Synchronous Distributed Key Generation without Broadcasts

Nibesh Shrestha[1], Adithya Bhat[2], Aniket Kate[3], and Kartik Nayak[4]

[1] Supra Research nibeshrestha2@gmail.com
[2] Visa Research haxolotl.research@gmail.com
[3] Purdue University aniket@purdue.edu
[4] Duke University kartik@cs.duke.edu

Abstract. Distributed key generation (DKG) is a key building block in developing many efficient threshold cryptosystems. This work initiates the study of communication complexity and round complexity of DKG protocols over a point-to-point (bounded) synchronous 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 any $t < n/2$ Byzantine faults among n parties. We present two variants of the protocol: (i) a protocol with worst-case $O(\kappa n^3)$ communication and $O(t)$ rounds, and (ii) a protocol with expected $O(\kappa n^3)$ communication and expected constant rounds. In the process of achieving our results, we design (1) a novel weak gradecast protocol with a communication complexity of $O(\kappa n^2)$ for linear-sized inputs and constant rounds, (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 constant rounds, and (4) a multi-valued validated Byzantine agreement (MVBA) protocol with $O(\kappa n^3)$ communication complexity for linear-sized inputs and expected constant rounds. Each of these primitives is of independent interest.

1 Introduction

The problem of distributed key generation (DKG) is setting 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 number of trust assumptions placed in cryptographic protocols such as threshold signatures [11, 55] and threshold encryption schemes [21]. These threshold cryptosystems can themselves be used to implement random beacons [15, 24], reduce the complexity of consensus protocols [56, 59], in multiparty computation protocols [36, 37], or to outsource management of secrets to multiple, semi-trusted authorities [25, 43].

Given its widespread applications and their recent adoption in practice (e.g., [24]), we need efficient solutions for DKG. An ideal solution for DKG would have low communication complexity, low latency, optimal resilience, and provide uniform randomness of generated keys such that the generated keys can be useful in a wider class of cryptosystems while being secure. This work focuses

on the synchronous network setting. Synchronous protocols have the advantage of tolerating up to a minority corruption. While a myriad of DKG protocols [16, 33, 35, 49, 52] have been proposed in this setting, existing solutions fall short in one way or the other. For example, Pedersen's DKG [52] produces non-uniform keys in the presence of the adversary, the DKG protocol due to Gennaro et al. [33] has high latency as it requires additional secret sharing using Feldman's VSS [27], and the protocol due to Gurkhan et al. [35] does not generate keys for discrete log-based cryptosystems.

Moreover, all the DKG protocols considered in the synchronous model assume a *broadcast channel* (that provides a consensus abstraction) and invoke $\Omega(n)$ broadcasts across two or more rounds [6], 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 (κ is a security parameter) in the absence of DKG-based threshold signatures, instantiating a broadcast channel with state-of-the-art Byzantine broadcast [2, 23] or Byzantine agreement [41] trivially blows up the communication complexity to $O(\kappa n^4)$.

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 DKG 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 has worst-cast $O(\kappa n^3)$ communication and $O(t)$ rounds whereas the second protocol has expected $O(\kappa n^3)$ communication and constant rounds in expectation.

1.1 Key Technical Ideas and Results

Our DKG protocols avoid the broadcast channel assumption and use a Byzantine consensus process in a non-black-box fashion to achieve $O(\kappa n^3)$ communication. Compared to the existing broadcast-based DKG protocols which require $\Omega(n)$ broadcasts over two or more rounds, our protocols require a single invocation of consensus instance. While DKG protocols [3, 42] without broadcast channel assumption have been explored in the asynchronous model, they either incur high communication [42] or do not generate keys for discrete log-based cryptosystems [3] or use stronger cryptographic assumptions [20]. More importantly, protocols designed for asynchronous or partially-synchronous settings can only tolerate up to $t < n/3$ Byzantine failures, which is sub-optimal for many DKG applications such as random beacons [24]. In the synchronous model, we provide the first solutions to DKG without a 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 [27, 51] 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`.

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

	Net.	Res.	Comm.	Round	Sim.	Dlog	Setup	Crypto Assumption
Pedersen [52]	sync.	1/2	$O(\kappa n^4)$	$O(t)$	✗	✓	PKI	DL
Gennaro et al. [33]	sync.	1/2	$O(\kappa n^4)$	$O(t)$	✓	✓	PKI	DL
Canetti et al. [16]	sync.	1/2	$O(\kappa n^4)$	$O(t)$	✓	✓	PKI	DL
Neji et al. [49]	sync.	1/2	$O(\kappa n^4)$	$O(t)$	✓	✓	PKI	RO+CDH
ETHDKG [54]	sync.	1/2	$O(\kappa n^4)$	$O(t)$	✗	✓	PKI	RO+CDH
Gurkhan et al. [35]	sync.	$\log n$	$\tilde{O}(\kappa n^3)$	$O(t)$	✗	✗	PKI	RO+SXDH+CBDH
NIDKG [34]	sync.	1/2	$O(\kappa n^4)$	$O(t)$	✓	✓	PKI	RO+DDH+...*
Hybrid-DKG [39]	psync.	1/3	$O(\kappa n^4)$	$O(t)$	✓	✓	PKI	RO+DL
Kokoris et al. [42]	async.	1/3	$O(\kappa n^4)$	$O(t)$	✗	✓	PKI	RO+DDH
Abraham et al. [3]	async.	1/3	$\tilde{O}(\kappa n^3)$	$E(O(1))$	✗	✗	PKI	RO+SXDH
Das et al. [20]	async.	1/3	$O(\kappa n^3)$	$E(O(\log n))$	✓	✓	PKI	RO+DCR+DDH
Das et al. [19]	async.	1/3	$O(\kappa n^3)$	$E(O(\log n))$	✓	✓	PKI	RO+DL
Our work (§ 9.1)	sync.	1/2	$O(\kappa n^3)$	$E(O(1))$	✓	✓	PKI+PoT	RO+CDH+q-SDH
Our work (§ 9.2)	sync.	1/2	$O(\kappa n^3)$	$O(t)$	✓	✓	PKI+PoT	RO+q-SDH

κ is the security parameter. **Net.** refers to the network model. **Res.** refers to the number of Byzantine faults tolerated in the system. **Comm.** refers to the communication complexity. **Sim.** means the protocol maintains secrecy which can be proven via a simulator. **Primitive** refers to the cryptographic primitives used. **PoT** refers to the power of tau setup required for bilinear accumulators. This setup can be removed by making use of Merkle trees at the cost of $\log n$ multiplicative communication overhead. **E(.)** implies "in expectation". *NIDKG assumes RO, rleaf-IND-CCA, DDH, Erasures, and one-more DH.

In our protocols, we replace broadcast channels with weaker primitives such as gradecast [28,41]. Thus, parties first perform secret sharing by using this weaker primitive to identify a set of at least $n - t$ parties who correctly shared their secrets, where t is the fault tolerance. During the sharing phase, no consensus primitives are invoked to agree on the set of qualified parties. The downside of this approach is that different honest parties may have different views regarding the acceptance of shared secrets. As a result, different honest parties obtain different sets 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 set where the input is their individual $\mathsf{AcceptList}$. Once, the Byzantine consensus primitive terminates and outputs a common set $\mathsf{AcceptList}_k$, 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

1. Communication optimal weak gradecast. As a building block, we first provide a communication optimal weak gradecast protocol satisfying the gradecast definition of Katz and Koo [41][5]. Our weak gradecast protocol incurs $O(n\ell+$

[5] This definition is slightly weaker than the one presented by Feldman and Micali [28].

κn^2) communication for ℓ bit input and does not require use of threshold signatures. In the same setting, the gradecast protocol of Katz and Koo [41] incurs a communication complexity of $O(\kappa n^3)$ even for a single bit input. With q-Strong Diffe Hellman (q-SDH) [12] setup assumption, we show the following result:

Theorem 1 (Informal). *Assuming a public-key infrastructure and a universal structured reference string under q-SDH assumption, there exists a gradecast protocol for an input of size ℓ bits with $O(n\ell + \kappa n^2)$ communication tolerating $t < n/2$ Byzantine faults.*

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 verifiable proof vouching that all parties in the set have correctly shared their secrets and these secrets are thus recoverable. We call this sub-protocol "Recoverable set of shares". Using our communication optimal gradecast, our recoverable set of shares protocol can be achieved in $O(\kappa n^3)$ communication and constant rounds.

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

		Network	Resilience	Communication	Round	Assumption
Cachin et al.	[14]	async.	1/3	$O(n^2\ell + \kappa n^2 + n^3)$	$E(O(1))$	Threshold setup
VABA	[4]	async.	1/3	$O(n^2\ell + \kappa n^2)$	$E(O(1))$	Threshold setup
DUMBO-MVBA	[44]	async.	1/3	$O(n\ell + \kappa n^2)$	$E(O(1))$	Threshold setup
Our work		sync.	1/2	$O(n^2\ell + \kappa n^3)$	$E(O(1))$	PKI

E(.) implies "in expectation".

3. Oblivious leader election. We design a communication efficient oblivious leader election (OLE) protocol with $O(\kappa n^3)$ communication and constant rounds. The OLE protocol elects a common honest leader with probability at least $\frac{1}{2}$. To the best of our knowledge, all prior OLE protocols requires n^2 weaker VSS instances and incurred $\Omega(n^4)$ communication [41] or required stronger cryptographic assumptions to achieve $O(\kappa n^3)$ communication [3]. In this work, we build an OLE protocol using only n weaker VSS instances and a non-interactive threshold signature scheme [15] to generate randomness. The security of our OLE protocol is based on the computational Diffie-Hellman (CDH) problem in the random oracle model. In particular, we show the following:

Theorem 2 (Informal). *Assuming a public-key infrastructure, a universal structured reference string under q-SDH assumption, random oracle, and CDH, there*

exists an oblivious leader election protocol with $O(\kappa n^3)$ communication and $O(1)$ rounds tolerating $t < n/2$ Byzantine faults.

4. 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) [14] in the literature.

MVBA was first formulated by Cachin et al. [14] to allow honest parties to decide on any externally valid values. Recent works [4, 44] have given communication efficient protocols for MVBA in the asynchronous model tolerating $t < n/3$ Byzantine faults. For long messages of size ℓ, the protocol due to Abraham et al. [4] incurs $O((\ell + \kappa)n^2)$ communication and the protocol due to Lu et al. [44] incurs $O(n\ell + \kappa n^2)$. Both of these works assume a threshold setup. Without threshold setup assumptions, the communication blows up by a factor of n in all of the above protocols.

To the best of our knowledge, no MVBA protocols have been formulated in the synchronous model tolerating $t < n/2$ faults. Recently, Nayak et al. [48] 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 construct the first MVBA protocol in the synchronous setting without threshold setup. Our MVBA protocol incurs expected $O(n^2\ell + \kappa n^3)$ communication for inputs of ℓ bit and expected 36 rounds. Specifically, we show the following result:

Theorem 3 (Informal). *Assuming a public-key infrastructure, random oracle, CDH, and a universal structured reference string under q-SDH assumption, there exists a multi-valued validated Byzantine agreement protocol for an input of size ℓ with expected $O(n^2\ell + \kappa n^3)$ communication and expected 36 rounds tolerating $t < n/2$ Byzantine faults.*

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 DKG protocol with expected $O(\kappa n^3)$ communication and expected 47 rounds.

Theorem 4 (Informal). *Assuming public-key infrastructure, random oracle, a universal structured reference string under q-SDH assumption and CDH, there exists a protocol that solves secure synchronous distributed key generation tolerating $t < n/2$ Byzantine faults with expected $O(\kappa n^3)$ communication and expected 47 rounds.*

Although the DKG protocol terminates in constant expected time, it can take linear time in the worst case. In this case, the protocol incurs $O(\kappa n^4)$ communication. As an alternative, we provide a protocol that incurs $O(\kappa n^3)$ communication in the worst-case. RandPiper [10] provides a Byzantine fault tolerant state machine replication (BFT SMR) protocol with $O(\kappa n^2)$ communication per epoch even for $O(n)$-sized input. Here, an epoch is a period that incurs 7 rounds. In this protocol, we execute the BFT SMR protocol for $t+1$ epochs with each epoch coordinated by a distinct leader. The leader proposes his set AcceptList 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 a universal structured reference string under q-SDH assumption there exists a protocol that solves secure synchronous distributed key generation tolerating $t < n/2$ Byzantine faults with $O(\kappa n^3)$ communication and $11 + 7(t+1)$ rounds.*

Limitations. In this work, we assume that the adversary is static, similar to several DKGs [33,35,38,49,52,54] in the literature. Canetti et al. [16] 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. Very recently, Bacho et al. [5] gave a relaxed definition of DKG and show that prior DKG protocols such as Gennaro et al [33] are adaptively-secure under this relaxed definition. It could be interesting to see if our protocols are adaptively-secure under their relaxed definition. In addition, our protocols make the q-SDH assumption. 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.

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 [16,18,26,33–35,49,52,54] 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 [2,23] or Byzantine agreement [41]. To the best of our knowledge, all optimally resilient deterministic Byzantine consensus protocols incur $O(\kappa n^3)$ communication without threshold signatures and $t+1$ rounds [23]. For randomized consensus protocols, the best known protocol with optimal resilience in this setting is Katz and Koo [41] which incurs $O(\kappa n^4)$ communication. Although, randomized consensus protocols terminate in expected constant rounds, n parallel instances of randomized consensus requires expected $O(\log n)$ rounds to terminate [9]. For the sake of simplicity, we assign a communication of $O(\kappa n^4)$ and $O(t)$ rounds for the DKG protocols that

use broadcast channel in Table 1. Compared to all prior DKG protocols, our protocols do not use broadcast channel and use Byzantine consensus protocols. In fact, our protocols require a single consensus invocation and incur either expected $O(\kappa n^3)$ communication and expected $O(1)$ rounds or worst-case $O(\kappa n^3)$ communication and $O(t)$ rounds. Our protocols are secure against static failures and generate uniform keys for discrete logarithm based cryptosystems.

We also argue that the protocols by Momose and Ren [47] and Tsimos et al. [58] are relevant but not sufficient to achieve our goals. Momose and Ren [47] 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 but with *sub-optimal* resilience and linear round complexity. Similarly, Tsimos et al. [58] present a communication-efficient broadcast protocol RandomBroadcast in the bulletin PKI setting. It works with $t < (1-\epsilon)$ resilience, $O(\kappa^2 n^2)$ communication, linear round complexity, and negligible error probability. Using RandomBroadcast to instantiate broadcast channels will result in DKG protocols with optimal resilience, $O(\kappa^2 n^3)$ communication, linear round complexity and negligible error probability. In contrast, our protocols have optimal resilience, $O(\kappa n^3)$ communication and expected $O(1)$ rounds (or $O(t)$ rounds).

Pedersen [52] 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 [27]. Gennaro et al. [33] 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. [16] extended Gennaro et al.'s DKG to handle adaptive corruptions.

Neji et al. [49] 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.

Gurkhan et al. [35] presented DKG protocol without a complaint phase by using publicly verifiable secret sharing (PVSS) [17] scheme. However, they tolerate only $\log n$ Byzantine faults and do not generate keys for discrete-logarithms based cryptosystems; reducing its usefulness.

Recently, Groth [34] presents a non-interactive DKG protocol with a refresh procedure that allows refreshing the secret key shares to a new committee. Erwig et al. [26] considers large scale non-interactive DKG protocol and handles mobile Byzantine faults. Both of above protocols assume broadcast channels.

Very recently, Cascudo et al. [18] presented a DKG protocol using PVSS [17] scheme which generates field elements as the secret keys. Their protocol assumes broadcast channel and invokes $\Omega(n)$ broadcasts; hence their protocol would require $O(\kappa n^4)$ communication and $O(t)$ rounds in the point-to-point model. Using our framework of reducing $O(n)$ broadcasts to a single broadcast, we can indeed use PVSS scheme (similar to their work) to obtain DKG protocol with improved

round complexity but would require additional cryptographic assumptions such as SXDH.

Several other works tackle the DKG problem from different angels. Kate et al. [39] reduced the size of input to the broadcast channel from $O(n)$ to $O(1)$ by using polynomial commitments [40]. Tomescu et al. [57] reduce the computational cost of dealings in Kate et al. [39] at the cost of a logarithmic increase in communication cost. Schindler et al. [54] 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. [39] 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. [42] 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. Abraham et al. [3] 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. Das et al. [20] gave the dlog-based DKG protocol with $O(\kappa n^3)$ communication and optimal resilience in the asynchronous model. However, their protocol incurs expected $O(\log n)$ round complexity and requires stronger Decisional Composite Residuosity (DCR) assumption. Very recently, Das et al. [19] gave the dlog-based DKG protocol with $O(\kappa n^3)$ communication and optimal resilience in the asynchronous model with discrete-log assumption. However, their construction still incurs $O(\log n)$ round complexity. We note that while DKG protocols have been designed with lesser assumption (i.e., DL assumption in Das et al. [19]) in the asynchronous model tolerating $t < n/3$ Byzantine failures, designing protocols tolerating $t < n/2$ Byzantine failures presents its own unique challenges and does not make our protocols sub-optimal.

Concrete round complexity. All prior synchronous DKG protocols invoke $\Omega(n)$ broadcasts over two or more rounds. Invoking broadcast channels with the state-of-the art Byzantine consensus protocols would require at least $2t + 2$ rounds. Our expected constant round DKG protocol requires only 47 rounds in expectation. When $t > 23$, the concrete round complexity of our protocol is less compared to other protocols. We can alternatively use PVSS scheme to share secrets (following Cascudo et al. [18]) to obtain improved concrete round complexity (but would require stronger SXDH assumptions).

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 [2, 4, 13, 29, 41, 47, 56, 59]. We review the most recent and closely related works.

Multi-valued validated Byzantine agreement was first introduced by Cachin et al. [14] 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 $O(n^2\ell + \kappa n^2 + n^3)$ communication for input of size ℓ. Later, Abraham et al. [4] gave an MVBA protocol with optimal resilience and $O((\ell + \kappa)n^2)$ communication in the same asynchronous setting. Lu et al. [44] extended the work of Abraham et al. [4] to handle long messages of size ℓ with a communication complexity of $O(n\ell + \kappa n^2)$. All of these protocols assumed threshold setup. In the absence of threshold setup, 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 $O(n^2\ell + \kappa n^3)$ for inputs of size ℓ and does not assume threshold setup and terminates in expected constant rounds.

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 $O(1)$ rounds. Feldman and Micali [29] 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 [41] 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 [41] and reduce its communication by linear factor while handling multi-valued input by designing a communication optimal gradecast protocol. A simple and efficient BBA tolerating $t < n/3$ Byzantine faults in the authenticated model was given by Micali [46]. Abraham et al. [2] 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.

Very recently, Abraham et al. [1] gave a BBA protocol in the authenticated model without PKI and digital signatures tolerating $t < n/3$ Byzantine faults. Their protocol has an expected communication complexity of $O(n^4 \log n)$ and expected constant rounds.

2.3 Related Work in the Gradecast Literature

Gradecast was originally introduced by Feldman and Micali [28] in the authenticated model with PKI and digital signatures. They gave a protocol tolerating $t < n/3$ Byzantine faults. Katz and Koo [41] gave a slightly relaxed definition of gradecast which allows honest parties with output any value with a grade of 1 when no honest party outputs a value with grade of 2. They gave a gradecast protocol tolerating $t < n/2$ Byzantine faults in the authenticated model with PKI and digital signatures. Their gradecast protocol incurs $O(\kappa n^3)$ communication even for a single bit. We also recall the gradecast protocol with multiple grades introduced by Garay et al. [32] and later improved by [31]. Their gradecast protocol supports arbitrary number of grades. Their protocol works in the authenticated model with PKI and digital signatures and has a communication

complexity of $O(g^*(\ell + \kappa)n^2)$ for input of size ℓ bit where g^* is the maximum grade supported. Compared to all these works, our gradecast protocol satisfies the definition of Katz and Koo [41] and tolerates $t < n/2$ Byzantine faults and incurs $O(n\ell + \kappa n^2)$ communication for inputs of size ℓ in authenticated model with PKI and digital signatures.

3 Model and Preliminaries

We consider a system consisting of n parties $(P_1, \ldots, P_n)$ with pair-wise reliable, authenticated point-to-point channels, 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. We assume a synchronous communication model. Thus, if an honest party sends a message at the beginning of some round, the recipient receives the message by the end of that round.

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 [33]. In particular, we assume that the adversary is unable to compute discrete logarithms modulo large (based on the security parameter κ) primes.

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. In addition, we use a hash function $H' : \mathbb{G} \rightarrow \{0,1\}^\kappa$ in our leader election protocol.

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.

3.1 Definitions

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

Definition 1 (Secure DKG for Dlog based cryptosystems [33]). *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 same value of public key $y = g^x \in \mathbb{G}$, where $x \in \mathbb{Z}_p$ is 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.

Weak Gradecast. Weak gradecast is a relaxed version of gradecast [28] introduced by Katz and Koo [41].

Definition 2 (Weak Gradecast [41]). *A protocol with a designated sender P_i holding an initial input v is a weak gradecast protocol tolerating $t < n/2$ Byzantine parties if the following conditions hold*

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 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.*

Oblivious leader election. An oblivious leader election protocol elects a common honest leader with some constant probability.

Definition 3 (Oblivious Leader Election [41]). *A protocol for parties $P_1, \ldots, P_n$ is an oblivious leader election protocol with fairness α tolerating t Byzantine failures if each honest party P_i outputs a value $v_i \in [n]$ and the following conditions holds with probability at least α:*

There exists a value $j \in [n]$ such that (i) each honest party P_i outputs $v_i = j$, and (ii) party P_j is honest.

Multi-valued validated Byzantine agreement. In an MVBA protocol, there is an external validity function ex-validation that every party has access to. Every honest party start with some externally valid input v_i, and on termination must output a value. An MVBA protocol has following properties:

Definition 4 (Multi-valued Validated Byzantine Agreement [4,44]). *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.*

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 [53]. 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 Section J.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 Section J.2.

In this paper, we use *collision free bilinear accumulators* from Nguyen [50] as cryptographic accumulators which generates constant sized witness, but requires q-SDH assumption. Alternatively, we can use Merkle trees [45] (and avoid q-SDH assumption) at the expense of $O(\log n)$ multiplicative communication.

Non-interactive threshold signature scheme. We use (t, n) *non-interactive threshold signature scheme* of Cachin et al. [15] in one of our protocols. The threshold signature scheme is secure against static adversary. The signature scheme consists of the following efficient algorithms: $\mathsf{KeyGen}_{\mathsf{TS}}$, $\mathsf{Sign}_{\mathsf{TS}}$, $\mathsf{ShareVerify}_{\mathsf{TS}}$, $\mathsf{Combine}_{\mathsf{TS}}$, $\mathsf{Verify}_{\mathsf{TS}}$. More details on these algorithms in provided in Section J.3.

Non-Interactive Proof-of-Equivalence of commitments [39]. Given two 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 it 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 Section J.4.

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 Secure DKG with Two Broadcast Rounds

We first present a secure DKG protocol assuming a broadcast channel motivated from Gennaro et al. DKG [33]. The presented DKG reduces the number of required rounds with broadcast to two, which is a significant improvement over [33] requiring three broadcast rounds in the best case and five broadcast rounds otherwise.[6] In later sections, we replace the broadcast channel with a novel consensus primitives to design communication-efficient DKG protocols.

[6] Using NIZK similar to us, the number of rounds for Gennaro et al. DKG [33] can be reduced to two in the best case and three otherwise in a rather straightforward man-

Gennaro et al. [33] presented a secure DKG protocol that produces uniform public keys based on Pedersen's VSS [51]. 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 [27] 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.

The protocol in Figure 1 improves upon the DKG protocol of Gennaro et al. [33] in the following ways.

Improving latency in the sharing phase. We improve latency by reducing information posted on the broadcast channel by using improved eVSS (iVSS) protocol [10] which requires only 2 broadcast rounds.[7] 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[8]. 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 [33], commitments to the secret share are commitments to the coefficients of a t-degree polynomial, which imply verifying a share requires $O(t)$ computations. This re-

ner; however, reducing to two broadcast rounds in all situations is the key challenge here.

[7] 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.

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

Sharing Phase

1. **Deal.** Each party (as a dealer) P_i selects two random polynomials $f_i(y)$, $f'_i(y) \in \mathbb{Z}_p[y]$ 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

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

$$\prod_{k=1}^{n} \mathcal{C}_{jk}^{\mathsf{Code}_k} = 1_{\mathbb{G}}, \text{ where } \{\mathsf{Code}_1, \ldots, \mathsf{Code}_n\} \in C^{\perp} \text{ using Equation (4)} \tag{2}$$

   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 $t + 1$ distinct $\langle \mathsf{vote}, i \rangle$ messages (called the vote-certificate), 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.

Fig. 1: **Secure distributed key generation in dlog-based cryptosystems**

sults 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 [17, 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 the following vector

$$C^{\perp} := \{\mathsf{Code}_1, \ldots, \mathsf{Code}_n; \mathsf{Code}_i = \mathsf{poly}(i) \prod_{j=1, j\neq i}^{n} 1/(i-j) \\ \mathsf{poly}(x) \text{ is a random polynomial of degree } n-t+1\} \quad (4)$$

is orthogonal to C. 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 [39]. Instead of executing an additional secret sharing, assuming random oracle, we make use of the NIZK proof of equivalence of commitments $\mathsf{NIZKPK}_{\equiv\mathsf{Com}}$ 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 $\mathcal{C}_{\langle g \rangle}(x_i)$.

We present detailed security analysis in Section K.

5 Communication Optimal Weak Gradecast

One of the main tools in the design of our communication efficient protocols is our communication optimal *weak gradecast* protocol. Gradecast (aka graded broadcast) is a relaxed version of broadcast introduced by Feldman and Micali [28] which can be obtained in constant number of rounds. Feldman and Micali [28] provided a gradecast protocol tolerating $t < n/3$ Byzantine faults in the *plain authenticated* model without PKI and digital signatures. Later, Katz and Koo [41] 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 [41] incurs $O(\kappa n^3)$ communication even for a single bit input in the absence of threshold signatures.

In this work, we present a gradecast protocol with an optimal communication complexity of $O(n\ell + \kappa n^2)$ for ℓ bit input.

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

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. Add an index j to each code word s_j to obtain $\mathcal{D} = [(1, s_1), \ldots, (n, s_n)]$. Compute accumulation value $z_i = \mathsf{eval}(a_k, \mathcal{D})$. If $z_e \neq z_i$ abort. Otherwise, compute witness w_j for each element $(j, s_j) \in \mathcal{D}$ using $\mathsf{CreateWit}$ function and send $\langle \mathsf{codeword}, \mathsf{mtype}, s_j, w_j, z_e, e \rangle_i$ to party P_j $\forall P_j \in \mathcal{P}$.
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 first accumulation value z_e, decode m using DEC function.

Fig. 2: **Deliver function**

Our gradecast (refer Figure 3) implements weaker gradecast [41] (Definition 2) which relaxes gradecast [28] 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 when no honest party outputs a value with a grade of 2. This weaker gradecast suffices for our purpose. In Section P, we show a quadratic lower bound on the communication complexity of weak gradecast for completeness.

Deliver. As a building block, we first present a $\mathsf{Deliver}$ function (refer Figure 2) used by an honest party to efficiently propagate long messages. This function is adapted from RandPiper [10] where linear-sized messages are propagated among all honest parties with $O(\kappa n^2)$ communication cost. The $\mathsf{Deliver}$ 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 $\mathsf{Deliver}$ 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 $\mathsf{Deliver}$ function is not invoked within an epoch. The $\mathsf{Deliver}$ function incurs 2 rounds.

The gradecast protocol is presented in Figure 3. In round 1, 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. All-to-all multicasting

Set $o_i = \perp$ and $g_i = \perp$. Each party P_i performs the following operations:

- **Round 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.
- **Round 2:** If party P_i receives $pr := \langle \mathsf{gcast}, v, z \rangle_j$ for the first time, then invoke $\mathsf{Deliver}(\mathsf{gcast}, pr, z)$.
- **Round 4:** If party P_i invoked $\mathsf{Deliver}$ in round 2 and no party P_j equivocation has been detected so far, set $o_i = v$ and $g_i = 2$. 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).
- **At any round:** If equivocating hashes signed by party P_j are detected, multicast the equivocating hashes.

Fig. 3: **Weak Gradecast with $O(n\ell + (\kappa + w)n^2)$ communication.**

of κ-sized signed hashes incurs only $O(\kappa n^2)$ in communication. The reduction in communication is obtained via the use of efficient erasure coding schemes [53], cryptographic accumulators [8] and multicast of equivocating hashes (if any).

In round 2, if party P_i receives $\langle \mathsf{gcast}, v, z \rangle_j$, it invokes $\mathsf{Deliver}$ to propagate long message v. Note that $\mathsf{Deliver}$ function requires 2 rounds. Round 3 accommodates steps of $\mathsf{Deliver}$ function invoked in rounds 2. In round 4, each party P_i sets its output value and grade as follows. If party P_i received $\langle \mathsf{gcast}, v, z \rangle_j$ in round 2 and did not detect any equivocation so far, it outputs value v with a grade of 2. Otherwise, party P_i outputs the first value it received with a grade of 1. If no value has been received, Pi outputs $\perp$ with a grade of 0. Note that if node Pi receives $\langle \mathsf{gcast}, v, z \rangle_j$ in round 1, but detects an equivocation at a later point, it outputs value v with a grade of 1.

The first value. If node P_i receives a valid code word corresponding to value v or $\langle \mathsf{gcast}, v, z \rangle_j$ before receiving any other values (i.e, $\langle \mathsf{gcast}, v', z' \rangle_j$ or a valid code word for any other value), then value v is the first value for P_i. Hereafter, node P_i only decodes value v when it receives $t + 1$ valid code words for it.

We present detailed security analysis in Section L.

6 Recoverable Set of Shares

In Section 4, we presented a secure DKG protocol by assuming broadcast channels. 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 [2, 23, 41]. 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 1), each honest party outputs a common set `QUAL` consisting of size at least $n - t$ parties such that the secrets shared by parties in set `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 primitive such as 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 set $\mathsf{AcceptList}_i$ of 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.

We stress that in recoverable set of shares protocol, honest parties need not agree on a common set and may output a different set of 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.

We call an $\mathsf{AcceptList}$ *certified* if it is accompanied by a set of signatures from at least $t + 1$ parties. The set of $t + 1$ signatures on $\mathsf{AcceptList}$ forms the certificate for $\mathsf{AcceptList}$ and denoted as $\mathcal{AC}(\mathsf{AcceptList})$.

Definition 5 (Recoverable Set of Shares). *Each party P_i, as a dealer, secret shares a uniformly random input s_i. Each honest party outputs an n element certifed list $\mathsf{AcceptList}_i$ with an entry corresponding to each party as a dealer such that $\mathsf{AcceptList}_i[j] \in \{0, 1, 2\}\ \forall j \in [n]$. A recoverable set of shares protocol tolerating t Byzantine failures satisfies the following properties:*

1. *If dealer P_j is honest, then each honest party P_i outputs $\mathsf{AcceptList}_i[j] = 2$.*
2. *A certified $\mathsf{AcceptList}_i$ must have $|\{h \mid \mathsf{AcceptList}_i[h] = 2\}| \geq n - t$.*
3. *If $\mathsf{AcceptList}_i$ is certified and $\mathsf{AcceptList}_i[j] = 2$, then secret s_j can be recovered from the secret shares s_{ji} received by each honest party P_i.*

Protocol details. At the start of the protocol (refer Figure 4), 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 $\mathcal{C}_{ik}\ \forall k \in \{1, \ldots n.\}$. 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 $P_j\ \forall j \in [n]$.

Each party P_i performs the following operations:

1. **(Round 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 $\langle \mathsf{propose}, \mathsf{VSS}.\vec{C}_i, z_{pi} \rangle_i$. Party 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. **(Round 2) Blame/Forward.** If 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 Equation (2)), then invoke $\mathsf{Deliver}(\mathsf{propose}, \mathsf{comm}_j, z_{pj}, -)$. If no valid secret shares has been received from party P_j, multicast $\langle \mathsf{blame}, j \rangle_i$ to all parties.
3. **(Round 3) Request open.** 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 blame are received for party P_j, do not send anything for dealer P_j until Round 6.
4. **(Round 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. **(Round 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. If no blames for party P_j has been received by round 3 and no party P_j equivocation has been detected, send $\langle \mathsf{vote}, H(\mathsf{comm}_j) \rangle_i$ to party P_j.
6. **(Round 6) Vote cert.** Upon receiving $t + 1$ distinct vote messages for comm_i (denoted by $\mathcal{C}(\mathsf{comm}_i)$), invoke weak gradecast (refer Figure 3) to propagate $\mathcal{C}(\mathsf{comm}_i)$.
7. **(Round 9) Propose Grade** Let $(o_{j,i}, g_{j,i})$ be the output of weak gradecast with party P_j as the sender. Set $\mathsf{AcceptList}_i[j] = g_{j,i}$. Multicast $\langle \mathsf{accept\text{-}list}, \mathsf{AcceptList}_i \rangle_i$.
8. **(Round 10) 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 $\mathsf{AcceptList}_i[h] \geq 1$ $\forall h \in [n]$.
9. **(At any round) Equivocation.** If equivocating hashes signed by party P_j are detected, multicast the equivocating hashes.

Fig. 4: **Recoverable Set of Shares**

If a party P_j receives valid secret share s_{ij}, s'_{ij} along with the proposal $\mathsf{comm}_i := \langle \mathsf{propose}, \mathsf{VSS}.\vec{C}_i, z_{pi} \rangle_i$ by the start of round 2, 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 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 if no party P_i has been detected by round 5. Additionally, if no blame messages are received for P_i by round 3, party P_j sends $\langle \mathsf{vote}, H(\mathsf{comm}_i) \rangle$ to party P_i at round 5.

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 $\mathcal{C}(\mathsf{comm}_i)$. 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 ≥ 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 outputs 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 [28, 29] 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 $t < n/4$ [28] and $t < n/3$ [29] resilience and involved multiple invocation of gradecast per secret sharing.

We present detailed security analysis in Section M.

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*. In the absence of an existing threshold (DKG) setup, the OLE protocol was designed via n^2 parallel invocations of weaker VSS primitives such as graded VSS [28] or moderated VSS [41] which trivially incurs $\Omega(n^4)$ communication. In a recent work, Abraham et al. [3] designed an OLE protocol tolerating $t < n/3$ Byzantine faults using Aggregatable PVSS [35] for the asynchronous model which incurs $O(\kappa n^3)$ communication. Their protocol invokes n^2 parallel invocation of Aggregatable PVSS. Since, Aggregatable PVSS [35] allows a linear number of secret sharings to be aggregated into a single transcript whose size is linear, their protocol incurs only $O(\kappa n^3)$ communication. However, Aggregatable PVSS requires additional cryptographic assumptions (i.e., SXDH assumption) which is not desirable. In this work, we build a communication efficient OLE protocol using only n parallel invocations of weaker VSS primitives and a non-interactive threshold signature scheme [15]. Note that our OLE protocol does not require a prior threshold (DKG) setup phase despite making use of threshold signatures. The security of our OLE protocol is based on the computational Diffie-Hellman (CDH) problem in the random oracle model. The resulting protocol incurs a communication complexity of $O(\kappa n^3)$ and constant rounds.

Construction. The starting point of our construction is the threshold coin-tossing scheme of Cachin et al. [15] which makes use of non-interactive threshold signature scheme. The threshold signature scheme requires a prior threshold setup which is essentially a DKG. The threshold setup establishes a tuple $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ of secret keys, a tuple $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ of verification keys. After the threshold setup phase, each party signs a common message (e.g., an epoch number) with its threshold secret key to obtain a threshold share. A combination of any $t+1$ valid threshold shares is then used to obtain a unique and random threshold signature σ. A random oracle $H'' : \mathbb{G} \rightarrow \{0,1\}$ is then used to generate an unbiased and unpredictable random bit from the threshold signature σ.

Note that the threshold signature scheme requires a prior threshold setup to establish a tuple $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ of secret keys, a tuple $(\mathsf{vk}_1, \ldots, \mathsf{vk}_n)$ of verification keys. We fulfill this requirement by using the output of recoverable set of shares protocol (from Section 6) to establish a local threshold setup corresponding to each party. In the recoverable set of shares protocol, each party P_i outputs an $\mathsf{AcceptList}_i$ along with $\mathcal{AC}(\mathsf{AcceptList}_i)$. An $\mathsf{AcceptList}_i$ (accompanied by $\mathcal{AC}(\mathsf{AcceptList}_i)$) consists of at least $n-t$ entries with grades of 2 and all honest parties must have received secret shares shared by parties in $\mathsf{AcceptList}_i$ whose grades are 2. Thus, each party P_j uses secret shares shared by parties in an $\mathsf{AcceptList}_i$ with grades of 2 to compute its secret key $\mathsf{sk}_{i,j}$ and verification key $\mathsf{vk}_{i,j} = g^{\mathsf{sk}_{i,j}}$ to establish local DKG setup $\mathsf{local\text{-}dkg}[i]$ corresponding to party P_i.

In order to setup $\mathsf{local\text{-}dkg}[i]$, party P_i first invokes weak gradecast to propagate its $\mathsf{AcceptList}_i$ (along with $\mathcal{AC}(\mathsf{AcceptList}_i)$). If party P_j outputs ($\mathsf{AcceptList}_i$,

Each party P_i performs the following operations:

1. Each party P_i invokes recoverable set of shares protocol (refer Figure 4). Each party P_i outputs ($\mathsf{AcceptList2}_i$ $\mathcal{AC}(\mathsf{AcceptList2}_i)$).
2. Each party P_i invokes weak gradecast to propagate ($\mathsf{AcceptList2}_i, \mathcal{AC}(\mathsf{AcceptList2}_i)$).
3. Let ($o_{j,i}$, $\mathsf{grade}_i[j]$) be the output with party P_j as dealer. Let $o_{j,i}$ contains $\mathsf{AcceptList2}_j$. If $\mathsf{grade}_i[j] \geq 1$, set $\mathsf{local\text{-}dkg}_i[j] = \mathsf{AcceptList2}_j$, $\mathsf{local\text{-}dkg\text{-}grade}_i[j] = \mathsf{grade}_i[j]$.
   - Set $\mathsf{sk}_{j,i} = \sum_{m \in \mathsf{AcceptList2}_j | \mathsf{AcceptList2}_j[m]=2} s_{mi}$, $\mathsf{vk}_{j,i} = g^{\mathsf{sk}_{j,i}}$, and $\mathsf{sk}'_{j,i} = \sum_{m \in \mathsf{AcceptList2}_j | \mathsf{AcceptList2}_j[m]=2} s'_{mi}$.
   - Compute $\mathcal{C}_{\langle g \rangle}(\mathsf{sk}_{j,i})$, $\mathcal{C}_{\langle g,h \rangle}(\mathsf{sk}_{j,i}, \mathsf{sk}'_{j,i}))$ and $\pi_{\equiv \mathsf{Com}\, j,i} = \mathsf{NIZKPK}_{\equiv \mathsf{Com}}(\mathsf{sk}_{j,i}, \mathsf{sk}'_{j,i}, g, h, \mathcal{C}_{\langle g \rangle}(\mathsf{sk}_{j,i}), \mathcal{C}_{\langle g,h \rangle}(\mathsf{sk}_{j,i}, \mathsf{sk}'_{j,i}))$. Multicast $(\mathsf{vk}_{j,i}, \pi_{\equiv \mathsf{Com}\, j,i})$ to all parties.

Fig. 5: Threshold setup protocol

$\mathcal{AC}(\mathsf{AcceptList}_i)$) with a grade of ≥ 1, it uses $\mathsf{AcceptList}_i$ to compute its secret key $\mathsf{sk}_{i,j}$ and verification key $\mathsf{vk}_{i,j} = g^{\mathsf{sk}_{i,j}}$ to establish local DKG setup $\mathsf{local\text{-}dkg}[i]$ corresponding to party P_i. Note that this establishes a separate threshold setup for each party P_i. With local DKG setup $\mathsf{local\text{-}dkg}[i]$ as the threshold setup for party P_i, parties then sign a common message to generate a unique and random threshold signature σ_i. Parties then use a random oracle $H' : \mathbb{G} \rightarrow \{0,1\}^\kappa$ to generate κ bit random coin value assigned to party P_i. Each party P_j uses the random coin value assigned to party P_i if it outputs ($\mathsf{AcceptList}_i$, $\mathcal{AC}(\mathsf{AcceptList}_i)$) with a grade of 2. From the set of parties for which party P_j outputs ($\mathsf{AcceptList}_k$, $\mathcal{AC}(\mathsf{AcceptList}_k)$) with a grade of 2, it selects the party with highest (or lowest) coin value as its leader. With probability at least $\frac{1}{2}$, all honest parties select a common honest leader using this approach.

Note that threshold coin-tossing scheme of Cachin et al. [15] produces a single bit output. However, it can also be used to generate κ bit strings using κ-bit hash function [14]. Looking ahead, the final DKG is also computed from one of the valid $\mathsf{AcceptList}$ output from the recoverable set of shares. Making use of the secret shares in an $\mathsf{AcceptList}$ output from the recoverable set of shares during this local DKG setup phase will leak the final public key before the final DKG is decided. Note that the final public key can be computed from $t+1$ verification keys. This allows the adversary ability to force the final DKG to have certain final public key. To circumvent this issue, we execute two separate instances of recoverable set of shares in parallel; one instance to setup local DKG instances and the other to setup the final DKG instance. To remove this ambiguity, we call the accept list output from the recoverable set of shares executed for local DKG as $\mathsf{AcceptList2}$ i.e. each party P_i outputs an $\mathsf{AcceptList2}_i$ along with $\mathcal{AC}(\mathsf{AcceptList2}_i)$.

Protocol details. The setup phase of the protocol is presented in Figure 5. Each party P_i invokes recoverable set of shares protocol and outputs $\mathsf{AcceptList2}_i$ (along with $\mathcal{AC}(\mathsf{AcceptList2}_i)$). Each party P_i then invokes weak gradecast to

propagate ($\mathsf{AcceptList2}_i$, $\mathcal{AC}(\mathsf{AcceptList2}_i)$). At the end of the setup phase, each party P_i sets up the local DKG instance for each party P_j (i.e., $\mathsf{local\text{-}dkg}_i[j]$) as $\mathsf{AcceptList2}_j$ if $\mathsf{local\text{-}dkg\text{-}grade}_i[j] \geq 1$. If $\mathsf{local\text{-}dkg\text{-}grade}_i[j] = 2$, due to weak gradecast properties, all honest parties have a common local DKG instance for party P_j (i.e., $\mathsf{local\text{-}dkg}[j]$). In addition, for an honest party P_j, all honest parties will have a common local DKG instance $\mathsf{local\text{-}dkg}[j]$. Each party P_i also computes required secret keys $\mathsf{sk}_{j,i}$, verification keys $\mathsf{vk}_{j,i}$ for local DKG instance $\mathsf{local\text{-}dkg}_i[j]$ computed from $\mathsf{local\text{-}dkg}_i[j]$ as shown in Figure 5.

Let sid be the input of party P_i.
Set $\mathcal{X}_i \leftarrow \emptyset$. Each party P_i performs following operations:

1. Perform $\sigma_{j,i} = \mathsf{Sign}_{\mathsf{TS}}(\mathsf{sk}_{j,i}, (j, \mathsf{sid}))$ and multicast $\sigma_{j,i}$ if $\mathsf{local\text{-}dkg\text{-}grade}_i[j] \geq 1$ $\forall j \in [n]$.
2. Upon receiving a set S of $t+1$ valid signature shares for party P_j, compute $\sigma_j = \mathsf{Combine}_{\mathsf{TS}}(\mathsf{pk}, \mathsf{sid}, S)$ and $\mathcal{X}_i[j] \leftarrow H'(\sigma_j)$.
3. Perform $\ell \leftarrow argmax_h\{\mathcal{X}_i[h] | \mathsf{local\text{-}dkg\text{-}grade}_i[h] = 2\}$. Output P_ℓ.

Fig. 6: Oblivious Leader Election

The OLE protocol is presented in Figure 6. The input to the protocol is a sequence id sid. Once the local DKG instances are setup, each party P_i uses its secret key $\mathsf{sk}_{j,i}$ to sign a common message i.e., (j, sid) (for party P_j) if $\mathsf{local\text{-}dkg\text{-}grade}_i[j] \geq 1$ to obtain a threshold share $\sigma_{j,i}$. A set of $t+1$ valid signature shares corresponding to $\mathsf{local\text{-}dkg}[j]$ is combined to form a single threshold signature σ_j and a hash $H'(\sigma_j)$ generates κ bit coin value for party P_j. We note that two or more parties could output the same grade list (i.e, $\mathsf{AcceptList2}$) in the recoverable set of shares protocol; hence their local DKG might be same. However, parties sign a distinct message e.g. (j, sid) for party P_j. Such generated threshold signatures are unique and random regardless of their local DKG instance being common; hence the coin value assigned to each party is also random. Honest parties consider coin values for party P_j only if $\mathsf{local\text{-}dkg\text{-}grade}_i[j] = 2$. Note that if $\mathsf{local\text{-}dkg\text{-}grade}_i[j] = 2$, a threshold signature σ_j will exist for party P_j. This is because all honest parties will have $\mathsf{local\text{-}dkg\text{-}grade}[j] \geq 1$ and a common $\mathsf{local\text{-}dkg}[j]$ due to weak gradecast properties and each honest party P_i will send their signature share $\sigma_{j,i}$. A coin value is then computed as $H'(\sigma_j)$. The party P_ℓ with highest coin value is elected as leader.

Round complexity and communication complexity. The threshold setup phase has a latency of 15 rounds to invoke recoverable set of shares, n parallel instances of weak-gradecast and distribute verification keys. The OLE protocol requires only 1 round to generate threshold signatures. The threshold setup phase invokes recoverable set shares, n parallel weak-gradecasts with an input of size $O(\kappa n)$ and sharing verification keys. This incurs $O(\kappa n^3)$ communication. The threshold signature generation incurs $O(\kappa n^3)$ communication.

We present detailed security analysis in Section N.

8 Multi-Valued Validated Byzantine Agreement

In Section 6, we presented a recoverable set of shares protocol where each honest party P_i outputs a (possibly different) set $\mathsf{AcceptList}_i$ along with $\mathcal{AC}(\mathsf{AcceptList}_i)$– 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 ack 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. [14] to allow honest parties to agree on any externally valid value. Recent works [4, 44] have proposed MVBA protocols for the asynchronous communication model tolerating $t < n/3$ Byzantine faults. 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 $O(n^2\ell + \kappa n^3)$ communication for inputs of size ℓ bits and expected constant rounds.

We extend the Binary Byzantine agreement (BBA) protocol of Katz and Koo [41] to MVBA for large ($\ell = \Theta(n)$) input. The BBA protocol of Katz and Koo [41] tolerates $t < n/2$ Byzantine faults and terminates in expected 4 epochs. Their 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 for a single bit input; thus, their protocol trivially incurs $O(\kappa n^4)$ communication. 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 [40] 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 [23], BA protocols use a common source of randomness called *common coin* to achieve agreement in constant expected rounds. The common coin is *weak* if all honest parties obtain a common honest leader with some constant probability (and with the remaining probability either the common leader is Byzantine or honest parties may disagree on the leader). 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 and blows up the communication complexity. In this work, we replace their weak common coin protocol with our communication efficient leader election protocol from Section 7 which outputs a common honest leader with probability at least $\frac{1}{2}$. Our OLE protocol incurs $O(\kappa n^3)$ communication and a single round after an initial setup phase (refer Figure 5) which incurs 15 rounds.

Our MVBA protocol in presented in Figure 7. The underlying consensus mechanism is identical to the BBA protocol of Katz and Koo [41]. In round 1, each party P_i invokes weak gradecast protocol to propagate its input v_i. Our

Let v_i be party P_i's input and e be the current epoch. Each party P_i sets $\mathsf{lock}_i \leftarrow \perp$. Each party P_i performs following operations.

1. **(Round 1) Propose.** Each party P_i invokes weak gradecast to propagate v_i.
2. **(Round 4) Update.** Let $(v_{j,i}, \mathsf{grade}_i[j])$ be the output with party P_j as the dealer. Let $\mathcal{S}_i^v := \{j : v_{j,i} = v \wedge \mathsf{grade}_i[j] = 2\}$ and $\tilde{\mathcal{S}}_i^v := \{j : v_{j,i} = v \wedge \mathsf{grade}_i[j] \geq 1\}$. If $\mathsf{lock}_i = \perp$, then:
   (a) If $|\tilde{\mathcal{S}}_i^v| > t$, update $v_i \leftarrow v$.
   (b) If $|\mathcal{S}_i^v| > t$, set $\mathsf{lock}_i \leftarrow 1$.
   Invoke weak gradecast (refer Figure 3) to propagate v_i.
3. **(Round 7) Update2.** Again, let $(v_{j,i}, \mathsf{grade}_i[j])$ be the output with party P_j as the dealer. Define $\mathcal{S}_i^v$ and $\tilde{\mathcal{S}}_i^v$ as above. If $\mathsf{lock}_i = \perp$ and $|\tilde{\mathcal{S}}_i^v| > t$, set $v_i \leftarrow v$. Multicast v_i.
4. **(Round 8) Leader election.** Invoke OLE protocol with input e.
5. **(Round 9) Terminate/Advance Epoch.** Let P_ℓ be the output of leader election protocol.
   (a) If $\mathsf{lock}_i = 0$, output v_i and terminate.
   (b) If $\mathsf{lock}_i = 1$, set $\mathsf{lock}_i = 0$. If $\mathsf{lock}_i = \perp$ and $|\mathcal{S}_i^v| \leq t$, $v_{\ell,i} \neq \perp$ and $\mathsf{ex\text{-}validation}(v_{\ell,i}) = \mathsf{true}$, update $v_i \leftarrow v_{\ell,i}$. Advance to epoch $e + 1$.
6. **(At any round) Equivocation.** If equivocating hashes signed by party P_j are detected, multicast the equivocating hashes.

Fig. 7: **MVBA** with $O(n^2\ell + \kappa n^3)$ communication and expected 4 epochs.

weak gradecast protocol incurs 4 rounds. Rounds 2 and 3 accommodates the steps of the weak gradecast protocol. Again in round 4, each party P_i invokes weak gradecast protocol to propagate its updated input v_i. Rounds 5 and 6 accommodates the steps of the weak gradecast protocol. In round 8, parties invoke the OLE protocol to elect a leader.

Round complexity. By Theorem 10, a common honest leader is selected with probability at least $\frac{1}{2}$ and all honest parties terminate in the next 2 epochs. Thus, the expected number of epochs required is 4 epochs.

We present detailed security analysis in Section O.

9 Distributed Key Generation

Finally, we present two communication efficient DKG protocols with $O(\kappa n^3)$ communication. The first protocol incurs expected $O(\kappa n^3)$ communication and terminates in expected constant rounds while the second protocol incurs $O(\kappa n^3)$ communication in the worst case 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 weaker consensus primitives and use 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

1. **Deal/Setup.** Each party P_i invokes recoverable set of shares protocol (refer Figure 4). Each party P_i outputs a set $\mathsf{AcceptList}_i$ with an $\mathsf{ack\text{-}cert}$ for $\mathsf{AcceptList}_i$ (i.e., $\mathcal{AC}(\mathsf{AcceptList}_i)$). Each party P_i also invokes threshold setup phase (refer Figure 5) in parallel.
2. **MVBA.** Each party P_i invokes MVBA (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.

Fig. 8: DKG with expected $O(\kappa n^3)$ communication and expected $O(1)$ rounds

secret keys from a common set of at least $n-t$ parties 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 DKG with $O(\kappa n^3)$ communication and expected $O(1)$ rounds

The DKG protocol uses recoverable set of shares protocol (refer Figure 4) to perform secret sharing. The threshold setup protocol (refer Figure 5) is also executed at the start of the execution. 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 $\mathsf{ex\text{-}validation}(\mathsf{AcceptList}_i) = \mathsf{true}$. 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 1.

Latency and communication complexity. The recoverable set of shares protocol has a round complexity of 10 rounds and $O((\kappa+w)n^3)$ communication. The threshold setup protocol incurs a communication of $O((\kappa+w)n^3)$ and 15 rounds; but is executed in parallel and completes before the OLE protocol is invoked in the MVBA protocol. Thus, it does not increase overall round complexity of the protocol. The MVBA protocol incurs expected 4 epochs (with each epoch being 9 rounds) 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 a single round. Thus, the protocol incurs $O((\kappa+w)n^3)$ communication and expected 47 rounds.

9.2 DKG with worst-case $O(\kappa n^3)$ communication and $O(t)$ rounds

While the above protocol terminates in expected 4 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 DKG protocol with guaranteed termination in $t+1$ epochs with $O(\kappa n^3)$ communication in the worst case. The 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 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 [10] to agree on a common set. We present a brief overview of the BFT SMR.

1. **Deal.** Each party P_i invokes recoverable set of shares protocol (refer Figure 4). 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 [10] 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.

Fig. 9: DKG with worst-case $O(\kappa n^3)$ communication and $t+1$ epochs

BFT SMR of RandPiper [10]. The BFT SMR protocol of RandPiper [10] 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 7 rounds. 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 [10] for more details.

In this 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 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 10 rounds 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 7 rounds. The reconstruction phase requires $O(\kappa n^2)$ communication and a single round . Thus, the protocol incurs $O(\kappa n^3)$ communication in the worst-case and $11 + 7 * (t + 1)$ rounds.

References

1. Abraham, I., Asharov, G., Patil, S., Patra, A.: Asymptotically free broadcast in constant expected time via packed vss. In: TCC'22. pp. 384–414. Springer (2022)
2. Abraham, I., Devadas, S., Dolev, D., Nayak, K., Ren, L.: Synchronous byzantine agreement with expected $O(1)$ rounds, expected $O(n^2)$ communication, and optimal resilience. Financial Cryptography and Data Security (FC) (2019)
3. Abraham, I., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Reaching consensus for asynchronous distributed key generation. In: PODC'21. pp. 363–373 (2021)
4. Abraham, I., Malkhi, D., Spiegelman, A.: Asymptotically optimal validated asynchronous byzantine agreement. In: PODC'19. pp. 337–346 (2019)
5. Bacho, R., Loss, J.: On the adaptive security of the threshold bls signature scheme. In: CCS'2022. pp. 193–207 (2022)
6. Backes, M., Kate, A., Patra, A.: Computational verifiable secret sharing revisited. In: Advances in Cryptology—ASIACRYPT. pp. 590–609. Springer (2011)
7. Backes, M., Kate, A., Patra, A.: Computational verifiable secret sharing revisited. In: ASIACRYPT'11. pp. 590–609. Springer (2011)
8. Barić, N., Pfitzmann, B.: Collision-free accumulators and fail-stop signature schemes without trees. In: EUROCRYPT'97. pp. 480–494 (1997)
9. Ben-Or, M., El-Yaniv, R.: Resilient-optimal interactive consistency in constant time. Distributed Computing **16**(4), 249–262 (2003)
10. Bhat, A., Shrestha, N., Luo, Z., Kate, A., Nayak, K.: Randpiper–reconfiguration-friendly random beacons with quadratic communication. In: CCS'21. pp. 3502–3524 (2021)
11. Boldyreva, A.: Threshold signatures, multisignatures and blind signatures based on the gap-diffie-hellman-group signature scheme. In: International Workshop on Public Key Cryptography. pp. 31–46. Springer (2003)
12. Boneh, D., Boyen, X.: Short signatures without random oracles and the sdh assumption in bilinear groups. Journal of cryptology **21**(2), 149–177 (2008)

13. Buchman, E.: Tendermint: Byzantine fault tolerance in the age of blockchains. Ph.D. thesis, thesis (2016)
14. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: CRYPTO'01. pp. 524–541. Springer (2001)
15. Cachin, C., Kursawe, K., Shoup, V.: Random oracles in constantinople: Practical asynchronous byzantine agreement using cryptography. Journal of Cryptology **18**(3), 219–246 (2005)
16. Canetti, R., Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Adaptive security for threshold cryptosystems. In: CRYPTO'99. pp. 98–116 (1999)
17. Cascudo, I., David, B.: Scrape: Scalable randomness attested by public entities. In: ACNS. pp. 537–556. Springer (2017)
18. Cascudo, I., David, B., Shlomovits, O., Varlakov, D.: Mt. random: multi-tiered randomness beacons. In: ANCS'23. pp. 645–674. Springer (2023)
19. Das, S., Xiang, Z., Kokoris-Kogias, L., Ren, L.: Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling. In: USENIX Security Symposium (2023)
20. Das, S., Yurek, T., Xiang, Z., Miller, A., Kokoris-Kogias, L., Ren, L.: Practical asynchronous distributed key generation. In: S&P'22. pp. 2518–2534. IEEE (2022)
21. Desmedt, Y., Frankel, Y.: Threshold cryptosystems. In: Advances in Cryptology, 9th Annual International Cryptology Conference. vol. 435, pp. 307–315 (1989)
22. Dolev, D., Reischuk, R.: Bounds on information exchange for byzantine agreement. Journal of the ACM (JACM) **32**(1), 191–204 (1985)
23. Dolev, D., Strong, H.R.: Authenticated algorithms for byzantine agreement. SIAM Journal on Computing **12**(4), 656–666 (1983)
24. Drand: Drand - a distributed randomness beacon daemon, `https://github.com/drand/drand`
25. D'Arco, P., Stinson, D.R.: On unconditionally secure robust distributed key distribution centers. In: ASIACRYPT'02. pp. 346–363 (2002)
26. Erwig, A., Faust, S., Riahi, S.: Large-scale non-interactive threshold cryptosystems through anonymity. IACR Cryptology ePrint Archive **2021**, 1290 (2021)
27. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: FOCS'87. pp. 427–438. IEEE (1987)
28. Feldman, P., Micali, S.: Optimal algorithms for byzantine agreement. In: STOC'88. pp. 148–161 (1988)
29. Feldman, P., Micali, S.: An optimal probabilistic protocol for synchronous byzantine agreement. SIAM Journal on Computing **26**(4), 873–933 (1997)
30. Fitzi, M., Hirt, M.: Optimally efficient multi-valued byzantine agreement. In: PODC'06. pp. 163–168 (2006)
31. Fitzi, M., Liu-Zhang, C.D., Loss, J.: A new way to achieve round-efficient byzantine agreement. In: PODC'21. pp. 355–362 (2021)
32. Garay, J.A., Katz, J., Koo, C.Y., Ostrovsky, R.: Round complexity of authenticated broadcast with a dishonest majority. In: FOCS'07. pp. 658–668. IEEE (2007)
33. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. Journal of Cryptology pp. 51–83 (2007)
34. Groth, J.: Non-interactive distributed key generation and key resharing. IACR Cryptol. ePrint Arch. **2021**, 339 (2021)
35. Gurkan, K., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Aggregatable distributed key generation. In: EUROCRYPT'21. pp. 147–176 (2021)
36. Hirt, M., Nielsen, J.B., Przydatek, B.: Cryptographic asynchronous multi-party computation with optimal resilience. In: EUROCRYPT'05. pp. 322–340 (2005)

37. Hofheinz, D., Müller-Quade, J.: A synchronous model for multi-party computation and the incompleteness of oblivious transfer. FCS'04 **4**, 117–130 (2004)
38. Kate, A., Goldberg, I.: Distributed key generation for the internet. In: 29th IEEE International Conference on Distributed Computing Systems. pp. 119–128 (2009)
39. Kate, A., Huang, Y., Goldberg, I.: Distributed key generation in the wild. IACR Cryptol. ePrint Arch. **2012**, 377 (2012)
40. Kate, A., Zaverucha, G.M., Goldberg, I.: Constant-size commitments to polynomials and their applications. In: ASIACRYPT'10. pp. 177–194. Springer (2010)
41. Katz, J., Koo, C.Y.: On expected constant-round protocols for byzantine agreement. In: Annual International Cryptology Conference. pp. 445–462 (2006)
42. Kokoris Kogias, E., Malkhi, D., Spiegelman, A.: Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures. In: CCS '20. pp. 1751–1767. New York, NY, USA (2020)
43. Lab, T.: Torus: Globally accessible public key infrastructure for everyone. `https://tor.us/` (2021)
44. Lu, Y., Lu, Z., Tang, Q., Wang, G.: Dumbo-mvba: Optimal multi-valued validated asynchronous byzantine agreement, revisited. In: PODC'20. pp. 129–138 (2020)
45. Merkle, R.C.: A digital signature based on a conventional encryption function. In: EUROCRYPT'87. pp. 369–378. Springer (1987)
46. Micali, S.: Byzantine agreement, made trivial (2016)
47. Momose, A., Ren, L.: Optimal communication complexity of byzantine consensus under honest majority. arXiv preprint arXiv:2007.13175 (2020)
48. Nayak, K., Ren, L., Shi, E., Vaidya, N.H., Xiang, Z.: Improved extension protocols for byzantine broadcast and agreement. In: DISC'20 (2020)
49. Neji, W., Blibech, K., Ben Rajeb, N.: Distributed key generation protocol with a new complaint management strategy. Security and communication networks **9**(17), 4585–4595 (2016)
50. Nguyen, L.: Accumulators from bilinear pairings and applications. In: Cryptographers' track at the RSA conference. pp. 275–292 (2005)
51. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140 (1991)
52. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: EUROCRYPT'91. p. 522–526 (1991)
53. Reed, I.S., Solomon, G.: Polynomial codes over certain finite fields. Journal of the society for industrial and applied mathematics **8**(2), 300–304 (1960)
54. Schindler, P., Judmayer, A., Stifter, N., Weippl, E.R.: Ethdkg: Distributed key generation with ethereum smart contracts. IACR Cryptol. ePrint Arch. p. 985 (2019)
55. Shoup, V.: Practical threshold signatures. In: EUROCRYPT 2000. vol. 1807, pp. 207–220. Springer (2000)
56. Shrestha, N., Abraham, I., Ren, L., Nayak, K.: On the Optimality of Optimistic Responsiveness. In: CCS'20. pp. 839–857 (2020)
57. Tomescu, A., Chen, R., Zheng, Y., Abraham, I., Pinkas, B., Gueta, G.G., Devadas, S.: Towards scalable threshold cryptosystems. In: S&P'20. pp. 877–893. IEEE (2020)
58. Tsimos, G., Loss, J., Papamanthou, C.: Gossiping for communication-efficient broadcast. In: CRYPTO'2022. pp. 439–469. Springer (2022)
59. Yin, M., Malkhi, D., Reiter, M.K., Gueta, G.G., Abraham, I.: Hotstuff: Bft consensus with linearity and responsiveness. In: PODC'19. pp. 347–356 (2019)

Supplementary Material

J Extended Preliminaries

J.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 $t \geq 2c + d$.

J.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 $d_i \notin \mathcal{D}$, 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.

In this work, we use bilinear accumulator from Nyugen [50] which satisfies the following property:

Lemma 1 (Collision-free accumulator [50]). *The bilinear accumulator is collision-free. That is, for any set of size n and a probabilistic polynomial-time adversary $\mathcal{A}$, the following function is negligible in κ:*

$$\mathsf{P}r\left[\begin{array}{c} ak \leftarrow \mathsf{Gen}(1^\kappa, n), \\ (\{d_1, \ldots, d_n\}, d', w') \leftarrow \\ \mathcal{A}(1^\kappa, n, ak), \\ z \leftarrow \mathsf{Eval}(ak, \{d_1, \ldots, d_n\}) \end{array} \middle| \begin{array}{c} \\ (d' \notin \{d_1, ..., d_n\}) \wedge \\ (\mathsf{Verify}(ak, z, w', d') = 1) \\ \\ \end{array} \right]$$

J.3 Non-interactive threshold signature scheme

The threshold signature scheme of Cachin et al. [15] consists of following interfaces:

- The randomized *key generation* algorithm $\mathsf{KeyGen_{TS}}$ that takes a security parameter κ as input and outputs a tuple $(\mathsf{sk}_1, \ldots, \mathsf{sk}_n)$ of secret keys, a tuple $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$ and a common public key pk.
- The deterministic signing algorithm $\mathsf{Sign_{TS}}$ that takes as input sk_i and a message m and outputs a signature σ_i on m.
- The deterministic *share verification* algorithm $\mathsf{ShareVerify_{TS}}$ that takes as input public key pk_i, a signature share σ_i and tuple (i, m). It outputs a bit $b \in \{0, 1\}$ indicating whether σ_i is a valid signature share on m under secret key sk_i.
- The deterministic *combining* $\mathsf{Combine_{TS}}$ takes as input a tuple of public keys $(\mathsf{pk}_1, \ldots, \mathsf{pk}_n)$, a message m, and a list of $t+1$ pairs (i, σ_i). It outputs either a signature σ on m or $\perp$, if (i, σ_i) contains ill-formed signature shares.
- The deterministic *verification* algorithm $\mathsf{Verify_{TS}}$ takes as input a signature σ, a message m and a common public key pk. It outputs a bit $b \in \{0, 1\}$ indicating whether σ is a valid signature on m.

The threshold signature scheme satisfies robustness (i.e., it is computationally infeasible for an adversary to produce $t+1$ valid signature shares such that the output of the share combining algorithm is not a valid signature) and unforgeability (i.e., it is computationally infeasible for the adversary to output a valid signature on a message m given t signature shares on m).

J.4 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 $\mathsf{H}_{\equiv\mathsf{Com}} : \mathbb{G}^6 \rightarrow \mathbb{Z}_p$ 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 $c = \mathsf{H}_{\equiv\mathsf{Com}}(g, h, \mathcal{C}_{\langle g \rangle}(s), \mathcal{C}_{\langle g,h \rangle}(s, r), t_1', t_2')$.

K Analysis of Secure DKG

We rely on the following Lemma of [51].

Lemma 2 ([51]). *Under the discrete-log assumption, Pedersen's VSS satisfies the 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 interpolates 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 2 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 6 *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 messages 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 $f \leq t$ 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.

Theorem 7. *Under discrete-log assumption and random oracle, the protocol in Figure 1 is a secure protocol for distributed key generation in dlog-based cryptosystem tolerating $t < n/2$ Byzantine faults.*

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 Theorem 6, all honest parties have received secret shares shared by party P_j. This implies party P_j is not disqualified

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 $t' \leq t$. Let $Y \in \mathbb{G}$ be the input public key and $\mathsf{H}_{\equiv\mathsf{Com}} : \mathbb{G}^6 \rightarrow \mathbb{Z}_p$ 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 1) 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 $P_i \in \texttt{QUAL} \cap \mathcal{G}, P_j \in \mathcal{B}$, and the commitments $\mathcal{C}_i$ for $P_i \in \texttt{QUAL}$.
   - $\mathcal{S}$ knows all $f_i(y)$ and $f'_i(y)$ for $P_i \in \texttt{QUAL}$ 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 1).
   (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, $x = \sum_{P_i \in \texttt{QUAL}} s_i$ such that $Y = g^x$.

Fig. 10: Simulator for Secure DKG

during the sharing phase. By part (i) of Lemma 2, 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 $P_j \in \texttt{QUAL}$. Since, the secret key x is sum of individual secret s_j contributed by $P_j \in \texttt{QUAL}$ 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 2, 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 $\texttt{QUAL}$ 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 [33,39]. 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 $\texttt{QUAL}$ is well-defined and $\mathcal{S}$ knows all $f_i(y)$ and $f'_i(y)$ for $P_i \in \texttt{QUAL}$ 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 the 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 $P_i \in \texttt{QUAL}$

$\mathcal{S}$ 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 $g^{f^*_n(0)} = g^{s^*_n} = \frac{Y}{\prod_{P_j \in \texttt{QUAL}\setminus\{P_n\}} g^{s_i}}$ 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 $f_n^{'*}(y)$.

L Analysis of Gradecast

Lemma 3. *Suppose party P_j is the designated sender. If an honest party invokes* Deliver *in round r for a value m sent by party P_j and no honest party has detected a party P_j equivocation by round $r+1$, then all honest parties will receive value m by round $r+2$.*

Proof. Suppose an honest party P_i invokes Deliver at round r for a value m 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 value m to every party $P_k \; \forall k \in [n]$ at round r. The code words and witness arrive at all honest parties by round $r+1$.

Since no honest party has detected a party P_j equivocation by round $r+1$, 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 value m or received the code word from some other party. In any case, all honest parties will forward their code word corresponding to value m by round $r+1$. Thus, all honest parties will have received $t+1$ valid code words for a common accumulation value z_e by round $r+2$ sufficient to decode value m.

Theorem 8. *The protocol in Figure 3 is a gradecast protocol satisfying Definition 2.*

Proof. Suppose party P_j is the designated sender with its input value v.

We first consider the case when an honest party P_i outputs value v with a grade $g_i = 2$. Honest party P_i must have invoked Deliver for value v by round 2 and did not detect a party P_j equivocation by round 4. This implies no honest party detected a party P_j equivocation by round 3. By Lemma 3, all honest parties receive value v by round 4. In addition, since party P_i invoked Deliver for value v by round 2, all honest parties receive a code word for value v by round 3. Thus, value v is the first value received by all honest parties. Since $v \neq \perp$, all honest parties will output value v with a grade ≥ 1.

Next, we consider the case when the designated sender is honest. Since, the sender is honest, it sends its input value v to all honest parties such that all honest parties receive value v in round 2. Thus, all honest parties invoke Deliver to propagate value v in round 2. Moreover, the honest sender does not equivocate. Thus, all honest parties output value v with a grade of 2 in round 4.

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

Lemma 4 (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 3 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 $O(n\ell + (\kappa + w)n^2)$.

M Analysis of Recoverable Set of Shares

Lemma 5. *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 round 5. 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 shares $s_{k,j}$, $s'_{k,j}$ and commitment comm_k and invoked $\mathsf{Deliver}(\mathsf{propose}, \mathsf{comm}_k, z_{pk}, -)$ at round 2. Moreover, party P_i did not detect party P_k equivocation by round 5. This implies no honest party detected party P_k equivocation by round 3. By Lemma 3, all honest parties receive the commitment comm_k by round 4. This proves part (i) of the Lemma.

For part (ii), party P_i can send vote message on two occasions: (a) when it does not detect a $\langle \mathsf{blame}, k \rangle$ by round 3 and party k equivocation by round 5, 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 round 5.

In case (a), party P_i did not detect a party k equivocation by round 5 and $\langle \mathsf{blame}, k \rangle$ by round 3. Observe that all honest parties must have received valid secret shares corresponding to the commitment comm_k; otherwise party P_i must have received $\langle \mathsf{blame}, k \rangle$ by round 3 (since honest parties send $\langle \mathsf{blame}, k \rangle$ if no valid secret shares are received at round 2). 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 round 5. 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; otherwise, party P_i would have received more than f $\langle \mathsf{blame}, k \rangle$ message by round 3. 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 6. *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 Lemma 5 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 7 (Liveness). *Each honest party* P_i *will receive an* ack-cert *for its grade list* $\mathsf{AcceptList}_i$.

Proof. Consider an honest party P_i. Party P_i will send valid commitment $\mathsf{VSS}.\vec{C}_i$ and secret shares s_{ij}, s'_{ij} to party $P_j\ \forall j \in [n]$ in round 1. All honest parties will receive their valid secret shares s_{ij}, s'_{ij} and commitment comm_i in round 2. Thus, no honest party will send $\langle \mathsf{blame}, i \rangle$ for party P_i.

Observe that up to t Byzantine parties can always send $\langle \mathsf{blame}, i \rangle$. Honest parties wait until round 3 to collect blame messages for any party. Honest parties forward $\langle \mathsf{blame}, i \rangle$ to party P_i which party P_i receives by round 4. 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 round 5. Thus, party P_j will send vote for party P_i which party P_i receives by round 6. This implies party P_i collects $t + 1$ distinct vote messages by round 6.

Party P_i invokes weak gradecast to propagate $\mathcal{C}(comm_i)$ which completes by round 9. Due to the properties of weak gradecast, 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 an honest party sets $\mathsf{AcceptList}_i[l] = 2$ for a Byzantine party P_l and receive $\mathcal{C}(\mathsf{comm}_l)$. Due to the properties of weak gradecast, all honest parties receive $\mathcal{C}(\mathsf{comm}_l)$ 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.

Party P_i multicasts its $\mathsf{AcceptList}_i$ in round 9. 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 $\mathsf{AcceptList}_i$ proposed by party P_i and party P_i will receive ack-cert for $\mathsf{AcceptList}_i$ the end of round 10.

Theorem 9. *The protocol in Figure 4 is a recoverable set of shares protocol satisfying Definition 5.*

Proof. Straight forward from Lemma 5, Lemma 6 and Lemma 7

Lemma 8 (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 Deliver for the first comm_j from party P_j for $j \in [n]$. 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 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 vote-cert to all other parties which incurs $O(\kappa n^3)$ in communication. Invoking 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.

N Analysis of OLE protocol

Our coin generation protocol is similar to the threshold coin-tossing scheme of [15]. In Cachin et al. [15], the coin value is a single bit computed from the threshold signature using $H'' : \mathbb{G} \to \{0, 1\}$. In our scheme, the coin value is a κ bit string computed from the threshold signature using a κ bit hash function $H' : \mathbb{G} \to \{0, 1\}^\kappa$ which is also secure [14]. We rely on the following Lemma of [15].

Lemma 9 ([15]). *In the random oracle model, the coin-tossing scheme of Cachin et al. [15] is secure i.e., satisfies robustness and unpredictability under CDH assumption.*

Theorem 10. *In the random oracle model and under CDH assumption, the protocol in Figure 6 is an oblivious leader election protocol with fairness at least $\frac{1}{2}$.*

Proof. We first show termination i.e., honest party P_i will obtain a threshold signature σ_j (and coin value for party P_j) if $\mathsf{local\text{-}dkg\text{-}grade}_i[j] = 2$. This is because all honest parties will have $\mathsf{local\text{-}dkg\text{-}grade}[j] \geq 1$ and a common $\mathsf{local\text{-}dkg}[j]$ due to weak gradecast properties. Thus, each honest party P_k will send their signature share $\sigma_{j,k}$ i.e., a set of $t+1$ valid signature shares will be available sufficient to obtain threshold signature σ_j (and coin value $H'(\sigma_j)$).

By Lemma 9 the threshold signature generation protocol satisfies robustness and unpredictability. Thus, the coin value generated from threshold signature is robust and unpredictable.

Observe that each party P_i signs a distinct message (i.e, (j, sid)) for each part P_j. Thus, the threshold signature σ_j for each party P_j is unique and random even if two or more parties have the same local DKG instance; hence each party P_j will be assigned random coin value ($H'(\sigma_j)$). Since, the coin value assigned to a party is random, the coin value assigned to an honest party will be a global maximum with probability at least $\frac{n-t}{n}$. The probability that coin values of any two parties can be maximum is bounded by $\frac{1}{2^\kappa}$. Thus, all honest parties select the coin value corresponding to a common honest leader with probability $\frac{n-t}{n} - \frac{1}{2^\kappa} \geq \frac{1}{2}$ when $\kappa = 2 \log n$.

O Analysis of MVBA

Lemma 10. *If an honest party sets* lock *to* 1 *with a value* v *in epoch* e*, then all honest parties adopt value* v *in epoch* e*.*

Proof. Suppose an honest party P_i sets lock_i to 1 in epoch e. Party P_i must have received value v from a set Q of at least $t+1$ parties such that $|\mathcal{S}_i^v| > t$. By the properties of weak gradecast, all other honest parties receive value v corresponding to parties in Q with a grade ≥ 1 (i.e., all other honest parties have $\mathsf{grade}[j] \geq 1 \ \forall j \in Q$) and $|\tilde{\mathcal{S}}^v| > t$ 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 invoke weak-gradecast to propagate value v at the end of the Update step. Since, honest parties do not equivocate and send value v in a timely manner, all honest parties output value v such that $\mathsf{grade}[j]$ to 2. Thus, $|\tilde{\mathcal{S}}_i^v| > t$ and $|\mathcal{S}_i^v| > t$ in the Update2 step. Since, $|\mathcal{S}_i^v| > t$, no honest party will adopt value v_ℓ selected from the proposal election protocol. Thus, all honest parties adopt value v in epoch e.

Lemma 11. *If all honest parties start an epoch* e *with same input* v*, then all honest parties decide value* v *and terminate by the end of epoch* $e+1$*.*

Proof. Suppose all honest parties start an epoch e with the same input v. All honest parties invoke weak-gradecast with value v in the Propose step. By the properties of weak gradecast, for an honest dealer, all honest parties output a grade of 2. 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 $|\tilde{\mathcal{S}}_i^v| > t$ If $\mathsf{lock} = \perp$, honest parties set lock to 1.

Similarly, all honest parties invoke weak-gradecast with value v in the Update2 step. By similar argument, 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 at the of Update 2 step. Moreover, no honest party will adopt the value output from the proposal election protocol.

Honest parties with $\mathsf{lock} = 0$, output v and terminate in epoch e. All the remaining honest parties with $\mathsf{lock} = 1$, set $\mathsf{lock} = 0$ and advances to epoch $e + 1$. In the next epoch, all the remaining honest parties have $\mathsf{lock} = 1$ and will not update its value and stick to value v. At the end of epoch $e + 1$, they set their lock $\mathsf{lock} = 0$, output value v and terminate. Thus, all honest parties output v and terminate by the end of epoch $e + 1$.

Theorem 11. *The protocol in Figure 7 solves MVBA.*

Proof. We first consider external 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 $|\mathcal{S}_i^v| > t$. 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. $\mathsf{lock}_i = 1$ for at least one honest party P_i with a value v in epoch e. By Lemma 10, all honest party adopt value v in epoch e and enter epoch $e + 1$ with same value v. By Lemma 11, all honest parties output value v and terminate by epoch $e + 2$.

Case II. $\mathsf{lock}_i = \perp$ 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 11, all honest parties output value v_ℓ and terminate in epoch $e + 2$.

If $|\mathcal{S}_i^v| > t$ for at least one honest party P_i in the Update2 step, by the properties of weak-gradecast, $|\tilde{\mathcal{S}}^v| > t$ 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_ℓ 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 11, all honest parties output value v and terminate by epoch $e + 2$.

Lemma 12 (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 invokes weak gradecast with $O(\ell)$-sized proposal. By Lemma 4, this step incurs $O(n^2\ell + (\kappa + w)n^3)$. Similarly, in the Update2 step, each party invokes weak gradecast with $O(\ell)$-sized proposal. By Lemma 4, this step also incurs $O(n^2\ell + (\kappa + w)n^3)$. The proposal election protocol has a communication complexity of $O(\kappa n^3)$. Thus, the total communication complexity of the protocol is $O(n^2\ell + (\kappa + w)n^3)$ bits per epoch.

P 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 [22].

Lemma 13. *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 ≥ 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 if an honest party outputs a value v with a grade of 2, then all honest parties need to output a value v with a grade ≥ 1. A contradiction.

Theorem 12. *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 [30]). From Lemma 13, 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.