Synchronous Distributed Key Generation without Broadcasts

Nibesh Shrestha[1], Adithya Bhat[2], Aniket Kate[2], and Kartik Nayak[3]

[1]Rochester Institute of Technology, nxs4564@rit.edu
[2]Purdue University, {bhat24, aniket}@purdue.edu
[3]Duke University, kartik@cs.duke.edu

Abstract

Distributed key generation (DKG) is an important building block in designing many efficient distributed protocols. In this work, we initiate the study of communication complexity and latency of distributed key generation protocols under a synchronous network in a point-to-point network. Our key result is the first synchronous DKG protocol for discrete log-based cryptosystems with $O(\kappa n^3)$ communication complexity (κ denotes a security parameter) that tolerates $t < n/2$ Byzantine faults among n parties. We show two variants of the protocol: a deterministic protocol with $O(t\Delta)$ latency and randomized protocol with $O(\Delta)$ latency in expectation where Δ denotes the bounded synchronous delay. In the process of achieving our results, we design (1) a gradecast protocol with optimal communication complexity of $O(\kappa n^2)$ for linear-sized inputs and latency of $O(\Delta)$, (2) a primitive called "recoverable set of shares" for ensuring recovery of shared secrets, (3) an oblivious leader election protocol with $O(\kappa n^3)$ communication and $O(\Delta)$ latency , and (4) a multi-valued validated Byzantine agreement (MVBA) protocol with $O(\kappa n^3)$ communication complexity for linear-sized inputs and $O(\Delta)$ latency in expectation. Each of these primitives may be of independent interest.

1 Introduction

The problem of distributed key generation (DKG) is to set up a common public key and its corresponding secret keys among a set of participating parties without a trusted entity. DKG protocols are used to reduce the amount of trust assumptions placed in other cryptographic protocols such as threshold signatures [10,47] and threshold encryption schemes [16]. The latter can itself be used to implement random beacons [19], reduce the complexity of consensus protocols [3, 50], in multiparty computation protocols [29, 30], or to outsource management of secrets to multiple, semi-trusted authorities [20, 35].

Given its widespread application, we need efficient solutions for DKG. An ideal solution for DKG would have low communication complexity, low latency, optimal resilience, provide uniform randomness of generated keys such that the generated keys can be useful in a wider class of cryptosystems while being secure. In this work, we focus on the synchronous network setting where messages sent by a sender will arrive at a receiver within a known bounded delay Δ. Synchronous protocols have the advantage of tolerating up to a minority corruption. While a myriad of DKG protocols [13,26,28,41,44] have been proposed in this setting, existing solutions fall short in one way or the other. For example, Pedersen's DKG [44] produces non-uniform keys in the presence of the adversary, the DKG protocol due to Gennaro et al. [26] has a high latency as it requires additional secret sharing using Feldman's VSS [22], and the protocol due to Gurkhan et al. [28] does not generate keys for discrete log-based cryptosystems. In addition, all the DKG protocols considered in the synchronous model assume a *broadcast channel* (that provides a consensus abstraction) and invoke $\Omega(n)$ broadcasts, where n is the number of parties. Since the best known Byzantine consensus protocols with optimal resilience incur at least $O(\kappa n^3)$ communication where κ is a security parameter, instantiating a broadcast channel with state-of-the-art Byzantine broadcast [1, 18] or Byzantine agreement [33] trivially

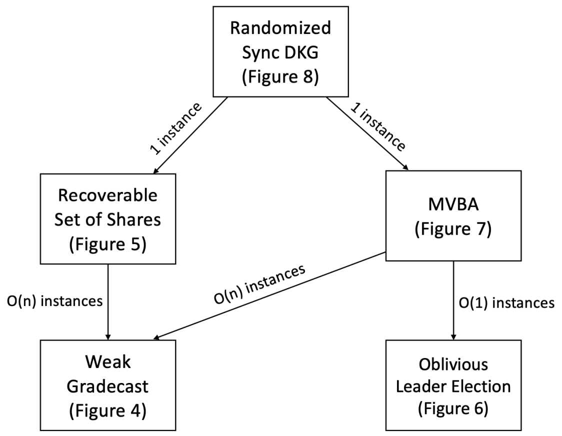
Randomized Sync DKG (Figure 8)
1 instance
1 instance
Recoverable Set of Shares (Figure 5)
MVBA (Figure 7)
O(n) instances
O(n) instances
O(1) instances
Weak Gradecast (Figure 4)
Oblivious Leader Election (Figure 6)

Figure 1: Overview of sub-protocols and their dependencies

blows up the communication complexity to $O(\kappa n^4)$[1]. Moreover, due to the use of multiple broadcast channel rounds, the latency of such protocols in a point-to-point network setting has not been explored. This leaves us with the following open question:

Can we design a synchronous distributed key generation protocol supporting a wide class of cryptosystems with $o(\kappa n^4)$ communication complexity, good latency and tolerating a minority corruption?

We answer this question positively by showing two DKG protocols for discrete log-based cryptosystems each with $O(\kappa n^3)$ communication complexity. The first protocol is deterministic and has $O(t\Delta)$ latency whereas the second protocol is randomized and has $O(\Delta)$ latency in expectation.

1.1 Key Technical Ideas and Results

Our DKG protocols do not use broadcast channels and use Byzantine consensus protocols in a non black-box fashion to achieve $O(\kappa n^3)$ communication. Compared to the broadcast channel-based DKG in the literature which requires $\Omega(n)$ broadcasts over two or more rounds, our protocols requires a single invocation of consensus instance. While DKG protocols [2,34] without broadcast channel assumption have been explored in the asynchronous model, they either incur high communication [34] or do not generate keys for discrete log-based cryptosystems [2]. In the synchronous model, we provide the first solutions to DKG without broadcast channel with all the desirable properties with $O(\kappa n^3)$ communication.

A typical approach among existing works is to perform n parallel verifiable secret sharings [22,43] such that all honest parties agree on a common set of qualified parties QUAL who correctly performed secret sharing and then compute final public key and secret keys from the secret shares of all parties in QUAL. In our protocols, we replace broadcast channels with weaker primitives such as multicast and gradecast [23,33]. Thus, parties first perform secret sharing by using these weaker primitives to identify a set of $n-t$ parties who correctly shared their secrets, where t is the fault tolerance. During the sharing phase, no consensus primitives is invoked to agree on set of qualified parties. The downside of this approach is that different honest parties may have different views regarding the acceptance of the shared secrets. As a result, different honest parties obtain different subsets of at least $n-t$ parties (say $\mathsf{AcceptList}_i$ for party P_i) who they accept to have performed secret sharing correctly. For DKG, it is required that all honest parties compute the final public key and secret keys from a common set of parties. Thus, we need to agree on a common set of parties too. Parties then use a Byzantine consensus primitive to agree on one common subset where the input is their individual $\mathsf{AcceptList}$. Once, the Byzantine consensus primitive terminates and outputs a common set

[1]Momose and Ren [39] gave a deterministic BA protocol with $O(\kappa n^2)$ communication with sub-optimal resilience of $t < (1-\epsilon)n/2$ for a small constant ϵ. Using their BA protocol to instantiate broadcast channels will result in DKG protocols with $O(\kappa n^3)$ communication with *sub-optimal* resilience and linear round complexity.

$\mathsf{AcceptList}_k$, the final public key and secret keys are computed from $\mathsf{AcceptList}_k$. Note that this approach requires only a single instance of Byzantine consensus.

Key Building Blocks

Communication optimal weak gradecast. As a building block, we first provide a communication optimal gradecast protocol satisfying the gradecast definition of Katz and Koo [33] who required a communication complexity of $O(\kappa n^3)$. (This definition is slightly weaker than the one presented by Feldman and Micali [23].) Specifically, we show the following result:

Theorem 1 (Informal). *Assuming a public-key infrastructure, there exists a gradecast protocol for an input of size ℓ with $O(n\ell + (\kappa + w)n^2)$ communication tolerating $t < n/2$ Byzantine faults, where κ is the security parameter and w is the size of witness.*[2]

Recoverable set of shares using weak gradecast. We use the gradecast primitive to perform communication efficient secret sharing. A consequence of using gradecast (instead of broadcast channels) is that parties may have different views regarding the acceptance of the shared secrets. For instance, each party P_i outputs a different set $\mathsf{AcceptList}_i$ and this set may also contain Byzantine parties. However, we still do guarantee that for any set output by any party (including Byzantine parties), there is a verifiable proof vouching that all parties in the subset have correctly shared their secret and these secrets are thus recoverable. We call this sub-protocol Recoverable set of shares. Using our communication efficient gradecast, our recoverable set of shares protocol can be achieved in $O(\kappa n^3)$ communication and constant latency.

Oblivious leader election. We design a communication efficient oblivious leader election (OLE) protocol (aka, common coin) with $O(\kappa n^3)$ communication and $O(\Delta)$ latency. The OLE protocol elects a common honest leader with probability at least $\frac{1}{2}$. Our OLE protocol uses Publicly Verifiable Secret Sharing scheme [14] whose security is not known under an adaptive adversaryy. This improves upon the OLE protocol of Katz and Koo [33] by a factor of n in communication at the cost of tolerating only static corruption. In particular, we show the following:

Theorem 2 (Informal). *Assuming a public-key infrastructure, there exists an oblivious leader election protocol with $O(\kappa n^3)$ communication and $O(\Delta)$ latency tolerating $t < n/2$ Byzantine faults, where κ is the security parameter.*

Agreeing on a recoverable set of shares using efficient multi-valued validated Byzantine agreement. Our next goal is to agree on one such set output by one of the parties. We stress that due to the proof associated with the output of the recoverable set of shares protocol, we can agree on the set output by any party, including a Byzantine party. However, here, the size of the set and its proof is linear, which can potentially worsen the communication complexity again. Thus, we need a consensus primitive that takes long messages as inputs and outputs one of the "valid" input values. Such a primitive is called *multi-valued validated Byzantine agreement* (MVBA) [12] in the literature.

MVBA was first formulated by Cachin et al. [12] to allow honest parties to decide on any externally valid values. Recent works [4, 36] have given communication efficient protocols for MVBA. For long messages of size ℓ, the protocol due to Abraham et al. [4] incurs $O((\ell + \kappa)n^2)$ ommunication and the protocol due to Luo et al. [36] incurs $O(\ell n + \kappa n^2)$. Both of these works assume a threshold setup. Without threshold setup assumptions, the communication blows up by a factor of n in all the above protocols.

To the best of our knowledge, no MVBA protocols have been formulated in the synchronous model tolerating $t < n/2$ Byzantine faults. A recent work [40] provides an efficient BA protocol for long messages. However, since it is a BA protocol, they output a value only when all honest parties start with the same large input. We give the first construction of MVBA protocol without threshold setup. Our MVBA protocol

[2]see Section 3 for details on witness.

incurs expected $O(\ell n^2 + \kappa n^3)$ communication and expected 18Δ time. Specifically, we show the following result:

Theorem 3 (Informal)**.** *Assuming a public-key infrastructure, there exists a multi-valued validated Byzantine agreement protocol for an input of size ℓ with expected $O(n^2\ell + (\kappa + w)n^3)$ communication and expected 18Δ tolerating $t < n/2$ Byzantine faults, where κ is the security parameter and w is the size of witness.*

Efficient distributed key generation. Using our recoverable set of shares protocol where parties output different sets of size at least $n - t$ parties and our MVBA protocol, honest parties can agree on a common set from which the final public key and secret keys are computed. In particular, we obtain a randomized DKG protocol with $O(\kappa n^3)$ communication and expected 36Δ time.

Theorem 4 (Informal)**.** *Assuming public-key infrastructure and random oracle, there exists a randomized protocol that solves secure synchronous distributed key generation tolerating $t < n/2$ Byzantine faults with expected $O((\kappa + w)n^3)$ communication and expected 36Δ time, where κ is the security parameter and w is the size of the witness.*

Although the randomized DKG protocol terminates in constant expected time, it can take a linear time in the worst case. In this case, the protocol incurs $O(\kappa n^4)$ communication. As an alternative, we provide a deterministic solution which incurs $O(\kappa n^3)$ communication. A recent work RandPiper [9] provides a BFT SMR protocol with $O(\kappa n^2)$ communication per epoch even for $O(n)$-sized input. Here, an epoch is a period which incurs some constant Δ time. In our deterministic DKG protocol, we execute the BFT SMR protocol for $t+1$ epochs with each epoch coordinated by a distinct leader. The leader proposes his set of $n-t$ parties along with the proof. Honest parties output the first committed set to compute the final public key and secret keys. In particular, we obtain the following result:

Theorem 5 (Informal)**.** *Assuming a public-key infrastructure and random oracle, there exists a deterministic protocol that solves secure synchronous distributed key generation tolerating $t < n/2$ Byzantine faults with $O((\kappa + w)n^3)$ communication and $18\Delta + (11(t + 1)\Delta)$, where κ is the security parameter and w is the size of the witness.*

A lower bound on the communication complexity of deterministic distributed key generation. We formalize a communication lower bound for a deterministic DKG protocol. Specifically, we show the following result:

Theorem 6. *There does not exist a deterministic protocol for secure distributed key generation tolerating t Byzantine parties with a communication complexity of at most $t^2/4$ messages.*

We remark that our deterministic DKG protocol incurs $O(\kappa n^3)$ communication and thus, our results are not tight. We leave open the problem of coming up with a better bound or designing a deterministic DKG protocol with improved communication complexity.

Summary of contributions. To summarize, we make the following contributions in this work:

1. As a warm up, we first present a secure DKG protocol assuming broadcast channels with two broadcast rounds in Section 4. This protocol lays a groundwork for our two DKG protocols which are the key contributions.

2. We present a communication optimal weak gradecast protocol in Section 5.

3. We present a recoverable set of shares protocol that optimizes round and communication complexity in the sharing phase in Section 6.

4. We present an oblivious leader election protocol in Section 7.

5. We present a multi-valued validated Byzantine agreement protocol in Section 8.

Table 1: **Comparison of related works on Distributed Key Generation**

		Network	**Resilience**	**Communication**	**Latency**	**Sim Sec.**	**Primitive**	**Dlog.**
Pedersen	[44]	sync.	1/2	$O(\kappa n^4)$	$O(t\Delta)$	✗	VSS	✓
Gennaro et al.	[26]	sync.	1/2	$O(\kappa n^4)$	$O(t\Delta)$	✓	VSS	✓
Neji et al.	[41]	sync.	1/2	$O(\kappa n^4)$	$O(t\Delta)$	✗	VSS	✓
ETHDKG	[46]	sync.	1/2	$O(\kappa n^4)$	$O(t\Delta)$	✗	VSS	✓
Gurkhan et al.	[28]	sync.	$\log n$	$O(\kappa n^3 \log n)$	$O(t\Delta)$	✗	PVSS	✗
NIDKG	[27]	sync.	1/2	$O(\kappa n^4)$	$O(t\Delta)$	✓	PVSS	✓
Hybrid-DKG	[31]	psync.	1/3	$O(\kappa n^4)$	$O(t)$ rnds	✓	VSS	✓
Kokoris et al.	[34]	async.	1/3	$O(\kappa n^4)$	$O(t)$ rnds	✗	VSS	✓
Abraham et al.	[2]	async.	1/3	$\tilde{O}(\kappa n^3)$	$O(1)$ rnds	✗	PVSS	✗
Das et al.	[15]	async.	1/3	$O(\kappa n^3 \log n)$	$O(\log n)$ rnds	✓	VSS	✓
Our work (rand.)		sync.	1/2	$O(\kappa n^3)$	$O(\Delta)$	✓*	VSS	✓
Our work (deter.)		sync.	1/2	$O(\kappa n^3)$	$O(t\Delta)$	✓	VSS	✓

κ is the security parameter denoting maximum of sizes of signatures, hashes, and other components used in the protocol. **Sim Sec.** means the protocol maintains secrecy which can be proven via a simulator. **Primitive** refers the cryptographic primitives used. **Dlog.** refers to the generation of keys for discrete log based cryptosystems. **rnds.** refers to rounds. Protocols expressed in terms of rounds are run in either partial synchrony or asynchrony and their time complexity cannot be expressed in terms of Δ. **rand.** implies randomized. **deter.** implies deterministic.

* Our common coin protocol uses PVSS whose simulation security is not known. While the simulator can use such a common coin as a black box, the randomized DKG protocol itself may not be simulation secure.

6. Using above primitives, we provide two communication and round efficient DKG protocols in Section 9.

7. Finally, we formalize a communication lower bound for a deterministic DKG in Section 10.

Limitations. In this work, we assume that the adversary is static. We do not know any adaptive attacks on our protocols, yet the existing proof techniques are insufficient to prove security against adaptive adversaries. Canetti et al. [13] show how to build adaptively secure DKG protocols and several of our techniques could be applicable in realizing their protocol in the point-to-point network setting. We also assume a q-SDH based CRS (Common Reference String) setup. This assumption is only used for bilinear accumulators which could be replaced with Merkle tree accumulators resulting in a $\log n$ multiplicative overhead in the communication complexity. We make use of Publicy verifiable secret sharing (PVSS) to build a common coin protocol and PVSS scheme is not known to be simulation secure. As a result, we are unsure of simulation security of our randomized DKG protocol.

2 Related Work

2.1 Related Works in Distributed Key Generation Literature

We review the most recent and closely related DKG protocols. An overview of the closely related work is provided in Table 1. While a myriad of DKG protocols [13, 21, 26–28, 41, 44, 46] have been proposed in the synchronous model, all of these protocols assume a broadcast channel. All of these protocols invoke $\Omega(n)$ parallel broadcasts. A natural choice to instantiate the broadcast channels is via Byzantine consensus primitives such as Byzantine Broadcast [1,18] or Byzantine agreement [33]. To the best of our knowledge, all deterministic Byzantine consensus protocols incur $O(\kappa n^3)$ communication without threshold signatures and $t+1$ rounds [6]. For randomized consensus protocols, the best known protocol with optimal resilience in this setting is Katz and Koo [33] which incurs $O(\kappa n^4)$ communication. Although, randomized consensus protocols terminate in expected constant rounds, n parallel instances of randomized consensus requires $\log n$ rounds to terminate. For the sake of simplicity, we assign a communication of $O(\kappa n^4)$ and $O(t\Delta)$ rounds for the DKG protocols that use broadcast channel in Table 1. Compared to all these protocols, our protocols do not use broadcast channel and use Byzantine consensus protocols. In fact, our protocols require a single consensus invocation and incurs $O(\kappa n^3)$ communication and expected constant rounds for randomized protocol and $O(t\Delta)$ rounds for deterministic protocol. Our protocols are secure against static failures and generates uniform keys for discrete logarithm based cryptosystems.

Pedersen [44] introduced the first efficient DKG protocol for discrete log cryptosystems in the synchronous setting. Their protocol is based on n parallel invocations of Feldman VSS [22]. Gennaro et al. [26] showed that Pedersen's DKG protocol can be biased by an adversary to generate non-uniform keys. To remove the bias, they proposed a new DKG protocol that requires additional secret sharing rounds; hence, is less efficient. Canneti et al. [13] extended Gennaro et al.'s DKG to handle adaptive corruptions.

Neji et al. [41] presented an efficient DKG protocol to remove the bias without the additional secret sharing round. However, in their protocol, honest parties still need to agree on whether to perform reconstruction for a secret shared by a party which requires additional consensus invocation. While they provide a simulator based proof to prove secrecy property of their protocol, we believe their protocol is not simulatable as an adversary playing last can influence the distribution.

Gurkhan et al. [28] presented DKG protocol without a complaint phase by using publicly verifiable secret sharing(PVSS) [14] scheme. However, they tolerate only $\log n$ Byzantine faults and does not generate keys for discrete-logarithms based cryptosystems; reducing its usefulness.

Recently, Groth [27] presents a non-interactive DKG protocol with a refresh procedure that allows refreshing the secret key shares to a new committee. Erwig et al. [21] considers large scale non-interactive DKG protocol and handles mobile Byzantine faults. Both of above protocols assume broadcast channels.

Several other works tackle the DKG problem from different angels. Kate et al. [32] reduced the size of input to the broadcast channel from $O(n)$ to $O(1)$ by using polynomial commitments [32]. Tomescu et al. [49] reduce the computational cost of dealings in Kate et al. [31] at the cost of a logarithmic increase in communication cost. Schindler et al. [46] instantiate the broadcast channel with the Ethereum blockchain. In Table 1, we replaced the Ethereum blockchain with Byzantine consensus primitives for fair comparison.

Kate et al. [31] gave the first practical DKG protocol in the partially synchronous communication model which requires $3t + 2f + 1$ parties to tolerate t Byzantine faults and f crash faults. Kokoris-Kogias et al. [34] gave the first DKG protocol in asynchronous communication model with optimal resilience ($t < n/3$). Their protocol has $O(\kappa n^4)$ communication and $O(t)$ rounds overhead. Recently, Abraham et al. [2] gave an improved DKG protocol with $O(\kappa n^3)$ communication and expected $O(1)$ round complexity. However, their protocol uses PVSS and hence does not generate keys for dlog-based cryptosystems. Concurrent to our work, Das et al. [15] gave the dlog-based DKG protocol with $O(\kappa n^3 \log n)$ communication and expected $O(\log n)$ round complexity in the asynchronous communication model.

2.2 Related Works in Byzantine Agreement Literature

There has been a long line of work in improving communication and round complexity of consensus protocols [1, 4, 11, 24, 33, 39, 48, 50]. We review the most recent and closely related works.

Multi-valued validated Byzantine agreement was first introduced by Cachin et al. [12] to allow honest parties to agree on any externally valid values. Their protocol works in asynchronous communication model and has optimal resilience ($t < n/3$) with $O(\ell n^2 + \kappa n^2 + n^3)$ communication for input of size ℓ. Later, Abraham et al. [4] gave an MVBA protocol with optimal resilience and $O(\ell n^2 + \kappa n^2)$ communication in the same asynchronous setting. Luo et al. [36] extended the work of Abraham et al. [4] to handle long messages of size ℓ with a communication complexity of $O(\ell n + \kappa n^2)$. All of these protocols assumed threshold signatures. In the absence of threshold signatures, the communication complexity blows up by a factor of n in all of these protocols.

To the best of our knowledge, no MVBA protocol has been formulated in the synchronous setting tolerating $t < n/2$ Byzantine faults. Our MVBA protocol incurs $O(\ell n^2 + \kappa n^3)$ for inputs of size ℓ and does not assume threshold signatures and terminates in expected constant rounds.

Table 2: **Comparison of related works on MVBA with ℓ-bit input**

		Network	**Res.**	**Communication**	**Latency**	**Assumption**
Cachin et al.	[12]	async.	1/3	$O(\ell n^2 + \kappa n^2 + n^3)$	$O(1)$ rnds	Threshold setup
VABA	[4]	async.	1/3	$O(\ell n^2 + \kappa n^2)$	$O(1)$ rnds	Threshold setup
DUMBO-MVBA	[36]	async.	1/3	$O(\ell n + \kappa n^2)$	$O(1)$ rnds	Threshold setup
Our work		sync.	1/2	$O(\ell n^2 + \kappa n^3)$	36Δ (exp)	PKI

κ is the security parameter denoting maximum of sizes of signatures, hashes, and other components used in the protocol. **Res.** refers to the number of Byzantine faults tolerated in the system. **rnds** refers to rounds and exp stands for "in expectation".

Our MVBA protocol can also be used for binary inputs as a Binary Byzantine Agreement (BBA) protocol tolerating $t < n/2$ Byzantine faults and terminating in expected $O(\Delta)$ rounds. Feldman and Micali [24] were the first to give a BBA protocol that terminates in constant expected rounds. Their protocol works in plain authenticated model without PKI and tolerates $t < n/3$ Byzantine faults (which is optimal). In the authenticated setting, Katz and Koo [33] gave a BBA protocol tolerating $t < n/2$ Byzantine faults terminating in expected constant rounds. Their protocol incurs $O(\kappa n^4)$ communication and terminates in expected 4 epochs. We extend the BBA protocol of Katz and Koo [33] and reduce its communication by linear factor while handling multi-valued input by designing a communication optimal gradecast protocol. We also reduce its round complexity by a factor of 2. A simple and efficient BBA tolerating $t < n/3$ Byzantine faults in the authenticated model was given by Micali [38]. Abraham et al. [1] reduced the round complexity of BBA protocol to expected 10 rounds. However, their protocol required a threshold setup to generate a perfect common coin; a perfect common coin ensures all honest parties output the same random value. Compared to their work, our work does not require a threshold setup and executes with a weak common coin while terminating in expected 14 rounds (equivalent of 18Δ).

3 Model and Preliminaries

We consider a system consisting of n parties in a reliable, authenticated all-to-all network, where up to $t < n/2$ parties can be Byzantine faulty. The model of corruption is static i.e., the adversary picks the corrupted parties before the start of protocol execution. The Byzantine parties may behave arbitrarily. A non-faulty party is said to be *honest* and executes the protocol as specified.

Messages exchanged between parties may take at most Δ time before they arrive, where Δ is a known maximum network delay. To provide safety under adversarial conditions, we assume that the adversary is capable of delaying the message for an arbitrary time upper bounded by Δ. In addition, we assume all honest parties have clocks moving at the same speed. They also start executing the protocol within Δ time from each other. This can be easily achieved by using the clock synchronization protocol [1] once at the beginning of the protocol.

Δ Synchrony Model. Our protocols are expressed in a Δ synchrony model where all parties execute an *epoch* for a certain amount of Δ time and all honest parties enter and exit an epoch within Δ time of each other. Within an epoch, parties are allowed to execute protocol steps when *events* are triggered i.e., when certain messages are received. This model differs from *lock-step* synchrony model [1, 18, 33] where honest parties are synchronized in each *round* and parties execute protocol steps only at the start of the round. In this regard, Δ synchrony model requires lesser time to execute a protocol.

PKI. Each party P_i has a public key. The parties and their public keys are common knowledge. We make use of digital signatures and PKI to prevent spoofing and replays and to validate messages. Message x sent by a party P_i is digitally signed by P_i's private key and is denoted by $\langle x \rangle_i$. We denote $H(x)$ to represent invocation of the random oracle H on input x.

Equivocation. Two or more messages of the same *type* but with different payload sent by a party is considered an equivocation. In order to facilitate efficient equivocation checks, the sender sends the payload along with signed hash of the payload. When an equivocation is detected, broadcasting the signed hash suffices to prove equivocation by the sender.

Setup. Let p be a prime number that is $\mathsf{poly}(\kappa)$ bits long, and $\mathbb{G}$ be a group of order p such that it is computationally infeasible except with negligible probability in κ to compute discrete log. Let $\mathbb{Z}_p$ denote its scalar field. Moreover, let g and h denote the generators of $\mathbb{G}$ where $a \in \mathbb{Z}_p$ such that $g^a = h$ is not known to any t subset of the nodes.

We make the standard computational assumption on the infeasibility to compute discrete logarithms called the discrete-log assumption [26]. In particular, we assume that the adversary is unable to compute discrete logarithms modulo large (based on the security parameter κ) primes.

3.1 Definitions

A distributed protocol DKG performed by n parties $(P_1, \ldots, P_n)$ generates private outputs $(x_1, \ldots, x_n)$ called the *shares* and a public output y.

Definition 3.1 (Secure Distributed Key Generation for Dlog based cryptosystems [26])**.** *A dlog based DKG protocol that distributes a secret x among n parties through shares $(x_1, \ldots, x_n)$ where x_i is a share output to party P_i is t-secure if in the presence of an adversary that corrupts up to t parties, the following requirements for correctness and secrecy are maintained.*

Correctness.

C1. *All subsets of $t+1$ shares provided by honest parties define the same unique secret key $x \in \mathbb{Z}_p$.*

C2. *All honest parties have the same value of public key $y = g^x \in \mathbb{G}$, where $x \in \mathbb{Z}_p$ is the unique secret guaranteed by (C1).*

C3. *x is uniformly distributed in $\mathbb{Z}_p$ (and hence y is uniformly distributed in $\mathbb{G}$).*

Secrecy. *No information on x can be learned by the adversary except for what is implied by the value $y = g^x$.*

More formally, the secrecy condition is expressed in terms of simulatability: for every (probabilistic polynomial-time) adversary $\mathcal{A}$ that corrupts up to t parties, there exists a (probabilistic polynomial-time) simulator $\mathcal{S}$, such that on input an element $y \in \mathbb{G}$, produces an output distribution which is polynomially indistinguishable from $\mathcal{A}$'s view of a run of the DKG protocol that ends with y as its public key output.

3.2 Primitives

In this section, we present several primitives used in our protocols.

Linear erasure and error correcting codes. We use standard $(t+1, n)$ Reed-Solomon (RS) codes [45]. This code encodes $t+1$ data symbols into code words of n symbols using ENC function and can decode the $t+1$ elements of code words to recover the original data using DEC function. More details on ENC and DEC are provided in Appendix A.1.

Cryptographic accumulators. A cryptographic accumulator scheme constructs an accumulation value for a set of values using Eval function and produces a witness for each value in the set using $\mathsf{CreateWit}$ function. Given the accumulation value and a witness, any party can verify if a value is indeed in the set using Verify function. More details on these functions are provided in Appendix A.2.

In this paper, we use *collision free bilinear accumulators* from Nguyen [42] as cryptographic accumulators which generates constant sized witness. Note that bilinear accumulators of Nguyen [42] requires q-SDH assumption. We can use Merkle trees [37] instead of q-SDH assumption at the expense of $O(\log n)$ multiplicative communication complexity.

Publicly Verifiable Secret Sharing – PVSS. A publicly verifiable secret sharing (PVSS) scheme allows any party to verify the correctness of the shares distributed by the dealer. Here, we assume existence of an aggregatable PVSS scheme which allows multiple PVSS sharing transcripts to be aggregated into a single transcript. One such scheme is the enhanced PVSS scheme of Gurkhan et al. [28] which allows aggregating linear number of PVSS transcripts to a single aggregated transcript whose size is still linear. Below, we provide interfaces to an aggregatable PVSS scheme.

- $\mathsf{PVSS.Setup}(\kappa, \mathsf{aux})$: Generates the scheme parameters $\mathsf{PVSS.pp}$. $\mathsf{PVSS.pp}$ is an implicit input to all other algorithms.
- $\mathsf{PVSS.Sh}(s)$: A probabilistic algorithm run by party P_i that takes as input a secret s and outputs a PVSS transcript pvss. The pvss also contains a description of the party who sent it.
- $\mathsf{PVSS.ShVerify}(\mathsf{pvss})$: A deterministic algorithm run by party P_i that returns 1 if it is convinced that the transcript pvss of party P_j is valid.
- $\mathsf{PVSS.Aggregate}(\mathcal{D})$: An algorithm run by Party P_i that takes as input a set $\mathcal{D}$ containing at least $t+1$ PVSS transcripts from different parties and outputs a aggregated transcript $\mathsf{agg\_pvss}$.
- $\mathsf{PVSS.Verify}(\mathsf{agg\_pvss})$: A deterministic algorithm that returns 1 if and only if the aggregated transcript $\mathsf{agg\_pvss}$ contains PVSS transcript that pass verification from at least $t+1$ different parties. Returns 0 otherwise.
- $\mathsf{PVSS.GetShare}(\mathsf{agg\_pvss}, \mathsf{sk}_i)$: A probabilistic algorithm run by Party P_i that takes as input an aggregated transcript, a secret key and returns its share share_i and a proof π_i. The share also contains a description of the party who sent it.
- $\mathsf{PVSS.VerifyShare}(\mathsf{agg\_pvss}, \mathsf{share}_j, \pi_j)$: A deterministic algorithm run by party P_i that takes as input aggregated transcript, a public key, a share, and a proof from Party P_j, and returns 1 to indicate acceptance or 0 to indicate rejection.
- $\mathsf{PVSS.Recon}(\mathcal{R})$: Given a set $\mathcal{R}$ of $t+1$ valid secret shares, reconstruct the shared secret.

Non-Interactive Proof-of-Equivalence of Commitments [31]. Given commitments $\mathcal{C}_{\langle g\rangle}(s) = g^s$ and $\mathcal{C}_{\langle g,h\rangle}(s,r) = g^s h^r$ to the same value s for generators $g, h \in \mathbb{G}$ and $s, r \in \mathbb{Z}_p$, a prover proves that she knows s and r such that $\mathcal{C}_{\langle g\rangle}(s) = g^s$ and $\mathcal{C}_{\langle g,h\rangle}(s,r) = g^s h^r$.

We denote this by $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}(s, r, g, h, \mathcal{C}_{\langle g\rangle}(s), \mathcal{C}_{\langle g,h\rangle}(s,r)) = \pi_{\equiv\mathsf{Com}} \in \mathbb{Z}_p^3$. A full construction of $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ is provided in Appendix A.3.

Normalizing the length of cryptographic building blocks. Let λ denote the security parameter, $\kappa_h = \kappa_h(\lambda)$ denote the hash size, $\kappa_a = \kappa_a(\lambda)$ denote the size of the accumulation value and witness of the accumulator and $\kappa_v = \kappa_v(\lambda)$ denote the size of secret share and witness of a secret. Further, let $\kappa = \mathsf{max}(\kappa_h, \kappa_a, \kappa_v)$; we assume $\kappa = \Theta(\kappa_h) = \Theta(\kappa_v) = \Theta(\kappa_a) = \Theta(\lambda)$. Throughout the paper, we can use the same parameter κ to denote the hash size, signature size, accumulator size and secret share size for convenience.

4 Warm Up: Secure Distributed Key Generation with Two Broadcast Rounds

We first present a secure DKG protocol assuming a broadcast channel motivated from Gennaro et al. DKG [26]. The presented DKG reduces the number of required rounds with broadcast to two, which is a significant improvement over [26] requiring three broadcast rounds in the best case and five broadcast

Sharing Phase

1. **Deal.** Each party (as a dealer) P_i selects two random polynomials $f_i(y)$, $f'_i(y)$ over $\mathbb{Z}_p$ of degree t:

$$f_i(y) = a_{i0} + a_{i1}y + \cdots + a_{it}y^t, \qquad f'_i(y) = b_{i0} + b_{i1}y + \cdots + b_{it}y^t$$

Let $s_i = a_{i0} = f_i(0)$. Party P_i posts $\mathcal{C}_{ik} = g^{f_i(k)}h^{f'_i(k)}\ \forall k \in \{1, \ldots, n\}$ on the broadcast channel. Party P_i computes the secret shares $s_{ij} = f_i(j)$, $s'_{ij} = f'_i(j)$ and sends s_{ij}, s'_{ij} privately to $P_j\ \forall j \in [n]$.

2. **Blame.** Each party P_i verifies that the commitment vector contains a t degree polynomial (Equation (2)). For $j \in [n]$, check if

$$g^{s_{ji}} \cdot h^{s'_{ji}} = \mathcal{C}_{ji} \tag{1}$$

$$\prod_{k=1}^{n} \mathcal{C}_{jk}^{\mathsf{Code}_k} = 1_{\mathbb{G}}, \text{ where } \{\mathsf{Code}_1, \ldots, \mathsf{Code}_n\} \in C^{\perp} \tag{2}$$

If the check fails for (dealer) party P_j, send $\langle \mathsf{blame}, j\rangle_i$ to all parties and collect all the blames.

3. **Forward blame.** If more than t blame messages are collected for party P_j as the dealer in the previous step, do not send anything for dealer P_j until the Decide step (Step 6).

Otherwise, for every $\langle \mathsf{blame}, j\rangle_k$ received from party P_k, forward the blame messages to the dealer P_j.

4. **Open.** Each party P_i, who as a dealer, received $\langle \mathsf{blame}, i\rangle_k$ from any party P_j, sends valid secret shares s_{ik}, s'_{ik} (that verifies Equation (1)) to party P_j.

5. **Vote.** If in Step 2, a party P_i received $\leq t$ $\langle \mathsf{blame}, j\rangle_k$ messages and party P_j sent valid secret shares s_{jk}, s'_{jk} for every $\langle \mathsf{blame}, j\rangle_k$ it forwarded to party P_j, send a vote $\langle \mathsf{vote}, j\rangle_i$ to party P_j. Forward the secret shares s_{jk}, s'_{jk} to party P_k.

6. **Decide.** If party P_i, as a dealer, receives $t+1$ $\langle \mathsf{vote}, i\rangle$ messages, post the vote-certificate on the broadcast channel.

Each party P_i marks a party P_j qualified if it receives a vote-certificate for party P_j on the broadcast channel; otherwise the party is disqualified. Party P_i builds a set of non-disqualified parties $\mathtt{QUAL}$.

Generating Public key

7. Party P_i sets its share of the secret as $x_i = \sum_{j\in\mathtt{QUAL}} s_{ji}$, and computes $x'_i = \sum_{j\in\mathtt{QUAL}} s'_{ji}$, $\mathcal{C}_{\langle g\rangle}(x_i) = g^{x_i}$, $\mathcal{C}_{\langle g,h\rangle}(x_i, x'_i) = g^{x_i}h^{x'_i}$ and $\pi_{\equiv\mathsf{Com}\,i} = \mathsf{NIZKPK}_{\equiv\mathsf{Com}}(x_i, x'_i, g, h, \mathcal{C}_{\langle g\rangle}(x_i), \mathcal{C}_{\langle g,h\rangle}(x_i, x'_i))$. Party P_i sends $(\mathcal{C}_{\langle g\rangle}(x_i)$, $\pi_{\equiv\mathsf{Com}\,i})$ to all parties.

8. Upon receiving a tuple $(\mathcal{C}_{\langle g\rangle}(x_j)$, $\pi_{\equiv\mathsf{Com}\,j})$, compute $\mathcal{C}_{\langle g,h\rangle}(x_j, x'_j) = g^{x_j}h^{x'_j}$ locally as follows:

$$g^{x_j}h^{x'_j} = \prod_{m\in\mathtt{QUAL}} \mathcal{C}_{mj} \tag{3}$$

Ensure $\pi_{\equiv\mathsf{Com}\,j}$ verifies $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ between $\mathcal{C}_{\langle g\rangle}(x_j)$ and $\mathcal{C}_{\langle g,h\rangle}(x_j, x'_j)$.

9. Upon receiving $t+1$ valid g^{x_j} values, perform Lagrange interpolation in the exponent to obtain $y = g^x$. Output y as the public key and x_i as the private key.

Figure 2: Secure distributed key generation in dlog-based cryptosystems

rounds otherwise.[3] In later sections, we replace the broadcast channel with our own Byzantine consensus primitives to design communication and round efficient DKG protocols. Gennaro et al. [26] presented a secure DKG protocol that produces uniform public keys based on Pedersen's VSS [43].

In their protocol, each party, as a dealer, selects a secret uniformly at random and shares the secret using Pedersen's VSS protocol. Since Pedersen's VSS provides information theoretic secrecy guarantees, the adversary has no information about the public key and hence cannot bias it. At the end of the secret sharing, a set of qualified parties QUAL who correctly shared their secret is defined. Once the set QUAL is fixed, parties in set QUAL invoke an additional round of secret sharing using Feldman's VSS [22] to generate the final public key. While this approach ensures generation of uniform keys and maintains secrecy, it adds additional overhead as it incurs more latency and communication to perform additional secret sharing. In addition to the above overhead, Pedersen VSS requires three broadcast rounds. In particular, parties post the commitment, complaints and secret shares corresponding to the complaints on to the broadcast channel during the sharing phase.

We improve upon the DKG protocol of Gennaro et al. [26] in the following ways. The improved protocol is described in Figure 2.

Improving latency in the sharing phase. We improve latency by reducing information posted on the broadcast channel by using improved eVSS (iVSS) protocol [9] which requires only 2 broadcast rounds.[4] Reducing the broadcast rounds greatly improves latency as broadcast channels are generally instantiated using Byzantine broadcast or Byzantine agreement protocols which have worst-case linear round complexity.

In iVSS, the dealer posts commitments on the broadcast channel and privately sends the secret shares to each party. Instead of posting the complaints on the broadcast channel, parties multicast blame message if they receive invalid secret shares or receive no secret shares at all. Parties then forward all blame messages to the dealer[5]. The dealer is expected to send secret shares corresponding to the blame messages (i.e., secret shares s_{ij}, s'_{ij} if a P_j sent blame message against dealer P_i). If the dealer sends all secret shares corresponding to the blame message it forwarded, a party sends a vote message to the dealer. Upon receiving $t+1$ vote messages, the dealer posts a vote-certificate containing $t+1$ vote messages. Honest parties consider the dealer to be honest if they see the vote-certificate on the broadcast channel.

Observe that using iVSS scheme, the dealer posts only the commitment and vote-certificate on the broadcast channel. This improves the sharing phase by one broadcast round.

Using commitments to evaluations instead of commitments to coefficients. In VSS such as Pedersen's VSS and Feldman's VSS and thus in [26], commitments to the secret share are commitments to the coefficients of the t-degree polynomial, which imply verifying a share requires $O(t)$ computations. This results in $O(nt)$ computations per VSS instance in the complaint stage (where every node verifies opening of up to t complaints) and during reconstruction. SCRAPE [14, Section 2.1] showed how to commit (using discrete log commitments) to evaluations instead of coefficients of the polynomial and verify that the committed evaluations are of a degree t polynomial by using the property of coding schemes: if C is the code space for an (n,t) sharing, then by sampling a vector $\{\mathsf{Code}_1, \ldots, \mathsf{Code}_n\} \in C^{\perp}$, we can check that the Pedersen's commitments to the evaluations are an (n,t) sharing (see Equation (1)). If λ is $\log_g h$, then commitments to evaluations form a polynomial $g^f h^{f'} = g^{f+\lambda f'}$ which is another (n,t) polynomial thereby allowing to use the coding technique. This is an information-theoretic technique and therefore does not affect the security of the underlying VSS.

Removing additional secret sharing while generating public key. We remove the additional secret sharing performed using Feldman's VSS by taking an alternate approach [31]. Instead of executing an additional secret sharing, assuming random oracle, we make use the NIZK proof of equivalence of commitments

[3]Using NIZK similar to us, the number of rounds for Gennaro et al. DKG [26] can be reduced to two in the best case and three otherwise in a rather straightforward manner; however, reducing two broadcast rounds in all situations is the key challenge here.

[4]Alternatively, we can use broadcast optimal VSS protocol of Backes et al. [7] which has 2 broadcast rounds. We prefer iVSS protocol for its simplicity.

[5]In an implementation, we can only forward up to t blames instead of all the blames.

$\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ to generate the public key. This approach does not require additional secret sharing via Feldman's VSS. Once the sharing phase is completed, a set of qualified parties $\mathtt{QUAL}$ is finalized. Then, each party P_i computes its share of the shared secrets i.e., $x_i = \sum_{P_j \in \mathtt{QUAL}} s_{ji}$ and $x'_i = \sum_{P_j \in \mathtt{QUAL}} s'_{ji}$ along with commitments $\mathcal{C}_{\langle g \rangle}(x_i)$, $\mathcal{C}_{\langle g,h \rangle}(x_i, x'_i)$. It then multicasts commitment of its share $\mathcal{C}_{\langle g \rangle}(x_i)$ and the corresponding $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ proof $\pi_{\equiv\mathsf{Com}\,i}$ to prove P_i knows x_i and x'_i.

All parties can compute the commitment $\mathcal{C}_{\langle g,h \rangle}(x_i, x'_i)$ locally as shown in Equation (3) and verify the correctness of commitment $\mathcal{C}_{\langle g \rangle}(x_i)$ using $\pi_{\equiv\mathsf{Com}\,i}$. The final public key Y is computed via Lagrange interpolation in the exponent using $t+1$ distinct commitments $\mathcal{C}_{\langle g \rangle}(x_i)$.

A proof of the following theorem is provided in Appendix B.1.

Theorem 7. *Under discrete-log assumption, the protocol in Figure 2 is a secure protocol for distributed key generation in dlog-based cryptosystem tolerating $t < n/2$ Byzantine faults.*

5 Communication Optimal Weak Gradecast

One of the main tools in the design of our communication efficient protocols is our communication optimal *gradecast* protocol. Gradecast (aka graded broadcast) is a relaxed version of broadcast introduced by Feldman and Micali [23] which can be obtained in constant number of rounds. Feldman and Micali [23] provided a gradecast protocol tolerating $t < n/3$ Byzantine faults in the *plain authenticated* model without PKI and digital signatures. Later, Katz and Koo [33] provided a slightly weaker gradecast protocol in the *authenticated* model tolerating $t < n/2$ Byzantine faults using PKI and digital signatures. The gradecast protocol of Katz and Koo [33] incurs $O(\kappa n^3)$ communication in the absence of threshold signatures. We present its communication optimal counterpart with $O(\kappa n^2)$ communication while propagating linear-sized input.

Definition 5.1 (Weak Gradecast [33]). *A protocol with a designated sender P_i holding an initial input v is a gradecast protocol tolerating $t < n/2$ Byzantine parties if the following conditions hold for any adversary controlling at most t Byzantine parties:*

1. *Each honest party P_j outputs a value v_j with a grade $g_j \in \{0, 1, 2\}$.*
2. *If the sender is honest, each honest party outputs v_i with a grade of 2.*
3. *If an honest party P_i outputs a value v with a grade of 2, then all honest parties output value v with a grade of ≥ 1.*

Our gradecast (refer Figure 4) implements weaker gradecast [33] (Definition 5.1) which relaxes gradecast [23] when no honest party outputs a grade of 2 and allows honest parties to output different values with a grade of 1. In particular, when an honest party P_j outputs a value v with a grade of 1, our primitive allows other honest parties to output a different value v' with a grade of 1 as long as no honest party outputs a value with a grade of 2. This weaker gradecast suffices for our purpose. In Appendix C, we show a quadratic lower bound on the communication complexity of weak gradecast for completeness.

$\mathsf{Deliver}(\mathsf{mtype}, m, z_e, e)$:

1. Partition input m into $t+1$ data symbols. Encode the $t+1$ data symbols into n code words $(s_1, \ldots, s_n)$ using ENC function. Compute witness $w_j \; \forall s_j \in (s_1, \ldots, s_n)$ using $\mathsf{CreateWit}$ function. Send $\langle \mathsf{codeword}, \mathsf{mtype}, s_j, w_j, z_e, e \rangle_i$ to party $j \; \forall j \in [n]$.

2. If party P_j receives the first valid code word $\langle \mathsf{codeword}, \mathsf{mtype}, s_j, w_j, z_e, e \rangle_*$ for the accumulator z_e, forward the code word to all the parties.

3. Upon receiving $t+1$ valid code words for the accumulator z_e, decode m using DEC function.

Figure 3: **Deliver function**

Deliver. As a building block, we first present a Deliver function (refer Figure 3) used by an honest party to efficiently propagate long messages. This function is adapted from RandPiper [9] where linear-sized messages are propagated among all honest parties with $O(\kappa n^2)$ communication cost. The Deliver function enables efficient propagation of long messages using erasure coding techniques and cryptographic accumulators. The input parameters to the function are a keyword mtype, long message m, accumulation value z_e corresponding to message m and epoch e in which Deliver function is invoked. The input keyword mtype corresponds to message *type* containing long message m sent by its sender. In order to facilitate efficient leader equivocation, the input keyword mtype, hash of long message m, accumulation value z_e, and epoch e are signed by the sender of message m. We omit epoch parameter when the Deliver function is not invoked within an epoch.

Set $o_i = \perp$ and $g_i = \perp$. Set epoch-timer to 5Δ and start counting down. Each party P_i performs the following operations:

1. If party P_j is the designated sender, then it multicasts its input value v in the form of $\langle \mathsf{gcast}, v, z\rangle_j$ where z is the accumulation value of v.

2. If epoch-timer $\geq 3\Delta$ and party P_i receives $pr := \langle \mathsf{gcast}, v, z\rangle_j$, invoke $\mathsf{Deliver}(\mathsf{gcast}, pr, z, \perp)$. Set grade-timer to 2Δ and start counting down. When grade-timer expires and no party P_j equivocation has been detected, set $o_i = v$ and $g_i = 2$.

3. When epoch-timer expires, let v_i be the first value received. If $v_i = \perp$, set $o_i = \perp$ and $g_i = 0$, else if $o_i = \perp$, set $o_i = v_i$ and $g_i = 1$. Output (o_i, g_i).

4. **(At any time)** If equivocating hashes signed by party P_j are detected, multicast the equivocating hashes.

Figure 4: **Weak Gradecast with $O(\ell n + (\kappa + w)n^2)$ communication.**

The gradecast protocol is presented in Figure 4. In the protocol, the designated sender P_j sends value v by multicasting $\langle \mathsf{gcast}, v, z\rangle_j$ where z is the accumulation value for value v. We note that the size of input value v can be large. To facilitate efficient equivocation checks, the sender P_j signs $\langle \mathsf{gcast}, H(v), z\rangle$ and sends v separately. Whenever an equivocation by the sender is detected, multicasting signed hashes suffices to prove equivocation by the sender. The reduction in communication is obtained via the use of efficient erasure coding schemes [45], cryptographic accumulators [8] and broadcast of equivocating hashes (if any). Broadcasting of equivocating hashes been explored in several efficient BFT protocols [3, 5, 48]. In this protocol, we assume that parties start executing a protocol instance within Δ time of each other. We note that we embed the gradecast protocol directly in the following protocols without separately invoking the primitive. We present it in Figure 4 for intuition.

Fact 8. *If an honest party invokes* Deliver *at time τ for an object b sent by party P_j and no honest party has detected a party P_j equivocation by time $\tau + \Delta$, then all honest parties will receive object b by time $\tau + 2\Delta$.*

Proof. Suppose an honest party P_i invokes Deliver at time τ for an object b sent by party P_j. Party P_i must have sent valid code words and witness $\langle \mathsf{codeword}, \mathsf{mtype}, s_k, w_k, z_e, e\rangle_i$ computed from object b to every party $P_k \in \mathcal{P}$ at time τ. The code words and witness arrive at all honest parties by time $\tau + \Delta$.

Since no honest party has detected a party P_j equivocation by time $\tau + \Delta$, it must be that either honest parties will forward their code word $\langle \mathsf{codeword}, \mathsf{mtype}, s_k, w_k, z_e, e\rangle$ when they receive the code words sent by party P_i or they already sent the corresponding code word when they either invoked Deliver for object b or received the code word from some other party P_j. In any case, all honest parties will forward their epoch e code word corresponding to object b by time $\tau + \Delta$. Thus, all honest parties will have received $t+1$ valid code words for a common accumulation value z_e by time $\tau + 2\Delta$ sufficient to decode object b by time $\tau + 2\Delta$. □

Theorem 9. *The protocol in Figure 4 is gradecast protocol satisfying Definition 5.1.*

Proof. We first consider the case when an honest party P_i outputs a value v_i with a grade of 2. If an honest party P_i outputs a value v_i with a grade of 2, then it must have received value v_i at some time τ such that its epoch-timer $\geq 3\Delta$ and invoked Deliver to deliver value v, and set grade-timer to 2Δ. In addition, party

P_i did not detect any party P_j (the designated sender) equivocation by time $\tau + 2\Delta$. This implies no other honest party detected a party P_j equivocation by time $\tau + \Delta$. By Fact 8, all honest parties receive value v by time $\tau + 2\Delta$. In addition, since party P_i invoked $\mathsf{Deliver}$ at time τ, all honest parties receive a code word for value v by time $\tau + \Delta$. Thus, value v is the first value received by all honest parties and all honest parties output value v with a grade of ≥ 1.

Next, we consider the case when the designated sender (party P_j) is honest. Since, the sender is honest, it sends its input value v to all honest parties such that their $\mathsf{epoch\text{-}timer} \geq 3\Delta$. Thus, all honest parties invoke $\mathsf{Deliver}$ and set $\mathsf{grade\text{-}timer}$ to 2Δ. Moreover, the honest sender does not equivocate. This implies all honest parties output value v with a grade of 2.

The case where each honest party outputs a value with a grade $\in \{0, 1, 2\}$ is trivial by design. □

Lemma 10 (Communication Complexity)**.** *Let ℓ be the size of the input, κ be the size of accumulator, and w be the size of witness. The communication complexity of the protocol in Figure 4 is $O(n\ell + (\kappa + w)n^2)$.*

Proof. At the start of the protocol, the sender multicasts its value of size ℓ to all party P_j $\forall j \in [n]$ along with κ sized accumulator. This step incurs $O(n\ell + \kappa n)$. Invoking $\mathsf{Deliver}$ on an object of size ℓ incurs $O(n\ell + (\kappa + w)n^2)$, since each party multicasts a code word of size $O(\ell/n)$, a witness of size w and an accumulator of size κ. Thus, the overall communication complexity is $O(n\ell + (\kappa + w)n^2)$. □

6 Recoverable Set of Shares

In Section 4, we presented a secure DKG protocol by assuming broadcast channels in the sharing phase. In general, broadcast channels are instantiated using Byzantine Broadcast (BB) or Byzantine agreement (BA) protocols. To the best of our knowledge, all known BB and BA protocols tolerating $t < n/2$ Byzantine faults incur $O(\kappa n^3)$ communication in the absence of threshold signatures [1, 18, 33]. The secure DKG protocol required $2n$ broadcasts. Thus, instantiating broadcast channel using BB or BA protocols for our secure DKG protocol trivially incurs $O(\kappa n^4)$ communication. In this section, we present a slightly weaker sharing protocol by appropriately replacing the broadcast channel with multicast and our weak gradecast. This protocol completes in constant rounds and acts as a building block towards constructing the DKG. We call this protocol *Recoverable Set of Shares.*

In the sharing phase of our secure DKG protocol with broadcast channels (Figure 2), each honest party outputs a common set $\mathtt{QUAL}$ consisting of size at least $n-t$ parties such that the secrets shared by parties in set $\mathtt{QUAL}$ can be reconstructed. In more detail, honest parties have a common decision on which parties correctly shared their secret at the end of the sharing phase. Requiring this agreement was free in the presence of broadcast channels; however, under a point-to-point network, it blows up communication complexity.

Thus, in our protocol, we instead rely on the use of weaker primitives such as multicast and gradecast instead of consensus to share secrets. As a result, each honest party P_i may have a different view regarding the acceptance of the shared secret. Thus, each honest party P_i outputs a possibly different subset $\mathsf{AcceptList}_i$ of size at least $n - t$ parties which they accept to have shared the secret correctly i.e., party P_i observes the secrets shared by parties in $\mathsf{AcceptList}_i$ can be reconstructed. It is in this regard, we call our protocol *recoverable* set of shares as the secret shared by parties in $\mathsf{AcceptList}_i$ can be reconstructed independent of whether these parties are present in $\mathsf{AcceptList}_j$ for $j \neq i$.

We stress that in recoverable set of shares protocol, honest parties need not agree on a common set and may output a different subset of size at least $n-t$ parties which they believe have shared the secret properly. To ensure that the final keys for DKG are generated for a common set, parties need to agree on one such set. In the following section, we present a multi-valued validated Byzantine agreement protocol to agree on a common set.

Protocol Details. Each honest party P_i starts the recoverable set of shares protocol (refer Figure 5) with its $\mathsf{epoch\text{-}timer}_i$ set to 16Δ and starts counting down. At the start of the protocol, each honest party P_i selects two random t degree polynomials $f_i(y) = \sum_k a_{ik} y^k$ over $\mathbb{Z}_p$ and $f'_i(y) = \sum_k b_{ik} y^k$ over $\mathbb{Z}_p$ such that $f_i(0) = s_i$ and $f'_i(0) = s'_i$. Party P_i generates the commitment $\mathcal{C}_{ik} = g^{f_i(k)} h^{f'_i(k)}$ $\forall k \in \{1, \ldots, n\}$. Let $\mathsf{VSS}.\vec{C}_i$ represent

Set $\mathsf{epoch\text{-}timer}_i$ to 16Δ and start counting down. Each party P_i performs following operation:

1. **Distribute.** Each party P_i selects two random polynomials $f_i(y)$, $f'_i(y)$ over $\mathbb{Z}_p$ of degree t:
$$f_i(y) = a_{i0} + a_{i1}y + \cdots + a_{it}y^t, \qquad f'_i(y) = b_{i0} + b_{i1}y + \cdots + b_{it}y^t$$
Let $s_i = a_{i0} = f_i(0)$. Party P_i generates the commitment $\mathcal{C}_{ik} = g^{f_i(k)}h^{f'_i(k)}\ \forall k \in \{1, \ldots, n\}$. Let $\mathsf{VSS}.\vec{C}_i$ represent $\mathcal{C}_{ik}\ \forall k \in \{1, \ldots n\}$. Party P_i multicasts its proposal $\langle \mathsf{propose}, \mathsf{VSS}.\vec{C}_i, z_{pi}\rangle_i$. Part P_i computes the shares $s_{ij} = f_i(j)$, $s'_{ij} = f'_i(j)$ and sends s_{ij}, s'_{ij} to $P_j\ \forall j \in [n]$.
2. **Blame/Forward.** If $\mathsf{epoch\text{-}timer}_i \geq 14\Delta$ and party P_i receives commitment $\mathsf{comm}_j := \langle \mathsf{propose}, \mathsf{VSS}.\vec{C}_j, z_{pj}\rangle_j$ and valid secret share s_{ji}, s'_{ji} (i.e., satisfy Equation (1) with $\mathsf{VSS}.\vec{C}_j$) and, then invoke $\mathsf{Deliver}(\mathsf{propose}, \mathsf{comm}_j, z_{pj}, -)$. If no valid secret shares has been received from party P_j until $\mathsf{epoch\text{-}timer} \geq 13\Delta$, multicast $\langle \mathsf{blame}, j\rangle_i$ to all parties.
3. **Request open.** Wait until $\mathsf{epoch\text{-}timer}_i \geq 11\Delta$. Collect all blames received so far. If up to t blame are received for party P_j, forward the blame messages to party P_j. If more than t blames are received for party P_j then do not send anything until 5. If no blames for party P_j or party P_j equivocation has been detected, send $\langle \mathsf{vote}, H(\mathsf{comm}_j)\rangle_i$ to party P_j.
4. **Open.** Party P_i sends secret shares s_{ik}, s'_{ik} to party P_j, for every blame $\langle \mathsf{blame}, i\rangle_k$ received from party P_j.
5. **Vote.** Upon receiving valid secret shares s_{jk}, s'_{jk} for every $\langle \mathsf{blame}, j\rangle_k$ it forwarded and no party P_j equivocation has been detected, send $\langle \mathsf{vote}, H(\mathsf{comm}_j)\rangle_i$ to party P_j. Forward secret share s_{jk} to party P_k for every $\langle \mathsf{blame}, j\rangle_k$ it received.
6. **Vote cert.** Upon receiving $t+1$ distinct vote messages for comm_i (denoted by $\mathcal{C}(\mathsf{comm}_i)$), multicast $\langle \mathsf{vote\text{-}cert}, \mathcal{C}(\mathsf{comm}_i), z_{vi}\rangle_i$.
7. **Grade.** If $\mathsf{epoch\text{-}timer}_i \geq 5\Delta$ and party P_i receives the first $vc_j := \langle \mathsf{vote\text{-}cert}, \mathcal{C}(\mathsf{comm}_j), z_{vj}\rangle_j$, invoke $\mathsf{Deliver}(\mathsf{vote\text{-}cert}, vc_j, z_{vj}, -)$. Set $\mathsf{accept\text{-}timer}[j]$ to 2Δ and start counting down. When $\mathsf{accept\text{-}timer}[j]$ reaches 0, if no party P_j equivocation has been detected, set $\mathsf{AcceptList}_i[j] = 2$. Let $\mathcal{C}(\mathsf{comm}_{j,i})$ be the first vote certificate received from party P_j. If $\mathcal{C}(\mathsf{comm}_{j,i}) = \perp$, set $\mathsf{AcceptList}_i[j] = 0$, else if $\mathsf{AcceptList}_i[j] \neq 2$, set $\mathsf{AcceptList}_i[j] = 1$.
8. **Propose Grade.** Wait until $\mathsf{epoch\text{-}timer}_i \geq 3\Delta$. Multicast $\langle \mathsf{accept\text{-}list}, \mathsf{AcceptList}_i\rangle_i$.
9. **Verify and Ack.** Upon receiving $\langle \mathsf{accept\text{-}list}, \mathsf{AcceptList}_j\rangle_j$ from party P_j, if the following conditions hold send $\langle \mathsf{ack}, H(\mathsf{AcceptList}_j)\rangle_i$ to party P_j.
   (a) $|\{h \mid \mathsf{AcceptList}_j[h] = 2\}| \geq n - t$
   (b) If $\mathsf{AcceptList}_j[h] = 2$ then $\mathsf{AcceptList}_i[h] \geq 1\ \forall h \in [n]$.
10. **(Non-blocking) Equivocation.** If equivocating hashes signed by party P_j are detected, multicast the equivocating hashes.

Figure 5: **Recoverable Set of Shares**

$\mathcal{C}_{ik}\,\forall k \in \{1,\ldots n.\}$. Party P_i multicasts the commitment in the form of a proposal $\langle \mathsf{propose}, \mathsf{VSS}.\vec{C}_i, z_{pi}\rangle_i$ where z_{pi} is the accumulation value of $\mathsf{VSS}.\vec{C}_i$. In order to facilitate efficient equivocation checks, party P_i signs $\langle \mathsf{propose}, H(\mathsf{VSS}.\vec{C}_i), z_{pi}\rangle$ separately and sends $\mathsf{VSS}.\vec{C}_i$ separately. Party P_i also privately sends secret share s_{ij}, s'_{ij} to party P_j $\forall j \in [n]$.

If a party P_j receives valid secret share s_{ij}, s'_{ij} along with the proposal $\mathsf{comm}_i := \langle \mathsf{propose}, \mathsf{VSS}.\vec{C}_i, z_{pi}\rangle_i$ in a timely manner (such that its $\mathsf{epoch\text{-}timer}_j \geq 14\Delta$), it invokes $\mathsf{Deliver}(\mathsf{propose}, \mathsf{comm}_i, z_{pi}, -)$ to propagate the commitment $\mathsf{VSS}.\vec{C}_i$; otherwise party P_j multicasts $\langle \mathsf{blame}, i\rangle_j$. Observe that we ignore the epoch e parameter in $\mathsf{Deliver}$ as the current protocol is not executed in an epoch.

Party P_j waits to collect any blame messages sent by other parties. If no blame message or party P_i equivocation have been detected within the waiting time, party P_j sends a vote $\langle \mathsf{vote}, H(\mathsf{comm}_i)\rangle_j$ to party P_i. If up to t blame messages are received for P_i, P_j forwards the blame messages to party P_i. Party P_i then privately sends secret shares s_{ik}, s'_{ik} to party P_j, for every blame $\langle \mathsf{blame}, i\rangle_k$ received from party P_j. Upon receiving valid secret shares for all $\langle \mathsf{blame}, i\rangle_k$ it forwarded, party P_j sends a vote $\langle \mathsf{vote}, H(\mathsf{comm}_i)\rangle$ to party P_i and also forwards secret shares s_{ik}, s'_{ik} to party P_k.

Party P_i then waits to collect $t+1$ vote messages for $H(\mathsf{comm}_i)$, denoted by $\mathcal{C}(\mathsf{comm}_i)$. A certificate on the comm_i implies that secret s_i shared by party P_i can be reconstructed later. Party P_i then "gradecasts" $\langle \mathsf{vote\text{-}cert}$, $\mathcal{C}(\mathsf{comm}_i), z_{vi}\rangle_i$ where z_{vi} is the accumulation value of $\mathcal{C}(\mathsf{comm}_i)$. Similar to the proposal, the hash of the certificate is signed to allow for efficient equivocation checks. It is important to note that two different certificates for the same commitment comm_i is still considered an equivocation.

Invocation of gradecast on $\mathcal{C}(\mathsf{comm}_i)$ ensures that if the party P_i is honest, all honest parties output a common $\mathcal{C}(\mathsf{comm}_i)$ with a grade of 2 and if an honest party P_k output $\mathcal{C}(\mathsf{comm}_i)$ with a grade of 2, all other honest parties output the certificate with a grade of ≥ 1.

Note that all parties (at least all honest parties) are executing the secret sharing phase. Thus, at the end of gradecast step, each honest party outputs at least $n-t$ certificates with a grade of 2 and output at most t values with a grade ≤ 2. We call the list of grades for party P_j as $\mathsf{AcceptList}_j$. This list is a set of parties which party P_j observes to have shared their secret properly and each secret can be reconstructed. Party P_j then multicasts its $\mathsf{AcceptList}_j$ to all other parties. Party P_k then checks the validity of $\mathsf{AcceptList}_j$ by checking if (i) $|\{h \,|\, \mathsf{AcceptList}_j[h] = 2\}| \geq n-t$, and (ii) if $\mathsf{AcceptList}_j[h] = 2$ then $\mathsf{AcceptList}_k[h] \geq 1$ $\forall h \in [n]$. The first check ensures that $\mathsf{AcceptList}_j$ contains at least $n-t$ entries with $\mathsf{AcceptList}_j[h] = 2$. This check trivially satisfies for $\mathsf{AcceptList}$ sent by an honest party as each honest party receives at least $n-t$ certificates with a grade of 2. Later, the DKG protocols use secrets from parties in $\mathsf{AcceptList}_j$ such that $\mathsf{AcceptList}_j[h] = 2$ to compute the final keys. This is required to ensure security of DKG protocol. The second check ensures that all the secrets corresponding to $\mathsf{AcceptList}_j[h] = 2$ are recoverable; observe that if $\mathsf{AcceptList}_j[h] = 2$ then $\mathsf{AcceptList}_k[h] \geq 1$ due to weak gradecast properties. This implies party P_k has received a $\mathcal{C}(\mathsf{comm}_h)$ from party P_h and $\mathcal{C}(\mathsf{comm}_h)$ implies the secret shared by party P_h can be reconstructed. If the checks pass, party P_k sends $\langle \mathsf{ack}, H(\mathsf{AcceptList}_j)\rangle_k$ to party P_j. A set of $t+1$ ack ($\mathsf{ack\text{-}cert}$) messages for $\mathsf{AcceptList}_j$ (denoted by $\mathcal{AC}(\mathsf{AcceptList}_j)$) implies at least one honest party has verified that all the secrets corresponding to $\mathsf{AcceptList}_j[h] = 2$ can be recovered.

The idea of using gradecast to perform secret sharing has been explored before in the works of Feldman and Micali [23, 24] to generate common source of randomness. Compared to their work, our protocols work in authenticated model with $t < n/2$ resilience and invoke a single gradecast per secret sharing. Their protocols work in *unauthenticated* model without PKI with $t < n/4$ [23] and $t < n/3$ [24] resilience and involved multiple invocation of gradecast per secret sharing.

6.1 Security Analysis

Fact 11. *If an honest party sends* vote *for a commitment* comm*, then (i) all honest parties receive* comm*, (ii) all honest parties receive their valid secret shares corresponding to commitment* comm.

Proof. Suppose an honest party P_i sends a vote for commitment $\mathsf{comm}_k := \langle \mathsf{propose}, \mathsf{VSS}.\vec{C}_k, z_{pk}\rangle_k$ at time τ. Party P_i must have received up to t blame messages for party P_k. This implies at least one honest party P_j received valid secret share $s_{k,j}$ and commitment comm_k when its $\mathsf{epoch\text{-}timer}_j \geq 14\Delta$ and invoked $\mathsf{Deliver}(\mathsf{propose}, \mathsf{comm}_k, z_{pk}, -)$. Let τ' be the time when party P_j invoked $\mathsf{Deliver}(\mathsf{propose}, \mathsf{comm}_k, z_{pk}, -)$. The earliest party P_i sends a vote for comm_k is when it waits until its $\mathsf{epoch\text{-}timer}_i \geq 11\Delta$ and does not detect any equivocation by party P_k or any blame messages for party P_k.

Note that honest parties may start the protocol within Δ time. Thus, when $\mathsf{epoch\text{-}timer}_i = 11\Delta$ for party P_i, party P_j may have $10\Delta \leq \mathsf{epoch\text{-}timer}_j \leq 12\Delta$. In any case, the time when P_i waits until $\mathsf{epoch\text{-}timer}_i \geq 11\Delta$ corresponds to at least $\tau' + 2\Delta$. Since party P_i did not detect party P_k equivocation by time $\tau' + 2\Delta$, no honest party detected party P_k equivocation by time $\tau' + \Delta$. By Fact 8, all honest parties receive the commitment comm_k by time $\tau' + 2\Delta$. This proves part (i) of the Lemma.

For part (ii), party P_i can send vote on two occasions: (a) when it does not detect a party k equivocation or $\langle \mathsf{blame}, k\rangle$ until its $\mathsf{epoch\text{-}timer}_i \geq 11\Delta$, and (b) when party k sent valid secret shares for every $\langle \mathsf{blame}, k\rangle$ message it forwarded and does not detect any party k equivocation by time τ.

In case (a), party P_i did not detect a party k equivocation or $\langle \mathsf{blame}, k\rangle$ until its $\mathsf{epoch\text{-}timer}_i \geq 11\Delta$ at time τ. Observe that all honest parties must have received valid secret shares corresponding to the commitment comm_k when $\mathsf{epoch\text{-}timer} \geq 14\Delta$; otherwise party P_i must have received $\langle \mathsf{blame}, k\rangle$ by time τ (since honest parties start protocol with Δ time difference and send $\langle \mathsf{blame}, k\rangle$ if no valid secret shares are received until $\mathsf{epoch\text{-}timer} \geq 14\Delta$). Thus, all honest parties receive valid secret shares corresponding to commitment comm_k.

In case (b), party P_i receives valid secret shares from party P_k for every $\langle \mathsf{blame}, k\rangle$ (up to t blame) messages it forwarded and detected no party k equivocation by time τ. Observe that party P_i received $f \leq t$ $\langle \mathsf{blame}, k\rangle$ messages and received valid secret shares for every $\langle \mathsf{blame}, k\rangle$ message it forwarded. This implies at least $n - t - f$ honest parties have received valid shares for commitment comm_k from party P_k such that $\mathsf{epoch\text{-}timer} \geq 14\Delta$; otherwise, party P_i would have received more than f $\langle \mathsf{blame}, k\rangle$ message by the time its $\mathsf{epoch\text{-}timer}_i = 11\Delta$. Since, party P_i forwards f received secret shares corresponding to f received $\langle \mathsf{blame}, k\rangle$, all honest parties receive valid secret shares corresponding to commitment comm_k. □

Lemma 12. *If an honest party sends an* ack *for a grade list* $\mathsf{AcceptList}_j$*, then all honest parties have valid secret shares corresponding to* comm_h *for all h such that* $\mathsf{AcceptList}_j[h] = 2$.

Proof. Suppose an honest party P_i sends an ack for a grade list $\mathsf{AcceptList}_j$. Then, it must be that if $\mathsf{AcceptList}_j[h] = 2$ then $\mathsf{AcceptList}_i[h] \geq 1$ $\forall h \in [n]$. Party P_i sets $\mathsf{AcceptList}_i[h] \geq 1$ when it receives a vote certificate $\mathcal{C}(\mathsf{comm}_h)$. If there is a vote certificate $\mathcal{C}(\mathsf{comm}_h)$ for value comm_h, then at least one honest party (say party P_k) must have voted for comm_h. By Fact 11 part (ii), all honest parties have valid secret shares corresponding to commitment comm_h. Thus, all honest parties have valid secret shares corresponding to comm_h for all h such that $\mathsf{AcceptList}_j[h] = 2$. □

Lemma 13 (Liveness)**.** *Each honest party P_i will receive an* $\mathsf{ack\text{-}cert}$ *for its grade list* $\mathsf{AcceptList}_i$.

Proof. Consider an honest party P_i. Let τ be the time when party P_i starts the protocol. Party P_i will send valid commitment $\mathsf{VSS}.\vec{C}_i$ and secret share s_{ij} to party P_j $\forall j \in [n]$ at time τ. All honest parties will receive their valid secret shares s_{ij} and commitment comm_i by time $\tau + \Delta$. Since honest parties start the protocol within Δ time, all honest parties receive valid secret shares and commitment when their $\mathsf{epoch\text{-}timer} \geq 14\Delta$. Thus, no honest party will send $\langle \mathsf{blame}, i\rangle$.

Observe that up to t Byzantine parties can always send $\langle \mathsf{blame}, i\rangle$. Honest parties wait until their $\mathsf{epoch\text{-}timer} \geq 11\Delta$ to collect blame messages for any party. At worst, this time corresponds to $\tau + 6\Delta$. Honest parties forward $\langle \mathsf{blame}, i\rangle$ to party P_i which party P_i receives by time $\tau + 7\Delta$. Party P_i forwards valid secret shares to party P_j for every $\langle \mathsf{blame}, i\rangle$ message it received from party P_j which party P_j receives by time $\tau + 8\Delta$. Thus, party P_j will send vote for party P_i which party P_i receives by time $\tau + 9\Delta$. This implies party P_i collects $t + 1$ distinct vote messages by $\tau + 9\Delta$.

Party P_i send $\mathsf{vote\text{-}cert}$ message vc_i which all parties receive by time $\tau + 10\Delta$. Thus, all honest parties receive vc_i such that their $\mathsf{epoch\text{-}timer} \geq 5\Delta$ (since honest parties start the protocol within Δ time). Thus, all honest parties will invoke $\mathsf{Deliver}$ to deliver vc_i. Moreover, honest party P_i does not equivocate. Thus, all honest parties set $\mathsf{AcceptList}[i]$ to 2.

Observe that for an honest party P_i, all honest parties set $\mathsf{AcceptList}[i]$ to 2. Thus, for any honest party P_j, all honest parties set $\mathsf{AcceptList}[j]$ to 2. This implies all honest parties will have $|\{h \mid \mathsf{AcceptList}_j[h] = 2\}| \geq n - t$.

Next, we consider the case when a Byzantine party (say, party P_l) sends $\mathsf{vote\text{-}cert}$ message vc_l to only party P_i. If honest party P_i sets $\mathsf{AcceptList}_i[l] = 2$, it must be that party P_i invoked $\mathsf{Deliver}$ to propagate vc_l when its $\mathsf{epoch\text{-}timer}_i \geq 5\Delta$ at some time τ' and did not detect any party P_l equivocation by time $\tau' + 2\Delta$. This implies no honest party detected party P_l equivocation by time $\tau' + \Delta$. By Fact 8, all honest parties receive $\mathsf{vote\text{-}cert}$ for party P_l and set $\mathsf{AcceptList}[l] \geq 1$. Thus, for every $\mathsf{AcceptList}_i[h] = 2$ then $\mathsf{AcceptList}[h] \geq 1$ for all honest parties.

Observe that party P_i multicasts $\mathsf{AcceptList}_i$ when its $\mathsf{epoch\text{-}timer}_i \geq 3\Delta$. At this time all honest parties have $\mathsf{epoch\text{-}timer} \geq 2\Delta$ (Since honest parties start the protocol within Δ time). Thus, all honest parties will receive $\mathsf{AcceptList}_i$ when their $\mathsf{epoch\text{-}timer} \geq \Delta$ and since $\mathsf{AcceptList}_i$ satisfies both the conditions $|\{h \mid \mathsf{AcceptList}_i[h] = 2\}| \geq n - t$ and $\mathsf{AcceptList}_i[h] = 2$ then $\mathsf{AcceptList}[h] \geq 1$, all honest parties will send ack for the grade list $\mathsf{AcceptList}_i$ proposed by party P_i and party P_i will receive $\mathsf{ack\text{-}cert}$ for $\mathsf{AcceptList}_i$ by the time $\mathsf{epoch\text{-}timer}_i$ expires. □

Lemma 14 (Communication Complexity). *Let ℓ be the size of commitment comm, κ be the size of secret share and accumulator, and w be the size of witness. The communication complexity of the protocol is $O(n^2\ell + (\kappa + w)n^3)$ bits per epoch.*

Proof. At the start of the protocol, each party P_i multicasts comm_i of size ℓ to all party P_j $\forall j \in [n]$ and sends secret share $s_{i,j}$ to party P_j $\forall j \in [n]$. This step incurs $O(n^2\ell + \kappa n^3)$. In the Forward step, parties invoke $\mathsf{Deliver}$ for the first comm_j from party P_j for $j \in [n]$. Invoking $\mathsf{Deliver}$ on an object of size ℓ incurs $O(n\ell + (\kappa + w)n^2)$, since each party multicasts a code word of size $O(\ell/n)$, a witness of size w and an accumulator of size κ. Thus, invoking $\mathsf{Deliver}$ on n commitments incurs $O(n^2\ell + (\kappa + w)n^3)$.

In the Blame step, honest parties may blame up to t Byzantine parties if they do not receive valid secret shares. Multicast of t blame from each party incurs $O(\kappa t n^2)$ communication. In addition, t Byzantine parties always can blame honest parties. Honest parties forward up to t $\langle \mathsf{blame}, j \rangle$ messages to party P_j. This incurs $O(\kappa t n^2)$ communication.

In the Private open step each party can send up to t secret shares to all other parties. This incurs $O(\kappa t n^2)$ for all parties. In the Vote cert step, each party multicasts $O(n)$-sized $\mathsf{vote\text{-}cert}$ to all other parties which incurs $O(\kappa n^3)$ in communication. Invoking $\mathsf{Deliver}$ on an $O(n)$-sized certificate incurs $O(n^2 + (\kappa + w)n^2)$. For n certificate, this step incurs $O(n^3 + (\kappa + w)n^3)$.

In the Propose grade step, each party multicast their grade list of size $O(n)$. Multicast of $O(n)$-sized grade list by n parties incurs $O(n^3)$ communication. Thus, the total communication complexity is $O(n^2\ell + (\kappa + w)n^3)$ bits. □

7 Oblivious Leader Election

In this section, we construct an oblivious leader election (OLE) (aka, common coin) protocol that outputs a common honest leader with some constant probability called the *fairness*. Our OLE protocol follows the same high level idea as prior common coin protocols [23, 33]. In summary, each party P_i selects n secrets each for party P_j for $j \in [n]$. A coin value x_j is assigned to party P_j which is a function of secrets shared by at least $t + 1$ parties; this suffices to ensure the assigned coin value x_j is random. A party having the highest (or lowest) coin value is selected to be the leader.

Traditionally, the common coin was designed via n^2 parallel invocations of weaker VSS primitives such as graded VSS [23] or moderated VSS [33] which trivally incurs $\Omega(n^4)$ communication. In this work, we build an OLE protocol using Aggregatable PVSS [28]. Aggregatable PVSS allows a linear number of secret sharings to be aggregated into a single transcript whose size is linear. Parties can publicly verify the correctness of the aggregated PVSS transcript and verify the aggregated PVSS transcript contains a required threshold of PVSS sharing instances. Moreover, the aggregated PVSS transcript can be efficiently propagated among parties without blowing up communication. A recent work [2] also designs an OLE protocol using Aggregatable

PVSS [28] for the asynchronous model. In their work, they additionally use threshold verifiable random functions [2] obtained from Aggregatable PVSS to generate a verifiable common coin. Our aggrement protocol in the following section does not require a verifiable common coin and can function with a coin common with a constant probability. In this regard, our common coin protocol is simpler compared to the common coin protocol of Abraham et al. [2].

Set $\mathcal{X}_i \leftarrow \emptyset$, $\mathsf{accept}_i[j] \leftarrow \perp$ and $\mathsf{valid\_shares}_i \leftarrow \perp$.
Each party P_i sets $\mathsf{epoch\text{-}timer}$ to 6Δ and performs following operations:

1. **Deal.** Each party P_i randomly selects n secrets $s_{i,1}, \ldots s_{i,n}$ and generates PVSS transcripts $(\mathsf{pvss}_{i,1}, \ldots, \mathsf{pvss}_{i,n}) \leftarrow \mathsf{PVSS.Sh}(s_{i,1}), \ldots, \mathsf{PVSS.Sh}(s_{i,n})$. Party P_i sends $\mathsf{pvss}_{i,j}$ to party P_j $\forall j \in [n]$.

2. **Propose.** Upon receiving $t+1$ distinct $\mathsf{pvss}_{j,i}$ such that $\mathsf{PVSS.ShVerify}(\mathsf{pvss}_{j,i}) = 1$, each party P_i aggregates them to obtain aggregated transcript $\mathsf{agg\_pvss}_i$ using $\mathsf{PVSS.Aggregate}$ function. Party P_i multicasts $\langle \mathsf{propose}, \mathsf{agg\_pvss}_i, z_{pi} \rangle_i$ where z_{pi} is the accumulation value for $\mathsf{agg\_pvss}_i$.

3. **Deliver.** If $\mathsf{epoch\text{-}timer} \geq 3\Delta$ party P_i receives the first valid aggregated transcript $tr_j := \langle \mathsf{propose}, \mathsf{agg\_pvss}_j, z_{pj} \rangle_j$ from party P_j, invoke $\mathsf{Deliver}(\mathsf{propose}, tr_j, z_{pj})$ for all aggregated transcripts from party P_j for $j \in [n]$ and set $\mathsf{grade\text{-}timer}_i[j]$ to 2Δ and start counting down. When $\mathsf{grade\text{-}timer}_i[j]$ expires, if no party j equivocation has been detected, set $\mathsf{accept}_i[j] = 2$.

4. **Evaluate.** Upon receiving a valid aggregated transcript $\mathsf{agg\_pvss}_j$ from party P_j, perform $(\mathsf{share}_{j,i}, \pi_{j,i}) \leftarrow \mathsf{PVSS.GetShare}(\mathsf{agg\_pvss}_j, \mathsf{sk}_i)$ for aggregated transcripts from party P_j for $j \in [n]$. Multicast $\langle \mathsf{share}_{j,i}, \pi_{j,i} \rangle_i$.

5. **Verify.** Upon receiving the first $\langle \mathsf{share}_{k,j}, \pi_{k,j} \rangle_j$ from party P_j, if $\mathsf{PVSS.VerifyShare}(\mathsf{agg\_pvss}_k, \mathsf{share}_{k,j}, \pi_{k,j}) = 1$, then $\mathsf{valid\_shares}_i[k] \leftarrow \mathsf{valid\_shares}_i[k] \cup \{(\mathsf{share}_{k,j}, \pi_{j,i})\}$. If $|\mathsf{valid\_shares}_i[k]| > t$, perform $(x_k, \pi_k) \leftarrow \mathsf{PVSS.Recon}(\mathsf{valid\_shares}_i[k])$ and $\mathcal{X}_i[k] \leftarrow x_k$.

6. **Output.** When $\mathsf{epoch\text{-}timer}$ expires, perform $\ell \leftarrow argmax_k\{\mathcal{X}_i[k] | \mathsf{accept}_i[k] = 2\}$. Output ℓ.

7. **(Non-blocking) Equivocation.** If equivocating hashes signed by any party P_j are detected, broadcast the equivocating hashes.

Figure 6: Oblivious Leader Election using Aggregatable PVSS

Protocol Details. In our oblivious leader election protocol (refer Figure 6), each party P_j randomly selects n secrets $s_{i,1}, \ldots, s_{i,n}$ and generates corresponding PVSS transcripts $(\mathsf{pvss}_{i,1}, \ldots, \mathsf{pvss}_{i,n})$ by using $\mathsf{PVSS.Sh}$ function. Party P_j then sends $\mathsf{pvss}_{i,j}$ to party P_j $\forall j \in [n]$.

Each party P_i then waits to receive $t+1$ valid PVSS transcripts from different parties and aggregates them to obtain an aggregated PVSS transcript $\mathsf{agg\_pvss}_i$ whose size is still linear. The secrets shared in these $t+1$ PVSS transcripts determine the coin value x_i assigned to party P_i. Ultimately, the value of this coin is used to decide the winner of leader election. In our protocol, the party with the highest such coin value is elected to be the leader. To prevent a Byzantine party from always selecting the best strategy, this coin value is computed from the secrets shared by at least $t+1$ parties out of which one party is guaranteed to be honest. Since, an honest party randomly selects his secrets, the assigned coin value is truly random.

Party P_i then "gradecasts" its aggregated transcript $\mathsf{agg\_pvss}_i$. Gradecast of the transcript $\mathsf{agg\_pvss}_i$ ensures that if an honest party P_j assigns a grade of 2 for $\mathsf{agg\_pvss}_i$, all honest parties receive the transcript and hence evaluate the transcript to get their secret shares. This in turn ensures that the secret shared in the transcript $\mathsf{agg\_pvss}_i$ can be reconstructed.

Upon receiving a valid aggregated transcript $\mathsf{agg\_pvss}_k$, honest parties P_j decrypts their secret share $\mathsf{share}_{j,k}$ along with a proof π_j and mulitcasts these to all parties. Upon receiving $t+1$ valid secret shares for $\mathsf{agg\_pvss}_k$, parties reconstruct the secret x_k shared via $\mathsf{agg\_pvss}_k$.

For aggregated PVSS transcripts $\mathsf{agg\_pvss}_i$ shared by an honest party P_j, all honest parties will assign a grade of 2. This implies all honest parties will at least have $n-t$ entries in accept with a value of 2. The secret shared in corresponding PVSS transcripts is guaranteed to be reconstructed. For aggregated PVSS transcripts $\mathsf{agg\_pvss}_h$ shared by a Byzantine party P_h, if an honest party P_j sets a grade of 2, all honest receive the same aggregated PVSS transcript $\mathsf{agg\_pvss}_h$ and all honest parties will evaluate the same transcript to obtain their secret shares. This ensures the reconstruction of secret x_h.

Since, the coin value x_i assigned to each party P_i is random, the probability that the coin value assigned to an honest party is the maximum value is at least $1/2$. Moreover, all honest parties have the coin values assigned to all other honest parties. Thus, when the coin value assigned to an honest party is the global maximum, all honest party output the common coin value and its corresponding party to be the common leader.

Theorem 15. *The protocol in Figure 6 is an oblivious leader election protocol with fairness at least* $1/2$.

Proof. By the properties of weak gradecast (refer Theorem 9), when the dealer is honest, all honest parties output the value sent by the dealer with a grade of 2. Since, there are at least $n - t$ honest parties and each honest party P_k sends $\mathsf{agg\_pvss}_k$, all honest parties will output P_k with a grade of 2. In addition, the coin value associated with an honest party can be reconstructed.

Since the coin value assigned to a party is computed from input contributions of at least $t + 1$ parties and an honest party selects secrets uniformly at random, each coin value is uniform and random. Thus, with probability at least $\frac{n-t}{n}$, the coin value assigned to a honest party will be global maximum. In addition, the coin values of two parties can be common with probability $\frac{1}{n^2}$. Thus, all honest parties output a common leader with fairness $\frac{n-t}{n} - \frac{1}{n^2} \geq \frac{1}{2}$. □

Lemma 16 (Communication Complexity)**.** *Let ℓ be the size of a PVSS transcript, κ be the size of accumulator and w be the size of witness. The communication complexity of the protocol is $O(n^2\ell + (\kappa + w)n^3)$ bits per epoch.*

Proof. At the start of the protocol, each party P_i sends $O(\ell)$-sized PVSS transcript to all other parties. This step incurs $O(n^2\ell)$ communication. In the Propose step, each party multicasts $O(\ell)$-sized aggregated PVSS transcript which incurs $O(n^2\ell)$ communication. In the Deliver step, each party invoke Deliver for one proposal from each party. Invoking Deliver on an object of size ℓ incurs $O(n\ell + (\kappa + w)n^2)$, since each party multicasts a code word of size $O(\ell/n)$, a witness of size w and an accumulator of size κ. Thus, invoking Deliver on n proposal incurs $O(n^2\ell + (\kappa + w)n^3)$.

In the Evaluate step, each party multicasts κ-sized share and w-sized proof to all parties for each aggregated PVSS transcript. For n aggregated PVSS transcripts, this step incurs $(\kappa + w)n^3$. Thus, the total communication complexity of the protocol is $O(n^2\ell + (\kappa + w)n^3)$ bits per epoch. □

8 Multi-Valued Validated Byzantine Agreement

In the previous section, we presented a recoverable set of shares protocol where each honest party P_i outputs a (possibly different) set $\mathsf{AcceptList}_i$ of size at least $n - t$ and its ack-cert $\mathcal{AC}(\mathsf{AcceptList}_i)$–both of which are linear sized. For DKG, all honest parties need to agree on a common set of parties whose secret shares are used to compute final secret keys and a public key. Thus, we need a consensus primitive that takes a different $O(n)$-sized input from each party and outputs a common set which is valid. Here, a valid set is accompanied by its certificate and can potentially also be the input of a Byzantine party. Such a consensus primitive is called a *multi-valued validated Byzantine agreement.*

Multi-valued validated Byzantine agreement (MVBA) was introduced by Cachin et al. [12] to allow honest parties to agree on any externally valid value. Recent works [4, 36] have proposed MVBA protocols for the asynchronous communication model with reduced communication assuming $t < n/3$ Byzantine faults. Abraham et al. [4] present a MVBA protocol with $O(\kappa n^2)$ communication for small size inputs and Luo et al. [36] present MVBA protocol for long message of size ℓ with $O(n\ell + \kappa n^2)$ communication and constant expected rounds. Both of the works assume threshold signatures to generate constant-sized certificates. In the absence of threshold signatures, the communication blows up linearly in both protocols. In addition, to the best of our knowledge, no MVBA protocol have been proposed in the synchronous communication model for $t < n/2$ case. In this paper, we present a synchronous MVBA protocol tolerating $t < n/2$ Byzantine faults with $O(\kappa n^3)$ communication and expected constant rounds.

Following Abraham et al. [4], we present an MVBA protocol with an external validity function ex-validation, that determines whether a value is valid or not.

Definition 8.1 (Multi-valued Validated Byzantine Agreement [4,36])**.** *A protocol solves multi-valued validated Byzantine agreement if it satisfies following properties except with negligible probability in the security parameter κ:*

- ***Validity.*** *If an honest party decides a value v, then* ex-validation(v) = *true.*
- ***Agreement.*** *No two honest parties decide on different values.*
- ***Termination.*** *If all honest parties start with externally valid values, all honest parties eventually decide.*

We extend the Binary Byzantine agreement (BBA) protocol of Katz and Koo [33] to MVBA for large ($\ell = \Theta(n)$) input. Their protocol tolerates $t < n/2$ Byzantine faults and terminates in expected 4 epochs. They present two BBA protocols. The first protocol involves invoking n parallel gradecasts; with each gradecast propagating small sized input. As mentioned before, their gradecast protocol incurs $O(\kappa n^3)$ communication; thus, their first protocol trivially incurs $O(\kappa n^4)$ communication. Their second protocol avoids the use of gradecast to reduce round complexity; however the protocol can output $\perp$ if honest parties do not start with the same input; which is not desired for our purpose. Thus, we make efficiency improvements on their first protocol to obtain our MVBA protocol.

We replace their gradecast protocol with our communication optimal gradecast protocol from Section 5. Our gradecast protocol incurs only $O(\kappa n^2)$ communication while propagating $O(n)$-sized input. Using our gradecast protocol allows BBA protocol of Katz and Koo [32] to handle large input while simultaneously reducing the communication to $O(\kappa n^3)$.

To circumvent the linear round lower bound for a deterministic BA protocol [6,18], BA protocols use a common source of randomness called *common coin* to achieve agreement in constant expected rounds. The common coin is *weak* if honest parties may output different random values and outputs a common random value with some constant probability. In Katz and Koo BBA, the weak common coin was obtained by invoking n^2 moderated VSS instances which incurs $\Omega(\kappa n^4)$ communication. We replace their common coin protocol with our communication efficient OLE protocol from Section 7 which has $O(\kappa n^3)$ communication.

Finally, as noted before, their BBA protocol terminates in expected 4 epochs. We reuse the idea of $\mathsf{BA}^\star$ protocol of Micali [38] where parties terminate early by multicasting a special message before they terminate. This allows our protocol to terminate in expected 2 epochs, i.e., improves the latency by a factor of two. Moreover, our protocol is described in the Δ synchrony model and requires fewer Δ wait compared to round-based protocol. The resulting protocol is an MVBA protocol with $O(\kappa n^3)$ communication and expected 2 epochs to terminate where each epoch requires 9Δ time.

Now, we present our MVBA protocol (refer Figure 7). Our MVBA protocol is executed in epochs with each epoch taking 9Δ time. Our MVBA protocol embeds the gradecast protocol from Section 5 directly. The propose and forward steps mimic the invocation of gradecast. In the propose step, each party P_i multicasts their input v_i in the form of $\langle \mathsf{propose}, v_i, z_{pe}, e\rangle_i$ where z_{pe} is the accumulation value of v_i. In order to facilitate efficient equivocation checks, party P_i signs $\langle \mathsf{propose}, H(v_i), e, z_{pe}\rangle$ and sends v_i separately. Similarly, when multicasting $\langle \mathsf{propose2}, v_i, z_{pe2}, e\rangle_i$ (z_{pe2} is the accumulation value of v_i), party P_i signs $\langle \mathsf{propose2}, H(v_i), z_{pe2}, e\rangle$ and sends v_i separately. In the terminate/advance epoch step, if $\mathsf{lock}_i = \mathsf{true}$, party P_i output v_i and multicasts a special message $\langle \mathsf{terminate}, v_i, z_{te}, e\rangle_i$ (z_{te} is the accumulation value of v_i) to signify party P_i terminates. In the following epochs, honest parties considers v_i to be input of party P_i.

Exact Round Complexity. By Theorem 15, a common honest leader is elected with probability 1/2 and all honest parties terminate in the next epoch. Thus, the expected number of epochs required is 2 epochs.

8.1 Analysis of MVBA

Fact 17. *If an honest party sets* lock *to* true *with a value v in epoch e, then all honest parties adopt value v in epoch e.*

Let v_i be party P_i's input and e be the current epoch. Each party P_i sets $\mathsf{epoch\text{-}timer}_e$ to 9Δ and $\mathsf{lock}_i \leftarrow \mathsf{false}$. In each epoch e, party P_i waits until $\mathsf{epoch\text{-}timer}_e \geq 7\Delta$ and invokes OLE protocol (refer Figure 6). For each epoch e, party P_i performs following operations.

1. **Propose.** Each party P_i multicasts $\langle \mathsf{propose}, v_i, z_{pe}, e\rangle_i$.
2. **Forward.** If $\mathsf{epoch\text{-}timer}_e \geq 7\Delta$ and party P_i receives the first party P_j proposal $pr_j := \langle \mathsf{propose}, v_j, z_{pe}, e\rangle_j$, invoke $\mathsf{Deliver}(\mathsf{propose}, pr_j, z_{pe}, e)$ and set $\mathsf{grade\text{-}timer}_e[j]$ to 2Δ and start counting down. When $\mathsf{grade\text{-}timer}_e[j]$ expires and no party P_j equivocation has been detected, set $\mathsf{grade}_i[j] = 2$. If party P_i receives party $j's$ proposal, set $\mathsf{grade}_i[j] = 1$.
3. **Update.** Wait until $\mathsf{epoch\text{-}timer}_e \geq 5\Delta$. Let $v_{j,i}$ be the value received from party P_j. Let $\mathcal{S}_i^v := \{j : v_{j,i} = v \wedge \mathsf{grade}_i[j] = 2\}$ and $\tilde{\mathcal{S}_i^v} := \{j : v_{j,i} = v \wedge \mathsf{grade}_i[j] \geq 1\}$.
   (a) If $|\tilde{\mathcal{S}_i^v}| > t$, update $v_i \leftarrow v$.
   (b) If $|\mathcal{S}_i^v| > t$, set $\mathsf{lock}_i \leftarrow \mathsf{true}$.

   Multicast $\langle \mathsf{propose2}, v_i, z_{pe2}, e\rangle_i$.
4. **Forward2.** If $\mathsf{epoch\text{-}timer}_e \geq 3\Delta$ and party P_i receives the first $pr2_j := \langle \mathsf{propose2}, v_j, z_{pe2}, e\rangle_j$ from party P_j, invoke $\mathsf{Deliver}(\mathsf{propose2}, pr2_j, z_{pe2}, e)$ and set $\mathsf{grade\text{-}timer}_e[j]$ to 2Δ and start counting down. When $\mathsf{grade\text{-}timer}_e[j]$ expires and no party P_j equivocation has been detected, set $\mathsf{grade}_i[j] = 2$. If party P_i receives party $j's$ proposal, set $\mathsf{grade}_i[j] = 1$.
5. **Update2.** Wait until $\mathsf{epoch\text{-}timer}_e \geq \Delta$. Define $\mathcal{S}_i^v$ and $\tilde{\mathcal{S}_i^v}$ as above. If $|\tilde{\mathcal{S}_i^v}| > t$, set $v_i \leftarrow v$. Multicast v_i.
6. **Terminate/Advance Epoch.** Let ℓ be the output of OLE protocol. When $\mathsf{epoch\text{-}timer}_e$ expires,
   (a) If $\mathsf{lock}_i = \mathsf{true}$, output v_i, multicast $\langle \mathsf{terminate}, v_i, z_{te}, e\rangle_i$ and terminate.
   (b) If $\mathsf{lock}_i = \mathsf{false}$, $|\mathcal{S}_i^v| \leq t$, $v_{\ell,i} \neq \perp$ and $\mathsf{ex\text{-}validation}(v_{\ell,i}) = \mathsf{true}$, update $v_i \leftarrow v_{\ell,i}$. Advance to epoch $e+1$, set $\mathsf{epoch\text{-}timer}_e$ to 9Δ and start counting down.
   In all following epochs, if party P_i received $\langle \mathsf{terminate}, v_j, z_{te}, e\rangle_j$ from party P_j, consider $v_{j,i}$ to be v_j.
7. **(Non-blocking) Equivocation.** If equivocating hashes signed by party P_j are detected, multicast the equivocating hashes.

Figure 7: **MVBA** with $O(\kappa n^3)$ communication and expected 2 epochs.

Proof. Suppose an honest party P_i sets lock_i to true in epoch e. Party P_i must have received value v from a set Q of at least $t+1$ parties such that $|\mathcal{S}_i^v| > t$ i.e., $\mathsf{Deliver}$ for value v from parties in Q completed without detecting an equivocation from all parties in Q. By Fact 8, all other honest parties receive value v corresponding to parties in Q. Thus, all other parties set $\mathsf{grade}[j] \geq 1\ \forall j \in Q$ i.e. $|\tilde{\mathcal{S}^v}| > t$ for all other honest parties and all honest parties adopt value v in the Update step.

Once all honest parties adopt value v in the Update step, they send value v in the $\mathsf{propose2}$ step. Since, honest parties do not equivocate and send value v in a timely manner, all honest parties receive value v such that $\mathsf{grade}[j]$ to 2. Thus, $|\tilde{\mathcal{S}_i^v}| > t$ and $|\mathcal{S}_i^v| > t$. Since, $|\mathcal{S}_i^v| > t$, no honest party will adopt value sent by the leader ℓ chosen from the OLE protocol. Thus, all honest parties adopt value v in epoch e. □

Lemma 18. *If all honest parties start an epoch e with same input v, then all honest parties decide value v and terminate at the end of epoch e.*

Proof. Suppose all honest parties start an epoch e with the same input v. This includes honest parties who have terminated with value v and sent $\mathsf{terminate}$ message. An honest party P_j that terminated sends $\langle \mathsf{terminate}, v_i, z_{te}, e-1\rangle_j$ to all other honest parties and all other honest parties consider value v as $v_{j,i}$.

Let τ be the time when the first honest party starts an epoch. Since honest parties are synchronized within Δ time, all honest parties start epoch e by time $\tau + \Delta$. Observe that honest parties $\mathsf{propose}$ value v at the start of an epoch. Thus, all honest parties $\mathsf{propose}$ value v by time $\tau + \Delta$ and all honest parties receive $\langle \mathsf{propose}, v_i, e, z_{pe}\rangle_i$ by time $\tau + 2\Delta$ such that $\mathsf{epoch\text{-}timer}_e \geq 7\Delta$. Thus, all honest parties will invoke Deliver for the proposed value v from all other honest parties. Moreover, honest parties do not equivocate. Thus, all honest parties will set $\mathsf{grade}[j] = 2$ for all other honest parties. Thus, for value v, all honest parties have $|\mathcal{S}_i^v| > t$ and $|\tilde{\mathcal{S}}_i^v| > t$, and set lock to true for value v.

Similarly, all honest parties send propose2 for value v when their $\mathsf{epoch\text{-}timer}_e = 5\Delta$. This implies all honest parties receive propose2 for value v such that $\mathsf{epoch\text{-}timer}_e \geq 3\Delta$. Thus, all honest parties will invoke Deliver for the proposed value v from all other honest parties. Since, honest parties do not equivocate, all honest parties will set $\mathsf{grade}[j] = 2$ for all other honest parties i.e., $|\mathcal{S}_i^v| > t$ and $|\tilde{\mathcal{S}}_i^v| > t$ for all honest parties. Moreover, no honest party will adopt value sent by the leader ℓ chosen from OLE protocol in epoch e.

Note that all honest parties set lock to true. Thus, all honest parties output v and terminate at the end of epoch e. □

Theorem 19. *The protocol in Figure 7 solves MVBA.*

Proof. We first consider validity i.e., if an honest party decides a value v, then $\mathsf{ex\text{-}validation}(v) = \mathsf{true}$. Observe that an honest party P_i decides a value v only when its sets $\mathsf{lock}_i = \mathsf{true}$. An honest party sets $\mathsf{lock}_i = \mathsf{true}$ only when it observes $|\mathcal{S}_i^v| \leq t$. Thus, at least one honest party P_j must have sent value v in Propose step. Honest party P_j sends value v either when its input at the start of the protocol execution is v in which case $\mathsf{ex\text{-}validation}(v) = \mathsf{true}$, or when its updates its value v_j to v at the end of an epoch. In the latter case, party P_j checks if $\mathsf{ex\text{-}validation}(v) = \mathsf{true}$.

Next, we consider agreement. Consider an epoch e and let P_ℓ be the common leader in epoch e elected via OLE protocol. There are two cases to consider.

Case I. $\mathsf{lock}_i = \mathsf{true}$ for at least one honest party P_i with a value v in some epoch e. By Fact 17, all honest part adopt value v in epoch e and enter epoch $e + 1$ with same value v. By Lemma 18, all honest parties output value v and terminate in epoch $e + 1$.

Case II. $\mathsf{lock}_i = \mathsf{false}$ for all honest parties in epoch e. If leader P_ℓ is honest, leader P_ℓ sends the same value v_ℓ to all parties. If $|\mathcal{S}_i^v| \leq t$ for all honest parties, then all honest parties adopt the value v_ℓ in epoch e. By Lemma 18, all honest parties output value v and terminate in epoch $e + 1$.

If $|\mathcal{S}_i^v| > t$ for at least one honest party P_i in the Update2 step, there exists a set Q of at least $t + 1$ who sent value v and party P_i did not detect equivocation from any party in set Q before its grade-timer expired. By Fact 8, all honest parties receive value v from parties in Q i.e. $|\tilde{\mathcal{S}}_i^v| > t$ for all honest parties. Thus, all honest parties including leader P_ℓ adopt value v in the Update2 step. If the leader P_ℓ is honest, it sends the same value v to all parties. Honest parties with $|\mathcal{S}_i^v| \leq t$ adopt value v sent by leader P_ℓ in epoch e which is the same value adopted by party P_i with $|\mathcal{S}_i^v| > t$. Thus, all honest parties have value v at the end of epoch e. By Lemma 18, all honest parties output value v and terminate in epoch $e + 1$.

□

Lemma 20 (Communication Complexity)**.** *Let ℓ be the size of input v for each party, κ be the size of accumulator and w be the size of witness. The communication complexity of the protocol is $O(n^2\ell + (\kappa + w)n^3)$ bits per epoch.*

Proof. At the start of the protocol, each party P_i multicasts $O(\ell)$-sized proposal to all other parties. This step incurs $O(n^2\ell)$ communication. In the Forward step, each party invokes Deliver for one proposal from each party. Invoking Deliver on an object of size ℓ incurs $O(n\ell + (\kappa + w)n^2)$, since each party multicasts a code word of size $O(\ell/n)$, a witness of size w and an accumulator of size κ. Thus, invoking Deliver on n proposal incurs $O(n^2\ell + (\kappa + w)n^3)$.

Similarly, in the Update step, each party multicasts $O(\ell)$-sized proposal to all other parties which incurs $O(n^2\ell)$ communication. In the Forward2 step, each party invokes Deliver for one proposal from each party. Thus, invoking Deliver on n proposal incurs $O(n^2\ell + (\kappa + w)n^3)$.

In Update2 step, each party multicasts $O(\ell)$-sized value which incurs $O(n^2\ell)$ communication. Thus, the total communication complexity of the protocol is $O(n^2\ell + (\kappa + w)n^3)$ bits per epoch. □

9 Distributed Key Generation

Finally, we present two communication efficient DKG protocols with $O(\kappa n^3)$ communication. The first protocol is randomized and terminates in expected constant epochs while the second protocol is deterministic and terminates in $t+1$ epochs. The DKG protocols in this section differs from the secure DKG protocol of Section 4 in the following ways. First, we replace the broadcast channel with Byzantine consensus primitives and requires a single invocation of consensus instance. Second, in the secure DKG protocol, the final public key and secret keys are computed from the secret shares of all honest parties. In particular, all honest parties belong to set `QUAL` and the public key and secret keys are computed from parties in `QUAL`. In contrast, the DKG protocols in this section compute the final public key and secret keys from a common set of size at least $n-t$ where at least $n-2t$ parties are honest (i.e., at least one honest party when $n = 2t+1$). This suffices to ensure construction of a secure DKG protocol.

9.1 Randomized DKG

1. **Deal.** Each party P_i invokes recoverable set of shares protocol (refer Figure 5). Each party P_i outputs a set $\mathsf{AcceptList}_i$ with an $\mathsf{ack\text{-}cert}$ for $\mathsf{AcceptList}_i$ (i.e., $\mathcal{AC}(\mathsf{AcceptList}_i)$).

2. **MVBA.** Each party P_i invokes MVBA (refer Figure 7) with input ($\mathsf{AcceptList}_i$, $\mathcal{AC}(\mathsf{AcceptList}_i)$). Let $\mathsf{AcceptList}_k$ be the output of all honest parties.

3. **Generating keys.** Let $x_i = \sum_{j \in \mathsf{AcceptList}_k | \mathsf{AcceptList}_k[j]=2} s_{ji}$ and $x'_i = \sum_{j \in \mathsf{AcceptList}_k | \mathsf{AcceptList}_k[j]=2} s'_{ji}$ be the sum of secret shares in $\mathsf{AcceptList}_k$. Compute $\mathcal{C}_{\langle g \rangle}(x_i)$, $\mathcal{C}_{\langle g,h \rangle}(x_i, x'_i)$ and $\pi_{\equiv \mathsf{Com}\, i} = \mathsf{NIZKPK}_{\equiv \mathsf{Com}}(x_i, x'_i, g, h, \mathcal{C}_{\langle g \rangle}(x_i), \mathcal{C}_{\langle g,h \rangle}(x_i, x'_i))$.
   - Multicast $(\mathcal{C}_{\langle g \rangle}(x_i), \pi_{\equiv \mathsf{Com}\, i})$ to all parties.
   - Verify the received $(\mathcal{C}_{\langle g \rangle}(x_i), \pi_{\equiv \mathsf{Com}\, j})$ as shown in Equation (3).
   - Upon receiving $t+1$ valid $\mathcal{C}_{\langle g \rangle}(x_i)$, interpolate them to obtain $y = g^x$. Set y as the public key and x_i as the private key.

Figure 8: Randomized DKG with $O(\kappa n^3)$ communication and expected $O(\Delta)$ epochs

The randomized DKG protocol uses recoverable set of shares protocol (refer Figure 5) to perform secret sharing. At the end of the recoverable set of shares, each honest party P_i outputs a (possibly different) set of at least $n-t$ parties ($\mathsf{AcceptList}_i$) which they observe to have correctly shared their secret along with an $\mathsf{ack\text{-}cert}$ for $\mathsf{AcceptList}_i$ ($\mathcal{AC}(\mathsf{AcceptList}_i)$). The $\mathsf{ack\text{-}cert}$ for $\mathsf{AcceptList}_i$ serves an external validity function to the MVBA protocol i.e., if there is an $\mathcal{AC}(\mathsf{AcceptList}_i)$ for $\mathsf{AcceptList}_i$, then $\mathsf{ex\text{-}validation}(\mathsf{AcceptList}_i) = true$. Note that both $\mathsf{AcceptList}_i$ and $\mathcal{AC}(\mathsf{AcceptList}_i)$ are linear sized. Each honest party P_i then invokes MVBA protocol with ($\mathsf{AcceptList}_i$, $\mathcal{AC}(\mathsf{AcceptList}_i)$) as input. At the end of MVBA protocol, each honest party outputs a common set $\mathsf{AcceptList}_k$. The final secret key and public key is then computed using secret shares shared by parties h such that $\mathsf{AcceptList}_k[h] = 2$ using the reconstruction protocol in Figure 2.

Latency and Communication Complexity. The recoverable set of shares protocol incurs a latency of 16Δ and $O(\kappa n^3)$ communication. The MVBA protocol incurs expected 2 epochs (with each epoch being 9Δ) and $O((\kappa+w)n^3)$ communication where the size of input is $O(\kappa n)$. The reconstruction phase requires $O(\kappa n^2)$ communication and 2Δ time in the worst case. Thus, the protocol incurs $O((\kappa+w)n^3)$ communication and expected 36Δ.

9.2 Deterministic DKG

While the above randomized protocol terminates in expected 2 epochs in the best case, it has probabilistic termination and may require a linear number of epochs in the worst case with a communication of $O(\kappa n^4)$. As an alternate solution, we present a deterministic DKG protocol with guaranteed termination in $t+1$ epochs with $O(\kappa n^3)$ communication. The deterministic DKG protocol is presented in Figure 9. In the protocol, honest parties execute the recoverable set of shares protocol and each honest party P_i outputs a (possibly

1. **Deal.** Each party P_i invokes recoverable set of shares protocol (refer Figure 5). Each party P_i output a set $\mathsf{AcceptList}_i$ with an $\mathsf{ack\text{-}cert}$ for $\mathsf{AcceptList}_i$.

2. **BFT SMR.** Each party P_i participates in BFT SMR (refer Figure 11) with input $\mathsf{AcceptList}_i$ and $\mathcal{AC}(\mathsf{AcceptList}_i)$. The BFT SMR protocol is executed in round-robin manner with first $t+1$ leaders. Let $\mathsf{AcceptList}_k$ be the first committed value of all honest parties.

3. **Generating keys.** Let $x_i = \sum_{j \in \mathsf{AcceptList}_k | \mathsf{AcceptList}_k[j]=2} s_{ji}$ and $x'_i = \sum_{j \in \mathsf{AcceptList}_k | \mathsf{AcceptList}_k[j]=2} s'_{ji}$ be the sum of secret shares in $\mathsf{AcceptList}_k$. Compute $\mathcal{C}_{\langle g \rangle}(x_i)$, $\mathcal{C}_{\langle g,h \rangle}(x_i, x'_i))$ and $\pi_{\equiv \mathsf{Com}\, i} = \mathsf{NIZKPK}_{\equiv \mathsf{Com}}(x_i, x'_i, g, h, \mathcal{C}_{\langle g \rangle}(x_i), \mathcal{C}_{\langle g,h \rangle}(x_i, x'_i)$.
   - Multicast $(\mathcal{C}_{\langle g \rangle}(x_i), \pi_{\equiv \mathsf{Com}\, i})$ to all parties.
   - Verify the received $(\mathcal{C}_{\langle g \rangle}(x_i), \pi_{\equiv \mathsf{Com}\, j})$ as shown in Equation (3).
   - Upon receiving $t+1$ valid $\mathcal{C}_{\langle g \rangle}(x_i)$, interpolate them to obtain $y = g^x$. Set y as the public key and x_i as the private key.

Figure 9: Deterministic DKG with $O(\kappa n^3)$ communication and $t+1$ epochs

different) set of at least $n-t$ parties ($\mathsf{AcceptList}_i$) which they observe to have correctly shared their secret along with an $\mathsf{ack\text{-}cert}$ for $\mathsf{AcceptList}_i$ ($\mathcal{AC}(\mathsf{AcceptList}_i)$). The tuple ($\mathsf{AcceptList}_i$, $\mathcal{AC}(\mathsf{AcceptList}_i)$) is input into a leader-based Byzantine fault tolerant state machine replication (BFT SMR) protocol of RandPiper [9] to agree on a common set. We present a brief overview of the BFT SMR.

BFT SMR of RandPiper [9]. The BFT SMR protocol of RandPiper [9] (refer Figure 11) is a communication efficient rotating-leader SMR protocol with $O(\kappa n^2)$ communication per epoch even for $O(n)$-sized input. The BFT SMR protocol has optimal resilience i.e., tolerates $t < n/2$ Byzantine faults. The leaders are rotated in each epoch; in their protocol, an epoch is a duration of 11Δ. When the leader of an epoch is honest, all honest parties commit the proposed value in the same epoch, whereas, when the leader of the epoch is Byzantine, some honest parties may require linear number of epochs to commit the proposed value. The BFT SMR utilizes the "block-chaining" paradigm i.e., each proposal is represented in the form of a block which explicitly extends a block B proposed earlier by including hash of previous block B. In this paradigm, when a block B is committed, all its ancestors are also committed. We refer the readers to the RandPiper [9] for more details.

In this deterministic DKG protocol, we execute the BFT SMR protocol for $t+1$ epochs. In each epoch, the epoch leader is expected to propose its ($\mathsf{AcceptList}$, $\mathcal{AC}(\mathsf{AcceptList})$). If the epoch leader is honest, all honest parties commit the proposed set in the same epoch; otherwise honest parties may require linear number of epochs when the leader is Byzantine to commit the proposed value or commit no value at all if the Byzantine leader does not propose. Since the BFT SMR protocol is executed for $t+1$ epochs, there is will be at least one honest leader; thus all honest parties commit at least one set. Honest parties output the first committed set and perform reconstruction using this set to generate the final secret key and public key.

Latency and Communication Complexity. The recoverable set of shares protocol incurs a latency of 16Δ and $O(\kappa n^3)$ communication. The BFT SMR protocol incurs $O(\kappa n^2)$ communication per epoch; $O(\kappa n^3)$ communication for $t+1$ epochs. The length of each epoch is 11Δ. The reconstruction phase requires $O(\kappa n^2)$ communication and 2Δ time in the worst case. Thus, the protocol incurs $O(\kappa n^3)$ communication and 18Δ + $((t+1) * 11\Delta)$ time.

10 A Lower bound on the Communication Complexity for Secure Distributed Key Generation

In this section, we formalize a communication lower bound for a deterministic protocol for secure distributed key generation. The proof of this lower bound is inspired by the well-known communication lower bound for Byzantine broadcast by Dolev and Reischuk [17]. In the lower bound proof, we argue properties C1 and C2 (refer Definition 3.1) for a secure DKG protocol map to the agreement property in Byzantine broadcast and properties C3 and secrecy (refer Definition 3.1) map to the validity property in Byzantine broadcast. We conclude a secure DKG protocol must incur $\Omega(t^2/4)$ communication.

Theorem 21. *There does not exist a deterministic protocol for distributed key generation tolerating t Byzantine parties with a communication complexity of at most $t^2/4$ messages.*

Proof. Suppose for the sake of contradiction, there exists such a protocol. By secrecy property in Definition 3.1, the unique secret x has to be generated by the input contributions of a set Q of least $t+1$ parties; otherwise, the adversary controlling t Byzantine parties can learn about the secret x. Moreover, the input contribution by at least one honest party $r \in Q$ is not predetermined and must be selected uniformly at random; otherwise, t Byzantine parties can bias the unique secret x and violate correctness property (C3). Since input contribution by the honest party $r \in Q$ is chosen uniformly at random, with high probability, there must be a unique secret $v \in \mathbb{Z}_p$ (and corresponding public key) that honest parties do not decide if they receive no messages. Consider the parties being partitioned into the following 2 sets – A: a set of $\lceil t/2 \rceil$ parties, B: all remaining parties.

We consider two executions where correctness is violated in the last execution. In the first execution (W1), all parties in A are Byzantine. Parties in A do not communicate with each other. Towards B, parties in A execute honestly except they ignore the first $\lceil t/2 \rceil$ messages from parties in B. Since, the maximum faults in W1 is $\lceil t/2 \rceil$, the protocol decides and assume all honest parties decide a common public key $y = g^v$ for some unique secret v.

Since the communication complexity of the protocol is at most $t^2/4$, there must exist a party (say s) in A that receives at most $t/2$ messages from parties in B; otherwise the communication complexity will be more than $t^2/4$. Let B_s be the set of all parties that send messages to party s in W1.

In the second execution (W2), all parties in $A \setminus \{s\}$ are Byzantine and all parties in B_s are Byzantine. The total number of Byzantine parties is $(\lceil t/2 \rceil - 1) + \lceil t/2 \rceil \leq t$ which is within allowed fault threshold t. All parties in B_s execute the protocol in the same way as in W1 except they do not send any messages to party s. Parties in $A \setminus \{s\}$ execute the protocol in the same way as in W1. Party s in W1 behave as an honest party which did not receive the first $\lceil t/2 \rceil$ messages which is similar to party s in W2 which receives no messages. Thus, parties in $B \setminus B_s$ cannot distinguish W1 and W2. Thus, they decide the same common public key y. Since, party s does not receive any messages in W2, it does not decide y. If it does not decide any public key or decides any other public key $y' \neq y$, the correctness property (C2) is violated. A contradiction. □

Remark. We note that our lower bound and upper bound protocol are not tight. It is an intriguing open question to either prove a cubic communication lower bound for a secure DKG or design a DKG protocol with quadratic communication.

Acknowledgements

We thank Ittai Abraham for helpful discussions and feedback related to this paper.

References

[1] Ittai Abraham, Srinivas Devadas, Danny Dolev, Kartik Nayak, and Ling Ren. Synchronous byzantine agreement with expected $O(1)$ rounds, expected $O(n^2)$ communication, and optimal resilience. *Financial Cryptography and Data Security (FC)*, 2019.

[2] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Reaching consensus for asynchronous distributed key generation, 2021.

[3] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, Ling Ren, and Maofan Yin. Sync hotstuff: Simple and practical synchronous state machine replication. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 654–667, 2020.

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

[5] Ittai Abraham, Kartik Nayak, and Nibesh Shrestha. Optimal good-case latency for rotating leader synchronous bft. *IACR Cryptology ePrint Archive*, 2021:1138, 2021.

[6] Marcos Kawazoe Aguilera and Sam Toueg. A simple bivalency proof that t-resilient consensus requires t+ 1 rounds. *Information Processing Letters*, 71(3-4):155–158, 1999.

[7] Michael Backes, Aniket Kate, and Arpita Patra. Computational verifiable secret sharing revisited. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 590–609. Springer, 2011.

[8] Niko Barić and Birgit Pfitzmann. Collision-free accumulators and fail-stop signature schemes without trees. In *International conference on the theory and applications of cryptographic techniques*, pages 480–494. Springer, 1997.

[9] Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. Randpiper – reconfiguration-friendly random beacons with quadratic communication. Cryptology ePrint Archive, Report 2020/1590, 2020. `https://eprint.iacr.org/2020/1590`, To appear in ACM SIGSAC CCS 2021.

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

[11] Ethan Buchman. *Tendermint: Byzantine fault tolerance in the age of blockchains*. PhD thesis, thesis, 2016.

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

[13] Ran Canetti, Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Adaptive security for threshold cryptosystems. In *Annual International Cryptology Conference*, pages 98–116. Springer, 1999.

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

[15] Sourav Das, Tom Yurek, Zhuolun Xiang, Andrew Miller, Lefteris Kokoris-Kogias, and Ling Ren. Practical asynchronous distributed key generation. *Cryptology ePrint Archive*, 2021.

[16] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In Gilles Brassard, editor, *Advances in Cryptology - CRYPTO '89, 9th Annual International Cryptology Conference, Santa Barbara, California, USA, August 20-24, 1989, Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 307–315. Springer, 1989.

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

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

[19] Drand. Drand - a distributed randomness beacon daemon.

[20] Paolo D'Arco and Douglas R Stinson. On unconditionally secure robust distributed key distribution centers. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 346–363, 2002.

[21] Andreas Erwig, Sebastian Faust, and Siavash Riahi. Large-scale non-interactive threshold cryptosystems through anonymity. *IACR Cryptology ePrint Archive*, 2021:1290, 2021.

[22] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438. IEEE, 1987.

[23] Paul Feldman and Silvio Micali. Optimal algorithms for byzantine agreement. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 148–161, 1988.

[24] Pesech Feldman and Silvio Micali. An optimal probabilistic protocol for synchronous byzantine agreement. *SIAM Journal on Computing*, 26(4):873–933, 1997.

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

[26] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[27] Jens Groth. Non-interactive distributed key generation and key resharing. *IACR Cryptol. ePrint Arch.*, 2021:339, 2021.

[28] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 147–176. Springer, 2021.

[29] Martin Hirt, Jesper Buus Nielsen, and Bartosz Przydatek. Cryptographic asynchronous multi-party computation with optimal resilience. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 322–340. Springer, 2005.

[30] Dennis Hofheinz and Jörn Müller-Quade. A synchronous model for multi-party computation and the incompleteness of oblivious transfer. *Proceedings of Foundations of Computer Security—FCS*, 4:117–130, 2004.

[31] Aniket Kate, Yizhou Huang, and Ian Goldberg. Distributed key generation in the wild. *IACR Cryptol. ePrint Arch.*, 2012:377, 2012.

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

[33] Jonathan Katz and Chiu-Yuen Koo. On expected constant-round protocols for byzantine agreement. In *Annual International Cryptology Conference*, pages 445–462. Springer, 2006.

[34] Eleftherios Kokoris Kogias, Dahlia Malkhi, and Alexander Spiegelman. Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In *CCS '20: Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, CCS 20, pages 1751–1767, New York, NY, USA, 2020. Association for Computing Machinery.

[35] Torus Lab. Torus: Globally accessible public key infrastructure for everyone. `https://tor.us/`, 2021.

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

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

[38] Silvio Micali. Byzantine agreement, made trivial, 2016.

[39] Atsuki Momose and Ling Ren. Optimal communication complexity of byzantine consensus under honest majority. *arXiv preprint arXiv:2007.13175*, 2020.

[40] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H Vaidya, and Zhuolun Xiang. Improved extension protocols for byzantine broadcast and agreement. *arXiv preprint arXiv:2002.11321*, 2020.

[41] Wafa Neji, Kaouther Blibech, and Narjes Ben Rajeb. Distributed key generation protocol with a new complaint management strategy. *Security and communication networks*, 9(17):4585–4595, 2016.

[42] Lan Nguyen. Accumulators from bilinear pairings and applications. In *Cryptographers' track at the RSA conference*, pages 275–292. Springer, 2005.

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

[44] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In *Proceedings of the 10th Annual International Conference on Theory and Application of Cryptographic Techniques*, EUROCRYPT'91, page 522–526, Berlin, Heidelberg, 1991. Springer-Verlag.

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

[46] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar R Weippl. Ethdkg: Distributed key generation with ethereum smart contracts. *IACR Cryptol. ePrint Arch.*, 2019:985, 2019.

[47] Victor Shoup. Practical threshold signatures. In Bart Preneel, editor, *Advances in Cryptology - EUROCRYPT 2000, International Conference on the Theory and Application of Cryptographic Techniques, Bruges, Belgium, May 14-18, 2000, Proceeding*, volume 1807 of *Lecture Notes in Computer Science*, pages 207–220. Springer, 2000.

[48] Nibesh Shrestha, Ittai Abraham, Ling Ren, and Kartik Nayak. On the Optimality of Optimistic Responsiveness. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pages 839–857, 2020.

[49] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 877–893. IEEE, 2020.

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

A Extended Preliminaries

A.1 Linear erasure and error correcting codes.

- ENC. Given inputs $m_1, \ldots, m_{t+1}$, an encoding function ENC computes $(s_1, \ldots, s_n) = \mathsf{ENC}(m_1, \ldots, m_{t+1})$, where $(s_1, \ldots, s_n)$ are code words of length n. A combination of any $t+1$ elements of n code words uniquely determines the input message and the remaining of the code word.

- DEC. The function DEC computes $(m_1, \ldots, m_{t+1}) = \mathsf{DEC}(s_1, ..., s_n)$, and is capable of tolerating up to c errors and d erasures in code words $(s_1, \ldots, s_n)$, if and only if $t \geq 2c + d$.

A.2 Cryptographic Accumulators.

Formally, given a parameter k, and a set D of n values $d_1, \ldots, d_n$, an accumulator has the following components:

- $\mathsf{Gen}(1^k, n)$: This algorithm takes a parameter k represented in unary form 1^k and an accumulation threshold n (an upper bound on the number of values that can be accumulated securely), returns an accumulator key a_k. The accumulator key a_k is part of the q-SDH setup and therefore is public to all parties.
- $\mathsf{Eval}(a_k, \mathcal{D})$: This algorithm takes an accumulator key a_k and a set $\mathcal{D}$ of values to be accumulated, returns an accumulation value z for the value set $\mathcal{D}$.
- $\mathsf{CreateWit}(a_k, z, d_i, \mathcal{D})$: This algorithm takes an accumulator key a_k, an accumulation value z for $\mathcal{D}$ and a value d_i, returns $\perp$ if $d_i \in \mathcal{D}$, and a witness w_i if $d_i \in \mathcal{D}$.
- $\mathsf{Verify}(a_k, z, w_i, d_i)$: This algorithm takes an accumulator key a_k, an accumulation value z for $\mathcal{D}$, a witness w_i and a value d_i, returns true if w_i is the witness for $d_i \in \mathcal{D}$, and false otherwise.

A.3 Construction of $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$.

$\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ is generated as follows:
- Pick $v_1, v_2 \in_R \mathbb{Z}_p$, and let $t_1 = g^{v_1}$ and $t_2 = h^{v_2}$.
- Compute hash $c = \mathsf{H}_{\equiv\mathsf{Com}}(g, h, \mathcal{C}_{\langle g \rangle}(s), \mathcal{C}_{\langle g,h \rangle}(s, r), t_1, t_2)$, where $\mathsf{H}_{\equiv\mathsf{Com}} : \mathbb{G}^6 \to \mathbb{Z}_p$ is a random oracle hash function.
- Let $u_1 = v_1 - c \cdot s$ and $u_2 = v_2 - c \cdot r$.
- Send the proof $\pi_{\equiv\mathsf{Com}} = (c, u_1, u_2)$ along with $\mathcal{C}_{\langle g \rangle}(s)$ and $\mathcal{C}_{\langle g,h \rangle}(s, r)$.

The verifier checks this proof (given $\pi_{\equiv\mathsf{Com}}$, g, h, $\mathcal{C}_{\langle g \rangle}(s)$, $\mathcal{C}_{\langle g,h \rangle}(s, r)$) as follows:
- Let $t_1' = g^{u_1}\mathcal{C}_{\langle g \rangle}(s)^c$ and $t_2' = h^{u_2}(\frac{\mathcal{C}_{\langle g,h \rangle}(s,r)}{\mathcal{C}_{\langle g \rangle}(s)})^c$.
- Accept the proof as valid if $c = \mathsf{H}_{\equiv\mathsf{Com}}(g, h, \mathcal{C}_{\langle g \rangle}(s), \mathcal{C}_{\langle g,h \rangle}(s, r), t_1', t_2')$.

B Analysis of Secure DKG

We rely on the following Lemma of [43].

Lemma 22 ([43]). *Under the discrete-log assumption, Pedersen's VSS satisfies following properties in the presence of a polynomially bounded adversary that corrupts up to t parties.*

(i) If the dealer is not disqualified during the sharing phase, then all honest parties hold secret shares that interpolate to unique polynomial of degree t. In particular, any $t + 1$ of these shares suffice to reconstruct the secret σ.

(ii) The protocol produces information (i.e., commitments $\mathcal{C}_k$ and secret shares σ_i) that can be used at reconstruction time to test for the correctness of each secret share; thus, reconstruction is possible, even in the presence of malicious parties, from any subset of shares containing at least $t + 1$ correct secret shares.

(iii) The view of the adversary is independent of the value of the secret σ, and therefore the secrecy of σ is unconditional.

Note that Lemma 22 also holds when using evaluations instead of coefficients as discussed in Section 9. The coding check (see Equation (2)) ensures that the shared commitments to evaluations are indeed a t degree polynomial except with $1/p$ probability in $\mathbb{Z}_p$. Since p is sufficiently large ($\mathsf{poly}(\kappa)$), the probability of the check failing is negligible in the security parameter.

Fact 23. *If a dealer P_i receives a vote-certificate, all honest parties must have received their corresponding secret shares s_{ij}, s'_{ij}.*

Proof. Suppose a dealer P_i receives a vote-certificate i.e, $t+1$ vote messages. At least one of the vote message is sent by an honest party (say P_j). An honest party P_j sends a vote message only when it receives no blame messages or receives up to t blame messages and dealer P_i sent secret shares s_{ik}, s'_{ik} for every $\langle \mathsf{blame}, i\rangle_k$ message it forwarded.

If party P_j received no blame messages, all honest parties must have received their corresponding secret shares s_{ij}, s'_{ij}; otherwise honest parties would have sent blame messages. On the other hand, if party P_j received $f \le t$ blame messages, $n-t-f$ honest parties must have received their corresponding secret shares; otherwise, these honest parties would have sent blame messages and party P_j would have received more than f blame messages. Since party P_j forwards secret shares s_{ik}, s'_{ik} to party P_k for every $\langle \mathsf{blame}, i\rangle_k$ message it received, all honest parties must have received corresponding secret shares. □

B.1 Proof of Theorem 7

Let $\mathcal{B}$ be the set of parties controlled by the adversary, and $\mathcal{G}$ be the set of honest parties (run by the simulator $\mathcal{S}$). Without of loss of generality, let $\mathcal{B} = [P_1, P_{t'}]$ and $\mathcal{G} = [P_{t'+1}, P_n]$, where $t' \ge t$. Let $Y \in \mathbb{G}$ be the input public key and $\mathsf{H}_{\equiv\mathsf{Com}} : \mathbb{G}^6 \to \mathbb{Z}_p$ is a random oracle hash table for $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$.

(1) Perform Step 1 through Step 6 on the behalf of the uncorrupted parties $P_{t'+1}, \ldots, P_n$ exactly as secure DKG protocol (refer Figure 2) until set `QUAL` is finalized. At the end of Step 6, the following holds:

- Set `QUAL` is well-defined with at least one honest party in it.
- The adversary's view consists of polynomials $f_i(y)$, $f'_i(y)$ for $P_i \in \mathcal{B}$, the secret shares s_{ij}, s'_{ij} for $P_i \in$ `QUAL` $\cap\, \mathcal{G}, P_j \in \mathcal{B}$, and the commitments $\mathcal{C}_i$ for $P_i \in$ `QUAL`.
- $\mathcal{S}$ knows all $f_i(y)$ and $f'_i(y)$ for $P_i \in$ `QUAL` as it knows $n - t'$ shares for each of those.

(2) Perform the following computations for each $i \in \{t+1, \ldots, n\}$ before Step 6 (refer Figure 2).

(a) Compute x_j for party $P_j \in \mathcal{B}$. Similarly, compute x_j for party $P_j \in [P_{t'+1}, P_t]$. Interpolate in the exponent $(0, Y)$ and (j, g^{x_j}) for $j \in [1, t]$ to compute $\mathcal{C}_{\langle g\rangle}(x_i^*) = g^{x_i^*}$.

(b) Compute the corresponding $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ by generating random challenges $c_i \in \mathbb{Z}_p$ and responses $u_{i,1}, u_{i,2} \in \mathbb{Z}_p$, computing the commitments $t_{i,1} = (g^{x_i^*})^{c_i} g^{u_{i,1}}$ and $t_{i,2} = \frac{\mathcal{C}_{\langle g,h\rangle}(x_i, x'_i)^{c_i}}{\mathcal{C}_{\langle g\rangle}(x_i^*)} h^{u_{i,2}}$ and include entry $\langle (g, h, \mathcal{C}_{\langle g\rangle}(x_i^*), \mathcal{C}_{\langle g,h\rangle}(x_i, x'_i), t_{i,1}, t_{i,2}), c_i\rangle$ in the hash table $\mathsf{H}_{\equiv\mathsf{Com}}$ so that $\pi_{\equiv\mathsf{Com}} = (c_i, u_{i,1}, u_{i,2})$.

(3) In the end, $x = \sum_{P_i \in \mathtt{QUAL}} s_i$ such that $Y = g^x$.

Figure 10: Simulator for Secure DKG

Proof. We first prove correctness of the protocol. Observe that all honest parties build the same set of non-disqualified parties `QUAL` in Step 6. This is true because the commitment to the shared polynomials and vote-certificates are posted on the broadcast channel and broadcast channel ensures all honest parties output a common value.

Note that if a party $P_j \in$ `QUAL`, it must have posted its commitment and vote-certificate on the broadcast channel. By Fact 23, all honest parties have received secret shares shared by party P_j. This implies party P_j is not disqualified during the sharing phase. By part (i) of Lemma 22, all honest parties hold correct secret shares and any $t+1$ of these secret shares suffices to reconstruct the secret s_j. This is true for all parties $P_j \in$ `QUAL`. Since, the secret key x is sum of individual secret s_j contributed by $P_j \in$ `QUAL` and each secret s_j can be reconstructed using Lagrange interpolation via a combination of $t+1$ secret shares provided by honest parties, the secret key x can be reconstructed via $t+1$ shares provided by honest parties. This proves property C1 of a secure DKG protocol.

By part (ii) of Lemma 22, there exists information (i.e., commitments) that can be used to verify correctness of each secret share. Observe that each honest party P_j sends g^{x_j} and $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ proof $\pi_{\equiv\mathsf{Com}j}$

at the end of sharing phase. Each party P_i can verify correctness of $\mathcal{C}_{\langle g \rangle}(x_j)$ by checking Equation (3). A valid $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ proof $\pi_{\equiv\mathsf{Com}j}$ proves in zero knowledge that party P_j knows x_j and x'_j thus proving the correctness of g^{x_j}. By using $t+1$ valid g^{x_j}, honest parties can compute the same g^x via Lagrange interpolation in the exponent which is the public key. This proves property C2 of a secure DKG protocol.

Observe that the secret key x is the sum of secrets shared by parties in $\mathtt{QUAL}$ which contains at least one honest party and honest parties select their secret uniformly at random. This suffices to prove property C3 of a secure DKG protocol.

We now prove secrecy. Our proof of secrecy is based on the proof of secrecy in earlier works [26, 31]. We provide a simulator $\mathcal{S}$ for our secure DKG protocol in Figure 10. Without loss of generality, we assume the adversary $\mathcal{A}$ compromises parties $P_1, \ldots, P_{t'}$, where $t' \leq t$, denoted by set $\mathcal{B}$. The rest of the parties $P_{t'+1}, \ldots, P_n$, denoted by set $\mathcal{G}$ are controlled by the simulator.

Informally, the simulator $\mathcal{S}$ with input Y runs as follows. $\mathcal{S}$ will run on the behalf of the honest parties $\mathcal{G}$ Step 1 until Step 6 following exactly the instructions. At this point, the set $\mathtt{QUAL}$ is well-defined and $\mathcal{S}$ knows all $f_i(y)$ and $f'_i(y)$ for $P_i \in \mathtt{QUAL}$ as it knows $n - t'$ shares for each of those. Observe that the view of adversary $\mathcal{A}$ that interacts with $\mathcal{S}$ is identical to the view of $\mathcal{A}$ that interacts with honest parties in a regular run of the protocol. In particular, $\mathcal{A}$ sees following distribution of data:

- Polynomials $f_i(y)$, $f'_i(y)$ for $P_i \in \mathcal{B}$
- Values $f_i(j)$, $f'_i(j)$ for $i \in \mathcal{G}$, $j \in \mathcal{B}$ and values $\mathcal{C}_i$ for $P_i \in \mathtt{QUAL}$

$\mathcal{S}$ will then change the secret shared by one honest party (say P_n) to "hit" the desired public key Y such that the above data distribution observed by $\mathcal{A}$ remains identical. For parties $P_i \in (\mathcal{G} \setminus \{P_n\})$, the input polynomial $f_i(y)$ and $f'_i(y)$ remains identical. Thus, their data distribution remains identical. For party P_n, the input polynomial is modified such that $g^{f^*_n(0)} = g^{s^*_n} = \frac{Y}{\prod_{P_j \in \mathtt{QUAL} \setminus \{P_n\}} g^{s_i}}$ and $f^*_n(j) = s_{nj}$ for $j \in [1, t]$. Define $f'^*(y)$ such that $f^*_n(y) + \lambda f'^*_n(y) = f_n(y) + \lambda f'_n(y)$, where $\lambda = \log_g(h)$. Observe that for these polynomials, the evaluations and commitments seen by parties in $\mathcal{B}$ is identical to the real run of the protocol.

Simulator $\mathcal{S}$ will then compute g^{x_j} for party $P_j \in [P_1, P_t]$ and interpolate in the exponent $(0, Y)$ and (j, g^{x_j}) for $j \in [1, t]$ to compute $\mathcal{C}_{\langle g \rangle}(x^*_i) = g^{x^*_i}$ and the corresponding $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ and publish these values. Observe that these values pass the verification in the real run of protocol.

It remains to be shown that polynomials $f^*_i(y)$ and $f'^*_i(y)$ belong to the right distribution. For $\mathtt{QUAL} \setminus (\mathcal{G} \setminus \{P_n\})$, this is trivially true as they are defined identically to $f_i(y)$ and $f'_i(y)$ which were chosen uniformly at random. For f^*_n, the polynomial evaluates to random values $f_n(j)$ at $j \in [1, t]$ and evaluates to $\log_g(s^*_n)$ required to hit Y. Finally, $f'^*_n(y)$ is defined as $f^*_n(y) + \lambda f'^*_n(y) = f_n(y) + \lambda f'_n(y)$, and since $f'_n(y)$ is chosen to be random, so is $f'^*_n(y)$. □

C A Lower Bound on the Communication Complexity of Weak Gradecast

In this section, we show a quadratic communication lower bound for the weak gradecast protocol. The proof of this lower bound is a trivial extension of the communication lower bound for Byzantine broadcast by Dolev and Reischuk [17].

Lemma 24. *There does not exist a protocol for weak gradecast tolerating t Byzantine parties with a communication complexity of at most $t^2/4$ messages.*

Proof. Suppose for the sake of contradiction, there exists such a protocol. Consider the parties being partitioned into the following two sets: A: a set of $\lceil t/2 \rceil$ parties, and B: all remaining parties which includes the designated sender r.

We consider two executions W1 and W2 where the third property of weak gradecast (i.e., if an honest party outputs a value v with a grade of 2, all other honest parties output value v with a grade of ≥ 1)

is violated in the W2. In the first execution (W1), all parties in A are Byzantine. Parties in A do not communicate with each other. Towards B, parties in A execute honestly except they ignore the first $\lceil t/2 \rceil$ messages from parties in B. The designated sender $r \in A$ sends value v to all parties. Since, the maximum faults in W1 is $\lceil t/2 \rceil$ and the designated sender is honest, all honest parties decide value v with a grade of 2.

Since the communication complexity of the protocol is at most $t^2/4$, there must exist a party (say s) in A that receives at most $t/2$ messages from parties in B; otherwise the communication complexity will be more than $t^2/4$. Let B_s be the set of all parties that send messages to party s in W1.

In the second execution (W2), all parties in $A \setminus \{s\}$ are Byzantine and all parties in B_s are Byzantine which includes the designated sender r. The total number of Byzantine parties is $(\lceil t/2 \rceil - 1) + \lceil t/2 \rceil \leq t$ which is within allowed fault threshold t. The designated sender r sends value v. The parties in B_s execute the protocol in the same way as in W1 except they do not send any messages to party s. Parties in $A \setminus \{s\}$ execute the protocol in the same way as in W1. Party s in W1 behave as an honest party which did not receive the first $\lceil t/2 \rceil$ messages which is similar to party s in W2 which receives no messages. Thus, parties in $B \setminus B_s$ cannot distinguish W1 and W2. Thus, they decide value v with a grade of 2. Since, party s does not receive any messages in W2, it does not decide v with a grade of ≥ 1. This violates the third property of weak gradecast where all honest parties need to output a common value v with a grade of 2. A contradiction. ☐

Theorem 25. *Let $\mathcal{CC}(\ell)$ be the communication complexity of weak gradecast for ℓ bit input. Then $\mathcal{CC}(\ell) = \Omega(n\ell + n^2)$*

Proof. Since each party must learn ℓ bit input, the protocol needs $\Omega(n\ell)$ bits (The argument follows from [25]). From Lemma 24, weak gradecast requires $\Omega(n^2)$ even for a single bit input. Thus, $\mathcal{CC}(\ell) = \Omega(n\ell + n^2)$ for ℓ bit input. ☐

D BFT SMR from RandPiper [9]

Let e be the current epoch and L_e be the leader of epoch e. For each epoch e, party P_i performs the following operations:

1. **Epoch advancement.** When $\textsf{epoch-timer}_{e-1}$ reaches 0, enter epoch e. Upon entering epoch e, send highest ranked certificate $\mathcal{C}_{e'}(B_l)$ to L_e. Set $\textsf{epoch-timer}_e$ to 11Δ and start counting down.

2. **Propose.** L_e waits for 2Δ time after entering epoch e and broadcasts $\langle \textsf{propose}, B_h, \mathcal{C}_{e'}(B_l), z_{pe}, e\rangle_{L_e}$ where B_h extends B_l. $\mathcal{C}_{e'}(B_l)$ is the highest ranked certificate known to L_e.

3. **Vote.** If $\textsf{epoch-timer}_e \geq 7\Delta$ and party P_i receives the first proposal $p_e = \langle \textsf{propose}, B_h, \mathcal{C}_{e'}(B_l), z_{pe}, e\rangle_{L_e}$ where B_h extends a highest ranked certificate, invoke $\textsf{Deliver}(\textsf{propose}, p_e, z_{pe}, e)$. Set $\textsf{vote-timer}_e$ to 2Δ and start counting down. When $\textsf{vote-timer}_e$ reaches 0, send $\langle \textsf{vote}, H(B_h), e\rangle_i$ to L_e.

4. **Vote cert.** Upon receiving $t+1$ votes for B_h, L_e broadcasts $\langle \textsf{vote-cert}, \mathcal{C}_e(B_h), z_{ve}, e\rangle_{L_e}$.

5. **Commit.** If $\textsf{epoch-timer}_e \geq 3\Delta$ and party P_i receives the first $v_e = \langle \textsf{vote-cert}, \mathcal{C}_e(B_h), z_{ve}, e\rangle_{L_e}$, invoke $\textsf{Deliver}(\textsf{vote-cert}, v_e, z_{ve}, e)$. Set $\textsf{commit-timer}_e$ to 2Δ and start counting down. When $\textsf{commit-timer}_e$ reaches 0, if no equivocation for epoch-e has been detected, commit B_h and all its ancestors.

6. **(Non-blocking) Equivocation.** Broadcast equivocating hashes signed by L_e and stop performing epoch e operations.

Figure 11: **BFT SMR Protocol** from RandPiper [9]