secrets

PUBLIC PARAMETERS: $pp := (n, t, \ell, g_0, g_1, \{pk_i\})$ for $i = 1, 2, \ldots, n$
PRIVATE PARAMETERS: Node i has secret key sk_i i.e., $pk_i = g_1^{sk_i}$

```
SHARING PHASE:
    // As dealer L with input s:
101: $\boldsymbol{c}, \boldsymbol{v}, \boldsymbol{\pi} \leftarrow$ PVSS.Share($pp, s, \ell, n$)
102: RBC($\boldsymbol{v} \| \boldsymbol{c} \| \boldsymbol{\pi}$) with predicate $P(\cdot)$ as given below.
    // predicate $P(\cdot)$ for node i
103: procedure $P(\boldsymbol{v} \| \boldsymbol{c} \| \boldsymbol{\pi})$
104:     return PVSS.Verify($pp, \ell, \boldsymbol{v}, \boldsymbol{c}, \boldsymbol{\pi}$)

RECONSTRUCTION PHASE:
    // every node i with key $pk_i, sk_i$
201: $\tilde{s}_i := \boldsymbol{c}[i]^{sk_i}$; $\tilde{\pi} :=$ dleq.Prove($sk_i, g_1, pk_i, \boldsymbol{c}[i], \tilde{s}_i$);
202: send $\langle$RECONSTRUCT, $\tilde{s}_i, \tilde{\pi}_i\rangle$ to all
203: upon receiving $\langle$RECONSTRUCT, $\tilde{s}_j, \tilde{\pi}_j\rangle$ from node $j$ do
204:     if dleq.Verify($\tilde{\pi}, g_1, pk_j, \boldsymbol{c}[j], \tilde{s}_j$) then
205:         $T = T \cup \{\tilde{s}_j\}$
206:         if $|T| \geq \ell$ then
207:             $g_1^s :=$ PVSS.Recon($T$)
208:             output $g_1^s$ and return
```

through RECOVER messages. If the polynomial $p(\cdot)$ has degree $\ell > t$, then $t+1$ points are insufficient to reconstruct the polynomial $p(\cdot)$ or the shares.

To address the above issue, we design a dual-threshold ACSS using a Publicly Verifiable Secret Sharing (PVSS) scheme. Intuitively, a PVSS scheme enables the dealer to generate a NIZK proof that a given ciphertext is an encryption of a valid share. Each node can then check these proofs in the predicate of the RBC protocol. Now, if the RBC successfully terminates, enough honest nodes have checked the validity of each share (using the NIZK proofs), which ensures that all honest nodes receive their shares. Moreover, if the RBC terminates, due to its Totality guarantee, every honest node will receive the encrypted shares of every node and can use its secret key to decrypt its share.

Another consequence of using PVSS is that our dual-threshold ACSS scheme is also publicly verifiable, i.e., any external verifier, who need not be a participant, can check that the dealer L acted honestly without learning any information about the shares or the secret.

For concreteness, we will use the PVSS scheme from Scrape [19], which is secure assuming the existence of a Random Oracle and the hardness of the Decisional Diffie-Hellman problem. Next, we describe the interface of Scrape's PVSS below and present our dual-threshold ACSS protocol in Algorithm 6. Let pp be the public parameters.

- PVSS.Share(pp, s, ℓ, n) $\rightarrow \boldsymbol{v}, \boldsymbol{c}, \boldsymbol{\pi}$: For a uniformly random $s \in \mathbb{F}$, the vector $\boldsymbol{v}$ is a commitment to a degree-ℓ random polynomial with $p(0) = s$. The vector $\boldsymbol{c}$ consists of encrypted shares for each node. The vector $\boldsymbol{\pi}$ consists of non-interactive zero-knowledge (NIZK) proofs that each component of $\boldsymbol{c}$ is a correct encryption of a share of s. Each of these proofs is $O(\kappa)$ bits long.
- PVSS.Verify($pp, \ell, \boldsymbol{v}, \boldsymbol{c}, \boldsymbol{\pi}$) $\rightarrow$ 0/1 : The PVSS.Verify function takes in the tuple $(\boldsymbol{v}, \boldsymbol{c}, \boldsymbol{\pi})$ and outputs 1 if and only if $\boldsymbol{v}$ is a

commitment to a degree-ℓ polynomial $p(\cdot)$ and each component of $\boldsymbol{c}$ is a valid encryption of a share of $p(0)$.

Our dual-threshold ACSS makes use of a non-interactive protocol for checking the equality of discrete logarithm. In particular, given a group $\mathbb{G}$ of prime order q, two uniformly random generators $g_0, g_1 \in \mathbb{G}$ and a tuple (g_0, x, g_1, y), a prover $\mathcal{P}$ wants to prove to a PPT verifier $\mathcal{V}$, in zero-knowledge, that there exists an witness α such that $x = g_0^\alpha$ and $y = g_1^\alpha$. We describe the interfaces of such a protocol and provide the detailed protocol in Appendix E.

- dleq.Prove$(\alpha, g_0, x, g_1, y) \rightarrow \pi$: Given tuple (g_0, x, g_1, y) and α where $\alpha = \log_{g_0} x = \log_{g_1} y$, dleq.Prove outputs an non-interactive zero-knowledge proof π that such an α exists.
- dleq.Verify$(\pi, g_0, x, g_1, y) \rightarrow 0/1$: Given a proof π and a tuple (g_0, x, g_1, y), dleq.Verify outputs 1 if $\log_{g_0} x = \log_{g_1} y$, and 0 otherwise.

The dealer L with a uniformly random secret $s \in \mathbb{F}$, first computes the PVSS shares for s using PVSS.Share. Let $(\boldsymbol{v}, \boldsymbol{c}, \boldsymbol{\pi})$ be the three vectors output of PVSS.Share. Next, L reliably broadcast $\boldsymbol{v}\|\boldsymbol{c}\|\boldsymbol{\pi}$ using our RBC protocol. During the RBC protocol, the predicate $P(\cdot)$ checks whether PVSS.Verify$(\boldsymbol{v}, \boldsymbol{c}, \boldsymbol{\pi})$ returns true.

During the reconstruction phase, each node decrypts its share using its secret key and generates a non-interactive zero-knowledge proof of correct decryption using the dleq.Prove algorithm. Let $\tilde{s}_i$ and $\tilde{\pi}_i$ be the decrypted share and its correctness proof of node i, respectively. Node i then sends $\langle \texttt{RECONSTRUCT}, i, \tilde{s}_i, \tilde{\pi}_i \rangle$ to all nodes. Upon receiving a `RECONSTRUCT` message, each node validates it using the dleq.Verify algorithm. Finally, after receiving ℓ valid `RECONSTRUCT` messages, each honest node reconstructs g_1^s using Lagrange interpolation.

Remark. The above dual-threshold ACSS protocol uses a random oracle in two places: the non-interactive variant of Chaum-Pedersen's "Σ" protocol for proving equality of discrete logarithm and Scrape's PVSS. If one wishes to avoid the random oracle, one can use the pairing-based equality of discrete logarithm and a recent PVSS scheme based on Bilinear decisional Diffie-Hellman (DBDH) from [23].

Dual-threshold ACSS for arbitrary secrets. Although our dual-threshold ACSS in Algorithm 6 only supports uniform secrets, it can be extended to arbitrary secrets $z \in \mathbb{Z}_q$ using techniques from [48]. Briefly, to share an arbitrary secret $z \in \mathbb{Z}_q$, the dealer first runs the sharing phase of Algorithm 6 for a random secret g_1^s along with a RBC on $z \cdot h^{-s}$. Upon reconstructing h^s, each honest node can use it to recover z. The security of this approach requires the Decisional Diffie-Hellman (DDH) assumption. We refer readers to [48] for more details on this approach.

We next analyze our dual-threshold ACSS protocol.

Lemma 5.6 (Termination and Correctness). *The dual-threshold ACSS protocol in Algorithm 6 guarantees termination and correctness for any $\ell < n - t$.*

Proof. Note that when the dealer is honest, the predicate $P(\cdot)$, i.e., the PVSS.Verify check will return true at every honest node. Thus, the Termination and Correctness of our dual-threshold ACSS follow directly from the Termination and Validity guarantees of our RBC protocol.

When the dealer is malicious, and the sharing phase terminates at an honest node, a similar argument as Lemma 5.4 implies that every honest node will eventually terminate the sharing phase. Furthermore, from Lemma 4.2, the PVSS.Verify check succeeds at at least $t + 1$ honest nodes. Hence, except for probability $\binom{2t+1}{t+1}\frac{1}{q^{t+1}}$, $\boldsymbol{v}$ is a commitment to a degree-ℓ polynomial. Furthermore, all these nodes check that $\boldsymbol{c}$ consists of encryptions of valid shares of all nodes. Also, the Totality of the RBC protocol guarantees that every node will eventually output $\boldsymbol{v}\|\boldsymbol{c}\|\boldsymbol{\pi}$; hence, each honest will eventually receive their encrypted share and send it to all nodes during the reconstruction phase.

For any given $\ell < n - t$, since there are at least $\ell + 1$ honest nodes in the system, during the reconstruction phase, each honest node will receive at least $\ell + 1$ valid decrypted shares, which are sufficient to reconstruct the committed polynomial (in the exponent) and recover the secret g_1^s. Note that all of these points lie on a fixed degree-ℓ polynomial, which gets finalized whenever the sharing phase terminates at one honest node. Moreover, during the reconstruction phase, every node validates the decrypted shares against the same polynomial commitment. As a result, any set of $\ell + 1$ valid shares will result in the same output. This implies that our dual-threshold ACSS in Algorithm 6 achieves both Termination and Correctness. □

Lemma 5.7 (Completeness). *The dual-threshold ACSS protocol in Algorithm 6 guarantees Completeness for any $\ell < n - t$.*

Proof sketch. When the dealer is honest, Completeness follows directly from the validity guarantee of our RBC protocol and the fact that PVSS.Verify check will be successful at every honest node. When the dealer is malicious, at the end of sharing phase, from the totality and agreement guarantees of our RBC protocol, every honest node will eventually receive an identical $\boldsymbol{v}\|\boldsymbol{c}\|\boldsymbol{\pi}$. Furthermore, Lemma 4.4 implies that the check PVSS.Verify$(\boldsymbol{v}, \boldsymbol{c}, \boldsymbol{\pi})$ was successful at least $t + 1$ honest nodes. Hence, due to the correctness guarantee of PVSS.Verify, $\boldsymbol{v}$ is a valid polynomial commitment to a degree-ℓ polynomial and $\boldsymbol{c}$ consists of correct encryptions of shares of every node. Thus, upon receiving $\boldsymbol{v}\|\boldsymbol{c}\|\boldsymbol{\pi}$, each node can locally decrypt the appropriate ciphertext in $\boldsymbol{c}$ to recover its share. Putting all these together, we get that Algorithm 6 guarantees Completeness. □

Lemma 5.8 (Secrecy). *The dual-threshold ACSS protocol in Algorithm 6 guarantees Secrecy for any threshold $\ell < n - t$.*

Proof sketch. When the secret is a uniformly random element $g_1^s \in \mathbb{G}$, IND1-Secrecy property (Definition F.2) of Scrape's PVSS implies that, to an adversary that corrupts up to ℓ nodes, assuming hardness of DDH, g_1^s is indistinguishable from a randomly chosen element in $\mathbb{G}$. Hence, when we use g_1^s as a one time pad in the transformation of [48], our dual-threshold ACSS ensures Secrecy for arbitrary secrets. □

Remark. Note that our dual-threshold ACSS ensures secrecy even in scenarios where the adversary corrupts more than t (up to ℓ) nodes during both sharing and reconstruction phase. In contrast, existing schemes [3, 16, 34] can not ensure secrecy if the adversary corrupts more than t nodes during the sharing phase. For termination of the sharing phase, all existing dual-threshold ACSS schemes (including ours) require that the adversary corrupts only up to t nodes during the sharing phase and up to $n - \ell - 1$ nodes during the reconstruction phase.

6 LOWER BOUNDS ON COMMUNICATION COMPLEXITY

In this section, we present communication complexity lower bound results for the problems studied in this paper. All the lower bounds we show hold even under synchrony, so they naturally apply to the harder network model of asynchrony.

For data dissemination, we show a lower bound of $\Omega(n|M|+n^2)$ in Theorem 6.2 below. Thus, our ADD protocol has optimal communication complexity for $|M| \geq \Theta(n \log n)$ (see remark in Section 3), and has a gap of at most $O(\log n)$ otherwise. Our lower bound proof draws inspiration from Byzantine agreement lower bound proof of Dolev and Reischuk [24].

To prove the lower bound for data dissemination (DD), we reformulate the problem slightly, which we refer to as the *Fixed-sender* Data Dissemination (f-DD) problem. It is easy to see that any protocol that solves DD also solves f-DD. Thus a lower bound for f-DD applies to DD as well.

Definition 6.1 (Fixed-sender Data Dissemination (f-DD)). Consider a network of n nodes consisting of $2t + 1$ senders whose input is M and $n - (2t + 1)$ recipients whose input is $\perp$. Let there be up to t malicious nodes among all nodes. A protocol Π solves Fixed-sender Data Dissemination (f-DD) if it ensures that every honest node eventually outputs M.

Theorem 6.2. *Any protocol that solves f-DD or DD for message M with $n \geq 3t + 1$ must incur a cost of $\Omega(n|M|+n^2)$.*

Proof. First, $\Omega(n|M|)$ is a trivial lower bound for f-DD, since $n-(2t+1) \geq t = \Omega(n)$ honest recipients need to receive the message M eventually, which incurs a cost of $\Omega(n|M|)$.

Next, we show $\Omega((t/2)^2) = \Omega(n^2)$ is also a lower bound for f-DD, even for 1-bit message. The proof is analogous to the $\Omega((t/2)^2)$ lower bound proof for Byzantine agreement or broadcast due to Dolev and Reischuk [24]. Suppose there exists protocol Π that solves DD for 1 bit using $< (t/2)^2$ messages. Let S be the set of recipient nodes who will output 1 if they do not receive any message during execution of Π. Without loss of generality, suppose that $|S| \geq t/2$ (otherwise a symmetric argument works as well). Consider the following two scenarios.

I. Let the sender's input message be 0. The adversary corrupts a set of recipient nodes $B \subseteq S$ where $|B| = t/2$. Let A denote the set of remaining nodes. Each corrupt node in B behave honestly except (i) they do not send messages to each other; and, (ii) ignore the first $t/2$ messages they receive. Then, all honest nodes will output 0. Note that, since Π uses less than $(t/2)^2$ messages and $|B| = t/2$, $\exists p \in B$ such that p receives less than $t/2$ messages.
II. Let the sender's input message be 0. Let $A(P)$ denote the set of nodes that (attempt to) send p messages in Scenario I. Since p receives $< t/2$ messages, $|A(p)| < t/2$. The adversary corrupts nodes in $A(p) \cup (B \setminus \{p\})$. This is allowed because $|A(p) \cup (B \setminus \{p\})| < t$. The corrupt nodes in $B \setminus \{p\}$ behave the same as in Scenario I except that they additionally ignore all messages from node p. The corrupted nodes in $A(p)$ behave the same as in Scenario I except that they do not send messages to p.

We claim that Scenario I and II are distinguishable to nodes in $A \setminus A(p)$. Nodes in $B \setminus \{p\}$ behave the same to all nodes. Nodes in $A(p)$ behave the same to $A \setminus A(p)$ by construction. Node p behaves the same since in both Scenarios it does not receive any message from others. Thus the claim holds.

Due to the claim above, in Scenario II, honest nodes in $A \setminus A(p)$ output 0 like they did in Scenario I. However, since p in Scenario II does not receive any message, it will output 1 by assumption, which violates the requirement of f-DD. Therefore, such a protocol Π does not exists, and $\Omega((t/2)^2) = \Omega(n^2)$ is a lower bound for f-DD even for 1-bit message.

Hence, any protocol that solves f-DD for a message M must incur a communication cost of $\Omega(n|M|+n^2)$. By definition, any protocol that solves DD also solves f-DD, and therefore the lower bound applies to DD as well. □

For RBC for long messages, a lower bound of $\Omega(n|M|+n^2)$ is straightforward: $\Omega(n|M|)$ comes from the fact that $\Omega(n)$ honest nodes each need to receive the message M from the broadcaster, and $\Omega(n^2)$ is a lower bound for solving RBC even for single-bit input [4, 24]. Hence, our RBC protocol is optimal when $|M| \geq \Theta(\kappa n)$, and has a gap of at most $O(\kappa)$ from the optimal otherwise. For all variants of secret sharing problems (AVSS, ACSS, Dual-threshold AVSS), there is a lower bound of $\Omega(n^2)$ according to [4]. Since a AVSS protocol is sufficient to solve RBC [4], the $\Omega(n^2)$ lower bound on RBC also applies to AVSS, and other stronger secret sharing problems as well. Therefore, our secret sharing protocol constructions are a factor of $O(\kappa)$ from the optimal.

7 RELATED WORK

To the best of our knowledge, the ADD problem has not been studied before. This may be in part because, despite being a simple primitive, its applications are not immediately apparent. In hindsight, the biggest conceptual barrier for us in this work was to realize the usefulness of ADD as opposed to designing protocols to solve it. For many of the applications we have identified for ADD in this work, we had to introduce additional techniques to address other efficiency bottlenecks in them.

Reliable broadcast. The problem of reliable broadcast (RBC) was introduced by Bracha [14]. In the same paper, Bracha provided an RBC protocol for a single bit with a communication cost of $O(n^2)$, thus $O(n^2|M|)$ for $|M|$ bits using a naïve approach. Almost two decades later, Cachin and Tessaro [16] improved the cost to $O(n|M|+\kappa n^2 \log n)$ assuming a collision-resistant hash function with κ being the output size of the hash. Hendricks et al. in [32] propose an alternate RBC extension protocol with a communication cost of $O(n|M|+\kappa n^3)$ using a erasure coding scheme where each element of a codeword can be verified for correctness.

Recently, assuming a trusted setup phase, hardness of q-SDH [11, 12] and Decisional Bilinear Diffie-Hellman (DBDH) [13], Nayak et al. [41] reduced the communication cost to $O(n|M|+\kappa n^2)$. Our RBC protocol achieves the best of both worlds, i.e., it does not require a setup, and has a communication cost of $O(n|M|+\kappa n^2)$.

Asynchronous VSS/CSS. The problem of asynchronous verifiable secret sharing has been studied for decades in many different settings [3–6, 15, 17, 18, 34, 42, 52]. The information-theoretically secure schemes [17, 18, 22, 42, 43] mostly have high communication cost or sub-optimal fault tolerance. Using cryptographic assumption such as collision resistant hash function and Decisional Diffie-Hellman assumption, Cachin et al. [15] proposed a AVSS scheme with communication cost of $O(n^3\kappa)$. Backes et al. [4] later improved it to $O(\kappa n^2)$ assuming a trusted setup phase and the hardness of q-SDH. Very recently, Alhaddad et al. [3] proposed a ACSS scheme for uniformly random secrets with a total communication cost of $O(\kappa n^2 \log n)$.

Some works focused on improving the amortized communication cost of AVSS for many secrets [3, 16, 52], such as amortized cost of $O(n^2\kappa)$ per secret in Cachin et al. [16], and $O(n\kappa)$ per secrets in both hbACSS [52] and Haven [3].

Dual-threshold ACSS. The problem of dual-threshold AVSS was introduced by Cachin and Tessaro [16] where they provide a dual-threshold for $t < n/4$ and $\ell < n/2$ with communication cost of $O(\kappa n^3)$. Only recently, Kokoris et al. [34] proposed the first dual-threshold ACSS protocol for $t < n/3$ and $\ell < 2n/3$ with a communication cost of $O(\kappa n^3)$ and Alhaddad et al. [3] improved the communication cost to $O(\kappa n^2 \log n)$. If we assume a trusted setup and use the polynomial commitment scheme of [33], then the total communication cost can be improved to $O(\kappa n^2)$. Moreover, during the sharing phase, all of the above mentioned schemes provide secrecy only against an adversary that corrupts up to t nodes. Contrary to the existing schemes, our dual-threshold ACSS ensures secrecy against an adversary that corrupts up to ℓ nodes.

Asynchronous Distributed Key Generation. There are relatively few works on asynchronous DKG [2, 18, 29, 34]. The ADKG construction of Canetti and Rabin [18] uses n^2 AVSS and is hence inefficient. Kokoris et al, [34] uses n dual-threshold ACSS with reconstruction threshold of $2t + 1$ and an asynchronous common subset protocol to design an ADKG scheme with a total communication cost of $O(\kappa n^4)$.

Very recently, Abraham et al. [2] proposed an ADKG protocol with a communication cost of $O(\kappa n^3 \log n)$. The factor of $\log n$ in the ADKG construction stems from the use of the prior best RBC protocol of Cachin and Tessaro [16]. Using our new RBC protocol in a blackbox manner, we can improve the communication cost of their ADKGto $O(\kappa n^3)$. A *limitation* of the Abraham et al ADKG [2] is that the secret key is a group element and not a field element. As a result, their ADKG protocol is not compatible with off-the-shelf threshold signature schemes such as [10]. Since we only replace the RBC of [2] with ours, we inherit this limitation.

Concurrent work. Concurrently and independently, Gao et al. [29] design a new AVSS scheme that has a communication cost of $O(\kappa n^2)$. Gao et al. [29] then use their improved AVSS scheme along with a novel reasonably-fair common coin protocol to design Asynchronous Byzantine Agrement (ABA) [35] and ADKG protocol both with an expected communication cost of $O(\kappa n^3)$. They also inherit the aforementioned limitation from Abraham et al's ADKG [2]. Gao et al. [29] additionally observes that the asynchronous Validated Byzantine Agreement (VBA) protocol in Abraham et al. [2] implicitly gives an ABA [35] protocol with a communication cost of $O(\kappa n^3 \log n)$ and no trusted setup. This means our RBC protocol can also improve the communication cost of VBA and ABA to $O(\kappa n^3)$.

8 DISCUSSION

Concrete communication cost our ADD and RBC protocols. Although we mostly discuss asymptotic cost in this paper, it is equally important from a practical point of view to make the hidden constants small. Indeed, this is the case with the primitives we construct in this work. In particular, our ADD with $n = 3t + 1$ has a concrete communication cost of $B_{\mathrm{ADD}} = 6n|M|+2n^2$. The factor 6 is due to the increase in the size of the message due to RSEnc and the fact that ADD has two rounds of communication. If we substitute B_{ADD} in our RBC protocol for long messages we get a communication cost of $n|M|+2\kappa n^2 + B_{\mathrm{ADD}}$. Here the $n|M|$ accounts for the communication cost the broadcaster incurs while sending M to all nodes. The $2\kappa n^2$ is the cost of Bracha's RBC on the hash(M). Similarly, our four-round RBC incurs the same communication cost of $7n|M|+2\kappa n^2 + 2n^2$.

Limitation of ADD. One limitation of using ADD in our RBC is its higher computation costs due to encoding and decoding of the message. Additionally, in the presence of malicious nodes, each honest node may have to try decoding t times. Contrary to this, in the RBC protocol of Cachin and Tessaro [15], each node needs to run the decoding algorithm only once.

9 CONCLUSION

In this paper we have introduced the problem of Data Dissemination, which seeks to disseminate a data blob M from a subset of honest subset of honest nodes to *all* honest nodes, despite the presence of some malicious nodes. We have presented an Asynchronous Data Dissemination (ADD) protocol for n parties with a communication cost of $O(n|M|+n^2)$. We then used our ADD protocol to improve the communication cost or trust assumption of RBC for long messages, AVSS, ACSS, dual-threshold ACSS, and ADKG.

We believe ADD can be useful in other applications that we did not study in this paper, e.g., in improving the communication cost of recent randomness beacon protocols for both synchronous [8] and partially synchronous networks [23]. Generally speaking, ADD will be useful in protocols that involve distribution of long common messages across all nodes. These messages include but are not limited to blocks in blockchain protocols, polynomial commitments, encrypted shares, NIZK proofs, etc.

ACKNOWLEDGMENTS

The authors would like to thank Vinith Krishnan, Andrew Miller, and Tom Yurek for many helpful discussions related to the paper.

REFERENCES

[1] Ittai Abraham, TH Hubert Chan, Danny Dolev, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. 2019. Communication complexity of byzantine agreement, revisited. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. 317–326.

[2] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Reaching Consensus for Asynchronous Distributed Key Generation. In *Proceedings of the 2021 ACM Symposium on Principles of Distributed Computing*. 363–373.

[3] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. 2021. High-Threshold AVSS with Optimal Communication Complexity. Cryptology ePrint Archive, Report 2021/118. (2021). https://eprint.iacr.org/2021/118.

[4] Michael Backes, Amit Datta, and Aniket Kate. 2013. Asynchronous computational VSS with reduced communication complexity. In *Cryptographers' Track at the RSA Conference*. Springer, 259–276.

[5] Soumya Basu, Alin Tomescu, Ittai Abraham, Dahlia Malkhi, Michael K Reiter, and Emin Gün Sirer. 2019. Efficient Verifiable Secret Sharing with Share Recovery in BFT Protocols. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 2387–2402.

[6] Michael Ben-Or, Ran Canetti, and Oded Goldreich. 1993. Asynchronous secure computation. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. 52–61.

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

[8] Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. 2021. RandPiper–Reconfiguration-Friendly Random Beacons with Quadratic Communication. In *(To appear) Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security.*

[9] George Robert Blakley. 1979. Safeguarding cryptographic keys. In *1979 International Workshop on Managing Requirements Knowledge (MARK)*. IEEE, 313–318.

[10] Alexandra Boldyreva. 2003. Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme. In *International Workshop on Public Key Cryptography*. Springer, 31–46.

[11] Dan Boneh and Xavier Boyen. 2004. Short signatures without random oracles. In *International conference on the theory and applications of cryptographic techniques*. Springer, 56–73.

[12] Dan Boneh and Xavier Boyen. 2008. Short signatures without random oracles and the SDH assumption in bilinear groups. *Journal of cryptology* 21, 2 (2008), 149–177.

[13] Dan Boneh, Ben Lynn, and Hovav Shacham. 2001. Short signatures from the Weil pairing. In *International conference on the theory and application of cryptology and information security*. Springer, 514–532.

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

[15] Christian Cachin, Klaus Kursawe, Anna Lysyanskaya, and Reto Strobl. 2002. Asynchronous verifiable secret sharing and proactive cryptosystems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security*. 88–97.

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

[17] Ran Canetti. 1996. *Studies in secure multiparty computation and applications*. Ph.D. Dissertation. Citeseer.

[18] Ran Canetti and Tal Rabin. 1993. Fast asynchronous Byzantine agreement with optimal resilience. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. 42–51.

[19] Ignacio Cascudo and Bernardo David. 2017. SCRAPE: Scalable randomness attested by public entities. In *International Conference on Applied Cryptography and Network Security*. Springer, 537–556.

[20] Miguel Castro, Barbara Liskov, et al. 1999. Practical Byzantine fault tolerance. In *Proceedings of the Third Symposium on Operating Systems Design and Implementation*. 173–186.

[21] David Chaum and Torben Pryds Pedersen. 1992. Wallet databases with observers. In *Annual International Cryptology Conference*. Springer, 89–105.

[22] Ashish Choudhury. 2020. Optimally-resilient Unconditionally-secure Asynchronous Multi-party Computation Revisited. *IACR Cryptol. ePrint Arch.* 2020 (2020), 906.

[23] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. 2021. SPURT: Scalable Distributed Randomness Beacon with Transparent Setup. Cryptology ePrint Archive, Report 2021/100. (2021). https://eprint.iacr.org/2021/100.

[24] Danny Dolev and Rüdiger Reischuk. 1985. Bounds on information exchange for Byzantine agreement. *Journal of the ACM (JACM)* 32, 1 (1985), 191–204.

[25] Sisi Duan, Michael K Reiter, and Haibin Zhang. 2018. BEAT: Asynchronous BFT made practical. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2028–2041.

[26] Amos Fiat and Adi Shamir. 1986. How to prove yourself: Practical solutions to identification and signature problems. In *Conference on the theory and application of cryptographic techniques*. Springer, 186–194.

[27] Adam Gągol, Damian Leśniak, Damian Straszak, and Michał Świętek. 2019. Aleph: Efficient atomic broadcast in asynchronous networks with byzantine nodes. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*. 214–228.

[28] Shuhong Gao. 2003. A new algorithm for decoding Reed-Solomon codes. In *Communications, information and network security*. Springer, 55–68.

[29] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2021. Efficient Asynchronous Byzantine Agreement without Private Setups. *arXiv preprint arXiv:2106.07831* (2021).

[30] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2020. Dumbo: Faster asynchronous bft protocols. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 803–818.

[31] Somayeh Heidarvand and Jorge L Villar. 2008. Public verifiability from pairings in secret sharing schemes. In *International Workshop on Selected Areas in Cryptography*. Springer, 294–308.

[32] James Hendricks, Gregory R Ganger, and Michael K Reiter. 2007. Verifying distributed erasure-coded data. In *Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing*. 139–146.

[33] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. 2010. Constant-size commitments to polynomials and their applications. In *International conference on the theory and application of cryptology and information security*. Springer, 177–194.

[34] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. 2020. 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*. 1751–1767.

[35] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems* 4, 3 (1982), 382–401.

[36] Donghang Lu, Thomas Yurek, Samarth Kulshreshtha, Rahul Govind, Aniket Kate, and Andrew Miller. 2019. Honeybadgermpc and asynchromix: Practical asynchronous mpc and its application to anonymous communication. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 887–903.

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

[38] Florence Jessie MacWilliams and Neil James Alexander Sloane. 1977. *The theory of error correcting codes*. Vol. 16. Elsevier.

[39] Robert J. McEliece and Dilip V. Sarwate. 1981. On sharing secrets and Reed-Solomon codes. *Commun. ACM* 24, 9 (1981), 583–584.

[40] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. 2016. The honey badger of BFT protocols. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 31–42.

[41] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H Vaidya, and Zhuolun Xiang. 2020. Improved Extension Protocols for Byzantine Broadcast and Agreement. In *34th International Symposium on Distributed Computing (DISC 2020)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik.

[42] Arpita Patra, Ashish Choudhary, and C Pandu Rangan. 2009. Efficient statistical asynchronous verifiable secret sharing with optimal resilience. In *International Conference on Information Theoretic Security*. Springer, 74–92.

[43] Arpita Patra, Ashish Choudhury, and C Pandu Rangan. 2015. Efficient asynchronous verifiable secret sharing and multiparty computation. *Journal of Cryptology* 28, 1 (2015), 49–109.

[44] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*. Springer, 129–140.

[45] David Pointcheval and Jacques Stern. 1996. Security proofs for signature schemes. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 387–398.

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

[47] Alexandre Ruiz and Jorge L Villar. 2005. Publicly verifiable secret sharing from Paillier's cryptosystem. In *WEWoRC 2005–Western European Workshop on Research in Cryptology*. Gesellschaft für Informatik eV.

[48] Berry Schoenmakers. 1999. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference*. Springer, 148–164.

[49] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.

[50] Lloyd R Welch and Elwyn R Berlekamp. 1986. Error correction for algebraic block codes. (Dec. 30 1986). US Patent 4,633,470.

[51] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. 2019. Hotstuff: Bft consensus with linearity and responsiveness. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*. ACM, 347–356.

[52] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew K Miller. 2021. hbACSS: How to Robustly Share Many Secrets. (2021), 159 pages.

A ANALYSIS OF ALGORITHM 4

In this section, we will analyze that Algorithm 4 implements RBC for long messages while keeping the round complexity at 4.

Lemma A.1. *Assuming a collision resistant hash function, if an honest node sends $\langle \texttt{READY}, m_i, h\rangle$ where $h = \mathsf{hash}(M)$, then m_i is the i^{th} symbol of $\mathsf{RSEnc}(M, n, t+1)$, and furthermore, no honest node sends a READY message for a different $h' \neq h$.*

Proof. Let i be the first honest node that sends a $\langle \texttt{READY}, *, h\rangle$. Then at least $2t+1$ nodes sent $\langle \texttt{ECHO}, *, h\rangle$ to node i. Now, for the sake of contradiction assume that an honest node i' is the first honest node that sends a $\langle \texttt{READY}, *, h'\rangle$ for $h' \neq h$. Again at least $2t+1$ nodes sent $\langle \texttt{ECHO}, *, h'\rangle$ to node i'. Then, by quorum intersection, at least $t+1$ nodes sent ECHO message for both h and h'. This is impossible as there are at most t malicious nodes and an honest node sends ECHO message at most once.

Note that an honest node i sends $\langle \texttt{READY}, m_i, h\rangle$ for $h = \mathsf{hash}(M)$ only upon receiving at least $t+1$ matching $\langle \texttt{ECHO}, m_i, h\rangle$. At least one of these ECHO message is from an honest node. Then, by the collision resistance property of the underlying hash function, m_i is the i^{th} symbol of $\mathsf{RSEnc}(M, n, t+1)$. □

Lemma A.2. *If an honest node i receives $t+1$ READY messages with a matching hash h, then node i will eventually receive $t+1$ matching $\langle \texttt{ECHO}, m_i, h\rangle$ messages and hence send $\langle \texttt{READY}, m_i, h\rangle$.*

Proof. Let j be the first honest node that sends $\langle \texttt{READY}, *, h\rangle$ message to all. Then, node j must have received at least $2t+1$ ECHO messages with matching h. At least $t+1$ of these ECHO messages are from honest nodes. All these honest node will send $\langle \texttt{ECHO}, m_i, h\rangle$ to node i. Hence, node i will eventually receive $t+1$ $\langle \texttt{ECHO}, m_i, h\rangle$ messages. □

Theorem A.3 (Totality and Agreement). *Algorithm 4 guarantees Totality and Agreement.*

Proof. An honest node outputs a message M only upon receiving at least $2t+1$ READY messages with a matching hash $h = \mathsf{hash}(M)$. At least $t+1$ of them are sent by an honest node. Hence, all honest nodes will receive at least $t+1$ READY messages with hash h. By lemma A.2, eventually all honest nodes will send READY messages with hash h. Hence, all honest nodes will receive READY messages from all other honest nodes. Furthermore, due to Lemma A.1, all these READY message contain correct symbols from the codeword $\mathsf{RSEnc}(M, n, t+1)$. Thus, every honest node will eventually output M such that $h = \mathsf{hash}(M)$. □

Theorem A.4 (Validity). *Algorithm 4 guarantees Validity.*

Proof. When the broadcaster is honest and has input M, at least $2t+1$ honest nodes will send ECHO messages with identical $h = \mathsf{hash}(M)$. Hence, all honest nodes will eventually send READY messages for h. By lemma A.1 no honest node will send READY message for $h' \neq h$. As a result, all honest node will receive at least $2t+1$ READY message for h with valid symbols in it, which is sufficient to recover M. □

Next, we will analyze the communication complexity of the protocol.

Let $h = \mathsf{hash}(p(\cdot))$ i.e., hash of the coefficients of the original polynomial $p(\cdot)$.

For $0 \le r \le t$:

(1) Let T_r be the subset of symbols received till iteration r. Wait until $|T_r| \ge (m/n)(n - t + r + 1)$.
(2) Run $\mathsf{RSDec}(t+1, r, T_r)$, and let $p_r(\cdot)$ be the output polynomial.
(3) If $\mathsf{hash}(p_r(\cdot)) = h^*$, output $p_r(\cdot)$. Otherwise, proceed to the next iteration.

Figure 3: Hash based Online Error Correction (HOEC).

Lemma A.5. *Assuming existence of collision resistant hash functions, Algorithm 4 solves RBC with communication complexity of $O(n|M| + \kappa n^2)$ where κ is the size of the output of the hash function.*

Proof. In algorithm 4 the broadcaster sends a single PROPOSE to all other nodes. Moreover, each honest node sends a single ECHO and READY message. The proposal message has a communication cost of $O(n|M|)$. Since $|m_i| = |M|/(t+1)$ and hash outputs are κ bits long, both ECHO and READY messages are $O(|M|/n + \kappa)$ bits long. Hence, each node incur a communication cost of $O(|M| + n\kappa)$ for sending ECHO and READY messages to all other nodes. Hence, the total communication cost is $O(n|M| + \kappa n^2)$. □

B ADD FOR HIGH THRESHOLD

Recall from §3 that it is impossible to solve ADD for $n \le 2t$. In this section we will describe how to extend our solution to ADD from $n > 3t$ to $n > 2t$. To do so, we will make use of a collision-resistant hash function. The detailed changes in our original protocol are as follows.

During the encoding phase, each sender encodes the message M with $m = n(t+1)/(n-2t)$ (due to reasons to be described later). Let $M' = \mathsf{RSEnc}(M, m, t+1)$ be the encoded message.

During the dispersal phase, each sender sends the i^{th} component of M' (i.e., m_i) to node i. Note that m_i consists of m/n symbols of the codeword, hence its size is

$$|m_i| = \frac{m|M|}{n(t+1)} = \frac{n(t+1)|M|}{(n-2t)n(t+1)} = \frac{|M|}{n-2t}$$

Furthermore, each sender additionally sends the cryptographic hash digest $h = \mathsf{hash}(M)$ to all other nodes. In particular, during the dispersal phase, each sender sends $\langle \texttt{DISPERSE}, m_i, h\rangle$ to node i. A recipient node j, upon receiving $t+1$ matching $\langle \texttt{DISPERSE}, m_j, h\rangle$ message, sets $m_j^* = m_j$ and $h^* = h$, which will be used during the reconstruction phase.

During the reconstruction phase, similar to our $t < n/3$ ADD, each node i sends $\langle \texttt{RECONSTRUCT}, m_i^*\rangle$ (cf. Algorithm 1) to all other nodes. The procedure to reconstruct the message differs from OEC. In particular, every node instead runs the hash-based OEC that we call HOEC and describe below.

The HOEC algorithm, given in Figure 3, takes up to $t+1$ iterations. In iteration r, each recipient node i waits to receive $n - t + r + 1$ RECONSTRUCT messages. For every received $\langle \texttt{RECONSTRUCT}, m_j\rangle$ message from node j, node i adds (j, m_j) to a set T_r. Node i then uses the RSDec algorithm on T_r to recover the original message.

Let $p_r(\cdot) := \mathsf{RSDec}(t+1, r, T_r)$ be the output of the decoding algorithm, then the recipient outputs the coefficients of $p_r(\cdot)$ as the final message only if $\mathsf{hash}(p_r(\cdot))$ matches h^*, set during the dispersal phase. Here, $\mathsf{hash}(p_r(\cdot))$ denotes the hash of the coefficients of the polynomial $p_r(\cdot)$. If the hashes do not match, the recipient starts the next iteration.

The proof remains almost identical to the proofs of ADD with $n \geq 3t+1$ nodes. Thus we briefly restate them below.

Lemma B.1. *At the end of the reconstruction phase, every honest node outputs M.*

Proof. Clearly, every honest sender outputs M. Thus, we again focus on honest recipient nodes.

We will first argue that an honest recipient node will never output a wrong message. Suppose node i outputs the polynomial $p_r(\cdot)$ as the message in the r^{th} iteration in the reconstruction phase. Then, $\mathsf{hash}(p_r(\cdot)) = h^* = \mathsf{hash}(p(\cdot))$. Hence, by collision resistance property of the hash function, we get $p_r(\cdot) = p(\cdot)$.

Next, we argue that every honest recipient node will eventually output a message. Note that Lemma 3.2 also holds for $h = \mathsf{hash}(M)$ in the high-threshold ADD protocol. Then as before, at the end of the dispersal phase, every honest node holds the correct component of $M' = \mathsf{RSEnc}(M, n, t+1)$ and will send it to all other nodes in its RECONSTRUCT messages. Every recipient node will eventually receive $(n-t)m/n$ correct symbols from all honest nodes and at most tm/n incorrect symbols from malicious nodes. Given $m \geq n(t+1)/(n-2t)$, the RSDec algorithm will correct all the errors and return the correct message. □

We will next analyze the communication complexity of our high-threshold ADD.

Lemma B.2. *Assuming the existence of collision resistant hash function, in a network of $n > 2t$ nodes where up to t nodes could be malicious, our high-threshold ADD has a total communication cost of $O(n^2|M|/(n-2t) + n^2\kappa)$. Here κ is the size of the output of the hash function.*

Proof. During the dispersal phase, each sender sends a message of size $O(|M'|/n + \kappa)$ to every other node. Hence, the total communication cost of every sender is $O(|M'|+n\kappa)$. Since there are $\Theta(n)$ senders, the total communication cost in the dispersal phase is $O(n|M'|+n^2\kappa)$. During the reconstruction phase, each nodes sends a message of size $O(|M'|/n)$ to every other node. Hence, the total communication cost during the reconstruction phase is $O(n|M'|)$

Since in our high-threshold ADD $|M'| = O(n|M|/(n-2t)$, it has a total communication cost of $O(n^2|M|/(n-2t) + n^2\kappa)$. □

C THRESHOLD SECRET SHARING

A (n, k) threshold secret sharing scheme allows a secret $s \in \mathbb{Z}_q$ to be shared among n nodes such that any k of them can come together to recover the original secret, but any subset of $k-1$ shares cannot be used to recover the original secret [9, 49]. We use the common Shamir secret sharing [49] scheme, where the secret is embedded in a random degree $k-1$ polynomial in the field $\mathbb{Z}_q$ for some prime q. Specifically, to share a secret $s \in \mathbb{Z}_q$, a polynomial $p(\cdot)$ of degree $k-1$ is chosen such that $s = p(0)$. The remaining coefficients of $p(\cdot)$, $a_1, a_2, \cdots, a_t$ are chosen uniformly randomly from $\mathbb{Z}_q$. The resulting polynomial $p(x)$ is defined as:

$$p(x) = s + a_1x + a_2x^2 + \cdots + a_{k-1}x^{k-1}$$

Each node is then given a single evaluation of $p(\cdot)$. In particular, the i^{th} node is given $p(i)$ i.e., the polynomial evaluated at i. Observe that given $t+1$ points on the polynomial $p(\cdot)$, one can efficiently reconstruct the polynomial using Lagrange Interpolation. Also note that when s is uniformly random in $\mathbb{Z}_q$, s is information theoretically hidden from an adversary that knows any subset of $k-1$ or less evaluation points on the polynomial other than $p(0)$ [49].

Let s be the secret a node (the dealer) and let g_0, g_1 two uniformly random and independent generators of a group $\mathbb{G}$.

$\mathsf{PedPolyCommit}(g_0, g_1, s) \to \{\boldsymbol{s}, \boldsymbol{c}, \boldsymbol{r}\}$

(1) Sample $r \leftarrow \mathbb{Z}_q$ and let $v_0 = \mathsf{commit}(s, r)$.
(2) Sample $a_k, b_k \in Z_q$ for $k = 1, 2, ..., t$ and let:

$$p(x) = s + a_1x + \ldots + a_tx^t; \text{ and}$$
$$\phi(x) = r + b_1x + \ldots + b_tx^t$$

(3) Let $v_k = \mathsf{commit}(a_k, b_k)$ for $k = 1, 2, ..., t$,
(4) Let $\boldsymbol{v} = \{v_0, v_1, \ldots, v_t\}$, $\boldsymbol{s} = \{p(1), p(2), \ldots, p(n)\}$, and let $\boldsymbol{r} = \{\phi(1), \phi(2), \ldots, \phi(n)\}$.
(5) **return** $\{\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{r}\}$

$\mathsf{PedEvalVerify}(g_0, g_1, \boldsymbol{v}, i, s_i, r_i) \to 0/1$:

(1) If

$$\mathsf{commit}(s_i, r_i) = g_0^{s_i} g_1^{r_i} = \prod_{j=0}^{t} v_j^{i^j}$$

return 1 otherwise **return 0.**

Given PedPolyCommit and PedEvalVerify be the polynomial commitment and evaluation scheme, respectively. The VSS scheme is defined as below.

$\mathsf{VSS.Share}(s, g_0, g_1, n, t)$:

(1) Let $\{\boldsymbol{v}, \boldsymbol{s}, \boldsymbol{r}\} := \mathsf{PedPolyCommit}(g_0, g_1, s)$
(2) **Broadcast** $\boldsymbol{v}$ to all nodes and **send** $\boldsymbol{s}[i], \boldsymbol{r}[i]$ to node i

$\mathsf{VSS.Verify}(g_0, g_1, \boldsymbol{v}, s_i, r_i) \to 0/1$:

(1) **Output** $\mathsf{PedEvalVerify}(g_0, g_1, \boldsymbol{v}, i, s_i, r_i)$.

Let T be the set of valid shares s_k where $|T| = t+1$, then

$\mathsf{VSS.Recon}(\{s_k\}_{k\in T}) \to s$:

(1) Output

$$\sum_{k\in T} s_k \cdot \mu_k = p(0) = s \tag{3}$$

where $\mu_k = \prod_{j\neq k} \frac{j}{j-k}$ for $k \in T$ are Lagrange coefficients.

Figure 4: Pedersen's VSS scheme [44]

D PEDERSEN'S VSS [44]

Let κ be the security parameter. Let $\mathbb{G}$ be a cyclic abelian group of prime order q and $\mathbb{Z}_q$ the group of integer modulo q. Let $g_0, g_1 \leftarrow \mathbb{G}$ be two uniform and independent element from $\mathbb{G}$. Before we describe Pedersen's VSS scheme, we will first briefly describe the commitment scheme for a arbitrary secret $s \in \mathbb{Z}_q$. To commit to a secret s, the committer samples a random $r \in \mathbb{Z}_q$ and computes

$$\mathsf{commit}(s, r) = v = g_0^s g_1^r$$

To reveal such a commitment later, the committer reveals (s, r) and the verifier checks whether $g_0^s g_1^r$ is equal to v or not. We refer to the reveal procedure as:

$$\mathsf{reveal}(v) := s, r \text{ such that } v = g_0^s g_1^r$$

Pedersen [44] illustrates that the commitment scheme described above information theoretically hides s and binds s to v for a computationally bounded prover, assuming the prover does not know the discrete logarithm of g_1 with respect to g_0, i.e., the prover can not efficiently compute $\log_{g_0} g_1$. We summarize the VSS scheme from [44] in Figure 4.

Observe that the commitment to the polynomial $p(\cdot)$ that embeds the secret s is linear in the number of nodes. Moreover, given the linear size commitment and a tuple (s_k, t_k), one can efficiently verify (without any extra information) whether s_k is equal to $p(k)$ or not. We will crucially use these properties to design our AVSS scheme with a total communication cost of $O(\kappa n^2)$.

Next, we briefly summarize the properties of the VSS scheme described in Figure 4. Informally, Lemma D.1 states that once the VSS.Share step terminated correctly, any set of $t + 1$ nodes can combine their shares to recover the secret. Theorem D.3 states that any subset of $t + 1$ nodes will reconstruct the same secret.

Lemma D.1 (Lemma 4.2 of [44]). *Let $S \subset \{1, 2, \ldots, n\}$ be a set of $t + 1$ nodes such that the verification was successful for these $t + 1$ nodes. Then these $t+1$ nodes can find a pair (s', t') such that $v = g_0^{s'} g_1^{t'}$.*

Definition D.2 (Uniqueness). For all subsets S_1 and S_2 of $\{1, 2, \ldots, n\}$ of size k such that all nodes in S_1 and S_2 accepted their shares in the verification protocol described above. Let s_i be the secret computed by the participants in S_i, then $s_1 = s_2$.

Theorem D.3 (Theorem 4.2 of [44]). *Under the assumption that the dealer can not find $\log_{g_0} g_1$ except with negligible probability in $|q|$, the verification protocol satisfies uniqueness.*

Let S be the subset of nodes with $|S| \leq t$, let view_S be the internal state of nodes in S and messages sent and received by nodes in S. Then, the next theorem ensures formally states the secrecy property of the VSS.

Theorem D.4 (Theorem 4.4 of [44]). *For all adversary $\mathcal{A}$, for any subset $S \subset [n]$ of size t and view_S, for all $s \in \mathbb{Z}_q$*

$$\Pr[\mathcal{A} \text{ has secret} \mid \mathsf{view}_S] = \Pr[\mathcal{A} \text{ has secret}]$$

E ZERO KNOWLEDGE PROOF OF EQUALITY OF DISCRETE LOGARITHM

Our dual-threshold ACSS protocol has a step that requires nodes to produce zero-knowledge proofs about the equality of discrete logarithms for a tuple of publicly known values. In particular, given a group $\mathbb{G}$ of prime order q, two uniformly random generators $g_0, g_1 \leftarrow \mathbb{G}$ and a tuple (g_0, x, g_1, y), a prover $\mathcal{P}$ wants to prove to a probabilistic polynomial time (PPT) verifier $\mathcal{V}$, in zero-knowledge, the knowledge of a witness α such that $x = g_0^\alpha$ and $y = g_1^\alpha$.

We will use the Chaum-Pedersen "Σ-protocols" [21], which assumes the hardness of the Decisional Diffie-Hellman (DDH) problem, and can be made non-interactive using the Fiat-Shamir heuristic [26].

Decisional Diffie–Hellman assumption. Given a group $\mathbb{G}$ with generator $g \in \mathbb{G}$ and uniformly random samples $a, b, c \leftarrow \mathbb{Z}_q$, the Decisional Diffie–Hellman (DDH) hardness assumes that the following two distributions D_0, D_1 are computationally indistinguishable: $D_0 = (g, g^a, g^b, g^{ab})$ and $D_1 = (g, g^a, g^b, g^c)$.

Protocol for equality of discrete logarithm. For any given tuple (g_0, x, g_1, y), the Chaum-Pedersen protocol proceeds as follows.

(1) $\mathcal{P}$ samples a random element $\beta \leftarrow \mathbb{Z}_q$ and sends (a_1, a_2) to $\mathcal{V}$ where $a_1 = g_0^\beta$ and $a_2 = g_1^\beta$.
(2) $\mathcal{V}$ sends a challenge $e \leftarrow \mathbb{Z}_q$.
(3) $\mathcal{P}$ sends a response $z = \beta - \alpha e$ to $\mathcal{V}$.
(4) $\mathcal{V}$ checks whether $a_1 = g_0^z x^e$ and $a_2 = g_1^z y^e$ and accepts if and only if both the equality holds.

As mentioned, this protocol can be made non-interactive in the Random Oracle model using the Fiat-Shamir heuristic [26, 45]. This protocol guarantees completeness, knowledge soundness, and zero-knowledge. The knowledge soundness implies that if $\mathcal{P}$ convinces the $\mathcal{V}$ with non-negligible probability, there exists an efficient (polynomial time) extractor that can extract α from the prover with non-negligible probability.

In our dual-threshold ACSS, we use the non-interactive variant of the protocol described above. In particular, for any given tuple (g_0, x, g_1, y) where $x = g_0^s$ and $y = g_1^s$, $\pi \leftarrow \mathsf{dleq.Prove}(s, g_0, x, g_1, y)$ generates the non-interactive zero proof π. The proof π is $O(\kappa)$ bits long. Given the proof π and (g_0, x, g_1, y), $\mathsf{dleq.Verify}(\pi, g_0, x, g_1, y)$ verifies the proof.

F PUBLICLY VERIFIABLE SECRET SHARING

We restate this section from [23]. Our (n, ℓ) dual-threshold ACSS scheme for $n \geq 3t + 1$ and $t \leq \ell < n - t$ crucially rely on on a (n, ℓ) publicly verifiable secret sharing (PVSS). In particular, we use the PVSS scheme from Scrape [19], which is an improvement over the Schoenmakers scheme [48]. The scheme allows a node (dealer) to share a secret $s \in \mathbb{Z}_q$ among n nodes, such that any subset of at least $\ell + 1$ nodes can reconstruct g_1^s. Here, g_1 is a random generator of $\mathbb{G}$. Additionally, any subset of ℓ or less nodes, can not learn any information about the secret s.

The threshold ℓ is chosen in a way such that valid contribution from at least $\ell + 1$ nodes are required to recover g_1^s.

A key property of a PVSS scheme is that, not only the recipients but any third party (with access to recipients' public keys) can verify, even before the reconstruction phase begins, that the dealer has generated the shares correctly without having plaintext access to the shares.

The PVSS scheme of Scrape [19] is non-interactive in the Random Oracle model and has three procedures: PVSS.Share, PVSS.Verify, and PVSS.Recon. A node (dealer) with public-private key pair pk, sk, uses PVSS.Share to share a secret s, other nodes or external users

Let s be the secret a node (the dealer) with public-private key pair (sk, pk) wants to share with set of nodes with public keys $\{pk_j\}_j$ for $j = 1, 2, \ldots, n$. Let g_0, g_1 be two randomly chosen generators of group $\mathbb{G}$.

PVSS.Share$(s, g_0, n, \ell, \{pk\}_{j,j=1,2,\ldots,n}) \rightarrow (\boldsymbol{v}, \boldsymbol{c}, \boldsymbol{\pi})$:

(1) Sample uniform random $a_i \in \mathbb{Z}$ for $i = 1, 2, \ldots, \ell$ and let

$$p(x) = s + a_1 x + \ldots + a_\ell x^\ell;$$

(2) Let $v_j := g_0^{p(j)}$; and $c_j := pk_j^{p(j)}$, for $j = 1, \ldots, n$.

(3) Let $\pi_j :=$ dleq.Prove$(p(j), g_0, v_j, pk_j, c_j)$

(4) Output $\boldsymbol{v} = \{v_1, v_2, \ldots, v_n\}$; $\boldsymbol{c} = \{c_1, c_2, \ldots, c_n\}$, and $\boldsymbol{\pi} = \{\pi_1, \pi_2, \ldots, \pi_n\}$.

PVSS.Verify$(g_0, g_1, n, \ell, \{pk\}_{j,j=1,2,\ldots,n}, \boldsymbol{v}, \boldsymbol{c}, \boldsymbol{\pi}) \longrightarrow 0/1$:

(1) Sample a random codeword $\boldsymbol{y}^\perp \in C^\perp$ where $\boldsymbol{y}^\perp = [y_1^\perp, y_2^\perp, \ldots, y_n^\perp]$ and check whether

$$\prod_{k=1}^{n} v_k^{y_k^\perp} = 1_{\mathbb{G}} \tag{4}$$

where $1_{\mathbb{G}}$ is the identity element of $\mathbb{G}$.

(2) Check whether dleq.Verify$(\pi_j, g_0, v_j, pk_j, c_j) = 1$ for all j.

(3) Output 1 if both checks pass, output 0 otherwise.

Let T be the set of valid tuples of the form $(\tilde{s}_i, \tilde{\pi}_i)$ where $\tilde{\pi}_i =$ dleq.Prove$(sk_i, g_1, pk_i, \tilde{s}_i, c_i)$ and $|T| = \ell + 1$, then

PVSS.Recon$(g_1, \{\tilde{s}_i\}_{i \in T}) \longrightarrow g_1^s$:

(1) Output

$$\prod_{i \in T} (\tilde{s}_i)^{\mu_i} = \prod_{i \in T} g_1^{\mu_i \cdot p(i)} = g_1^{p(0)} \tag{5}$$

where $\mu_i = \prod_{j \neq i} \frac{j}{j-i}$ for $i \in T$ are Lagrange coefficients.

Figure 5: Scrape's PVSS scheme.

use PVSS.Verify to validate the shares, and PVSS.Recon is used to recover g_1^s. We describe them in detail in Figure 5.

The verification procedure of Scrape's PVSS uses properties of error correcting code, specifically the Reed Solomon code [46]. They use the observation by McEliece and Sarwate [39] that sharing a secret s using a degree ℓ polynomial among n nodes is equivalent to encoding the message $(s, a_1, a_2, \cdots, a_t)$ using a $[n, \ell + 1, n - \ell]$ Reed Solomon code [46].

Let C be a $[n, k, d]$ linear error correcting code over $\mathbb{Z}_q$ of length n and minimum distance d. Also, let $C^\perp$ be the dual code of C i.e., $C^\perp$ consists vectors $\boldsymbol{y}^\perp \in \mathbb{Z}_q^n$ such that for all $\boldsymbol{x} \in C$, $\langle \boldsymbol{x}, \boldsymbol{y}^\perp \rangle = 0$. Here, $\langle \cdot, \cdot \rangle$ is the inner product operation. Scrape's PVSS.Verify uses the following basic fact (Lemma F.1) of linear error correcting code. We refer readers to [19, Lemma 1] for its proof.

Lemma F.1. *If $\boldsymbol{x} \in \mathbb{Z}_q^n \setminus C$, and $\boldsymbol{y}^\perp$ is chosen uniformly at random from $C^\perp$, then the probability that $\langle \boldsymbol{x}, \boldsymbol{y}^\perp \rangle = 1$ is exactly $1/q$.*

The PVSS scheme of Scrape guarantees the IND1-Secrecy property as defined in Definition F.2. Intuitively, for any (n, ℓ) PVSS scheme, IND1-secrecy ensures that prior to the reconstruction phase, the public information together with the secret keys sk_i of any set of at most ℓ players gives no information about the secret. Formally this is stated as in the following indistinguishability based definition adapted from [31, 47]:

Definition F.2. (IND1-Secrecy) A (n, ℓ) PVSS is said to be IND1-secret if for any probabilistic polynomial time adversary $\mathcal{A}$ corrupting at most ℓ parties, $\mathcal{A}$ has negligible advantage in the following game played against an challenger.

(1) The challenger runs the Setup phase of the PVSS as the dealer and sends all public information to $\mathcal{A}$. Moreover, it creates secret and public keys for all honest nodes, and sends the corresponding public keys to $\mathcal{A}$.

(2) $\mathcal{A}$ creates secret keys for the corrupted nodes and sends the corresponding public keys to the challenger.

(3) The challenger chooses values s_0 and s_1 at random in the space of secrets. Furthermore it chooses $b \longleftarrow \{0, 1\}$ uniformly at random. It runs the phase of the protocol with s_0 as secret. It sends $\mathcal{A}$ all public information generated in that phase, together with s_b.

The advantage of $\mathcal{A}$ is defined as $|\Pr[b = b'] - 1/2|$.

Theorem F.3. (IND1-Secrecy [19, Theorem 1]) *Under the Decisional Diffie-Hellman assumption, the PVSS protocol in [19] guarantees IND1-secrecy against a static probabilistic polynomial time adversary that can collude with up to ℓ nodes.*