Verifiable Secret Sharing Simplified

Sourav Das[1] Zhuolun Xiang[2] Alin Tomescu[2] Alexander Spiegelman[2] Benny Pinkas[2,3] Ling Ren[1]
[1]University of Illinois at Urbana-Champaign [2]Aptos Labs [1]Bar-Illan University
{souravd2,renling}@illinois.edu,{xiangzhuolun,tomescu.alin,sasha.spiegelman}@gmail.com,benny@pinkas.net

ABSTRACT

Verifiable Secret Sharing (VSS) is a fundamental building block in cryptography. Despite its importance and extensive studies, existing VSS protocols are often complex and inefficient. Many of them do not support dual threads, are not publicly verifiable, or do not properly terminate in asynchronous networks. This paper presents a new and simple approach for designing VSS protocols in synchronous and asynchronous networks. Our VSS protocols are optimally fault-tolerant, i.e., they tolerate a 1/2 and a 1/3 fraction of malicious nodes in synchronous and asynchronous networks, respectively. They only require a public key infrastructure and the hardness of discrete logarithms. Our protocols support dual thresholds, and their transcripts are publicly verifiable. We implement our VSS protocols and evaluate them in a geo-distributed setting with up to 256 nodes. The evaluation demonstrates that our protocols offer asynchronous termination and public verifiability with performance that is comparable to that of existing asynchronous VSS schemes that lack these features. Compared to the existing asynchronous VSS schemes with similar guarantees, our approach lowers the bandwidth usage and latency by up to 90%.

1 INTRODUCTION

A Verifiable Secret Sharing (VSS) scheme lets a party holding a secret, commonly referred to as a dealer, share the secret in a verifiable manner among a set of nodes where a fraction of the nodes, including the dealer, could be malicious [26, 34, 56]. The secret sharing process is verifiable in the sense that each node can verify the validity and correctness of its share. VSS is a fundamental building block for secure-multiparty computation (MPC) [8], threshold cryptography [61], Byzantine fault tolerant algorithms [16], distributed key generation (DKG) [37], randomness beacon [25], and so on.

Over the years, numerous works have studied VSS with different properties and in different settings, such as different cryptographic assumptions, network conditions, fault-tolerance, and so on [5, 11, 26, 32, 34, 37, 49, 50, 56, 62, 65–67]. In this paper, we focus on VSS protocols that use Shamir secret sharing [61], are secure against a computationally bounded adversary, and have optimal fault tolerance in both synchronous and asynchronous networks. We also seek to achieve several additional desirable properties of VSS, which we will briefly discuss next.

A desirable property of VSS protocols is *completeness* which ensures that every honest node receives its share of the secret. Applications such as DKG, MPC, and proactive secret sharing crucially rely on the completeness property.

Another desirable property of VSS, especially asynchronous VSS (AVSS), is the support for dual thresholds [15]. Briefly, in an asynchronous network of $n \geq 3t + 1$ nodes where at most t nodes are malicious, a dual-threshold AVSS scheme with parameter $\ell \in [t, n - t)$ guarantees secrecy against any coalition of up to ℓ nodes. Dual-threshold AVSS with $\ell = n-t-1$ is used to design high-threshold asynchronous DKG [33], which is in turn used to achieve better secrecy in threshold cryptosystems [63] and better efficiency in Byzantine fault tolerant (BFT) algorithms [16, 17, 64]. Dual-threshold VSS is also useful in designing optimal fault-tolerant BFT systems that rely on sampling for scalability, an approach that is getting wide adoption in recent proof-of-stake blockchains [6, 24, 38].

Finally, some randomness beacon [11, 30] and DKG protocols [41, 44, 48] also require the VSS transcript to be *publicly verifiable* by any external entity. A VSS scheme with a publicly verifiable transcript is also called a Publicly Verifiable Secret Sharing (PVSS) scheme.

In this paper, unless stated otherwise, we always consider VSS protocols with the completeness property, and primarily study VSS protocols that support dual thresholds in asynchrony and provide publicly verifiable transcripts.

Existing work. Despite years of efforts, there are no VSS schemes that satisfy all our requirements (see §2 for a detailed discussion). For example, the historically dominant approach of designing synchronous VSS protocols relies on interactive complaints [9, 11, 34, 36, 37, 47, 56]. This approach incurs high latency, is fairly complex, and is not publicly verifiable. Moreover, when extended to asynchronous networks, this approach suffers from a subtle termination issue [32, 43, 62, 65] (more details in §2) and does not support dual thresholds. Several recent asynchronous VSS designs deviate from the interactive complaint framework. But these schemes rely on trusted setups and bilinear pairing for efficiency [4, 5, 50, 67], and they also do not support dual thresholds or public verifiability. On the other hand, existing publicly verifiable VSS uses *verifiable encryption* schemes to let the dealer prove statements over encrypted data, making them expensive [33, 35, 41, 48] or suitable only for limited applications [21, 22, 60].

Our contributions. We present a new and simple approach for designing VSS protocols for synchronous and asynchronous networks. Our VSS protocols are optimally fault-tolerant, i.e., they tolerate 1/2 and 1/3 fractions of malicious nodes in synchronous and asynchronous networks, respectively. Our VSS protocols guarantee completeness and have efficient publicly verifiable transcripts. Our asynchronous protocol also guarantees asynchronous termination without relying on additional cryptographic setups or bilinear pairings and only assumes public key infrastructure.

Our VSS protocols achieve the above-mentioned properties while maintaining the same asymptotic communication and computation costs of best-known VSS protocols. More precisely, in a synchronous network with n nodes, our VSS protocol incurs a communication cost of $O(\kappa n^2 + C_{\text{BB}}(\kappa n))$. Here κ is a computational security parameter and $C_{\text{BB}}(x)$ is the communication cost of broadcasting a message of size x via a Byzantine broadcast channel. Our asynchronous VSS (AVSS) protocol incurs a communication cost of $O(\kappa n^2)$.

We then augment our AVSS to support *dual thresholds* for any secrecy threshold $\ell \in [t, n-t)$. Our augmented AVSS protocol maintains the total communication cost of $O(\kappa n^2)$ without relying on a trusted setup. Our dual-threshold AVSS protocol has the following nice properties: (i) The best-case performance with any ℓ is the same as our low-threshold AVSS, where the best-case is when the network is synchronous and the number of malicious nodes is less than $2t - \ell$; and, (ii) the worst-case performance degrades gradually with ℓ, and is the same as our low-threshold AVSS for $\ell = t$. In contrast, existing dual-threshold AVSS protocols [33, 42, 48] incur a high cost independently of ℓ, and their performance does not improve even under the best-case scenario.

Another useful property of our VSS scheme is that, if we use an external broadcast channel (e.g., in a blockchain setting), nodes only need to communicate with the dealer. This also means that assuming the presence of an external broadcast channel, the synchronous timing assumption needs to apply only between the dealer and other nodes. This assumption is less stringent than requiring bounded communication delays between all pairs of nodes. This property also makes the implementation simpler, as only the dealer needs to establish communication with the other nodes.

As an independent contribution, we design an efficient verifiable encryption scheme for Pedersen commitments. Existing verifiable encryption schemes are designed for the non-hiding Feldman commitment scheme and cannot be used to encrypt messages with low entropy [18, 35, 42, 48]. Our verifiable encryption scheme addresses this limitation and supports arbitrary message distribution and is thus more suitable for general applications, including VSS.

Evaluation. We implement our VSS protocol in Rust and evaluate it with up to 256 nodes in geographically distributed Amazon EC2 instances. Our evaluation illustrates that our AVSS protocol has performance that is comparable to that of the best known AVSS schemes [32, 65] while additionally achieving asynchronous termination and public verifiability. Compared to an existing VSS protocol with these properties [42], our VSS scheme has 5-11× better latency, and uses about 8× less bandwidth.

Paper organization. The rest of the paper is organized as follows. First, we review related work in more detail in §2. In §3, we formally define the problem of verifiable secret sharing (VSS) and provide an overview of our new approach. We describe the required preliminaries in §4. We then describe our synchronous VSS in §5, asynchronous VSS in §6, and dual-threshold asynchronous VSS protocol in §7. We then present our implementation and evaluation results in §8. Finally, we conclude with a discussion in §9.

2 RELATED WORK

VSS protocols consist of two phases: *Sharing* and *Reconstruction*. During the sharing phase, nodes along with the dealer run a protocol so that each node receives its share of the secret at the end of the sharing phase. In the reconstruction phase, nodes interact to recover the shared secret. We categorize existing VSS schemes into three approaches based on the design of their sharing phase. We describe each approach and outline its core idea, advantages, and disadvantages below.

Complaint-based VSS. Historically, the most common approach to designing VSS protocols is to rely on interactive complaints [9, 11, 34, 36, 37, 47, 56]. Briefly, in these protocols the dealer embeds the secret into a univariate low degree polynomial and publishes a commitment to the polynomial via a broadcast channel. The dealer additionally sends each node its share using a private channel. Upon receiving its share and the commitment, each node validates them for correctness. Nodes that receive no share or invalid shares from the dealer publish complaints against the dealer. The dealer responds to the complaints by revealing the share of each complaining node. Intuitively, these protocols rely on complaints to ensure completeness, i.e., prevent malicious dealers from sending valid shares to only a subset of the honest nodes.

While this approach provides reasonable efficiency in synchronous networks, they do not extend well to the more realistic partially synchronous and asynchronous networks. Asynchronous VSS (AVSS) protocols that rely on complaints to provide completeness [32, 43, 62, 65] suffer from a subtle termination issue (even without batching) that prevents honest nodes from terminating the protocol, even after outputting their shares. More concretely, these protocols have a step where, after outputting their share, honest nodes wait for either acknowledgments or complaints from all other nodes before terminating. This step is crucial because, in the case of complaints, nodes must assist the complaining nodes in recovering their shares. This allows malicious nodes to prevent honest nodes from terminating by simply not sending acknowledgments or complaints.

In addition, complaint-based VSS protocols have other limitations: they do not support dual thresholds and are not publicly verifiable.

Verifiable Encryption-based VSS. One approach to VSS design that addresses the above issues, is to use *verifiable encryption* (VE) (see Definition 4). Briefly, in a VE-based VSS scheme, the dealer locally generates a transcript that includes encryptions of the shares of all nodes, each under the public key of the corresponding node, along with a non-interactive zero-knowledge (NIZK) proof of the correctness of the encrypted shares. The dealer then publishes the transcript to all the nodes using a broadcast channel. Upon receiving the transcript over the broadcast channel, each node validates the correctness of all encrypted shares using the NIZK proof and recovers its own share by decrypting its encrypted share.

Existing VE-based schemes achieve several nice properties. First, they are non-interactive, i.e., only the dealer broadcasts a single message in the entire protocol. Second, they are also publicly verifiable. Third, the same protocol approach, with appropriate instantiations of the broadcast channel, works in both synchronous and asynchronous networks. However, VE-based protocols are generally inefficient or rely on non-standard assumptions, particularly due to their reliance on NIZK over encrypted data [23, 33, 35, 41, 42, 48]. Some works [21, 22, 60] bypass this efficiency issue by weakening the VSS functionality. More precisely, these schemes require the VSS secret to be an elliptic curve group element rather than an element in a field. Hence, they are not compatible with off-the-self threshold cryptosystems whose keys are field elements [14, 37].

Bivariate polynomial-based AVSS. A more recent approach to designing AVSS is to rely on a bivariate polynomial [4, 5, 50, 67]. In these schemes, the dealer embeds its secret as the constant term of a random low-degree bivariate polynomial. The dealer then publishes

a commitment to the bivariate polynomial using reliable broadcast. Additionally, the dealer privately sends partial evaluations of the polynomials to each node. Each node, upon receiving its partial evaluation, communicates with others to recover its share of the secret. Intuitively, the sharing phase terminates only when the dealer sends valid partial evaluations to a majority of the honest nodes. By sending valid partial evaluations to the majority of the honest nodes, the dealer provides these nodes with sufficient information to assist each other in recovering their shares.

Unlike complaint-based AVSS schemes, this approach guarantees asynchronous termination, i.e., a node can terminate the protocol after outputting its share. However, these approaches require the dealer to perform $O(n^2)$ group exponentiations. Moreover, these protocols require a trusted setup and strong cryptographic assumptions in the Algebraic Group Model for efficient communication. More precisely, Haven [5] and Bingo [4] assume hardness of q-SDH in a pairing-friendly group and require a powers-of-tau setup [49] to achieve $O(\kappa n^2)$ total communication. Without the setup, the state-of-the-art protocol Haven incurs $O(\kappa n^2 \log n)$ total communication cost and has $O(n^2)$ per-node computation cost. Lastly, these protocols are not publicly verifiable.

One approach to achieve public verifiability in the complaint-based and bivariate-based VSS schemes is to use additional rounds of (multi-)signatures. Concretely, once the VSS finishes, nodes send signatures to each other. Each node then waits to receive $n - t$ valid signatures, validate them, and add them as part of its transcript. Indeed, this will work. However, applying this technique to an existing scheme will inherit the drawbacks of that scheme, such as non-termination or worse efficiency. Our scheme is a simpler and more efficient way to achieve public verifiability.

Other related works. A number of works have studied VSS protocols with information-theoretic security [8, 19, 20, 27, 29, 40, 45, 54, 55], in both synchronous and asynchronous networks. However, these have high worst-case communication costs, only guarantee security with abort, or have sub-optimal fault tolerance. A series of works [7, 15, 32] study VSS protocols without completeness, and the latest among them achieve [32] a communication cost of $O(\kappa n^2)$ assuming collision resistance hash functions and hardness of discrete logarithm.

3 DEFINITIONS AND OVERVIEW

Let $\mathbb{G}$ be an elliptic curve group of order q with $\mathbb{F}$ as its scalar field. Let $g, h \in \mathbb{G}$ be two uniformly random and independent generators. We use κ to denote the security parameter. For example, when we use a signature scheme, κ denotes the size of the secret key. Similarly, we also use κ to denote the size of an element in $\mathbb{F}$ or $\mathbb{G}$. For any integer a, we use $[a]$ to denote the ordered set $\{1, 2, \ldots, a\}$. Also, for two integers a and b where $a < b$, we use $[a, b]$ to denote the ordered set $\{a, a+1, \ldots, b\}$.

3.1 Threat Model

We consider a network of n nodes denoted by $\{1, 2, \ldots, n\}$, where each node is connected with the dealer via a pairwise private and authenticated channel. We assume nodes have access to a broadcast channel that the dealer can use to send a value to all nodes. A broadcast channel ensures that the dealer cannot send inconsistent values to different nodes. We can efficiently realize such optimal fault-tolerant broadcast channels in synchronous and asynchronous networks by running a Byzantine broadcast [51, 53] and a reliable broadcast [13, 32], respectively. We will give their interfaces in Appendix A. When such external broadcast channels are unavailable, and nodes implement the broadcast channels themselves, as in our experiments (see §8), we assume that nodes are pairwise connected, i.e., form a complete graph.

We consider a *static* adversary $\mathcal{A}$ that can corrupt a threshold fraction of the nodes upfront. For our synchronous VSS protocol, we assume that $\mathcal{A}$ can corrupt less than half of the nodes, i.e., at most t out of $n \geq 2t + 1$ nodes. Also, let Δ be an upper bound on the delay between the honest dealer and any honest node. For our AVSS and dual-threshold AVSS protocols, we assume that for $n \geq 3t + 1$, at most t nodes are malicious. Each node i has its private signing key sk_i and the corresponding public verification key pk_i. We also assume a public key infrastructure (PKI), i.e., all nodes have access to $\{\mathsf{pk}_j\}_{j \in [n]}$.

3.2 Definition of Verifiable Secret Sharing

Definition 1 (Verifiable Secret Sharing)**.** A verifiable secret sharing (VSS) protocol consists of two phases: *Sharing* and *Reconstruction*. During the sharing phase, a dealer L shares a secret $s \in \mathbb{F}$. During the reconstruction phase, nodes interact to recover the secret. We say that a VSS protocol is t-resilient if the following properties hold with probability $1 - \mathsf{negl}(\kappa)$ against any probabilistic polynomial time (PPT) adversary $\mathcal{A}$ that corrupts up to t nodes:

- **Correctness.** If L is honest and has a secret s, then the sharing phase will result in all honest nodes eventually outputting a share of s. Once the sharing phase finishes, if all honest nodes start the reconstruction phase, they will output s.
- **Completeness:** If any honest node outputs in the sharing phase, then there exists a secret $\tilde{s} \in \mathbb{F}$ such that all honest nodes eventually output a share of $\tilde{s}$. Also, $\tilde{s}$ is guaranteed to be reconstructed during the reconstruction phase.
- **Secrecy.** If L is honest, there exists a PPT simulator $\mathcal{S}$ which interacts with an ideal functionality $\mathcal{F}_{\text{VSS}}$ and outputs a view of $\mathcal{A}$, such that the $\mathcal{A}$'s view in the real-world protocol and the simulated protocol are indistinguishable.
- **Termination.** All honest nodes will eventually terminate the Sharing phase.

We will define the functionality $\mathcal{F}_{\text{VSS}}$ and its variants when we analyze its Secrecy property.

VSS protocols in synchronous and asynchronous networks can tolerate up to 1/2 and 1/3 fractions of failures, respectively [3]. It is well known that the standard Termination property is impossible in asynchronous networks since it is impossible to tell apart a slow dealer from a malicious one. Thus, AVSS protocols instead guarantee the asynchronous termination property, similar to that of reliable broadcast [13].

- **Asynchronous termination.** If any honest node outputs in the sharing phase, then all honest nodes will eventually terminate the sharing phase.

Many applications of VSS additionally require the VSS scheme to be publicly verifiable, as defined below.

Definition 2 (Publicly verifiable)**.** A publicly verifiable secret sharing (PVSS) protocol outputs a transcript that enables any third party, not just the original nodes, to verify that the dealer has ensured each node receives its share.

Another desirable property of AVSS protocol is dual-threshold, as defined below.

Definition 3 (Dual-threshold AVSS)**.** A (n, ℓ, t) dual-threshold AVSS for $n \geq 3t+1$ is a t-resilient AVSS scheme where for any given $\ell \in [t, n-t)$, the secrecy of the secret holds against any coalition of up to ℓ nodes. We refer to ℓ as the *secrecy* threshold.

Remark. The dual-threshold guarantees achieved by some VSS and DKG protocols [4, 5, 31, 50] are weaker than Definition 3. Those schemes achieve a secrecy threshold of $\ell > t$ only after the protocol terminates. During the protocol execution, their secrecy threshold is t. In contrast, Definition 3 requires a secrecy threshold of ℓ even during the protocol execution.

3.3 Overview of Our Approach

Our starting point is the classical complaint-based synchronous VSS schemes described in §2. In those schemes, nodes publish complaints if they receive an invalid share or no share from the dealer. The dealer responds to complaints by publishing the shares of the complaining nodes. If the dealer fails to do so, it is considered malicious, and nodes output default values. This approach prevents a malicious dealer from violating completeness while still ensuring secrecy. This is because honest nodes will not complain against an honest dealer, thereby safeguarding the shares of honest nodes. Moreover, when the dealer is malicious, secrecy is vacuous.

Note from §2 that the conflict is always between achieving completeness and ensuring secrecy. Without secrecy, achieving completeness is trivial: the dealer simply broadcasts shares of everyone (or even the secret) to all. With this in mind, let us take another look at the complaint-based schemes. Here, the dealer reveals shares of a subset of parties, and the protocol ensures that an honest dealer only reveals shares of malicious nodes. Our approach achieves a similar property but uses a different approach, as we describe next.

The first crucial change we introduce is that, instead of sending explicit complaints, we only send explicit acknowledgments. The absence of an acknowledgment is in some way a complaint. Specifically, the dealer computes the shares of its secret using a low-degree polynomial, along with a commitment to the polynomial. The dealer, instead of publishing the commitment, first privately sends each node i the commitment along with the share of node i. Each node, upon receiving its share of the secret, validates it for correctness. Upon successful validation, the node responds to the dealer with a signed acknowledgment. This acknowledgment can serve as proof that node i has received its valid share corresponding to the commitment.

The dealer waits to receive an appropriate number of signed acknowledgments. (The dealer cannot wait for acknowledgments from all nodes because malicious nodes may never send acknowledgments.) Next is where our second crucial change comes in. The dealer then publishes, using a broadcast channel, the VSS transcript, which consists of the commitment to the polynomial, the signed acknowledgments it has received, and the shares of nodes who did not respond with a signed acknowledgment. Looking ahead, we will argue that despite the dealer publicly revealing shares of a subset of nodes, an adversary does not learn enough points on an honest dealer's polynomial, so secrecy is maintained.

Upon receiving the transcript over the broadcast channel, nodes validate it by checking that, for each node $i \in [n]$, either its signature or its share of the secret is included in the transcript. Upon successful validation, each node outputs the commitment and its share and terminates the sharing phase. If the validation fails, a node outputs a default value. Intuitively, completeness is satisfied because a node either explicitly acknowledges receiving its share or will receive its share from the validated transcript.

It is easy to see that the transcript the dealer broadcasts is publicly verifiable. The public verification check of the transcript is precisely the verification check each node performs on the transcript before terminating the sharing phase.

Based on these insights, designing a synchronous VSS protocol is straightforward. In a synchronous network of $n = 2t + 1$ nodes, with pair-wise latency Δ, the dealer shares its secret using a degree t polynomial. The dealer then waits for 2Δ time units to receive signed acknowledgments from all honest nodes and reveal the remaining shares using a broadcast channel.

However, this approach fails in asynchronous networks with $n = 3t + 1$. Under asynchrony, the dealer needs to make progress upon receiving $n - t = 2t + 1$ signed acknowledgments. Note that t of these $2t + 1$ acknowledgments could be from malicious parties. Now, if the dealer reveals the remaining t honest shares, it would reveal a total of $2t$ shares to $\mathcal{A}$, which is sufficient to reconstruct the degree t polynomial the dealer uses to share its secret. We address this issue by requiring the dealer to share its secret using a degree $2t$ polynomial. This prevents $\mathcal{A}$ from learning the secret even after learning $2t$ shares.

Finally, to construct a dual-threshold AVSS with secrecy threshold ℓ for $\ell \in [t, n-t)$, we combine ideas from verifiable encryption-based VSS with our low-threshold AVSS, i.e., AVSS with $\ell = t$. More precisely, for any ℓ, the dealer still uses a degree $2t$ polynomial to share its secret, but crucially does not reveal all remaining t shares after receiving $2t + 1$ signed acknowledgments. Instead, the dealer publicly reveals only $2t - \ell$ of the remaining t shares, encrypts, and broadcasts the remaining $t - (2t - \ell) = \ell - t$ shares using a verifiable encryption scheme. Intuitively, this ensures that any coalition of at most ℓ nodes learns at most $2t$ points on the polynomial. The protocol still ensures completeness because the nodes whose shares are not revealed by the dealer will receive their share from the verifiable encryptions revealed by the dealer. Since the leader broadcasts $\ell - t$ shares using verifiable encryption, the performance degrades gradually with ℓ. And if the leader receives more than $2t + 1$ signed acknowledgments (e.g., in the best case with a synchronous network and few malicious parties), the performance will further improve.

4 PRELIMINARIES

4.1 Threshold Secret Sharing

A $(n, d + 1)$ threshold secret sharing scheme allows a secret $s \in \mathbb{F}$ to be shared into n shares such that any set of $d + 1$ shares are sufficient to recover the original secret, but any set of d shares give no

information about the original secret [12, 61]. We use the common Shamir secret sharing [61] scheme, where the secret is embedded in a random degree d polynomial in the field $\mathbb{F}$. Specifically, to share a secret $s \in \mathbb{F}$, a polynomial $p(\cdot)$ of degree d is chosen such that $s = p(0)$ and other coefficients are chosen uniformly randomly from $\mathbb{F}$. The i-th share of the secret is then $p(i)$, i.e., the polynomial evaluated at i. Given $d+1$ points on the polynomial $p(\cdot)$, one can efficiently reconstruct the polynomial using Lagrange interpolation. Also note that s is information-theoretically hidden from an adversary that knows d or fewer evaluation points on the polynomial other than $p(0)$ [61].

4.2 Polynomial Commitment Scheme

The dealer in our VSS scheme commits to its secret by committing to a polynomial $p(\cdot)$ of degree d. A polynomial commitment scheme PC has the following interface.

- $\mathsf{PC.Setup}(1^\kappa) \rightarrow pp$. On input the security parameter κ, outputs the public parameters for the polynomial commitment scheme.
- $\mathsf{PC.Commit}(pp, p(\cdot), n) \rightarrow (\boldsymbol{v}, \boldsymbol{w})$. On input the public parameters pp, number of evaluations n, and the polynomial $p(\cdot)$, outputs the commitment $\boldsymbol{v}$ of the polynomial $p(\cdot)$ and witness $\boldsymbol{w}$.
- $\mathsf{PC.Open}(pp, \boldsymbol{w}, p(\cdot), i) \rightarrow (p(i), \pi)$. On input the index i and the polynomial $p(\cdot)$, outputs $p(i)$, and a valid opening proof π.
- $\mathsf{PC.DegCheck}(pp, \boldsymbol{v}, d) \rightarrow 0/1$. On input the polynomial commitment $\boldsymbol{v}$ and a degree d, outputs 1 if $\boldsymbol{v}$ is a commitment to a polynomial of degree at most d, and outputs 0 otherwise.
- $\mathsf{PC.Verify}(pp, \boldsymbol{v}, i, u, \pi) \rightarrow 0/1$. On input the polynomial commitment $\boldsymbol{v}$ to a polynomial $p(\cdot)$, outputs 1 if $u = p(i)$ and 0 otherwise.

Batch interfaces. As we briefly describe in §3.3, the dealer in our VSS protocols provides opening proofs for a batch of indices and each node verifies them locally. Thus, we use the batched interfaces PC.BatchOpen and PC.BatchVerify for better exposition. Briefly, PC.BatchOpen takes a set I of indices along with $(\boldsymbol{v}, \boldsymbol{w})$ and outputs $(\boldsymbol{s}, \boldsymbol{\pi})$. Here $\boldsymbol{s}$ is the vector of openings for each index in I, and $\boldsymbol{\pi}$ consists of corresponding opening proofs. Similarly, PC.BatchVerify takes as input a set I of indices along with $(\boldsymbol{s}, \boldsymbol{\pi})$, and outputs 1 if all the opening proofs are valid. We formally define these interfaces in Appendix A.3 and present mechanisms to verify a batch of polynomial evaluations more efficiently than verifying each evaluation independently.

A polynomial commitment scheme PC is secure if it satisfies the *Completeness, Evaluation binding*, and *Hiding* [49]. Intuitively, the completeness property ensures that verification of honestly generated commitments and opening proofs are always successful. The evaluation binding property prevents $\mathcal{A}$ from successfully opening to two different values at the same index. Lastly, the hiding property guarantees that the commitment $\boldsymbol{v}$ reveals no information about the polynomial.

In Appendix A.2, we describe a concrete polynomial commitment scheme that combines ideas from the classic Pedersen's polynomial commitment and SCRAPE's low-degree test [21].

5 SYNCHRONOUS VSS

Our synchronous VSS protocol is given in Algorithm 1. We assume $n = 2t+1$. The CRS $(\mathbb{G}, \mathbb{F}, g, h)$ is the output of the polynomial commitment's setup phase $\mathsf{PC.Setup}(1^\kappa)$. Let Δ be the upper bound on the delay between the honest dealer and any honest node. For any node $i \in [n]$, let $\mathsf{sk}_i, \mathsf{pk}_i$ be its private signing key and public verification key.

Algorithm 1 Synchronous VSS

PUBLIC PARAMETERS: $n \geq 2t+1$, $\{\mathsf{pk}_i\}_{i\in[n]}$, maximum network latency Δ, and public parameters $(\mathbb{G}, \mathbb{F}, g, h)$ of the polynomial commitment scheme.
PRIVATE INPUT: Signing key sk_i.

SHARING PHASE:

// Dealer L at time $\tau = 0$ and with input m:
Sample a t-degree random polynomial $s(\cdot)$ with $s(0) = m$
$\boldsymbol{v}, \boldsymbol{w} \leftarrow \mathsf{PC.Commit}(s(\cdot), n)$
for $i = 1, 2, \ldots, n$ **do**
  Let $\pi_i = \mathsf{PC.Open}(s(\cdot), i, \boldsymbol{w})$
  send $\langle \mathsf{SHARE}, \boldsymbol{v}, s(i), \pi_i \rangle$ to node i

// Each node i
upon receiving $\langle \mathsf{SHARE}, \boldsymbol{v}, s(i), \pi_i \rangle$ from dealer L **do**
  Check $\mathsf{PC.DegCheck}(\boldsymbol{v}, t) = 1$
  Check $\mathsf{PC.Verify}(\boldsymbol{v}, i, s(i), \pi_i) = 1$
  if both checks are succesful **then**
    Let $\sigma_i = \mathsf{sign}(\mathsf{sk}_i, \boldsymbol{v})$
    send $\langle \mathsf{ACK}, \sigma_i \rangle$ to L

// Dealer L at time $\tau = 2\Delta$
Let $\boldsymbol{\sigma}$ be the set of received valid signatures on $\boldsymbol{v}$.
Let I be the indices of nodes with *missing* signatures.
Let $\boldsymbol{s}, \boldsymbol{\pi} = \mathsf{PC.BatchOpen}(p(\cdot), I, \boldsymbol{w})$.
send $(\boldsymbol{v}, I, \boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{\pi})$ using the broadcast channel.

// Each node i once the broadcast outputs $(\boldsymbol{v}, I, \boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{\pi})$.
Check if each $\sigma \in \boldsymbol{\sigma}$ is valid and $|\boldsymbol{\sigma}| \geq t+1$.
Check if $\mathsf{PC.BatchVerify}(\boldsymbol{v}, I, \boldsymbol{s}, \boldsymbol{\pi})$.
Check that I includes all nodes with missing signatures.
if all the checks pass **then**
  Output $(\boldsymbol{v}, s(i), \pi_i)$; **return**
else
  Output 0 as the default share; **return**

RECONSTRUCTION PHASE:

// every node i after finishing the sharing phase
send $\langle \mathsf{RECON}, s(i), \pi_i \rangle$ to all.
upon receiving $\langle \mathsf{RECON}, s(j), \pi_j \rangle$ from node j **do**
  if $\mathsf{PC.Verify}(\boldsymbol{v}, s(j), \pi_j)$ **then**
    $T = T \cup \{s_j\}$
    if $|T| \geq t+1$ **then**
      output $s(0)$ using Lagrange interpolation; **return**

5.1 Design

Sharing phase. Let $m \in \mathbb{F}$ be the message the dealer L wants to share. L samples a degree-t polynomial

$$s(x) = m + s_1 x + s_2 x^2 + \cdots + s_t x^t \tag{1}$$

with uniformly random $s_i \in \mathbb{F}$ for each $i \in [n]$. L then computes the commitment of $s(\cdot)$ along with the commitment witness as $\boldsymbol{v}, \boldsymbol{w} \leftarrow \mathsf{PC.Commit}(s(\cdot), n)$.

At time $\tau = 0$, L computes the opening proof $\pi_i = \mathsf{PC.Open}(s(\cdot), i, \boldsymbol{w})$ for each $i \in [n]$ and sends the tuple $\langle \mathsf{SHARE}, \boldsymbol{v}, s(i), \pi_i \rangle$ to node i. Node i, upon receiving the SHARE message from L, validates that $\boldsymbol{v}$ is a polynomial of degree t by checking that $\mathsf{PC.DegCheck}(\boldsymbol{v}, t) = 1$, and checks that its share is valid using $\mathsf{PC.Verify}(\boldsymbol{v}, i, s(i), \pi_i)$. If both

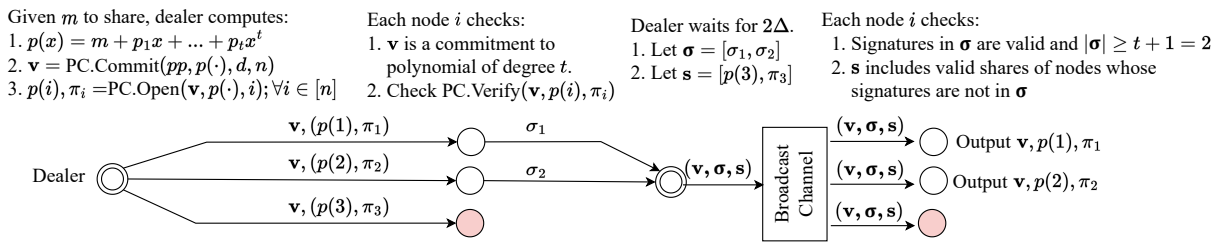


Figure 1: Our synchronous VSS protocol involves three nodes, one of which is malicious (shaded red in the diagram).

these checks are successful, node i sends a message $\langle \mathsf{ACK}, \sigma_i \rangle$ where σ_i is its signature on $\boldsymbol{v}$.

L waits for 2Δ units of time to collect ACK messages. Here, for ease of exposition, we assume Δ has accounted for the time required to validate $\boldsymbol{v}$ and check the validity of a share. At time $\tau = 2\Delta$, let $\boldsymbol{\sigma}$ be the set of valid signatures L receives and let I be the set of nodes from whom L does not receive valid signatures. L then computes $(\boldsymbol{s}, \boldsymbol{\pi})$ = PC.BatchOpen$(s(\cdot), I, \boldsymbol{w})$ where $\boldsymbol{s}$ is of size $|I|$ and consists of $s(k)$ for each $k \in I$, and $\boldsymbol{\pi}$ is the opening proof. Then, L sends the message $\langle \boldsymbol{v}, I, \boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{\pi} \rangle$ using the broadcast channel.

When the broadcast channel outputs $\langle \boldsymbol{v}, I, \boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{\pi} \rangle$, each node locally checks that: (i) $\boldsymbol{\sigma}$ is a valid set of signatures on $\boldsymbol{v}$ and $|\boldsymbol{\sigma}| \geq t+1$; (ii) I includes all nodes whose signatures are not included in $\boldsymbol{\sigma}$; and (iii) $\boldsymbol{s}$ includes valid shares of nodes in I with respect to $\boldsymbol{v}$, i.e., PC.BatchVerify$(\boldsymbol{v}, I, \boldsymbol{s}, \boldsymbol{\pi})$. If all these checks are successful, node i outputs its share $s(i)$, the commitment $\boldsymbol{v}$, and the opening proof π_i. A node gets these from either the broadcast message or the SHARE message it received from the dealer.

Using multisignatures. One simple concrete optimization is to have each node sign its ACK message using a multisignature scheme. More precisely, the ACK message from node i includes its partial signature on $\boldsymbol{v}$. L then broadcast the multisignature σ on $\boldsymbol{v}$ instead of broadcasting a list of signatures.

Reconstruction phase. Let T be a set of $t + 1$ nodes (including itself) from which node i receives valid shares $s(j)$. Upon receiving $t+1$ such valid shares, node i computes the secret m using Lagrange interpolation as $m = \sum_{k \in T} \mu_k s(k)$, where $\mu_k = \prod_{j \neq k} \frac{j}{j-k}$ are the Lagrange coefficients.

Optimized reconstruction. In certain situations, it is possible to optimize the reconstruction phase. A node may not need to always wait for $t + 1$ RECON messages. If, during the sharing phase, the dealer has already revealed k shares as part of $\boldsymbol{s}$, a node only needs to wait for $t + 1 - k$ RECON messages for shares not included in $\boldsymbol{s}$.

5.2 Analysis

Correctness. An honest dealer L will receive signed ACK messages from all honest nodes within 2Δ time under synchrony. Since there are at least $t + 1$ honest nodes, $|\boldsymbol{\sigma}| \geq t + 1$. Let $\langle \boldsymbol{v}, I, \boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{\pi} \rangle$ be the transcript broadcast by L. By the Validity property of Byzantine agreement, each honest node will output $\langle \boldsymbol{v}, I, \boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{\pi} \rangle$. Then, by the Correctness property of the signature scheme and the Completeness property of the polynomial commitment scheme, every honest node will accept the VSS transcript and output its share.

Finally, during the reconstruction protocol, each honest node will multicast a valid RECON message. Thus, every honest node will receive at least $t+1$ valid shares, which is sufficient to reconstruct the degree t polynomial $s(\cdot)$, and hence $s(0)$. Moreover, the Evaluation binding of the polynomial commitment ensures that honest nodes only accept valid shares on the committed polynomial. This implies that all honest nodes output the same unique secret $s(0)$.

Termination. Follows directly from the Termination property of the Byzantine broadcast scheme (cf. Definition 5).

Completeness. An honest party outputs its share only upon receiving a valid transcript $(\boldsymbol{v}, I, \boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{\pi})$ over the Byzantine broadcast channel. The Agreement property of the Byzantine broadcast guarantees that every honest node outputs the same transcript, and hence the same polynomial commitment. Successful validation of the transcript implies that at least $t + 1$ node, hence at least one honest node, signed the commitment. This implies with $1 - \mathsf{negl}(\kappa)$ probability, $\boldsymbol{v}$ is a commitment to a polynomial of degree at most t. Also, for each node $i \in [n]$, either a signature of i or its valid share is included in $\boldsymbol{\sigma}$. In the former case, assuming the existential unforgeability of the signature scheme, node i already received its share. In the latter case, node i will receive its valid share from $\boldsymbol{s}$.

During the reconstruction phase, each honest node will reconstruct the degree t polynomial $p(\cdot)$ corresponding to the commitment $\boldsymbol{v}$. Hence, each node will output the unique secret $s(0)$.

Secrecy. We prove Secrecy using *simulatability*: for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ that corrupts up to t nodes, there exists an ideal world PPT simulator $\mathcal{S}_{\text{VSS}}$ that interacts with the ideal functionality $\mathcal{F}_{\text{VSS}}$ (cf. Figure 9) and produces a view such that $\mathcal{A}$'s view in the simulated world is identical to a run of the Sharing phase. We formally prove Secrecy in Appendix C.

Performance. We will analyze our performance using Figure 7 as the polynomial commitment scheme. The dealer performs $O(n \log n)$ field operations to compute shares of each node (using FFT). The dealer then performs $O(n)$ group exponentiations to compute the commitments and $O(n)$ signature verifications. Since group exponentiation is more expensive than $\log n$ field operations, we treat the dealer's computation cost as $O(n)$ group exponentiations. The running time of each node is as follows. Each node performs $O(n)$ group exponentiations to verify the polynomial commitment, signatures of $O(n)$ nodes, and shares of $O(n)$ nodes. Finally, the dealer privately sends an $O(\kappa n)$-bit commitment to each node and broadcasts an $O(\kappa n)$-bit transcript. Hence, the total communication cost of our VSS protocol is $O(\kappa n^2 + C_{BB}(\kappa n))$ where $C_{BB}(a)$ is the communication cost of broadcasting a message of length a.

Combining all the above, we get the following theorem.

Theorem 1 (Synchronous VSS). *In a synchronous network of $n \geq 2t + 1$ nodes among which at most t nodes are malicious, assuming a polynomial commitment scheme, a signature scheme, and a Byzantine broadcast channel, Algorithm 1 implements a t-resilient publicly*

Algorithm 2 Asynchronous VSS

PUBLIC PARAMETERS: $n \geq 3t+1$, $\{\mathsf{pk}_i\}_{i\in[n]}$, and public parameters of the polynomial commitment scheme pp.
PRIVATE INPUT: Signing key sk_i.

SHARING PHASE:

// Dealer L with input m:

Sample a $2t$-degree random polynomial $s(\cdot)$ with $s(0) = m$
$\boldsymbol{v}, \boldsymbol{w} \leftarrow \mathsf{PC.Commit}(s(\cdot), n)$
for $i = 1, 2, \ldots, n$ **do**
  Let $\pi_i \leftarrow \mathsf{PC.Open}(s(\cdot), i, \boldsymbol{w})$
  send $\langle \mathsf{SHARE}, \boldsymbol{v}, s(i), \pi_i \rangle$ to node i

// Each node i

upon receiving $\langle \mathsf{SHARE}, \boldsymbol{v}, s(i), \pi_i \rangle$ from dealer L **do**
  Check $\mathsf{PC.DegCheck}(\boldsymbol{v}, 2t) = 1$
  Check $\mathsf{PC.Verify}(\boldsymbol{v}, i, s(i), \pi_i) = 1$
  if both the checks pass **then**
    Let $\sigma_i = \mathsf{sign}(\mathsf{sk}_i, \boldsymbol{v})$
    send $\langle \mathsf{ACK}, \sigma_i \rangle$ to L

// Dealer L waits for $2t+1$ valid signatures on $\boldsymbol{v}$

Let $\boldsymbol{\sigma}$ be the set of valid signatures on $\boldsymbol{v}$.
Let I be the indices of nodes with missing signatures.
Let $\boldsymbol{s}, \boldsymbol{\pi} = \mathsf{PC.BatchOpen}(p(\cdot), I, \boldsymbol{w})$.
send $(\boldsymbol{v}, I, \boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{\pi})$ using a reliable broadcast channel.

// Each node i once the broadcast outputs $(\boldsymbol{v}, I, \boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{\pi})$.

Check if each $\sigma \in \boldsymbol{\sigma}$ is valid and $|\boldsymbol{\sigma}| \geq 2t+1$.
Check that I includes all nodes with missing signatures.
Check if $\mathsf{PC.BatchVerify}(\boldsymbol{v}, I, \boldsymbol{s}, \boldsymbol{\pi})$.
if all the checks pass **then**
  Output $(\boldsymbol{v}, s(i), \pi_i)$; **return**

RECONSTRUCTION PHASE:

// every node i after finishing the sharing phase

send $\langle \mathsf{RECON}, s(i), \pi_i \rangle$ to all.
upon receiving $\langle \mathsf{RECON}, s(j), \pi_j \rangle$ from node j **do**
  if $\mathsf{PC.Verify}(\boldsymbol{v}, s(j), \pi_j)$ **then**
    $T = T \cup \{s_j\}$
    if $|T| \geq 2t+1$ **then**
      output $s(0)$ using Lagrange interpolation; **return**

verifiable VSS protocol with $O(\kappa n^2 + C_{BB}(\kappa n))$ communication cost. Here κ is the security parameter, and $C_{BB}(a)$ is the communication cost of broadcasting a message of length a using the broadcast channel.

6 ASYNCHRONOUS VSS

In this section, we will describe the modifications to make our protocol in an asynchronous or partial synchronous network. As we mention in §3, we seek to design an AVSS with the Completeness property. Since AVSS with completeness implies an asynchronous reliable broadcast (RBC), $n/3$ is the maximum number of failures any AVSS protocol can tolerate [13]. Throughout this section, we will assume $n = 3t+1$.

6.1 Design

The natural attempt to adapt the synchronous VSS in an asynchronous network of $n = 3t+1$ is to let the dealer share its secret using a degree t polynomial and keep the rest of the protocol as is. However, as we briefly mention in §3.3, this approach will not work. In asynchrony, there is no fixed upper bound on the message delays, so the dealer cannot wait to receive acknowledgments from all honest nodes. Instead, the dealer must move on upon receiving only $n-t = 2t+1$ signed acknowledgments. But t of these $n-t$ signed acknowledgments could be from malicious nodes, and the missing t acknowledgments correspond to honest but slow nodes. In this case, an honest dealer would reveal to $\mathcal{A}$ a total of $2t$ shares on a degree t polynomial, which is sufficient for $\mathcal{A}$ to recover the secret.

We address this issue with the following key observation: We let the dealer share the secret using a degree $2t$ polynomial (instead of degree t). The rest of the protocol, given in Algorithm 2, follows a similar structure, with a few natural changes highlighted in gray compared to Algorithm 1. The dealer waits for $n-t$ valid signed acknowledgments instead of a pre-specified time bound, and publishes the t shares from the t slow nodes. Intuitively, by using a degree $2t$ polynomial, we ensure that $\mathcal{A}$ does not learn the secret even after learning $2t$ shares. Finally, using a degree $2t$ polynomial does not affect the reconstructability of the secret as $n-t \geq 2t$, i.e., there are enough honest nodes to reconstruct the secret.

We want to note that although the dealer in Algorithm 2 shares its secret using a degree $2t$ polynomial, the protocol is not dual-threshold. This is because $\mathcal{A}$ learns up to $2t$ points on the polynomial by corrupting only t nodes.

Reducing the storage costs. In the AVSS scheme in Algorithm 2, each node stores the entire $\boldsymbol{v}$, which is $O(\kappa n)$ for Pedersen polynomial commitment. We can reduce the storage cost to $O(\kappa)$, using error-correcting code [58] and online error correction [20], similar to AVSS protocols such as [32, 65]. More specifically, each node encodes $\boldsymbol{v}$ using a $[n, t, n-t]$ Reed-Solomon code. Let $\hat{\boldsymbol{v}}$ be the encoded commitment. Each node i then stores $\hat{\boldsymbol{v}}[i]$ and deletes the rest of $\hat{\boldsymbol{v}}$. During the reconstruction phase, each node i sends $\langle \mathsf{RECON}, \hat{\boldsymbol{v}}[i], s(i), \pi_i \rangle$ to all. Upon receiving RECON messages, nodes first recover $\boldsymbol{v}$ using online error correction and then reconstruct the polynomial.

6.2 Analysis

Correctness. Since $n-t \geq 2t+1$, an honest dealer L will eventually receive $2t+1$ signed acknowledgments. Then, using a similar argument as our synchronous VSS, each honest node will eventually output and accept the transcript broadcast by the honest dealer. Similarly, during the reconstruction phase, each node will eventually receive $2t+1$ valid shares, which is sufficient to reconstruct the degree $2t$ polynomial $p(\cdot)$, and hence $s(0)$. Also, honest nodes will accept only valid shares and hence will output the same unique secret shared by the dealer.

Asynchronous Termination. Follows directly from the Totality property of the Byzantine RBC (cf. Definition 6).

Completeness. Follows using a similar argument as the synchronous VSS protocol.

Secrecy. We will prove the Secrecy in Appendix C.

Performance. The computation cost of the dealer and nodes are similar to that of the synchronous VSS protocol, except the dealer uses a degree $2t$ degree polynomial to share its secret. Precisely,

both the dealer and nodes need to perform $O(n)$ group exponentiations. In terms of the bandwidth cost, the dealer sends $O(\kappa n)$ length private message to each node and $O(\kappa n)$ bit long message using a broadcast channel. Thus, using the broadcast channel from [32], the total communication cost is $O(\kappa n^2)$. Finally, our AVSS requires two additional rounds of communication compared to the VE-based approach and the optimistic path of the state-of-the-art complaint-based AVSS scheme. As we will illustrate in §8.2, these additional rounds of communication are not a bottleneck.

Combining all the above, we get the following theorem.

Theorem 2 (Asynchronous VSS)**.** *In an asynchronous network of $n \geq 3t + 1$ nodes among which at most t nodes are malicious, assuming a polynomial commitment scheme, a signature scheme, and a Byzantine reliable broadcast channel, Algorithm 2 implements a t-resilient publicly verifiable asynchronous VSS protocol with $O(\kappa n^2)$ communication costs. Here κ is the security parameter.*

7 DUAL-THRESHOLD AVSS

In this section, we use our approach to design an (n, ℓ, t) dual-threshold AVSS scheme.

Protocol intuition. For any given ℓ, the dealer in our dual-threshold AVSS shares its secret using a degree $2t$ polynomial and follows the AVSS protocol until it receives $2t + 1$ signed acknowledgments. Then, unlike the AVSS scheme, the dealer does not reveal all remaining t shares. Instead, the dealer publicly reveals only $2t - \ell$ of the remaining t shares and shares the remaining $t-(2t-\ell) = \ell-t$ shares using a verifiable encryption scheme. More precisely, for each of the remaining $t - (2t - \ell) = \ell - t$ shares, the dealer encrypts it with the public key of the corresponding recipient node and computes a NIZK proof of its correctness. Intuitively, by publicly revealing only $2t - \ell$ shares, we ensure that any coalition of ℓ nodes learns at most $2t$ points on the polynomial. The protocol still ensures Completeness, as the nodes whose shares are not revealed by the dealer will receive their share from the verifiable encryptions.

7.1 Verifiable Encryption of Committed Messages

Our dual-threshold AVSS scheme relies on verifiable encryptions for the Pedersen commitment scheme, as defined below.

Definition 4 (Verifiable Encryption of a Committed Message)**.** Verifiable encryption (VE) of a committed message involves three parties: a prover $\mathcal{P}$, a verifier $\mathcal{V}$, and a receiver $\mathcal{R}$. The receiver $\mathcal{R}$ has a public-private key pair (pk, sk). Let Cm be a commitment scheme. Given (v, c, pk), $\mathcal{P}$ wants to convince $\mathcal{V}$ that c is a public key encryption of a message s under public key pk, and that v is an commitment to s and $\mathcal{P}$ knows s. A verifiable encryption scheme provides the following interfaces.

- $\mathsf{VE.Setup}(1^\kappa, \mathsf{Cm}) \rightarrow pp_{\mathrm{VE}}$. On input the security parameter κ, and the commitment scheme Cm, the algorithm outputs the public parameters pp_{VE}.
- $\mathsf{VE.KeyGen}(pp_{\mathrm{VE}}) \rightarrow (\mathsf{pk}, \mathsf{sk})$. The algorithm outputs a public-private key pair for the encryption scheme.
- $\mathsf{VE.EncProve}(pp_{\mathrm{VE}}, \mathsf{pk}, s, v, w) \rightarrow (c, \pi_{\mathrm{VE}})$: The algorithm takes as input the message s, commitment v with witness w, where $v, w \leftarrow \mathsf{Cm.Commit}(s)$. It outputs an encryption c of the tuple $(s, \pi = \mathsf{Cm.Open}(v, s, w))$ along with a NIZK proof π_{VE} of their correct encryptions.
- $\mathsf{VE.Verify}(pp_{\mathrm{VE}}, \mathsf{pk}, v, c, \pi_{\mathrm{VE}}) \rightarrow 0/1$. The algorithm outputs 1, if π_{VE} is a valid proof that there exists α, π such that $\alpha, \pi = \mathsf{VE.Dec}(\mathsf{sk}, c)$ and $\mathsf{Cm.Verify}(v, \alpha, \pi) = 1$. Note that π_{VE} needs to be verifiable without access to the secret key or the underlying message α.
- $\mathsf{VE.Dec}(\mathsf{sk}, c) \rightarrow s, \pi$: Given the ciphertext c and a secret key sk, the algorithm outputs a decryption of c using sk.

A verifiable encryption scheme is secure if it satisfies the standard *Completeness, Soundness*, and *Zero-knowledge* properties of verifiable computation schemes [39]. Intuitively, the Completeness property ensures that verification of an honestly generated π_{VE} is always successful, even if a malicious node generates the public key. The soundness property prevents a malicious prover from convincing an honest node about the correctness of an incorrectly generated ciphertext. Stating differently, if VE.Verify is successful for a ciphertext c and public key pk, then a node with secret key sk will always be able to recover its share and the opening proof. Lastly, the Zero-knowledge property guarantees that the ciphertext c and the proof π_{VE} reveal no information about the share other than whatever is revealed by the polynomial commitment scheme.

Batch verifiable encryptions. Looking ahead, the dealer in our dual-threshold VSS computes the verifiable encryptions for a batch of shares. Thus, we define the VE scheme to additionally support batched interfaces VE.BatchEncProve and VE.BatchVerify. Trivially, every VE can be modified to support VE.BatchEncProve and VE.BatchVerify by internally invoking the VE.EncProve and VE.Verify for each index in the batch, respectively. We define these additional interfaces is to support the design of batch encryption and verification that are more efficient than the trivial approach.

- $\mathsf{VE.BatchEncProve}(pp_{\mathrm{VE}}, I, \mathsf{pk}_I, \boldsymbol{s}, \boldsymbol{v}, \boldsymbol{w}) \rightarrow (\boldsymbol{c}, \pi_{\mathrm{VE}})$. On input a vector $\boldsymbol{s}$ of messages, their commitments $\boldsymbol{v}$, corresponding witness $\boldsymbol{w}$, the algorithm outputs encryptions $\boldsymbol{c}$ for each $s \in \boldsymbol{s}$, along with a NIZK proof π_{VE} that satisfy VE.BatchVerify.
- $\mathsf{VE.BatchVerify}(pp_{\mathrm{VE}}, I, \mathsf{pk}_I, \boldsymbol{v}, \boldsymbol{c}, \pi_{\mathrm{VE}}) \rightarrow 0/1$. The algorithm outputs 1 if π_{VE} is a valid proof that, for each $i \in I$ there exists (α_i, π_i) such that $\alpha_i, \pi_i = \mathsf{VE.Dec}(\mathsf{sk}_i, \boldsymbol{c}_i)$ and $\mathsf{PC.Verify}(\boldsymbol{v}, \alpha_i, \pi_i) = 1$

Constructions. Only a few VE schemes are known for discrete logarithm-based commitment schemes [18, 35, 42, 48]. These VE schemes are designed to work with the Feldman commitment scheme, where the dealer commits to a secret s as g^s. Note that the Feldman commitment scheme is not hiding. For instance, if the secret has low entropy, an adversary can recover the committed message by running a brute-force search on possible messages. As a result, these VE schemes cannot be directly used in general VSS schemes with arbitrary message distributions. Indeed, these VE schemes were designed for VSS schemes for Distributed Key Generation (DKG) protocols [33, 42, 48], where the shared secret is a random element from a large field.

Our dual-threshold AVSS requires a VE scheme for the Pedersen commitment scheme, where commitments are $g^s h^r$. To our knowledge, no such VE scheme has been described. We present

Algorithm 3 Dual-threshold AVSS

PUBLIC PARAMETERS: $n \geq 3t+1$, $\ell \geq t$, $\{pk_i\}_{i\in[n]}$, polynomial commitment PC and verifiable encryption VE.
PRIVATE INPUT: Signing key sk_i.

SHARING PHASE:

// Line 101 to 111 same as Algorithm 2

// Dealer L waits for $2t+1$ valid signatures

112: Let $\boldsymbol{\sigma}$ be the set of valid signatures on $\boldsymbol{v}$.
113: Let I be the indices of nodes with *missing* valid signatures.
114: Partition I into subsets I_R and I_{VE} with $|I_R| = 2t - \ell$
115: $\boldsymbol{s}, \boldsymbol{\pi} \leftarrow \mathsf{PC.BatchOpen}(\boldsymbol{v}, I_R, p(\cdot), \boldsymbol{w})$
116: Let $\boldsymbol{s}_{I_{\mathsf{VE}}} \leftarrow \{p(i)\}$ for all $i \in I_{\mathsf{VE}}$.
117: $\boldsymbol{c}, \pi_{\mathsf{VE}} \leftarrow \mathsf{VE.BatchEncProve}(I_{\mathsf{VE}}, \mathsf{pk}_{I_{\mathsf{VE}}}, \boldsymbol{s}_{I_{\mathsf{VE}}}, \boldsymbol{v}_{I_{\mathsf{VE}}}, \boldsymbol{w}_{I_{\mathsf{VE}}})$
118: **send** $(\boldsymbol{v}, I_R, I_{\mathsf{VE}}, \boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{\pi}, \boldsymbol{c}, \pi_{\mathsf{VE}})$ using a reliable broadcast.

// Node i upon broadcast outputs $(\boldsymbol{v}, I_R, I_{\mathsf{VE}}, \boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{\pi}, \boldsymbol{c}, \pi_{\mathsf{VE}})$.

119: Check if each $\sigma \in \boldsymbol{\sigma}$ is valid and $|\boldsymbol{\sigma}| \geq 2t+1$.
120: Check $I_R \cup I_{\mathsf{VE}}$ includes all nodes with missing signatures.
121: Check if $\mathsf{PC.BatchVerify}(\boldsymbol{v}, I_R, \boldsymbol{s}, \boldsymbol{\pi})$.
122: Check if $\mathsf{VE.BatchVerify}(I_{\mathsf{VE}}, \mathsf{pk}_{I_{\mathsf{VE}}}, \boldsymbol{v}, \boldsymbol{c}, \pi_{\mathsf{VE}})$.
123: **if** all the checks pass **then**
124: **if** received no valid SHARE message and $(p(i), \pi_i) \notin \boldsymbol{s}$ **then**
125: Let $p(i), \pi_i \leftarrow \mathsf{VE.Dec}(\boldsymbol{c}[i], sk_i)$
126: **output** $(\boldsymbol{v}, p(i), \pi_i)$; **return**

RECONSTRUCTION PHASE: *// Identical to Algorithm 2*

modifications to Groth's VE [42] to make it compatible with the Pedersen commitment scheme in Appendix B.

Remark. If our dual-threshold VSS scheme is used to share secrets with high entropy, we can also employ existing VE schemes, such as those mentioned in [18, 35, 42, 48].

7.2 Dual-threshold AVSS Design

Let L be the dealer of the (n, ℓ, t) dual-threshold AVSS scheme (cf. Definition 3). Let PC and VE be the polynomial commitment and verifiable encryption scheme, respectively. We summarize our scheme in Algorithm 3 where we highlight the changes with respect to Algorithm 2 in gray.

Sharing phase. The first part of the Sharing phase is the same as the AVSS protocol in Algorithm 2. L shares its secret using a degree $2t$ polynomial $p(\cdot)$, computes its commitment $\boldsymbol{v}, \boldsymbol{w} \leftarrow \mathsf{PC.Commit}(p(\cdot), n)$, and then sends $\langle \mathsf{SHARE}, \boldsymbol{v}, p(i) \rangle$ to each node. Each node i upon receiving the SHARE message, validates it as in Algorithm 2, computes $\sigma_i = \mathsf{sign}(\mathsf{sk}_i, \boldsymbol{v})$, and responds to L with $\langle \mathsf{ACK}, \sigma_i \rangle$.

L waits for $2t+1$ valid signed acknowledgements. Let $\boldsymbol{\sigma}$ be the set of valid acknowledgments, and let $I \subset [n]$ be the set of nodes from whom L does not receive ACK messages. Note that these include nodes who sent invalid ACK messages as well as nodes whose messages have not arrived. Next, L arbitrarily partitions I into two disjoint subsets I_R and I_N, such that $|I_R| = 2t - \ell$ and $|I_{\mathsf{VE}}| = \ell - t$. L then computes $\boldsymbol{s}, \boldsymbol{\pi} \leftarrow \mathsf{PC.BatchOpen}(p(\cdot), I_R, \boldsymbol{w})$ and $\boldsymbol{c}, \pi_{\mathsf{VE}} \leftarrow \mathsf{VE.BatchEncProve}(p(\cdot), \boldsymbol{v}, I_{\mathsf{VE}})$.

L then reliably broadcast the dual-threshold AVSS transcript $(\boldsymbol{v}, I_R, I_{\mathsf{VE}}, \boldsymbol{\sigma}, \boldsymbol{s}, \boldsymbol{\pi}, \boldsymbol{c}, \pi_{\mathsf{VE}})$ to all nodes. Upon receiving the transcript, nodes validate it by checking that: (i) $\boldsymbol{\sigma}$ is a valid set of signatures on $\boldsymbol{v}$ and $|\boldsymbol{\sigma}| \geq 2t+1$; (ii) $I_R \cup I_{\mathsf{VE}}$ includes all nodes whose signatures are not included $\boldsymbol{\sigma}$; (iii) $\boldsymbol{s}$ includes of valid shares of nodes in I_R with respect to $\boldsymbol{v}$ i.e., $\mathsf{PC.BatchVerify}(\boldsymbol{v}, I_R, \boldsymbol{s}, \boldsymbol{\pi})$; (iv) $\boldsymbol{c}$ includes verifiable ciphertexts using VE.BatchVerify.

Upon successful verification, each node i locally outputs the commitment $\boldsymbol{v}$, its share $p(i)$, along with the commitment opening proof π_i to be used during the reconstruction phase. Node i either receives $p(i), \pi_i$ from SHARE message, or computes $p(i), \pi_i \leftarrow \mathsf{VE.Dec}(\boldsymbol{c}[i], \mathsf{sk}_i)$.

Reconstruction phase. The reconstruction phase is identical to the reconstruction phase of our AVSS scheme.

7.3 Optimization for Common Case Execution

In the dual-threshold AVSS we have described so far, the dealer L always verifiably encrypts $\ell - t$ of the remaining shares, which can be expensive for both L and other nodes. The following optimizations can significantly lower the number of shares L needs to encrypt in the common case: when the number of active failures is low and the network between L and most honest nodes is synchronous.

In the optimized design, in addition to waiting for $2t+1$ signed acknowledgments, the dealer L also waits for the network latency 2Δ, whichever occurs later. Let $2t+1+k$ for $k \geq 0$ be the number of signed acknowledgments the dealer receives. L then verifiably encrypts shares of $\max\{0, \ell - (t+k)\}$ nodes. This implies that with more signed acknowledgments, L needs to verifiably encrypt fewer shares. In the best-case scenario, i.e., when L receives $\ell - t$ additional signed acknowledgments, it need not compute any verifiable encryptions. Thus, in the best case, we get the dual-threshold property for free.

Remark. The optimization we describe above is also applicable to the AVSS scheme in §6. Also, the storage cost optimization we describe in §6.1 also applies to our dual-threshold AVSS scheme.

7.4 Analysis

Correctness and Asynchronous termination. Follows from similar arguments as the AVSS protocol.

Completeness. The soundness guarantees of the VE scheme ensure that nodes whose signature or share is not included in the VSS transcript will still receive its valid share upon decryption. This, combined with an argument similar to the synchronous VSS protocol, guarantees Completeness.

Secrecy. We prove Secrecy in Appendix C.

Performance. The computation cost of the dealer and nodes is similar to that of the AVSS protocol, except the transcript includes verifiable encryptions for a subset of nodes. Since the (amortized) computation cost of both computing verifiable encryptions and verifying them is linear in the number of encrypted shares [42, 48], both the dealer and nodes need to perform $O(n)$ group exponentiations. Additionally, the dealer sends a private message of length $O(\kappa n)$ to each node and a broadcast channel message of length $O(\kappa n)$ bits. Thus, using the broadcast channel from [32], the total communication cost is $O(\kappa n^2)$.

Combining all the above, we get the following theorem.

Theorem 3 (Dual-threshold AVSS). *In an asynchronous network of $n \geq 3t + 1$ nodes among which at most t nodes are malicious, assuming a polynomial commitment scheme, a signature scheme, a Byzantine reliable broadcast channel, and a Verifiable Encryption scheme, Algorithm 3 implements a t-resilient publicly verifiable (n, ℓ, t) dual-threshold AVSS protocol for any $\ell \in [t, n-t)$ with $O(\kappa n^2)$ communication costs. Here κ is the security parameter.*

8 IMPLEMENTATION AND EVALUATION

We evaluate our VSS schemes and the baseline VSS schemes by implementing them in Rust. Our implementation is publicly available at https://github.com/sourav1547/e2e-vss. Our implementation uses the `blstrs` library [1], which implements efficient finite field and elliptic curve arithmetic. We also use (for both our implementation and the baselines) the multi-exponentiation of group elements using Pippenger's method [10, §4] for efficiency. For our dual threshold AVSS, we implement the verifiable encryption scheme we describe in Appendix B. For networking, we re-use the `network` crate from the open-source implementation of [46]. We will also separately benchmark the computation costs of the various steps of both our and baseline VSS schemes. We implement the asynchronous reliable broadcast (RBC) protocol from [32], with the optimistic path where nodes first run the Bracha's RBC [13] on the cryptographic hash of the dealer's message, and outputs it if the RBC output matches with the hash of the dealer's message. We use the Schnorr signature using Ed25519 elliptic curve [59] as the signature scheme.

Baselines. The first baseline VSS protocol we compare with is the complaint-based AVSS protocol of [65] with optimizations from [32], and here on, we will refer to it as the *complaint* based VSS. Recall from §2, this scheme relies on complaints and does not terminate even with a single faulty node. We chose this as one of our baselines as it is the most efficient AVSS scheme, and by comparing it with this scheme, we seek to demonstrate that our AVSS guarantees asynchronous termination and public verifiability with comparable performance. We measure the baseline's best-case performance, i.e., without any faulty nodes. We implement the polynomial commitment scheme in Figure 7 instead of standard Pedersen commitment to coefficients. Although committing to the evaluation points increases the dealing time, we adopt this approach as it lowers the computation cost during the complaint and reconstruction phase.

Our second baseline is the VE-based VSS scheme from [42] with our modifications in Appendix B. This scheme achieves similar properties to our scheme: it supports dual-threshold, is publicly verifiable, and works in synchronous and asynchronous networks. We want to note that this scheme has a parameter m that indicates the number of chunks we divide a secret into. A smaller m results in quicker dealing and verification time and a smaller transcript size but leads to longer worst-case decryption times. For our evaluations, we opt for $m = 16$ to favor the baseline, i.e., to give it faster dealing and verification time, shorter transcript size, and hence lower latency and bandwidth usage in the absence of failures. However, with $m = 16$, in the worst case, a node would have to perform more than 2^{21} group exponentiations to decrypt its shares.

We do not implement synchronous VSS schemes due to the lack of agreed-upon choice of synchronous broadcast protocols. Here, we provide some estimates for the state-of-the-art synchronous VSS protocol iVSS in [11]. It requires the dealer and non-dealer nodes to verify $O(n)$ signatures. Additionally, each non-dealer node verifies $O(n)$ shares of other parties. Hence, we expect the per-node computation cost to be similar to ours. iVSS requires six rounds of communication, including two broadcast rounds. Hence, we expect its communication costs to be higher than ours. Also, synchronous VSS schemes using the framework of [57] require n parallel broadcasts, and hence $O(n^3)$ communication costs.

Finally, in §8.3, we will provide estimates of our performance comparison with various other AVSS schemes and also discuss why we do not implement them in our framework.

8.1 Computation Costs Measurement

We measure the computation costs of *dealing, verification*, and *reconstruction* of the VSS schemes. We run these benchmarks on an Amazon Web Services (AWS) c5.4xlarge virtual machine with 16 vCPUs, 32GB RAM, and Amazon Linux 2-Kernel. We describe each of these metrics and our associated results.

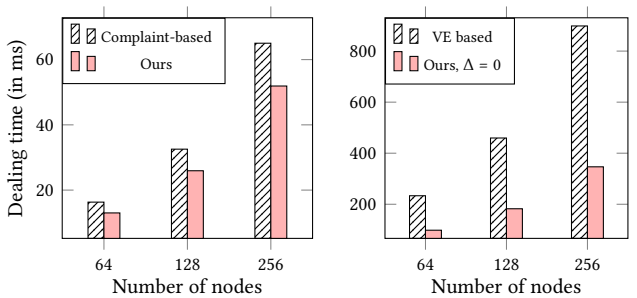
Dealing time. The dealing time measures the computation cost of the dealer in preparing the transcript, i.e., the time the dealer takes: (i) to compute the polynomial commitment, (ii) to compute the opening proofs for each non-dealer node and (iii) to verify `ACK` messages (wherever applicable). For dual-threshold AVSS, the dealing time also includes the computation time required to generate verifiable encryptions of a subset of shares.

We report the dealing time (in milliseconds) in Figure 2. For the low threshold scheme, we expected a similar dealing time between our approach and the complaint-based AVSS, as the dealer in the complaint-based AVSS also needs to compute the public key encryptions of each share, which has costs similar to verifying `ACK` messages. The slight discrepancy is due to using different elliptic curve groups for the public key operations. More precisely, we use the ElGamal encryption scheme in `bls12381` elliptic curve group as the encryption scheme in the complaint-based scheme, whereas our scheme uses Schnorr signatures in `ed25519` elliptic curve group. Each group operation in `bls12381` is more expensive. Hence, we observe a slightly higher dealing time in the complaint-based scheme.

For dual threshold with $\ell = 2t$, our AVSS dealing time is about 40% of the VE-based baseline. This is because the dealer in our dual-threshold AVSS scheme verifiably encrypts $\ell - t$ shares instead of all n shares. We reiterate that for $\ell > t$, in the best-case scenario, our dual-threshold AVSS has a dealing time comparable to our low-threshold AVSS. Hence, in the best case, our dual-threshold AVSS improves the dealing time by 17×.

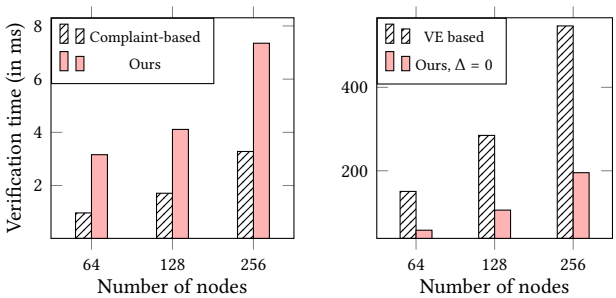
Verification time. The verification time measures the computation cost experienced by the non-dealer nodes. It refers to the time a node takes: (i) to verify the degree of the committed polynomial, (ii) sign the polynomial commitment and verify signatures of other nodes, (iii) validate the revealed shares (including its own), and (iv) and the verifiable encryptions (applicable only to dual-threshold AVSS) provided by the dealer.

We report the verification time (in milliseconds) in Figure 3. For $\ell = t$, our verification time is about 3× worse than the best-case verification time of the complaint-based scheme. This is because



(a) Low-threshold, $\ell = t$ **(b) Dual-threshold, $\ell = 2t$**

Figure 2: Dealing time



(a) Low-threshold, $\ell = t$ **(b) Dual-threshold, $\ell = 2t$**

Figure 3: Verification time

Table 1: AVSS reconstruction time (in milliseconds). For synchronous VSS, our reconstruction time is the same as the baseline.

Scheme	$n = 64$	$n = 128$	$n = 256$
Baseline	3.62	7.07	14.22
Ours	7.06	14.05	28.44

each node in our schemes needs to additionally validate the signatures and shares of other nodes revealed by the dealer. However, the absolute verification time is very small, e.g., only 7.35 milliseconds for 256 nodes. For high threshold $\ell = 2t$, compared to VE-based VSS schemes, the verification time of our protocol is 3× and 47-60× better in the worst and best case, respectively.

Reconstruction time. The reconstruction time measures the computation cost of reconstructing a secret from its shares. It includes the cost of: (i) verifying shares from each node, (ii) computing appropriate Lagrange coefficients, (iii) and the final inner product. Note that the reconstruction time of a VSS scheme depends on the degree of the polynomial used to share the secret and the cost of verifying each share. Since our synchronous VSS protocol uses the same polynomial degree and the same share verification procedure as existing synchronous VSS schemes, it does not add any additional overhead. On the other hand, the dealer in our AVSS scheme uses a degree $2t$ polynomial, compared to the degree t polynomial used by all existing AVSS schemes.We report the reconstruction time (in milliseconds) in Table 1. As expected, our reconstruction time is twice as expensive as the baseline. Nevertheless, the absolute values are small, e.g., 29 milliseconds for 256 nodes.

8.2 Geo-Distributed End-to-End Evaluation

With our end-to-end evaluation, we seek to show that our scheme maintains the bandwidth usage and latency of the most efficient AVSS scheme while ensuring asynchronous termination and public verifiability. Moreover, we will also illustrate that our scheme significantly improves the performance over the VE-based scheme, which also achieves these properties.

Experimental setup. We evaluate the VSS schemes end-to-end with 64, 128, and 256 nodes. For any given $n \geq 3t + 1$, depending upon the VSS scheme, we evaluate them with varying reconstruction threshold $\ell \in [t, n - t - 1]$. We also evaluate our dual-threshold ACSS with the common case optimization we discuss in §7.3 where the dealer either waits to receive $n - t$ valid ACK messages or for 2Δ units of time to as many ACK messages possible, whichever happens later. More precisely, we report our evaluation results for Δ values of 125, 137 and 150, for 64, 128, and 256 nodes, respectively.

We run all nodes on c5.4xlarge instances with one node per VM. In our experiments, we let one among the n ACSS recipients to be the ACSS dealer. We place the recipients evenly across eight AWS regions: Canada, Ireland, North California, North Virginia, Oregon, Ohio, Singapore, and Tokyo. We create an overlay network among the parties where all parties are pair-wise connected, i.e., they form a complete graph.

Metrics. We evaluate VSS schemes using two key metrics: *bandwidth usage* and *latency* of the sharing phase. We explain each of these metrics below.

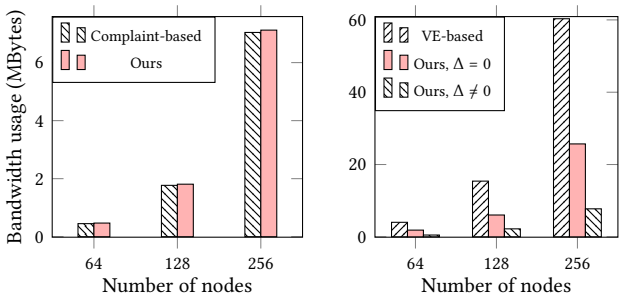
The bandwidth usage is the amount of data a node sends and receives during the sharing phase. For the dealer, this includes the data it sends over the private channel to each non-dealer node, the bandwidth usage for receiving ACK messages, and during the broadcast. For a non-dealer node, this includes bandwidth usage for receiving messages from the dealer, sending acknowledgment signatures (if any), and during the broadcast protocol.

Latency is measured as the time between the dealer starting the sharing phase of the ACSS protocol and the time the recipient nodes finish the sharing phase. As a result, the latency subsumes the computation cost of dealer and non-leader nodes and the communication latency.

We report our results after averaging over ten executions. We also want to note that during our 256-node experiments, we observed some unresponsive nodes for both the baseline and our schemes. However, since such nodes are less than 1/3 of the total nodes, our experiments gracefully finish. The slow nodes, however, increase the overall latency and bandwidth usage, especially in our dual-threshold AVSS experiments. Our results also include the latency and bandwidth usages from runs with slow nodes, where we discard the measurements from unresponsive nodes.

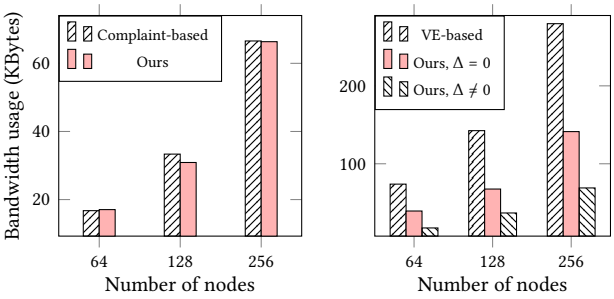
Results. We report the bandwidth usage of dealer (in Megabytes) and non-dealer nodes (in Kilobytes) in Figures 4 and 5, respectively. The bandwidth usage of our low-threshold scheme is comparable to that of the complaint-based scheme. This is because the dealer in the complaint-based scheme also broadcasts the public key encryptions of the shares and incurs a bandwidth usage similar to receiving and sending ACK messages in our scheme.

For dual threshold with $\ell = 2t$ and $\Delta = 0$, the dealer in our VSS scheme incurs less than 50% bandwidth usage compared to the VE-based scheme. Again, this was expected, as our dealer only broadcasts 1/3 verifiable encryptions. For $\Delta \neq 0$, i.e., when the dealer



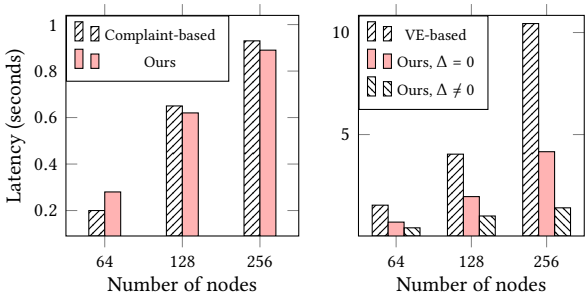
(a) Low-threshold **(b) Dual-threshold, $\ell = 2t$**

Figure 4: Dealer's Bandwidth usage in Sharing phase



(a) Low-threshold, $\ell = t$ **(b) Dual-threshold, $\ell = 2t$**

Figure 5: Receivers bandwidth usage in Sharing phase



(a) Low-threshold **(b) Dual-threshold**

Figure 6: End-to-end Latency of Sharing phase

waits to receive additional ACK messages, our dealer's bandwidth usage is much smaller and is only slightly higher than our low-threshold scheme. For example, with 256 nodes, our low-threshold AVSS dealer incurs 7.12 Megabytes of bandwidth usage, compared to 7.75 Megabytes for our dual-threshold AVSS dealer with $\Delta = 150$ milliseconds. This is because, in the latter case, the dealer receives all most ACK messages by waiting for 150 milliseconds and broadcasts very few verifiable encryptions.

We report the latency results in Figure 6. Note that our low-threshold AVSS scheme has comparable latency to the complaint-based scheme despite having one additional round trip delay. This is because, in our implementation, the dealing time of the complaint-based scheme is slightly higher than our dealing time (see §8.1). This also implies that our additional round-trip delay is not a bottleneck.

The VE-based scheme has much higher latency, i.e., 5×-11× higher than our low-threshold VSS scheme. Interestingly, this is much higher than expected, given the VE-based scheme's dealing and verification time. Upon further investigation, we note that the additional latency is due to the propagation latency of their large VSS transcripts. This also explains why our dual-threshold VSS scheme with $\Delta = 0$ has approximately 2× better latency, and our scheme with $\Delta \neq 0$ improves has 3-7× better end-to-end latency. This also concludes that both computation time and bandwidth usage are bottlenecks for AVSS schemes.

8.3 Additional Comparisons

Comparison with Class-group based VSS [48]. Very recently, Kate et al. [48] improved the efficiency of [42] for high-entropy secrets using a non-standard class-group assumption. Here, we will estimate how it compares to our scheme based on their C++ implementation. The VSS transcript size of [48] is $219(n + 1) + 48n$ bytes (assuming we commit to evaluation points instead of coefficients), which is approximately 3× higher than the transcript size of our low-threshold AVSS scheme. Regarding dealing and verification time, [48] reports 2.7× improvement over [42]. Since our dealing time is 17× better than [42], we anticipate that our dealing time is 6× better than that of [48]. Similarly, we expect our verification time to be 2-3× better. We achieve these improvements while relying on the standard discrete logarithm assumption.

Bivariate polynomial based AVSS. We do not implement the bivariate polynomial-based VSS schemes [4, 5] as they are very inefficient, even if we rely on a trusted setup. We will illustrate this below using a careful performance estimate of Bingo, the state-of-the-art bivariate-based AVSS scheme [4].

In Bingo, the dealer performs $O(n^2 \log n)$ field operations to evaluate the bivariate polynomials using FFT and n $2n$-wide multi-exponentiations to compute the polynomial commitments. Based on [2] for BLS21-381, we estimate that the dealing time for 256 nodes will be more than 1400 milliseconds (21× higher than ours) in an AWS EC2 m6g.8xlarge instance (more powerful machine than ours). Also, each non-dealer node must perform $t + n$ KZG polynomial commitment verifications [49], which would take more than 170 milliseconds (23× higher than ours).

9 DISCUSSION AND CONCLUSION

Interactive vs. non-interactive protocols. In existing VE-based VSS [33, 42, 48] the dealer sends a single message over the broadcast channel. On the other hand, our VSS protocols require interaction between the dealer and the other nodes (but not among the nodes). As a result, our protocols are slightly more complex to implement. Yet, we believe that the substantial performance improvements offered by our protocols outweigh the added complexity. Designing a more efficient non-interactive public verifiable secret sharing scheme remains a fascinating open question.

Applications with polynomials of an arbitrary degree. Although our AVSS scheme shares the secret using a degree $2t$ polynomial, applications such as asynchronous DKG, asynchronous proactive secret sharing, etc., need not use a degree $2t$ polynomials. Instead, these applications can share their secret using an arbitrary degree polynomial, using the degree switching trick of [31].

Conclusion. We have presented a simple approach to design three efficient verifiable secret sharing protocols for various settings, i.e., under synchrony, asynchrony, and asynchrony with dual-threshold. Unlike existing schemes, our VSS protocols do not rely on complaints and require only a single broadcast. Our protocols output efficient publicly verifiable transcripts and support dual-threshold

in asynchrony. Moreover, our asynchronous VSS protocols ensure termination, fixing a shortcoming in many existing schemes. Our protocols have comparable performance to state-of-the-art counterparts without those properties and provide significant performance improvements over schemes with similar properties.

ACKNOWLEDGMENTS

This work is funded in part by a VMware early career faculty grant, a Chainlink Labs Ph.D. fellowship, and the National Science Foundation award #2240976.

REFERENCES

[1] 2020. blstrs library. (2020). https://docs.rs/blstrs/latest/blstrs/
[2] 2023. zkalc is a cryptographic calculator! (2023). https://github.com/mmaker/zkalc
[3] Ittai Abraham, Danny Dolev, and Gilad Stern. 2020. Revisiting asynchronous fault tolerant computation with optimal resilience. In *Proceedings of the 39th Symposium on Principles of Distributed Computing*. 139–148.
[4] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. 2023. Bingo: Adaptively Secure Packed Asynchronous Verifiable Secret Sharing and Asynchronous Distributed Key Generation. In *Annual International Cryptology Conference*. Springer.
[5] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. 2021. High-threshold avss with optimal communication complexity. In *International Conference on Financial Cryptography and Data Security*. Springer, 479–498.
[6] Orestis Alpos, Christian Cachin, Simon Holmgaard Kamp, and Jesper Buus Nielsen. 2023. Practical Large-Scale Proof-of-Stake Asynchronous Total-Order Broadcast. *Cryptology ePrint Archive* (2023).
[7] 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.
[8] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC '88)*. New York, NY, USA, 1–10.
[9] Fabrice Benhamouda, Shai Halevi, Hugo Krawczyk, Alex Miao, and Tal Rabin. 2022. Threshold Cryptography as a Service (in the Multiserver and YOSO Models). In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 323–336.
[10] Daniel J Bernstein, Jeroen Doumen, Tanja Lange, and Jan-Jaap Oosterwijk. 2012. Faster batch forgery identification. In *Progress in Cryptology-INDOCRYPT 2012: 13th International Conference on Cryptology in India, Kolkata, India, December 9-12, 2012. Proceedings 13*. Springer, 454–473.
[11] Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. 2021. Randpiper–reconfiguration-friendly random beacons with quadratic communication. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*. 3502–3524.
[12] George Robert Blakley. 1979. Safeguarding cryptographic keys. In *1979 International Workshop on Managing Requirements Knowledge (MARK)*. IEEE, 313–318.
[13] Gabriel Bracha. 1987. Asynchronous Byzantine agreement protocols. *Information and Computation* 75, 2 (1987), 130–143.
[14] Luis TAN Brandao, Luis TAN Brandao, Michael Davidson, and Apostol Vassilev. 2020. NIST roadmap toward criteria for threshold schemes for cryptographic primitives. (2020).
[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, Klaus Kursawe, Frank Petzold, and Victor Shoup. 2001. Secure and efficient asynchronous broadcast protocols. In *Annual International Cryptology Conference*. Springer, 524–541.
[17] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2000. Random oracles in constantipole: practical asynchronous byzantine agreement using cryptography. In *Proceedings of the nineteenth annual ACM symposium on Principles of distributed computing*. 123–132.
[18] Jan Camenisch and Victor Shoup. 2003. Practical verifiable encryption and decryption of discrete logarithms. In *Annual International Cryptology Conference*. Springer, 126–144.
[19] Ran Canetti. 1996. *Studies in secure multiparty computation and applications*. Ph.D. Dissertation. Citeseer.
[20] 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.
[21] 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.
[22] Ignacio Cascudo and Bernardo David. 2020. ALBATROSS: publicly attestable batched randomness based on secret sharing. In *Advances in Cryptology–ASIACRYPT 2020: 26th International Conference on the Theory and Application of Cryptology and Information Security, Daejeon, South Korea, December 7–11, 2020, Proceedings, Part III 26*. Springer, 311–341.
[23] Ignacio Cascudo and Bernardo David. 2023. Publicly Verifiable Secret Sharing over Class Groups and Applications to DKG and YOSO. *Cryptology ePrint Archive* (2023).
[24] Pyrros Chaidos and Aggelos Kiayias. 2021. Mithril: Stake-based threshold multisignatures. *Cryptology ePrint Archive* (2021).
[25] Kevin Choi, Aathira Manoj, and Joseph Bonneau. 2023. SoK: Distributed Randomness Beacons. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE.
[26] Benny Chor, Shafi Goldwasser, Silvio Micali, and Baruch Awerbuch. 1985. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*. IEEE, 383–395.
[27] Ashish Choudhury. 2020. *Optimally-resilient unconditionally-secure asynchronous multi-party computation revisited*. Technical Report. Cryptology ePrint Archive, Report 2020/906, 2020. https://eprint. iacr. org ....
[28] Ivan Damgård. 2002. On Σ-protocols. *Lecture Notes, University of Aarhus, Department for Computer Science* (2002), 84.
[29] Ivan Damgård and Jesper Buus Nielsen. 2007. Scalable and unconditionally secure multiparty computation. In *Annual International Cryptology Conference*. Springer, 572–590.
[30] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. 2022. Spurt: Scalable distributed randomness beacon with transparent setup. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2502–2517.
[31] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. 2023. Practical Asynchronous High-threshold Distributed Key Generation and Distributed Polynomial Sampling. In *32st USENIX Security Symposium (USENIX Security 23)*. USENIX Association.
[32] Sourav Das, Zhuolun Xiang, and Ling Ren. 2021. Asynchronous Data Dissemination and its Applications. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*.
[33] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical asynchronous distributed key generation. In *2022 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2518–2534.
[34] Paul Feldman. 1987. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*. IEEE, 427–438.
[35] Pierre-Alain Fouque and Jacques Stern. 2001. One round threshold discrete-log key generation without private channels. In *International Workshop on Public Key Cryptography*. Springer, 300–316.
[36] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. 1996. Robust threshold DSS signatures. In *Advances in Cryptology—EUROCRYPT'96: International Conference on the Theory and Application of Cryptographic Techniques Saragossa, Spain, May 12–16, 1996 Proceedings 15*. Springer, 354–371.
[37] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology* 20, 1 (2007), 51–83.
[38] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling byzantine agreements for cryptocurrencies. In *Proceedings of the 26th symposium on operating systems principles*. 51–68.
[39] Shafi Goldwasser, Silvio Micali, and Chales Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing*. 291–304.
[40] Vipul Goyal, Yifan Song, and Chenzhi Zhu. 2020. Guaranteed output delivery comes free in honest majority MPC. In *Annual International Cryptology Conference*. Springer, 618–646.
[41] Jens Groth. 2010. Short Pairing-Based Non-interactive Zero-Knowledge Arguments.. In *Asiacrypt*, Vol. 6477. Springer, 321–340.
[42] Jens Groth. 2021. Non-interactive distributed key generation and key resharing. *IACR Cryptol. ePrint Arch.* 2021 (2021), 339.
[43] Jens Groth and Victor Shoup. 2022. Design and analysis of a distributed ecdsa signing service. *Cryptology ePrint Archive* (2022).
[44] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Aggregatable distributed key generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 147–176.
[45] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. 2008. Asynchronous multi-party computation with quadratic communication. In *International Colloquium on Automata, Languages, and Programming*. Springer, 473–485.
[46] ISTA-SPiDerS. 2022. APSS. https://github.com/ISTA-SPiDerS/apss. (2022).
[47] Aniket Kate and Ian Goldberg. 2009. Distributed key generation for the internet. In *2009 29th IEEE International Conference on Distributed Computing Systems*.

IEEE, 119–128.
[48] Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. 2023. Non-interactive VSS using Class Groups and Application to DKG. *Cryptology ePrint Archive* (2023).
[49] 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.
[50] 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.
[51] Leslie Lamport, Robert Shostak, and Marshall Pease. 2019. The Byzantine generals problem. In *Concurrency: the works of leslie lamport*. 203–226.
[52] Atsuki Momose and Ling Ren. 2021. Optimal Communication Complexity of Authenticated Byzantine Agreement. In *35th International Symposium on Distributed Computing*.
[53] 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*.
[54] 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.
[55] 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.
[56] Torben Pryds Pedersen. 1991. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*. Springer, 129–140.
[57] Torben Pryds Pedersen. 1991. A threshold cryptosystem without a trusted party. In *Workshop on the Theory and Application of of Cryptographic Techniques*. Springer, 522–526.
[58] 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.
[59] Claus-Peter Schnorr. 1990. Efficient identification and signatures for smart cards. In *Advances in Cryptology—CRYPTO'89 Proceedings 9*. Springer, 239–252.
[60] Berry Schoenmakers. 1999. A simple publicly verifiable secret sharing scheme and its application to electronic voting. In *Annual International Cryptology Conference*. Springer, 148–164.
[61] Adi Shamir. 1979. How to share a secret. *Commun. ACM* 22, 11 (1979), 612–613.
[62] Victor Shoup and Nigel P Smart. 2023. Lightweight Asynchronous Verifiable Secret Sharing with Optimal Resilience. *Cryptology ePrint Archive* (2023).
[63] Robin Vassantlal, Eduardo Alchieri, Bernardo Ferreira, and Alysson Bessani. 2022. Cobra: Dynamic proactive secret sharing for confidential bft services. In *2022 IEEE symposium on security and privacy (SP)*. IEEE, 1335–1353.
[64] 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.
[65] Thomas Yurek, Licheng Luo, Jaiden Fairoze, Aniket Kate, and Andrew Miller. 2022. hbACSS: How to Robustly Share Many Secrets. In *Proceedings of the 29th Annual Network and Distributed System Security Symposium*.
[66] Haibin Zhang, Sisi Duan, Chao Liu, Boxin Zhao, Xuanji Meng, Shengli Liu, Yong Yu, Fangguo Zhang, and Liehuang Zhu. 2022. Practical Asynchronous Distributed Key Generation: Improved Efficiency, Weaker Assumption, and Standard Model. *Cryptology ePrint Archive* (2022).
[67] Jiaheng Zhang, Tiancheng Xie, Thang Hoang, Elaine Shi, and Yupeng Zhang. 2022. Polynomial Commitment with a {One-to-Many} Prover and Applications. In *31st USENIX Security Symposium (USENIX Security 22)*. 2965–2982.

A ADDITIONAL PRELIMINARIES

A.1 Broadcast Channel

Our synchronous VSS and AVSS protocols make black box invocations to a Byzantine broadcast and Byzantine reliable broadcast protocol, respectively. We use state-of-the-art broadcast extension protocols, i.e., for long messages [32, 53]. For completeness, we include the definitions of Byzantine (reliable) broadcast below.

Definition 5 (Byzantine Broadcast)**.** A Byzantine broadcast is a protocol for a set of nodes $\{1, \ldots, n\}$ including a designated broadcaster who holds an initial input, is a Byzantine broadcast protocol if the following properties hold

- **Agreement.** If an honest node outputs a message M and another honest node outputs M', then $M = M'$.
- **Validity.** If the sender is honest and has input M, all honest nodes output M.
- **Termination.** Every honest node outputs a message.

Definition 6 (Byzantine Reliable Broadcast)**.** A protocol for a set of nodes $\{1, \ldots, n\}$ including a designated broadcaster who holds an initial input, is a Byzantine reliable broadcast protocol if the following properties hold

- **Agreement** and **Validity.** Same as Byzantine broadcast.
- **Totality.** If an honest node outputs a message, then every honest node eventually outputs a message.

The optimal fault-tolerant synchronous Byzantine broadcast [52, 53] achieves $O(n|M|+\kappa n^2)$ communication cost for a message M assuming powers-of-tau [49] and q-SDH, and $O(n|M|+\kappa n^2 \log n)$ communication cost assuming collision resistant hash function. The optimal fault-tolerant asynchronous Byzantine reliable broadcast [32] achieves a communication cost of $O(n|M|+\kappa n^2)$ for a message M assuming collision-resistant hash functions.

A.2 Pedersen's polynomial commitment

Constructions. In Figure 7 we describe a concrete polynomial commitment scheme that combines ideas from the classic Pedersen's polynomial commitment and SCRAPE's low-degree test [21]. The resulting scheme has a linear-sized commitment and constant-sized opening proof. The commitment includes n values of the polynomials in the exponent, and the low degree is verified by multiplying these values in the exponent with a random word from the dual code and checking that the result is $1_{\mathbb{G}}$, i.e., the identity element of $\mathbb{G}$. The scheme is information-theoretically hiding and evaluation binding assuming hardness of discrete logarithm [56].

Remark. An alternative approach would have been to commit to $d+1$ coefficients of the polynomials in the exponent. This would have made the commitment shorter and would have eliminated the need for low-degree verification. On the other hand, the opening phase would have become more costly: verifying each opened value would have required $O(d)$ exponentiations instead of one.

A.3 Batched interface for polynomial commitment

As we briefly describe in §3.3, in our VSS schemes, the dealer reveals shares for a list of nodes for everyone to verify. Thus, we introduce the following additional interface for batched opening and verification. Specifically, for any set $I \subseteq [n]$, we require the polynomial commitment to provide the following interfaces.

- PC.BatchOpen$(pp, \boldsymbol{w}, p(\cdot), I = \{i_1, \ldots, i_k\}) \rightarrow (\boldsymbol{u}, \boldsymbol{\pi})$. On input the set of indices I, the polynomial $p(\cdot)$ and witness $\boldsymbol{w}$, outputs $\boldsymbol{u} = [p(i_1), \ldots, p(i_k)]$ along with batch opening proof $\boldsymbol{\pi} = [\pi_{i_1}, \ldots, \pi_{i_n}]$.
- PC.BatchVerify$(pp, \boldsymbol{v}, I = \{i_1, \ldots, i_k\}, \boldsymbol{u}, \boldsymbol{\pi}) \rightarrow 0/1$. On input the commitment $\boldsymbol{v}$ to a polynomial $p(i)$, outputs 1 if $\boldsymbol{u}[j] = p(i_j)$ for all $i_j \in I$, and outputs 0 otherwise.

We describe the concrete instantiations of the batched interfaces in Figure 8. Here we use a random linear combination to verify

PC.Setup(1^λ): Output $pp = (\mathbb{G}, \mathbb{F}, g, h)$, for an elliptic curve group $\mathbb{G}$ with scalar field $\mathbb{F}$, and uniformly random and independent generators $g, h \in \mathbb{G}$.

PC.Commit($pp, p(\cdot), d, n$): Let $p(\cdot)$ be the polynomial of degree d. Sample a random polynomial $r(\cdot)$ of degree d. Let $\boldsymbol{v}$ be the commitment to $p(\cdot)$ where

$$\boldsymbol{v} = \left[g^{p(1)}h^{r(1)}, g^{p(2)}h^{r(2)}, \ldots, g^{p(n)}h^{r(n)}\right]$$

Output $(\boldsymbol{v}, \boldsymbol{w}) = (\boldsymbol{v}, r(\cdot))$.

PC.Open($pp, i, p(\cdot), r(\cdot)$): Output $(u, \pi) = (p(i), r(i))$.

PC.DegCheck($pp, \boldsymbol{v}, d$): Sample a random degree $d = n - t - 2$ polynomial $z(\cdot)$ in $\mathbb{F}$. Output 1 if

$$\prod_{i\in[n]} \boldsymbol{v}[i]^{z(i)\cdot\lambda_i} = 1_{\mathbb{G}} \tag{2}$$

for $\lambda_i = \prod_{j\in[n], j\neq i} 1/(i-j)$; otherwise output 0.

PC.Verify($pp, \boldsymbol{v}, i, u, \pi$): Output 1 if $\boldsymbol{v}[i] = g^z h^\pi$; otherwise output 0.

Figure 7: Pedersen's polynomial commitment scheme combined with SCRAPE's low degree test.

PC.BatchOpen($pp, \boldsymbol{w}, p(\cdot), I = \{i_1, \ldots, i_k\}$): Output $(\boldsymbol{s}, \boldsymbol{\pi})$ where

$$\boldsymbol{s} = [p(i_1), \ldots, p(i_k)]; \text{ and } \boldsymbol{\pi} = [r(i_1), \ldots, r(i_k)] \tag{3}$$

PC.BatchVerify($pp, \boldsymbol{v}, I = \{i_1, \ldots, i_k\}, \boldsymbol{s}, \boldsymbol{\pi}$): Given a subset $I \subseteq [n]$, let $k = |I|$. Assert $k = |\boldsymbol{v}| = |\boldsymbol{\pi}|$. Sample a uniform random vector $[\gamma_1, \ldots, \gamma_k] \in \mathbb{F}^k$. Let $s = \sum_{j\in[k]} \gamma_j s_j$ and $\pi = \sum_{j\in[k]} \gamma_j \pi_j$. Output 1, if the following holds, otherwise output 0.

$$\prod_{j\in[k]} \boldsymbol{v}[i_j]^{\gamma_j} = g^s h^\pi \tag{4}$$

Figure 8: Batched interfaces for the Polynomial commitment.

all the openings using a single multi-exponentiations of width k instead of $2k$ exponentiations.

B VERIFIABLE ENCRYPTIONS OF DISCRETE LOGARITHM

B.1 Verifiable Encryption Scheme of [42]

The VE scheme of Groth [42] works with Feldman commitment where a message $s \in \mathbb{F}$ is committed as g^s for some pre-specified generator $g \in \mathbb{G}$. We can not use it to design a VSS protocol as the Feldman commitment scheme is not hiding for messages with small entropy; an adversary can exhaustively search the message space to derive a matching commitment. Nevertheless, we will use Groth's VE to design a VE that works with the Pedersen commitment scheme. Next, we will briefly describe the relations $\mathcal{P}$ in Groth's VE proves and discuss how our modifications require $\mathcal{P}$ to prove a similar relation.

Let $v = g^s$ be the commitment to the secret s. $\mathcal{P}$ computes, among other things, the ElGamal encryption of v, i.e., $c_v = (c_{v,0}, c_{v,1}) = (g^a, v\mathsf{pk}^a)$. Here $\mathsf{pk} = g^{\mathsf{sk}}$ is the public key of the recipient with secret key sk. $\mathcal{P}$ then computes the NIZK proof in two parts: *Proof of correct sharing* and *Proof of correct chunking*.

Proof of correct sharing. In the first part, for the tuple (v, c_v, pk), $\mathcal{P}$ proves, using a Σ-protocol, that c_v is an ElGamal encryption of v for the public key pk.

Proof of correct chunking. In the second part, $\mathcal{P}$ proves that the ciphertext is decryptable. Let $\boldsymbol{c}_v$ be a vector of ElGamal ciphertexts where each ciphertext encrypts a small number of bits (called chunks) of s. Let $(\mathsf{pk}, c_v, \boldsymbol{c}_v)$ be the entire ciphertext (of commitment and each chunk of s), then $\mathcal{P}$ proves that $s \leftarrow \mathsf{Dec}(\mathsf{sk}, c_v, \boldsymbol{c}_v)$. We refer the reader to [42, §6.5]) for more details.

B.2 VE for Pedersen commitments

Our new VE for Pedersen commitment maintains the two-part structure of Groth's VE. Looking ahead, we provide support for the Pedersen commitment scheme only by changing the protocol for proof of correct sharing. Moreover, our modification adds only two group elements and a single field element to the Groth's VE proof. We discuss our changes next.

Proof of correct sharing. Let $g^s h^r$ be the Pedersen commitment to s. Let $v = g^s$ and $u = h^r$, hence the commitment to s is $v \cdot u$. In our scheme, the ciphertext also contains the ElGamal encryption of u, i.e. $c_u = (c_{u,0}, c_{u,1}) = (g^b, u\mathsf{pk}^b)$, along with $c_v = (c_{v,0}, c_{v,1}) = (g^a, v\mathsf{pk}^a)$. Now, $\mathcal{P}$ and $\mathcal{V}$ locally computes c_{vu}, where,

$$c_{vu} = (c_{v,0} \cdot c_{u,0}, c_{v,1} \cdot c_{u,1}) = (g^{a+b}, vu \cdot \mathsf{pk}^{a+b})$$

$\mathcal{P}$ in our VE then uses the protocol for proof of correct sharing of Groth's VE (with standard modifications [28]) for the tuple $(vu, c_{vu}, \mathsf{pk})$ to prove that c_{uv} is an ElGamal encryption of vu for public key pk.

Proof of correct chunking. Since the ciphertext of our VE remains unchanged (with the exception of one additional ElGamal encryption), a tempting approach is to directly use the protocol for proof of correct chunking of Groth's VE protocol as the second part of our VE scheme. Intuitively, proof of correct chunking protocol of Groth's VE guarantees that a node with secret sk will be able to decrypt s as $\mathsf{Dec}(sk, c_v, \boldsymbol{c}_v)$. Although it is true, there is one subtle issue.

Eventually, to reconstruct the secret, we require each node to reveal its share along with a opening proof. For Pedersen commitment $g^s h^r$, the natural opening proof is r. This implies that to fully support Pedersen commitments, we need to add additional information $\boldsymbol{c}_u$ to the ciphertext and the NIZK proof such that $(s, r) \leftarrow \mathsf{Dec}(\mathsf{sk}, c_u, c_v, \boldsymbol{c}_v, \boldsymbol{c}_u)$.

The obvious approach is to repeat the protocol to prove the decryptability of c_v for c_u, as well. However, this would increase the computation cost of dealing and verifying the transcript and the transcript size by a factor of 2. Next, we describe our approach that addresses this issue without increasing the ciphertext size, thus avoiding the 2× overhead.

Our key observation is that the opening proof of a Pedersen commitment $g^s h^r$ need not be r. Instead, it can be $(u = h^r, \pi_u)$ where π_u proves that u is correctly computed. Thus, in our VE, we let $\mathcal{R}$ recover (u, π_u), where $\mathcal{R}$ uses π_u to convince others regarding the correctness of u.

Computing u is trivial as it is the ElGamal decryption of c_u using the secret key sk. We define π_u as the tuple $(\mathsf{pk}^b, \pi_{\mathsf{pk}})$ where π_{pk} is a discrete logarithm equality (DLEq) proof for the tuple

Functionality $\mathcal{F}_{\text{VSS}}$

Parameters: Maximum number of malicious nodes t, the total number of nodes $n \geq 2t+1$. Let $\mathbb{G}$ be an elliptic curve group of order q with scalar field $\mathbb{F}$.

(1) Wait for secret s from the dealer.
(2) Wait for C with $|C| \leq t$, the set of nodes $\mathcal{A}$ will corrupt.
(3) Compute (n, t) Shamir secret shares of s over the field $\mathbb{F}$. Let $s(x)$ be the degree t polynomial with $s = s(0)$.
(4) Send $s(j)$ to each honest node j. Send $\{s(i)\}_{i \in C}$ to $\mathcal{A}$.

Figure 9: Functionality for the Sharing phase of synchronous VSS.

Inputs. C, $\mathbb{F}$ and $\mathbb{G}$. **Notation.** Let $\mathcal{H} = [n] \setminus C$

(1) Sample signing and public key $(\text{sk}_j, \text{pk}_j)$ for each $j \in \mathcal{H}$. Send the public keys to $\mathcal{A}$.
(2) Send C to $\mathcal{F}_{\text{VSS}}$ and receive $\{s(i)\}_{i \in C}$.
(3) Sample uniformly random generators $g, h \leftarrow \mathbb{G}$.
(4) Sample a polynomial $\hat{s}(\cdot)$ of degree t such that $\hat{s}(i) = s(i)$ for each $i \in C$. Additionally, sample a uniform random polynomial $\hat{r}(\cdot)$ of degree t.
(5) Compute the commitment $\boldsymbol{v} = [v_1, v_2 \ldots, v_n]$ where $v_i = g^{\hat{s}(i)} h^{r(i)}$ for each $i \in [n]$.
(6) Simulate the dealer by sending $\boldsymbol{v} = [v_1, v_2, \ldots, v_n]$ as the polynomial commitment. Participate in the rest of the protocol on behalf of the honest parties.

Figure 10: Synchronous VSS simulator $\mathcal{S}_{\text{VSS}}$

$(g, \text{pk}, c_{u,0}, \text{pk}^b)$. More precisely, π_{pk} convinces any verifier that $\log_g \text{pk} = \log_{c_{u,0}} \text{pk}^b$.

Each node upon receiving $\pi_u = (\text{pk}^b, \pi_{\text{pk}})$, checks the correctness of the DLEq relation using π_{pk} and $c_{u,0}$. Upon successful validation, the node computes $h^r = c_{u,1}/\text{pk}^b$. Finally, the node checks the correctness of s by checking whether $g^s h^r = vu$.

C SECRECY PROOFS

We prove Secrecy of our VSS protocols using *simulatability*: for every probabilistic polynomial-time (PPT) adversary $\mathcal{A}$ that statically corrupts up to t nodes, there exists an ideal world PPT simulator that interacts with the ideal functionality and produces a view such that $\mathcal{A}$'s view in the simulated world is indistinguishable to a run of the Sharing phase.

Secrecy of Synchronous VSS. We prove Secrecy of our synchronous VSS with respect to $\mathcal{F}_{\text{VSS}}$ ideal functionality (cf. Figure 9). Let $\mathcal{S}_{\text{VSS}}$ be corresponding simulator. $\mathcal{S}_{\text{VSS}}$ simulate $\mathcal{A}$'s view using the Pedersens' polynomial commitment scheme. We summarize $\mathcal{S}_{\text{VSS}}$ in Figure 10, and prove the following theorem.

Lemma 1 (Synchronous VSS Secrecy)**.** *$\mathcal{A}$'s view in its interaction with $\mathcal{S}_{\text{VSS}}$ is identically distributed to its view in the real protocol.*

Proof. Let $h = g^\alpha$ for some non-zero $\alpha \in \mathbb{F}$. For any fixed commitment $\boldsymbol{v}$, consider the probability of outputting $\boldsymbol{v}$ and $s(i)$ for each $i \in C$ in a real protocol. For a fixed polynomial $s(\cdot)$, there exists a unique polynomial $r(\cdot)$ that outputs $\boldsymbol{v}$ as the commitment. Since the dealer in the honest protocol samples $r(\cdot)$ uniformly at random, in the real protocol $\Pr[\boldsymbol{v}, \{s(i)\}_{i \in C}]_{\text{real}} = 1/|\mathbb{F}|^{t+1}$.

Now consider the probability of the same event in the simulated view. For a fixed $\hat{s}(\cdot)$, a unique degree t polynomial $\hat{r}(\cdot)$ exists that

Functionality $\mathcal{F}_{\text{AVSS}}$

Parameters: Maximum number of malicious nodes t, the total number of nodes $n \geq 3t+1$. Let $\mathbb{G}$ be an elliptic curve group of order q with scalar field $\mathbb{F}$.

(1) Wait for secret s from the dealer.
(2) Wait for C and $\mathcal{H}_R$ from $\mathcal{A}$. Here C is the set of nodes $\mathcal{A}$ will corrupt and $\mathcal{H}_R$ is the set of honest nodes whose shares the the functionality will reveal. Check that $|C| \leq t$, $|C \cup \mathcal{H}_R| \leq 2t$. Let $C_0 = C \cup \mathcal{H}_R$.
(3) Compute $(n, 2t)$ Shamir secret shares of s over the field $\mathbb{F}$. Let $s(x)$ be the degree $2t$ polynomial with $s = s(0)$.
(4) Send $s(j)$ to each honest node j. Send $\{s(i)\}_{i \in C_0}$ to $\mathcal{A}$.

Figure 11: Functionality for the Sharing phase of our AVSS.

Inputs. C, $\mathcal{H}_R$, $\mathbb{F}$ and $\mathbb{G}$. **Notation.** Let $\mathcal{H} = [n] \setminus C$ and let $C_0 = C \cup \mathcal{H}_R$.

(1) Sample signing and public key $(\text{sk}_j, \text{pk}_j)$ for each $j \in \mathcal{H}$. Send the public keys to $\mathcal{A}$.
(2) Send $(C, \mathcal{H}_R)$ to $\mathcal{F}_{\text{AVSS}}$ and receive $\{s(i)\}_{i \in C_0}$.
(3) Sample uniformly random generators $g, h \leftarrow \mathbb{G}$.
(4) Sample a polynomial $\hat{s}(\cdot)$ of degree $2t$ such that $\hat{s}(i) = s(i)$ for each $i \in C_0$. Additionally, sample a uniform random polynomial $\hat{r}(\cdot)$ of degree $2t$.
(5) Compute the commitment $\boldsymbol{v} = [v_1, v_2 \ldots, v_n]$ where $v_i = g^{\hat{s}(i)} h^{r(i)}$ for each $i \in [n]$.
(6) Simulate the dealer by sending $\boldsymbol{v} = [v_1, v_2, \ldots, v_n]$ as the polynomial commitment. Participate in the rest of the protocol on behalf of the honest parties.

Figure 12: Asynchronous VSS simulator $\mathcal{S}_{\text{AVSS}}$

results in $\boldsymbol{v}$ as the commitment. In particular, the unique $\hat{r}(\cdot)$ is:

$$\hat{r}(x) = r(x) + \frac{s(x) - \hat{s}(x)}{\alpha} \tag{5}$$

Since $\mathcal{S}_{\text{VSS}}$ samples $\hat{r}(\cdot)$ uniformly at random,

$$\begin{aligned} \Pr[\boldsymbol{v}, \{\hat{s}(i)\}_{i \in C}]_{\text{id}} &= \Pr\left[\hat{r}(x) = r(x) + \frac{s(x) - \hat{s}(x)}{\alpha}\right]_{\text{id}} \\ &= 1/|\mathbb{F}|^{t+1} \end{aligned} \tag{6}$$

Equation (6) implies that the polynomial commitment and shares seen by $\mathcal{A}$ are identically distributed in real and simulated view. Since $\mathcal{S}_{\text{VSS}}$ simulates the rest of the protocol as per protocol specification, the distribution of the remaining messages seen by $\mathcal{A}$ is also identical in both the real and simulated world. □

Secrecy of Asynchronous VSS. We prove Secrecy of our AVSS scheme with respect to $\mathcal{F}_{\text{AVSS}}$ ideal functionality (cf. Figure 11). Let $\mathcal{S}_{\text{AVSS}}$ be corresponding simulator. $\mathcal{S}_{\text{AVSS}}$ also uses the polynomial commitment scheme from Figure 7 to simulate $\mathcal{A}$'s view. We summarize $\mathcal{S}_{\text{AVSS}}$ in Figure 12, and prove the following.

Lemma 2 (Asynchronous VSS Secrecy)**.** *$\mathcal{A}$'s view in its interaction with $\mathcal{S}_{\text{AVSS}}$ is identically distributed to its view in the real protocol.*

Proof. Follows using a similar argument as the proof of Lemma 1. □

Secrecy of the dual-thresold AVSS. We prove Secrecy of our dual-threshold AVSS scheme with respect to $\mathcal{F}_{\text{DtAVSS}}$ ideal functionality

Functionality $\mathcal{F}_{\text{DtAVSS}}$

Parameters: The maximum number of malicious nodes t, the total number of nodes $n \geq 3t+1$, and maximum coalition size $\ell \in [t, n-t)$. Let $\mathbb{G}$ be an elliptic curve group of order q with scalar field $\mathbb{F}$.

(1) Wait for secret s from the dealer.
(2) Wait for C, $\mathcal{H}_R$, and $\mathcal{H}_C$ from $\mathcal{A}$. Here C is the set of nodes $\mathcal{A}$ will corrupt and $\mathcal{H}_R$ is the set of honest nodes whose shares the the functionality will reveal. Also, let $\mathcal{H}_C$ is the set of honest nodes who will collude with $\mathcal{A}$ to learn the secret. Let $C_0 = C \cup \mathcal{H}_C \cup \mathcal{H}_R$
(3) Assert that $|C| \leq t$, $|C \cup \mathcal{H}_C| \leq \ell$, and $|C \cup \mathcal{H}_C \cup \mathcal{H}_R| \leq 2t$.
(4) Compute $(n, 2t)$ Shamir secret shares of s over the field $\mathbb{F}$. Let $s(x)$ be the degree $2t$ polynomial with $s = s(0)$.
(5) Send $s(j)$ to each honest node j. Send $\{s(i)\}_{i \in C_0}$ to $\mathcal{A}$.

Figure 13: Dual-threshold AVSS functionality.

Inputs. $C, \mathcal{H}_C, \mathcal{H}_R, \mathbb{F}$ and $\mathbb{G}$.
Notation. Let $\mathcal{H} = [n] \setminus C$ and let $C_0 = C \cup \mathcal{H}_R \cup \mathcal{H}_C$.

(1) Sample signing and public key $(\mathsf{sk}_j, \mathsf{pk}_j)$ for each $j \in \mathcal{H}$. Send the public keys of all nodes to $\mathcal{A}$. Additionally, send sk_j for each $j \in \mathcal{H}_C$ to $\mathcal{A}$.
(2) Send $(C, \mathcal{H}_R, \mathcal{H}_C)$ to $\mathcal{F}_{\text{DtAVSS}}$ and receive $\{s(i)\}_{i \in C_0}$.
(3) Sample uniformly random generators $g, h \leftarrow \mathbb{G}$.
(4) Sample a polynomial $\hat{s}(\cdot)$ of degree $2t$ such that $\hat{s}(i) = s(i)$ for each $i \in C_0$. Additionally, sample a uniform random polynomial $\hat{r}(\cdot)$ of degree $2t$.
(5) Compute the commitment $\boldsymbol{v} = [v_1, v_2 \ldots, v_n]$ where $v_i = g^{\hat{s}(i)} h^{r(i)}$ for each $i \in [n]$.
(6) Simulate the dealer by sending $\boldsymbol{v} = [v_1, v_2, \ldots, v_n]$ as the polynomial commitment. Simulate the dual-threshold VSS protocol on behalf of honest nodes up until receiving $2t+1$ signed acknowledgments.
(7) Let $\mathcal{H}_E$ be the set of nodes whose shares $\mathcal{S}_{\text{DtVSS}}$ will verifiably encrypt. For each node $j \in \mathcal{H}_E$, use $\hat{s}(j)$ and $\hat{r}(j)$ as inputs to the VE scheme.

Figure 14: Dual-threshold AVSS simulator $\mathcal{S}_{\text{DtVSS}}$

(cf. Figure 13). Let $\mathcal{S}_{\text{DtVSS}}$ be corresponding simulator. $\mathcal{S}_{\text{DtVSS}}$ also uses Pedersens' polynomial commitment from Figure 7 and the VE scheme from Appendix B to simulate $\mathcal{A}$'s view. We summarize $\mathcal{S}_{\text{DtVSS}}$ in Figure 14, and prove the following.

Lemma 3 (Dual-threshold AVSS)**.** *$\mathcal{A}$'s view in its interaction with $\mathcal{S}_{\text{DtVSS}}$ is computationally indistinguishable from its view in the real protocol.*

Proof. We will prove this using a sequence of hybrids using the verifiable encryption scheme from Appendix B.

Hybrid 0. This corresponds to the real-world execution.

Hybrid 1. Same as Hybrid 0, except we will change the NIZK proof of correct sharing of the VE scheme with a simulated proof. Hybrid 1 is indistinguishable from Hybrid 0 due to the zero-knowledge property of the NIZK scheme.

Hybrid 2 to Hybrid $k+1$. Without loss of generality let $\mathcal{H}_E = 1, 2, \ldots, k$. Hybrid $i+1$ for any $i \in [1, k]$ is the same as Hybrid i, except it swaps out VE of $s(i)$ and with VE of $\hat{s}(i)$. For each $i \in [1, k]$, Hybrid $i+1$ is indistinguishable from Hybrid i due to the CPA security of the VE scheme.

Hybrid $k+2$ to Hybrid $2k+1$. Hybrid $k+i+1$ for any $i \in [1, k]$ is the same as Hybrid $k+i$, except it swaps out the encryption of $r(i)$ and with encryption of $\hat{r}(i)$. For each $i \in [1, k]$, Hybrid $k+i+1$ is indistinguishable from Hybrid $k+i$ due to the CPA security of the ElGamal encryption scheme

Hybrid $2k+2$. Same as Hybrid $2k+1$, except change the Pedersen commitment $\{g^{s(i)} h^{r(i)}\}_{i \in [n]}$ to $\{g^{\hat{s}(i)} h^{\hat{r}(i)}\}_{i \in [n]}$. Using a similar argument as Proof of Lemma 1, Hybrid $2k+2$ is identically distributed to Hybrid $2k+1$.

Hybrid $2k+3$. Same as Hybrid $2k+2$, except we will change the simulated NIZK proof of correct sharing of the VE scheme with a real NIZK proof. Hybrid $2k+3$ is indistinguishable from Hybrid $2k+2$ due to the zero-knowledge property of the NIZK scheme. Moreover, Hybrid $2k+3$ is the simulated transcript. □