GRandLine: Adaptively Secure DKG and Randomness Beacon with (Almost) Quadratic Communication Complexity

Renas Bacho
CISPA Helmholtz Center for Information Security, Saarland University
Saarbrücken, Germany
renas.bacho@cispa.de

Christoph Lenzen
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
lenzen@cispa.de

Julian Loss
CISPA Helmholtz Center for Information Security
Saarbrücken, Germany
loss@cispa.de

Simon Ochsenreither
Saarland University
Saarbrücken, Germany
s.ochsenreither@gmail.com

Dimitrios Papachristoudis
Saarbrücken, Germany
jim.papachristoudis@gmail.com

***Abstract*—A *randomness beacon* is a source of continuous and publicly verifiable randomness which is of crucial importance for many applications. Existing works on distributed randomness beacons suffer from at least one of the following drawbacks: (i) security only against a static/non-adaptive adversary, (ii) each epoch takes many rounds of communication, or (iii) computationally expensive tools such as Proof-of-Work (PoW) or Verifiable Delay Functions (VDF). In this paper, we introduce GRandLine, the first adaptively secure randomness beacon protocol that overcomes all these limitations while preserving simplicity and optimal resilience in the synchronous network setting. We achieve our result in two steps. First, we design a novel distributed key generation (DKG) protocol GRand that runs in $\mathcal{O}(\lambda n^2 \log n)$ bits of communication but, unlike most conventional DKG protocols, outputs both secret and public keys as group elements. Second, following termination of GRand, parties can use their keys to derive a sequence of randomness beacon values, where each random value costs only a single asynchronous round and $\mathcal{O}(\lambda n^2)$ bits of communication. We implement GRandLine and evaluate it using a network of up to 64 parties running in geographically distributed AWS instances. Our evaluation shows that GRandLine can produce about 2 beacon outputs per second in a network of 64 parties. We compare our protocol to the state-of-the-art randomness beacon protocols in the same setting and observe that it vastly outperforms them.**

***Index Terms*—Adaptive Security, Randomness Beacon, Aggregatable PVSS, DKG, Pairing-Based Cryptography**

1. Introduction

Distributed randomness plays a crucial role in many cryptographic and distributed system applications. A randomness beacon [1], [2], [3] is a source of *public*, *unpredictable*, and *unbiased random* values that can be used by anyone in a secure manner. A well-designed randomness beacon protocol ensures that random values are generated in a decentralized and secure manner, preventing a threshold of t out of n collaborating parties from biasing or predicting the random outputs. Randomness beacon protocols have wide-ranging applications in both academia and industry. In many consensus protocols [4], [5] they are used to securely choose a leader or a committee among participating parties (e.g., through a verifiable random function [6]) to perform specific tasks. In this manner, these protocols can achieve better efficiency and circumvent impossibility results that apply to their deterministic counterparts. In mix networks [7], randomness beacons are used to shuffle messages and thus provide unlinkability between sender and receiver of a message. In privacy-oriented cryptocurrencies and voting systems [5], [8], randomness beacons provide user anonymity and unlinkability to their actions. In recent years, randomness beacons have attracted significant interest and numerous protocols have been proposed [9], [10]. However, existing randomness beacon protocols found in the literature suffer from at least one of the following drawbacks: (i) they achieve security only against a static, non-adaptive adversary, (ii) each epoch takes many rounds of communication, or (iii) they rely on computationally expensive tools such as Proof-of-Work (PoW) or Verifiable Delay Functions (VDF). Motivated by this unsatisfactory state of affairs, we give a novel randomness beacon protocol GRandLine which improves upon the-state-of-the-art by combining, for the first time, *all* of the following properties:[1]

- *Adaptive Security.* We prove GRandLine secure in the presence of an adaptive adversary.
- *One-Round Epoch.* Each epoch in GRandLine takes only a single asynchronous round of communication and is non-interactive. It only requires synchrony for its

[1] In the following, we only compare to adaptively secure protocols.

pre-processing phase. This sets GRandLine apart from all other protocols but Drand [11].

- *Low Communication.* GRandLine has $\mathcal{O}(\lambda n^2)$ bits communication cost per epoch. This sets GRandLine apart from BRandPiper [1] and RandShare [12] which have $\mathcal{O}(\lambda f n^2)$ and $\mathcal{O}(\lambda n^4)$ bits communication cost per epoch, respectively. Here, $f \leq t$ denotes the actual number of faults in the system.
- *Responsive.* Our protocol progresses at the actual speed of the network and is not bound to a much larger known upper bound on the network delay. This sets GRandLine apart from RandRunner [13], BRandPiper [1], and OptRand [3]. The latter achieves responsiveness only when there are $t < n/4$ actual corrupt parties in the system.
- *Optimal Resilience.* Our protocol has optimal resilience threshold $t < n/2$ in the synchronous network. This sets GRandLine apart from SPURT [2] and RandShare [12] which tolerate only sub-optimal $t < n/3$.
- *No Heavy Tools.* Our protocol only uses lightweight cryptography such as hash functions and pairings. Notably, we do not rely on expensive tools such as Proof-of-Work or VDF. This sets GRandLine apart from RandChain [14] and RandRunner [13] which rely on Proof-of-Work and VDF, respectively.
- *Subcubic Pre-Processing.* Our protocol has a pre-processing phase with $\mathcal{O}(\lambda n^2 \log n)$ bits communication cost. This sets GRandLine apart from Drand [11], OptRand [3], and BRandPiper [1] which all have $\mathcal{O}(\lambda n^3)$ bits communication cost for pre-processing.

We achieve our result in two steps. First, we design a novel distributed key generation (DKG) protocol GRand with $\mathcal{O}(\lambda n^2 \log n)$ bits communication cost that, unlike most of the conventional DKG protocols, outputs both secret and public keys as group elements. It is the first DKG protocol (in any network setting) that achieves subcubic communication cost with optimal resilience threshold. Second, we give a novel construction that allows us to use GRand as setup for a non-interactive and unique *locally verifiable* threshold signature[2] from which we naturally derive a simple one-round randomness beacon. For a detailed comparison of existing work on randomness beacons, we refer to Table 1.

1.1. Technical Overview

The idea of using a threshold signature scheme with unique signatures (per message m and public key pk) and a non-interactive signing procedure is a well-known approach to generate one-round distributed randomness. It is most commonly used in consensus protocols and dates back to the seminal work of Cachin et al. [15]. For epoch $e \geq 1$, this works as follows:

- Each party P_i non-interactively creates a signature share σ_i on the message $m := e$ and sends σ_i to all other parties.
- Upon receiving $t+1$ valid shares $\{\sigma_i\}_{i \in \mathcal{S}}$, a party locally reconstructs the full signature σ. The randomness beacon value is then computed as hash $O_e := \mathsf{H}(\sigma)$.

In this manner, one obtains a simple and efficient randomness beacon protocol which is also used by many blockchain and consensus protocols [4], [5]. This construction relies on a setup in which a secret key sk is (t, n)-secret shared among all parties. In a fully distributed system, this is commonly established via a DKG protocol for field elements [21]. Concretely, this means that at the end of the DKG protocol each party P_i holds a secret key share $sk_i \in \mathbb{Z}_p$ such that $sk_i = f(i)$ for a polynomial $f \in \mathbb{Z}_p[X]$ of degree t. Further, the public key shares $pk_i = \omega^{sk_i} \in \mathbb{G}$ are publicly known where $\mathbb{G}$ is a prime order p group with generator ω. Unfortunately, even the most efficient DKG protocols [22], [23] incur a communication cost of $\mathcal{O}(\lambda n^3)$ to generate their keys. So what does that mean for us?

Challenges in Subcubic DKG. A common approach to generate a secret key $sk \in \mathbb{Z}_p$ shared among a set of n parties out of which at most t can behave maliciously is as follows. Each party P_i samples a random value $r_i \leftarrow_\$ \mathbb{Z}_p$ and shares it among all parties using a VSS scheme for which r_i lies on some polynomial $f_i \in \mathbb{Z}_p[X]$ of degree t. Then, parties agree on a subset $I \subset [n]$ of at least $t+1$ dealers whose VSS sharings completed successfully. Finally, each party P_i combines the shares it received from dealers in I to obtain a share sk_i of the final secret sk. Crucially, we have the guarantee that the secret key sk can be reconstructed even when corrupt parties refuse to participate in the reconstruction. However, the best known VSS schemes have quadratic communication cost [22], [24], which leads to cubic communication cost in the overall protocol. One way to overcome this issue is to let a randomly sampled (and sometimes anonymous) committee of small size (e.g., in the range of $\mathcal{O}(\lambda)$) perform the task of sharing a secret. This technique is commonly used in consensus protocols to boost its scalability [6], most notably in the Algorand blockchain. However, in order to achieve security against an adaptive adversary, these protocols come with undesirable features such as sub-optimal corruption threshold of $t < (1-\varepsilon)n/3, \varepsilon > 0$, reliance on secure erasures of internal states, and impractical primitives such as fully homomorphic encryption (FHE) [25]. Further, to sample the committee in the first place, many protocols rely on an initial seed of common randomness, which creates a circularity (without assuming some form of trusted setup).

Starting Point: Aggregatable PVSS. Publicly verifiable secret sharing (PVSS) schemes [26] are VSS schemes with the additional property that any third party can verify that the sharing has been done correctly. In particular, this avoids the need for a complaint phase as is needed in regular VSS schemes, which greatly simplifies constructions based on PVSS schemes. Recently, Gurkan et al. [27] introduced a PVSS scheme that supports aggregation of several PVSS transcripts while preserving security (called *aggregatable PVSS* or simply *APVSS*). From that the authors design

[2] By “locally verifiable” we mean that the final threshold signature does not have an efficient verification algorithm, but the partial signatures have.

Table 1: Comparison table of existing distributed randomness beacon protocols.

Protocol	**Network**	**Resil.**	**Adapt.**	**Unpred.**	**Resp.**	**Rounds**	**Commun.**	**Crypto. Primit.**	**Setup**	**Preproc.**
Cachin et al. [15]	*async*	1/3	✗	✓	✓	1	$\mathcal{O}(\lambda n^2)$	Uniq. Th. Signature	*CRS*	$\mathcal{O}(\lambda n^3)$
RandHerd [12]	*async*	1/3	✗	✓	✓	ABA	$\mathcal{O}(\lambda c^2 \log n)$	PVSS & Th. Schnorr	*CRS*	$\mathcal{O}(\lambda n^3)$
Herb [16]	*sync*	1/3	✗	✓	✗	$t+1$	$\mathcal{O}(\lambda n^3)$	Thresh. ElGamal	*CRS*	$\mathcal{O}(\lambda n^3)$
Drand [11]	*sync*	1/2	✗	✓	✓	1	$\mathcal{O}(\lambda n^2)$	Thresh. BLS	*CRS*	$\mathcal{O}(\lambda n^3)$
SPURT [2]	*part sync*	1/3	✗	✓	✓	9	$\mathcal{O}(\lambda n^2)$	PVSS & Pairing	*CRS*	✗†
Algorand [6]	*part sync*	1/3	✗	$\Omega(t)$	✗	BC	$\mathcal{O}(\lambda c n)$	VRF	*Seed*	$\mathcal{O}(\lambda n^3)$
HydRand [17]	*sync*	1/3	✗	$t+1$	✗	3	$\mathcal{O}(\lambda n^2)$	PVSS	*Seed*	$\mathcal{O}(\lambda n^3)$
OptRand [3]	*sync*	1/2	✗	✓	(✓)	11	$\mathcal{O}(\lambda n^2)$	PVSS & Pairing	*q-SDH*	$\mathcal{O}(\lambda n^3)$
RandShare [12]	*async*	1/3	✓	✓	✓	ABA	$\mathcal{O}(\lambda n^4)$	VSS	*CRS*	✗
SPURT [18]	*part sync*	1/3	✓	✓	✓	9	$\mathcal{O}(\lambda n^2)$	PVSS & Pairing	*CRS & AGM*	✗†
RandChain [14]	*sync*	1/2	✓	$\mathcal{O}(\lambda)$	✓	Δ_{PoW}	$\mathcal{O}(\lambda n)$	PoW & VDF	*CRS*	✗◇
RandRunner [13]	*sync*	1/2	✓	$t+1$	✗	Δ_{VDF}	$\mathcal{O}(\lambda n^2)$	Trapdoor VDF	*Seed*	$\mathcal{O}(\lambda n^3)$
BRandPiper [1]	*sync*	1/2	✓	✓	✗	11	$\mathcal{O}(\lambda f n^2)$	VSS	*q-SDH*	$\mathcal{O}(\lambda f n^3)$
OptRand [18]	*sync*	1/2	✓	✓	(✓)	11	$\mathcal{O}(\lambda n^2)$	PVSS & Pairing	*q-SDH & AGM*	$\mathcal{O}(\lambda n^3)$
Drand [19]	*sync*	1/2	✓	✓	✓	1	$\mathcal{O}(\lambda n^2)$	Thresh. BLS	*CRS & AGM*	$\mathcal{O}(\lambda n^3)$
GRandLine	*sync*	1/2	✓	✓	✓	1	$\mathcal{O}(\lambda n^2)$	PVSS & Pairing	*CRS & AGM*	$\mathcal{O}(\lambda n^2 \log n)$

Resil. denotes the Byzantine resilience threshold. **Adapt.** denotes adaptive adversary. **Unpred.** denotes unpredictability. **Resp.** denotes responsiveness. Asynchronous protocols are by default responsive. OptRand is responsive only when there are $t < n/4$ corrupt parties in the system. **Rounds** denotes the number of (a)synchronous rounds per epoch. In RandHerd and RandShare, parties run n asynchronous Byzantine agreement (ABA) instances in parallel which leads to expected $\mathcal{O}(\log n)$ rounds per epoch. Algorand assumes a broadcast channel BC which leads to $t+1$ rounds per epoch when implemented with an actual broadcast protocol. In RandChain and RandRunner, each epoch takes one computational round to evaluate the VDF or PoW which is much larger than a synchronous network round. **Comm.** denotes the communication cost in bits. In RandHerd and Algorand, c denotes the average size of a randomly chosen committee. In BRandpiper, $f \leq t$ denotes the actual number of faults in the system. **Crypto. Primit.** denotes the cryptographic primitives in usage. **Setup** denotes the setup assumption. *CRS* denotes a common reference string setup. *Seed* denotes an initial random seed used to run the protocol. *q-SDH* denotes the powers-of-tau setup [20]. **Preproc.** denotes the communication cost for pre-processing. This can either be a DKG, a SMR, or some other distributed protocol that generates the initial random seed. Since the protocols with an initial seed do not specify how to obtain it (other than by trusted setup), we assume the most efficient DKG for this task. † SPURT guarantees only a weak form of liveness: t out of n consecutive epochs might fail to produce an output. ◇ RandChain uses Nakamoto consensus and also suffers from blockchain-related attacks.

a DKG protocol whose secret and public keys both are group elements in an underlying pairing group. Crucially, the authors leverage the property of aggregation from their APVSS scheme in order to reduce the communication and computation cost of parties by relying on gossiping techniques rather than all-to-all communication. However, their techniques only work against a static adversary that corrupts a mere $\log n$ parties maliciously. Additionally, their protocol assumes a one-round broadcast channel and has critical robustness issues.[3] Is there a way to regain all the desirable features simultaneously?

Recursion in APVSS to the Rescue. To solve the problem, we take inspiration from the world of recursive algorithms. In a recursive algorithm, the function calls itself with smaller input values in such a way that eventually a base case is reached which is easy to solve. The result of the function for the current input is then obtained from simple operations to the returned value for the smaller input. This technique has also found application in distributed protocols in order to improve communication cost. For this, we briefly recall the well-known Phase-King protocol [28] for (binary) Byzantine agreement and its more efficient recursive variant [29].[4]

(Recursive) Phase-King. The standard, non-recursive protocol runs over $t+1$ phases with a different leader (called the king) in each phase in a round-robin fashion. Each phase follows the same two-step structure. First, the parties run a weak form of Byzantine agreement called graded consensus, where each party outputs a value v_i along with an associated bit grade g_i that reflects the party's confidence in its output value. After graded consensus, the leader proposes its own value to all parties to establish agreement among the honest parties in case they are split between different values. In this case, a party accepts the leader's value only if its own grade is 0. This approach establishes consensus as soon as an honest party takes its turn as leader. To provide this guarantee, the protocol runs for $t+1$ phases which leads to cubic communication cost. Interestingly, this protocol can be made more efficient using recursion. In the recursive variant, the leader is replaced by one half of the entire system, whose value (that he wishes to propose) is generated by the recursive invocation of the protocol. Essentially, in this manner there are only two phases, with the first half of the system emulating the leader of the first phase and the second half of the system emulating the leader of the second phase. Since at least one of these halves has an honest majority (and thus emulates an honest leader), the protocol terminates after these two phases. In this manner, the communication cost of the protocol can be brought down to only quadratic.

Having recalled the (recursive) Phase-King protocol, we explain how we use a similar idea to design a DKG protocol whose secret and public keys both are group elements. Concretely, we want to devise a recursive protocol that allows parties to agree on an aggregated PVSS transcript AT with contribution from at least one honest party. From this transcript AT each party can locally and without any further interaction derive its share of the secret by a simple

[3] It has failure probability of n^{-c} where $c \in [4, n/\log n]$ is some success parameter chosen before the execution of the protocol.

[4] We note that these protocols work in the information-theoretic setting and therefore assume $t < n/3$. We only use them for exposition reasons.

decryption operation (we will explain this in more detail soon). In order to achieve our goal in an efficient way (with the hope to not exceed quadratic communication cost), we carefully put together aggregation properties of PVSS and recent techniques from the theory of verifiable information dispersal. In more detail, our protocol works as follows. We split the system into two halves and run the protocol recursively and in parallel in both halves separately, so that each half ends up with a single transcript. In the next step, for each half, the protocol emulates an efficient single-sender broadcast protocol for long messages [30] to transmit its transcript to all parties. For this step, we use recent techniques from the theory of information dispersal that rely on erasure codes and cryptographic accumulators. Finally, all parties aggregate these two transcripts and end up with a single (aggregated) PVSS transcript AT. Our DKG protocol GRand has a communication cost of $\mathcal{O}(\lambda n^2 \log n)$ bits and terminates in $\mathcal{O}(n)$ rounds.

From Aggregated PVSS to DKG. For the following discussion, let $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be an asymmetric pairing of prime order p groups with generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$. We elaborate on what we said before. For this, recall that a PVSS transcript generated by some dealing party P_* consists of a vector of commitments $\mathbf{C} = (C_1, \dots, C_n)$ in $\mathbb{G}_1$, a vector of encrypted shares $\mathbf{E} = (E_1, \dots, E_n)$ in $\mathbb{G}_2$, and some auxiliary data π which usually includes some proof. The commitments are generated as $C_i = g^{f(i)}$ for all $i \in [n]$ where $f \in \mathbb{Z}_p[X]$ is some polynomial of degree t chosen (randomly) by the dealer P_*, while the encryptions are computed as $E_i = pk_i^{f(i)}$. Here, (pk_i, sk_i) is the key pair of party P_i from a plain PKI setup such that $pk_i = h^{sk_i} \in \mathbb{G}_2$. The describing property of a PVSS scheme is that any subset $\mathcal{S}$ of at least $t+1$ parties can pool their decrypted shares $\{D_i\}_{i \in \mathcal{S}}$ to reconstruct the secret D_0 encoded in the transcript, while this remains infeasible with t or less such shares. By combining several PVSS transcripts with contribution from at least one honest dealing party, we have the guarantee that the secret of the aggregated transcript AT remains hidden from the adversary. In most applications of PVSS, the transcript AT is used to get one-time randomness by direct reconstruction of the secret (and possibly subsequent hashing). We are taking a different, more involved route. Specifically, we think of the commitments $C_1, \dots, C_n$ as public key shares and of the decryptions $D_1, \dots, D_n$ as secret key shares. That is, each party P_i outputs a secret key share $SK_i := D_i \in \mathbb{G}_2$, a vector of public key shares $(PK_1, \dots, PK_n)$ where $PK_j := C_j \in \mathbb{G}_1$ for all $j \in [n]$, and a public key $PK := C_0 \in \mathbb{G}_1$ (computed by Lagrange interpolation in the exponent from $C_1, \dots, C_n$). Thus, our DKG protocol GRand outputs its secret and public keys in a group rather than a field.

From One to Infinity: Towards a Simple Randomness Beacon. Clearly, GRand is different from most DKG protocols found in the literature that output secret keys in a field rather than a group. However, one of our key insights is that this setup is enough to generate a stream of one-round randomness values. Our idea is inspired from the threshold BLS signature [31] and its application to randomness generation as introduced by Cachin et al. Specifically, each party P_i non-interactively creates a threshold BLS signature share $\sigma_i := \mathsf{H}_1(m)^{sk_i}$ on the epoch number $m := e \in \mathbb{Z}_{\geq 1}$ with its secret key share sk_i and multicasts (i.e., sends it to all parties) it. Upon receiving $t+1$ valid shares $\{\sigma_i\}_{i \in \mathcal{S}}$ (which is checked by a pairing equation $e(g, \sigma_i) = e(pk_i, \mathsf{H}_1(m))$ from P_i's public key share pk_i), a party locally reconstruct the full signature $\sigma = \mathsf{H}_1(m)^{sk}$ by Lagrange interpolation in the exponent and derives the randomness beacon value as another hash $O_e = \mathsf{H}_2(\sigma)$. With our DKG protocol GRand that generates keys (PK_i, SK_i) as group elements, the operation $\mathsf{H}_1(m)^{SK_i}$ is not possible. However, when we think of the operation of „raising the group element $\mathsf{H}_1(m)$ to a power of sk_i“ as an abstract group action $sk_i \odot \mathsf{H}_1(m)$, we realize that the action[5] $SK_i \odot \mathsf{H}_1(m)$ defined as $e(\mathsf{H}_1(m), SK_i) \in \mathbb{G}_T$ is possible. Therefore, we let each party P_i non-interactively create a share σ_i as $e(\mathsf{H}_1(m), SK_i)$. And upon receiving $t+1$ valid such shares, each party can locally reconstruct the full signature by Lagrange interpolation in the exponent as $\sigma = e(\mathsf{H}_1(m), SK)$. Again, the randomness beacon value for epoch e is then derived as another hash $O_e := \mathsf{H}_2(\sigma)$. Intuitively, since the secret key SK is hidden from the adversary, the signature σ should remain unpredictable so that O_e gives a random and unbiased randomness value. However, there is one crucial issue with this approach: the verification of *beacon shares* σ_i. Previously, it was possible to verify such a (signature) share by a pairing check, but now the beacon share σ_i is an element in the target group $\mathbb{G}_T$ itself so that there is possibly no way to verify correctness of $e(\mathsf{H}_1(m), SK_i)$ without providing additional elements. In fact, σ_i cannot even be distinguished from a random element in $\mathbb{G}_T$ without breaking the decisional Diffie-Hellman (DDH) assumption in the group $\mathbb{G}_T$.

A Simple Two-Step Trick. In order to resolve this issue, we use the following two-step approach. After the DKG setup, each party P_i locally samples an element $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random and sends $\mathsf{cm}_i := (g^{\alpha_i}, h^{-\alpha_i} SK_i)$ to all parties. Correctness of its second component can be checked via a pairing equation. Crucially, this requires only a single round of communication (no *broadcast* in the sense of consensus is needed!) and therefore does not add any more overhead. These additional elements allow parties to verify received beacon shares. Concretely, each party computes $\vartheta_i := (\mathsf{H}_1(m)^{\alpha_i}, \sigma_i)$ along with a non-interactive zero-knowledge (NIZK) proof of discrete logarithm equality $\pi_i := \mathsf{Dleq}(g, g^{\alpha_i}, \mathsf{H}_1(m), \mathsf{H}_1(m)^{\alpha_i})$ to prove correctness of $\mathsf{H}_1(m)^{\alpha_i}$. Upon receiving such a tuple (ϑ_i, π_i), any party can verify the correctness of the beacon share σ_i using a pairing equation that involves the element $\mathsf{cm}_{i,2}$. Having done this, each party can compute the randomness beacon value O_e for epoch $e = m$ as

[5] Note this does not define a group action in the mathematical sense. Here, we use the term *group action* only informally to convey the intuition.

described before. Intuitively, security is preserved because the elements $g^{\alpha_i}, \mathsf{H}_1(1)^{\alpha_i}, \mathsf{H}_1(2)^{\alpha_i}, \ldots$ do not reveal too much information about α_i, thus making it hard for the adversary to compute SK_i from the (randomized) element $h^{-\alpha_i} SK_i$ (this is a highly simplified argument and the actual analysis is much more involved). Overall, an epoch takes only a single round of communication in which each party P_i sends two group elements $\vartheta_i := (\mathsf{H}_1(m)^{\alpha_i}, \sigma_i)$ and a simple NIZK proof π_i to the other parties. Further, (PK_i, cm_i) can be thought of as an updated public key share of P_i which is three group elements, and verification of a beacon share σ_i takes two pairing operations (the same as threshold BLS!) and verification of the NIZK proof π_i.

1.2. Organization of this Article

The rest of the paper is organized as follows. In Section 2, we define our model and cover relevant preliminaries, including cryptographic and consensus primitives. In Section 3, we design a novel DKG protocol GRand whose secret and public keys are group elements. In Section 4, we design a simple one-round randomness beacon GRandLine on top of GRand. Due to space constraints, we defer the security and complexity analysis of our construction to Appendix D. Finally, we implement GRandLine and compare it to the state-of-the-art randomness beacons in the same setting. In Appendix A, we give a detailed discussion on existing work in randomness beacon and DKG protocols. In Appendix B, we provide definitions for additional cryptographic and consensus primitives. In Appendix C, we design a Byzantine agreement protocol for use in GRand.

2. Preliminaries and Model

Throughout the paper, we consider a complete network $\mathcal{P}$ of n parties connected by pairwise authenticated channels, i.e. the receiver of a message is aware of the sender's identity. We assume known party identifiers, w.l.o.g. from P_1 to P_n. An unknown subset of these parties is faulty and controlled by an adversary.

General Notation. Let λ denote the security parameter. Throughout the paper, we assume that global parameters $par = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, e)$ are fixed and known to all parties. Here, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a type 3 asymmetric pairing of prime order p cyclic groups with generators $g \in \mathbb{G}_1, h \in \mathbb{G}_2$. That means there is no efficiently computable homomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ and vice versa. For concrete choices, we will assume $\lambda = 128$ and that $\mathbb{G}_1, \mathbb{G}_2$ are instantiated with a 256-bit elliptic curve. We use $\mathbb{G}$ to denote a group specified by par. For two integers $a \leq b$, we define the set $[a, b] := \{a, \ldots, b\}$; if $a = 1$, we write this set as $[b]$, and if $a = 0$, we write it as $[\![b]\!]$. For an element x in a set S, we write $x \leftarrow_{\$} S$ to mean that x was sampled from S uniformly at random. All our algorithms may be randomized (unless stated otherwise) and written in uppercase letters. By $x \leftarrow \mathsf{A}(x_1, \ldots, x_n)$ we mean running algorithm A on inputs $(x_1, \ldots, x_n)$ and uniformly random coins and then assigning the output to x. If A has oracle access to some algorithm B during its execution, we write $x \leftarrow \mathsf{A}^{\mathsf{B}}(x_1, \ldots, x_n)$. We write $\mathbf{G}^{\mathsf{A}}$ to denote the output of the game $\mathbf{G}$ involving algorithm A. Further, we denote by $\mathcal{LC}$ the Reed-Solomon code over $\mathbb{Z}_p$ of length n and dimension $t+1$ defined as

$$\mathcal{LC} := \{(f(1), \ldots, f(n)) \mid f \in \mathbb{Z}_p[X]_{(t)}\},$$

where $\mathbb{Z}_p[X]_{(t)}$ denotes the set of all polynomials in $\mathbb{Z}_p[X]$ of degree at most t. Its dual code $\mathcal{LC}^{\perp}$ is defined as

$$\mathcal{LC}^{\perp} := \{(\mu_1 r(1), \ldots, \mu_n r(n)) \mid r \in \mathbb{Z}_p[X]_{(n-t)}\},$$

where $\mu_i := \prod_{j \in [n] \setminus \{i\}} 1/(i-j)$. We measure the communication complexity of our distributed protocols in bits.

Public Key Infrastructure. We assume that the parties have established a public key infrastructure (PKI) via a public bulletin board. This means that every party P_i has a public-secret key pair (pk_i, sk_i), where pk_i is known to all parties. For this, we assume that each party generates its keys locally (where corrupt parties may choose their keys arbitrarily) and then makes its public key known to everybody using a public bulletin board. Further, we assume that the pairs (pk_i, sk_i) also (implicitly) include verification-signing key pairs (vk_i, dk_i) used for a digital signature scheme to provide authentication. In particular, we assume that parties sign each message before they send it to other parties. As common in this line of work [32], we treat signatures as information-theoretic objects with perfect unforgeability and perfect correctness (cf. Appendix B). When we want to make the signature of party P_i on message m explicit, we also write $\langle m \rangle_i$ for that.

Communication Model. We assume a *synchronous* communication model, i.e., computation proceeds in compute-send-receive rounds of a priori known length Δ. When a correct party sends a message m at the beginning of a round, the message is guaranteed to be received by the end of that round. In particular, messages sent by a correct party cannot be dropped from the network and are always delivered. Correct parties have local clocks that move at the same speed and they start the protocol at the same time.

Adversarial Model. We assume an adversary who can take full control of up to $t = \lceil n/2 \rceil - 1$ out of the n parties in the network and may cause them to deviate from the protocol arbitrarily; we call this a *t-bounded* adversary. The adversary is *adaptive* and chooses the faulty parties at any time during the execution of the protocol. We assume a *strongly adaptive* adversary: when it corrupts a party, it can delete or substitute any undelivered messages that this party previously sent (while being correct). Furthermore, we assume that the adversary is in full control over message delays, subject to the network delay Δ. This means that it can observe and deliver messages sent to and from correct parties far quicker than in time Δ. In particular, we assume the adversary to be *rushing*: in any synchronous round of a protocol execution, it can observe the messages of all the correct parties and then decide on what messages to deliver to correct parties for that round. We refer to the correct parties as *honest* and the faulty parties as *corrupt*.

Algebraic Group Model. The algebraic group model (AGM) [33] was introduced as a model in between the generic group model (GGM) [34] and the standard model. In the AGM, all algorithms are treated as *algebraic*. This means whenever an algorithm outputs a group element, it must also output a representation of that element relative to all of the inputs the algorithm has received up to that point.

Definition 1 (Algebraic Algorithm)**.** *An algorithm* A *is called* algebraic *(over the group* $\hat{\mathbb{G}}$*) if for all group elements* $\zeta \in \hat{\mathbb{G}}$ *that* A *outputs, it additionally outputs a vector* $\mathbf{z} = (z_1, \ldots, z_m)$ *of integers such that* $\zeta = \prod_i g_i^{z_i}$*, where* $(g_1, \ldots, g_m) \in \hat{\mathbb{G}}^m$ *is the list of group elements* A *has received so far.*

Computational Assumptions. We rely on the newly introduced *co-one-more discrete logarithm (Co-OMDL) assumption* [18] that generalizes the ordinary one-more discrete logarithm (OMDL) assumption to the setting of two prime order groups for which the challenge is simultaneously given in both groups with access to a discrete logarithm oracle in one of them. If both groups coincide, then this is just the standard OMDL assumption. The authors also provide a proof of hardness of Co-OMDL in the generic group model of Shoup. In the following, we denote by DL_g an oracle that on input an element $\xi = g^z \in \mathbb{G}_1$ returns its discrete logarithm z in base g.

Definition 2 (Co-One-More Discrete Logarithm Problem)**.** *For an algorithm* A *and* $n \in \mathbb{N}$*, we define the co-one-more discrete logarithm experiment* $n\text{-}\mathbf{COMDL}^{\mathsf{A}}$ *as follows:*

- Offline Phase. *Sample* $(z_1, \ldots, z_n) \leftarrow_{\$} \mathbb{Z}_p^n$ *uniformly at random and set* $\xi_i := (g^{z_i}, h^{z_i}) \in \mathbb{G}_1 \times \mathbb{G}_2$ *for* $i \in [n]$.
- Online Phase. *Run* A *on input* $(par, \xi_1, \ldots, \xi_n)$*. In this phase,* A *gets access to the oracle* DL_g.
- Output Determination. *Let* $(z_1', \ldots, z_n')$ *denote the output of* A*. Return 1 if (i)* $z_i' = z_i$ *for* $i \in [n]$*, and (ii)* DL_g *was queried at most* $n - 1$ *times during the online phase. Return 0 otherwise.*

We say that the co-one-more discrete logarithm problem of degree n *is* (ε, T)-hard *if for all algorithms* A *running in time at most* T*,* $\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}} = 1] \leq \varepsilon$*. Conversely, we say that an algorithm* A (ε, T)-solves *the co-one-more discrete logarithm problem of degree* n *if it runs in time at most* T *and* $\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}} = 1] > \varepsilon$.

2.1. Cryptographic Primitives

In this section, we formally define syntax and security notions of the cryptographic primitives used in the paper. We defer definitions and security notions for additional primitives to Appendix B.

(Aggregatable) Publicly Verifiable Secret Sharing. In a verifiable secret sharing (VSS) scheme, a dealer distributes shares of a secret among a group of parties such that it can be reconstructed only if a threshold of these parties collaborate. In a publicly verifiable secret sharing (PVSS) scheme, any third party can verify the correctness of the sharing, thus avoiding the need for a complaint phase as in VSS schemes. Henceforth, we consider PVSS schemes that support aggregation of several sharings while preserving public verifiability (called *aggregatable* PVSS scheme).

Definition 3 (Aggregatable PVSS Scheme)**.** *Let* $\hat{\mathbb{G}}$ *be a cyclic group of prime order* p *specified by* par*. A* (t, n)-threshold aggregatable PVSS (APVSS) scheme over $\hat{\mathbb{G}}$ *is a tuple of algorithms* $\mathsf{APVSS} = (\mathsf{Keys}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Dist}, \mathsf{Agg}, \mathsf{ConId}, \mathsf{Ver}, \mathsf{Rec})$ *with the following properties:*

- Keys*: The randomized* key generation algorithm *takes as input system parameters* par *and an identity index* $i \in [n]$*. It outputs a public key* pk_i *and a secret key* sk_i*.*
- Enc*: The randomized* encryption algorithm *takes as input a public key* pk_i *and a message* m*. It outputs a ciphertext* c*.*
- Dec*: The deterministic* decryption algorithm *takes as input a secret key* sk_i *and a ciphertext* c*. It outputs a message* m *(optionally with a proof of correct decryption). We require that for all messages* m*,*
$$\Pr[\mathsf{Dec}_{sk_i}(\mathsf{Enc}_{pk_i}(m)) = m] = 1.$$
- Dist*: The randomized* secret sharing algorithm *takes as input a secret key* sk_i *and public keys* $pk_1, \ldots, pk_n$*. It outputs a vector of encrypted shares* $\mathbf{E} = (\mathsf{Enc}_{pk_1}(S_1), \ldots, \mathsf{Enc}_{pk_n}(S_n))$ *and a proof* π*, where* $S_1, \ldots, S_n$ *are shares of a secret* $S \in \hat{\mathbb{G}}$*. We refer to* $T := (\mathbf{E}, \pi)$ *as a* PVSS transcript*.*
- Agg*: The deterministic* aggregation algorithm *takes as input PVSS transcripts* $(\mathbf{E}_1, \pi_1), \ldots, (\mathbf{E}_k, \pi_k)$*,* $k \in \mathbb{N}$*. It outputs an* (aggregated) PVSS transcript $T := (\mathbf{E}, \pi)$.
- ConId*: The deterministic* contributor identifier algorithm *takes as input an (aggregated) PVSS transcript* $T = (\mathbf{E}, \pi)$ *and a public key* pk_i*. It outputs 1 (accept) or 0 (reject). In the first case, we refer to* P_i *as a* contributor *to* T.[6]
- Ver*: The deterministic* verification algorithm *takes as input public keys* $pk_1, \ldots, pk_n$*, and an (aggregated) PVSS transcript* $T = (\mathbf{E}, \pi)$*. It outputs 1 (accept) or 0 (reject). In the first case, we call the transcript* T valid *(relative to* $pk_1, \ldots, pk_n$*); otherwise we call it* invalid*.*
- Rec*: The deterministic* reconstruction algorithm *takes as input* $t + 1$ *shares* $S_1, \ldots, S_{t+1}$*. It outputs a secret* $S \in \hat{\mathbb{G}}$*.*

Discussion. We defer formal definitions for correctness and secrecy of an APVSS scheme to Appendix B. Our definition above (and the ones in the appendix) are based on the ones from [18]. Essentially, the only difference to their definitions is the following. Their aggregation algorithm

[6] We remark that ConId could return 1 on an invalid transcript.

takes exactly $t + 1$ transcripts as input, in contrast to ours that can take any finite number of transcripts as input, even a single one. In particular, our definition generalizes theirs and we do not need to explicitly separate anymore between a standard PVSS transcript and an aggregated one. However, when we want to emphasize that the transcript was formed by aggregation of several (possibly themselves aggregated) transcripts, we will make this explicit and call the transcript aggregated. Having said this, we appropriately adapted some other algorithms and security notions to our generalized setting. Further, for an APVSS scheme, we require the secrecy notion of *aggregated unpredictability* as defined in [18] (slightly adapted). This notion captures malleability attacks and prohibits any t-bounded (i.e., corrupting at most t parties) adversary from learning the secret of an aggregated transcript that has contribution from at least one honest party (even if the adversary is allowed to contribute to the aggregation itself).

Linear Erasure and Error Correcting Codes. We use standard (q, b)-Reed-Solomon (RS) codes. This primitive allows to encode b data symbols into code words of q symbols such that b elements of the code word suffice to recover the original data. In our protocols, we use Reed-Solomon codes with varying (q, b). Concretely, we use codes with q being the number of parties in some subset of parties $\mathcal{Q} \subseteq \mathcal{P}$ (called *committee*) and b being $\lceil q/2 \rceil$. In the special case $\mathcal{Q} = \mathcal{P}$, we have $(q, b) := (n, t + 1)$.

Definition 4 (Reed-Solomon Code.)**.** *A Reed-Solomon code is a tuple of deterministic algorithms* $\Sigma = (\mathsf{Encode}, \mathsf{Decode})$ *with the following properties:*

- Encode*: The deterministic* encoding algorithm *takes as input b data symbols $(m_1, \ldots, m_b)$. It outputs a code word $(s_1, \ldots, s_q)$ of length q. Knowledge of any b elements of the code word uniquely determines the input message and the remaining of the code word.*
- Decode*: The deterministic* decoding algorithm *takes as input a code word $(s_1, \ldots, s_q)$ of length q. It outputs b data symbols $(m_1, \ldots, m_b)$. This algorithm tolerates up to c errors and d erasures in a code word $(s_1, \ldots, s_q)$ if and only if $q - b \geq 2c + d$.*

Cryptographic Accumulator. A cryptographic accumulator allows to accumulate several elements from some set D into an accumulated value z. Further, for each element in D it allows to generate a compact proof of membership in D. An example of cryptographic accumulators are Merkle trees, where the root is the accumulator value and the authentication paths are membership proofs for the leaves. We assume that it is hard to forge invalid proofs of membership (cf. Appendix B).

Definition 5 (Cryptographic Accumulator)**.** *A cryptographic accumulator scheme is a tuple of probabilistic polynomial-time algorithms* $\Sigma = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Wit}, \mathsf{Ver})$ *with the following properties:*

- Gen*: The randomized* accumulator key generation algorithm *takes as input the security parameter λ and an accumulation threshold n. It outputs a (public) accumulator key ak.*
- Eval*: The deterministic* evaluation algorithm *takes as input an accumulator key ak and a set $D = \{d_1, \ldots, d_n\}$ of elements to be accumulated. It outputs an accumulation value z for the set D.*
- Wit*: The possibly randomized* witness creation algorithm *takes as input an accumulator key ak, an accumulation value z for D, and an element d_i. It outputs $\perp$ if $d_i \notin D$, and a witness w_i otherwise.*
- Ver*: The deterministic* verification algorithm *that takes as input an accumulator key ak, an accumulation value z for D, a witness w_i, and an element d_i. It outputs 1 (accept) if w_i is a valid proof for membership $d_i \in D$ and 0 (reject) otherwise.*

In this paper, we use an accumulator scheme with (non-) membership proofs and accumulation value each of size $\mathcal{O}(\lambda)$. This can be implemented using the accumulator scheme of [35] built upon class groups of unknown order. Alternatively, one could use standard Merkle trees at the cost of $\mathcal{O}(\log n)$ overhead in communication.

2.2. Consensus Primitives

In this section, we formally define syntax and security notions of the consensus primitives used in the paper.

Byzantine Agreement. A Byzantine agreement (BA) protocol allows a set of parties, each holding an input $v_i \in V$, to agree on a common output value $v \in V$. Here, V is a predefined value set with $|V| \geq 2$ (that might include a default value $\perp$). We formally define a Byzantine agreement protocol as follows.

Definition 6 (Byzantine Agreement)**.** *Let* Π *be a protocol executed by parties $P_1, \ldots, P_n$, where each party P_i holds an input $v_i \in V$. We define the following properties for* Π*:*

- Validity. Π *is* t-valid *if the following holds whenever at most t parties are corrupted: if every honest party has the same value v as input, then every honest party outputs v.*
- Consistency. Π *is* t-consistent *if the following holds whenever at most t parties are corrupted: every honest party that outputs a value outputs the same value v.*
- Termination. Π *is* t-terminating *if the following holds whenever at most t parties are corrupted: every honest party terminates with an output value $v \in V \cup \{\perp\}$.*

We say that Π *is a* t-secure *Byzantine agreement protocol if it is t-valid, t-consistent, and t-terminating.*

Distributed Randomness Beacon. A randomness beacon is a distributed protocol that allows a system of n parties to generate a sequence of unpredictable and unbiased random values, one for each epoch. Each party P_i has a local log that is defined as a write-once array $\Sigma_i = (\Sigma_i[1], \Sigma_i[2], \ldots)$ with $\Sigma_i[\ell]$ being its beacon output at epoch $\ell \geq 1$. Initially, each value is set to $\perp$. We say that party P_i outputs a beacon value in epoch ℓ if it writes a value on $\Sigma_i[\ell]$. A secure randomness beacon has to satisfy the properties of consistency, availability, bias-resistance, and d-unpredictability. We formally define these security notions as follows.

Definition 7 (d-Secure Randomness Beacon)**.** *Let $\mathcal{RB}$ be an epoch-based protocol executed by parties $P_1, \ldots, P_n$. We define the following security properties for $\mathcal{RB}$:*

- Consistency. *$\mathcal{RB}$ is (t, L)-consistent if the following holds whenever at most t parties are corrupted: if an honest party outputs a value $\sigma_\ell \in \{0,1\}^\lambda$ in epoch $\ell \in [L]$, then all honest parties output σ_ℓ in epoch ℓ.*
- Availability. *$\mathcal{RB}$ is (t, L)-available if the following holds whenever at most t parties are corrupted: for each $\ell \in [L]$, every honest party outputs a value $\sigma_\ell \in \{0,1\}^\lambda$ in epoch ℓ.*
- Bias-Resistance. *$\mathcal{RB}$ is (t, ε, T, L)-bias-resistant if it is (t, L)-available, (t, L)-consistent, and the following holds for all algorithms A, D such that A is t-bounded and both A and D run in time at most T. Denote by $\Sigma_{\mathsf{A},L}$ the probability distribution induced by the outputs of an honest party in an execution of $\mathcal{RB}$ until epoch L with A as adversary. Then*

$$\left| \Pr_{\sigma \leftarrow \Sigma_{\mathsf{A},L}} [\mathsf{D}(\sigma) = 1] \quad - \Pr_{u \leftarrow U_L} [\mathsf{D}(u) = 1] \right| \leq \varepsilon,$$

  where U_L denotes the uniform distribution over the L-fold Cartesian product of $\{0,1\}^\lambda$ with itself.
- d-Unpredictability. *$\mathcal{RB}$ is $(t, \varepsilon, T, L, q_h, d)$-unpredictable if it is (t, L)-available, (t, L)-consistent, and for all $\ell \in [L]$ and algorithms A that run in time at most T and make at most q_h random oracle queries, A's success probability in the d-unpredictability experiment defined hereafter is at most ε.*

*We say that $\mathcal{RB}$ is a $(t, \varepsilon, T, L, q_h, d)$-*secure *randomness beacon protocol if it is (t, ε, T, L)-bias-resistant, $(t, \varepsilon, T, L, q_h, d)$-unpredictable, (t, L)-available, and (t, L)-consistent.*

Definition 8 (d-Unpredictability for $\mathcal{RB}$)**.** *Let $\mathcal{RB}$ be an epoch-based protocol as defined above. For an algorithm A and $\ell \in [L]$, we define the d-unpredictability experiment d-$\mathbf{Unpred}^{\mathsf{A},\ell}_{\mathcal{RB},t}$ as follows:*

- Offline Phase. *For all $i \in [n]$, generate keys as $(pk_i, sk_i) \leftarrow \mathsf{Keys}(par, i)$. On input par and $\{pk_i\}_{i \in [n]}$, A returns an index set $\mathcal{C} \subset [n]$ of initially corrupted parties along with updated public keys $\{\hat{pk}_i\}_{i \in \mathcal{C}}$. Set $pk_i := \hat{pk}_i$ for all $i \in \mathcal{C}$. Initiate an execution of $\mathcal{RB}$ with A controlling parties in $\mathcal{C}$.*
- Random Oracle Queries. *At any point of the experiment, A gets access to an oracle of the following type: When A submits a query m, check if $H[m] = \bot$. In this case, set $H[m] \leftarrow \{0,1\}^\lambda$. Return $H[m]$.*
- Online Phase. *Run $\mathcal{RB}$ with A. When A outputs a value (σ'_e, e) for an $e > \ell$, the experiment ends with output* 0 *in case there is an honest party that has output a value $\sigma_{\ell+1}$ for epoch $\ell + 1$. Continue the execution of $\mathcal{RB}$ for another $e - \ell$ epochs.*
- Corruption Queries. *During the online phase, A may corrupt a party P_i by submitting an index $i \in [n] \setminus \mathcal{C}$. In this case, return the internal state of P_i and set $\mathcal{C} := \mathcal{C} \cup \{i\}$. Henceforth, A fully controls P_i.*
- Output Determination. *Return 1 if $|\mathcal{C}| \leq t$, $e \geq \ell + d$, $L \geq e$, and $\sigma'_e = \sigma_e$. Otherwise, return 0.*

Discussion. We briefly elaborate on the security notions defined above. Consistency and availability guarantee that each honest party outputs the same value $\sigma_e \in \{0,1\}^\lambda$ in each epoch $e \geq 1$. Bias-resistance guarantees that the beacon outputs are indistinguishable from uniformly random numbers. This property ensures that the adversary has no power in biasing the beacon output, even when controlling up to t parties in the system. On the other hand, this notion does not prohibit the adversary from learning the beacon output some epochs ahead of the honest parties. That is ensured by the notion of d-unpredictability, which states that the adversary does not learn the beacon output d epochs before the honest parties. Conversely, an adversary could predict the beacon output some epochs ahead of the honest parties, e.g. by corrupting the next t leaders whose previously committed values determine the next t beacon outputs as in GRandPiper [1] or HydRand [17], without actually having the power to bias it.

3. Distributed Key Generation

In this section, we design a novel distributed key generation (DKG) protocol whose secret and public keys both are *group elements*. This is different from most conventional DKG protocols that output secret keys in a field rather than a group. One of our key insights, however, is that this setup is enough to generate a (pairing-based) unique threshold signature from which we naturally derive our one-round randomness beacon. We first define a DKG protocol in our context. Recall our notation $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$.

Definition 9 (DKG Protocol)**.** *Let Π be a protocol executed by parties $P_1, \ldots, P_n$, where for all $i \in [n]$, P_i outputs a* secret key share *SK_i, a vector of* public key shares *$(PK_1, \ldots, PK_n)$, and a* public key *PK. We define the following properties for Π:*

- Consistency. *Π is* t-consistent *if the following holds whenever at most t parties are corrupted: all honest parties output the same public key PK and the same vector of public key shares $(PK_1, \ldots, PK_n)$.*
- Correctness. *Π is* t-correct *if the following holds whenever at most t parties are corrupted: there exists a polynomial $f \in \mathbb{Z}_p[X]$ of degree t such that $\forall i \in [n]$, $SK_i = h^{f(i)}$ and $PK_i = g^{f(i)}$. Moreover, $PK = g^{f(0)}$.*
- Termination. *Π is* t-terminating *if the following holds whenever at most t parties are corrupted: if all honest parties start the protocol, then they all terminate with an output.*
- Secrecy. *Π is* (t, ε, T)-secret *if for all algorithms A that run in time at most T, the success probability in the following experiment is at most ε.*
  - Offline Phase. *Initialize a corruption index set $\mathcal{C} := \varnothing$ and let $\mathcal{H} := [n] \setminus \mathcal{C}$. Run A on input par.*

- Corruption Queries. *At any point of the experiment,* A *may corrupt a party by submitting an index* $i \in \mathcal{H}$*. In this case, return the internal state of* P_i *and set* $\mathcal{C} := \mathcal{C} \cup \{i\}$*. Henceforth,* A *fully controls* P_i*.*
- Online Phase. *Initiate an execution of* Π *with* A *having full control over parties in* $\mathcal{C}$*. Let* $y = g^x \leftarrow \Pi$ *be the public key output by* Π*.*
- Output Determination. *Let* s' *denote the output of* A*. Then,* A *is considered successful if and only if* $|\mathcal{C}| \leq t$ *and* $s' = h^x$ *(which is the secret key).*

We say that Π *is a* (t, ε, T)-secure *DKG protocol if it is t-consistent, t-correct, t-terminating, and* (t, ε, T)*-secret.*

3.1. Components of our DKG Protocol

In the following two sections, we design a novel DKG protocol, called GRand, with a communication complexity of $\mathcal{O}(\lambda n^2 \log n)$. At the heart of GRand lies a recursive protocol that allows to aggregate PVSS transcripts from an underlying APVSS scheme. Further, GRand makes use of a Byzantine agreement protocol BA and an algorithm Deliver that allows to efficiently propagate long messages. In the following, we describe the components of GRand.

Aggregatable PVSS Scheme. Our APVSS scheme is similar to the one of Bhat et al. [3] which is essentially (the pairing-based version of) SCRAPE augmented with a signed NIZK proof of knowledge of discrete logarithm for $\zeta = g^\alpha$ where $\alpha := f(0)$ for a randomly chosen degree-t polynomial $f \in \mathbb{Z}_p[X]$. The difference between our schemes is that the reconstructed secret of our scheme is h^α, while the reconstructed secret in their scheme is $e(\hat{g}, h^\alpha)$ where $\hat{g} \in \mathbb{G}_1$ is an additional generator. The reason for this is their security proof which crucially relies on that design and the additional generator. However, since in our randomness beacon we never explicitly reconstruct the secret, it does not make much of a difference for our security proof and thus we can sidestep the need for a further generators in the source groups. We provide a formal description of our APVSS scheme in Figure 1 below.

Deliver Protocol. The protocol design was introduced in [1] and is based on erasure codes and cryptographic accumulators. Deliver is a two-round protocol that is invoked by a party P_i that wants to efficiently broadcast a long message m to all parties in some set $\mathcal{Q}$. Contrary to [1], we make use of Deliver for sets of varying sizes. We parameterize Deliver by a set $\mathcal{Q}$ of q parties among which it is executed. It is invoked by a party $P_i \in \mathcal{Q}$ and takes as input a long message m, the accumulation value z for an encoding of m, and $\mathcal{Q}$ along with implicit parameters $q = |\mathcal{Q}|$ and $b = \lceil q/2 \rceil$. For this, P_i first splits m into b data symbols and encodes these into q code words using an (q, b)-erasure code RS. Then, P_i sends the j-th code word, the accumulation value z for the set of q code words along with a witness to $P_j \in \mathcal{Q}$. Upon receiving a valid triple of this type, P_j forwards it to all other parties in $\mathcal{Q}$. Finally, upon receiving b valid code words corresponding to the accumulation value z, P_j reconstructs the full message m using the decoding algorithm. In this way, message m can efficiently reach all honest parties in $\mathcal{Q}$ when the sender P_i was honest. We provide a formal description of Deliver in Figure 2 below.

Byzantine Agreement. For this task, we use a variant of the recursive Phase-King protocol as given by Momose and Ren [36].[7] The protocol uses compact certificates of size $\mathcal{O}(\lambda)$ that prove knowledge of a threshold of signatures from other parties on the same message. We implement these certificates with succinct non-interactive arguments of knowledge (SNARK, cf. Appendix B) that do not require trusted setup. Efficient examples of such transparent SNARKs are well-known and given in [37], [38]. On the other hand, when the network of parties is of small size $n \in \mathcal{O}(\lambda)$, we can instead resort to an aggregatable signature such as standard BLS by simply adding the n-bit long vector of signers to the aggregated signature. Crucially, the protocol (in either regime) has a communication complexity of $\mathcal{O}(\lambda n^2)$ and terminates in a linear number of rounds. We provide the protocol description along with a discussion on its security and complexity in Appendix C.

Recursive PVSS Aggregation. We give an intuitive description of the protocol GenAPVSS (cf. Figure 3). The protocol is parameterized by a set $\mathcal{Q}$ of q parties among which it is executed. At the end, each party outputs the same PVSS transcript AT where possibly $AT = \perp$. The protocol begins by dividing $\mathcal{Q}$ into two equally-sized disjoint subsets $\mathcal{Q} = \mathcal{Q}_1 \sqcup \mathcal{Q}_2$ (each called a *committee*) among which the protocol GenAPVSS is executed recursively. Upon termination of both recursive calls $l \in [2]$, parties $P_i \in \mathcal{Q}_l$ output a transcript T_l. Now the main part of the protocol begins. The idea is for parties from $\mathcal{Q}_l$ to send their transcript T_l to the other committee $\mathcal{Q}_{1-l}$'s members in an efficient and secure way. If we let each party from respective committee simply send its transcript to the other committee, then that would lead to cubic or higher communication which is not what we want. To resolve this issue, we follow a four-step approach. First, each party $P_i \in \mathcal{Q}_l$ generates an accumulation value z_l of small size for its (large-sized) transcript T_l and sends it to all parties in the larger system $\mathcal{Q}$. Here, another party $P_j \in \mathcal{Q}$ only accepts z_l (with implicit committee assignment $z_l \rightarrow \mathcal{Q}_l$) if it was received from a majority of parties from $\mathcal{Q}_l$. Following this, all parties (in the system $\mathcal{Q}$) establish consensus on both accumulation values z_1 and z_1, each being assigned to the respective committee, by running two instances of the Byzantine agreement protocol BA in parallel. As a result, each party keeps the same values z_1 and z_2 with implicit committee assignments $z_1 \rightarrow \mathcal{Q}_1$ and $z_2 \rightarrow \mathcal{Q}_2$, respectively. In the second step, each party $P_i \in \mathcal{Q}_l$ propagates its transcript T_l to all parties in $\mathcal{Q}$ using Deliver on input $(\mathcal{Q}, T_l, z_l)$. This ensures efficient delivery of the large-sized transcript T_l to all other parties. To retain security, as an adversarial-controlled committee could

[7] The recursive Phase-King protocol is defined in the information-theoretic setting and therefore assumes $t < n/3$. Momose and Ren [36] design a computationally secure variant with optimal resilience $t < n/2$.

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be a pairing with generators $g \in \mathbb{G}_1, h \in \mathbb{G}_2$, and let (pk_i, sk_i) be the key pair of party P_i where $pk_i = h^{sk_i}$. The *secret sharing algorithm* Dist (invoked by some dealing party P_L) takes as input a secret key sk_L and public keys $pk_1, \ldots, pk_n$. It outputs a PVSS transcript $T_L := \{\mathbf{C}, \mathbf{E}, \pi\}$ as follows.

(1) Sample a polynomial $f(X) = \alpha + \alpha_1 X + \ldots + \alpha_t X^t \in \mathbb{Z}_p[X]$ of degree t uniformly at random.
(2) Set the commitments $C_i := g^{f(i)} \in \mathbb{G}_1$ and encrypted shares $E_i := pk_i^{f(i)} \in \mathbb{G}_2$ for all $i \in [n]$.
(3) Compute a NIZK proof of knowledge $\theta = (c, r)$ of discrete logarithm for $\zeta := g^{\alpha}$. Set $\pi := \langle \zeta, \theta \rangle_L$.

The *aggregation algorithm* Agg takes as input $k \geq 1$ transcripts $\{\mathbf{C}_i, \mathbf{E}_i, \pi_i\}_{i \in [k]}$ and outputs a transcript $T := \{\mathbf{C}, \mathbf{E}, \pi\}$. Let $\mu_1, \ldots, \mu_{t+1}$ denote the Lagrange coefficients for the set $\{1, \ldots, t+1\}$ at the point $x = 0$.

(4) Set $C_i := \mathbf{C}_{1,i} \cdot \ldots \cdot \mathbf{C}_{k,i}$ and $E_i := \mathbf{E}_{1,i} \cdot \ldots \cdot \mathbf{E}_{k,i}$ for all $i \in [n]$. Further, set $\pi := (\pi_1, \ldots, \pi_k)$. Publish the (aggregated) transcript $T := \{\mathbf{C}, \mathbf{E}, \pi\}$ where $\mathbf{C} = (C_1, \ldots, C_n)$ and $\mathbf{E} = (E_1, \ldots, E_n)$.

The *contributor identifier algorithm* ConId takes as input a transcript $T := \{\mathbf{C}, \mathbf{E}, \pi\}$ and a public key pk_i.

(5) Sparse π as $(\pi_1, \ldots, \pi_k)$. For all $j \in [k]$, check if the signature on π_j verifies using pk_i. If one of these checks succeeds, output 1 (contribution from P_i). Otherwise, output 0 (no contribution from P_i).

The *verification algorithm* Ver takes as input public keys $pk_1, \ldots, pk_n$ and a transcript $T = \{\mathbf{C}, \mathbf{E}, \pi\}$. It outputs 1 (accept) or 0 (reject). Let $\mathcal{LC}$ be the linear code as defined in Section 2 and let $\mathcal{LC}^{\perp}$ be its dual code.

(6) Sample $(\nu_1, \ldots, \nu_n) \leftarrow_{\$} \mathcal{LC}^{\perp}$ and check if $C_1^{\nu_1} \cdot \ldots \cdot C_n^{\nu_n} = 1$. For all $i \in [n]$, check if $e(g, E_i) = e(C_i, pk_i)$.
(7) Sparse π as $(\pi_1, \ldots, \pi_k)$ and check if $\zeta_1 \cdot \ldots \cdot \zeta_k = C_0$ (obtained by Lagrange interpolation in the exponent from $\mathbf{C}$). Further, check if the NIZK proofs θ_i and signatures on π_i (using pk_i) for all $i \in [k]$ verify.
(8) If one of the above checks fails, output 0 (invalid transcript). Otherwise, output 1 (valid transcript).

The *decryption algorithm* Dec (on input encrypted shares $\mathbf{E} = (E_1, \ldots, E_n)$) and the *reconstruction algorithm* Rec.

(9) On input a secret key sk_i, compute the secret share $S_i = E_i^{1/sk_i}$. A secret share S_j (with corresponding index j) is considered valid if $e(C_j, h) = e(g, S_j)$. Otherwise, the share is considered invalid.
(10) On input $t+1$ valid secret shares $\{S_i\}_{i \in I}$, reconstruct the secret S by Lagrange interpolation in the exponent for the set I. Concretely, the reconstructed secret is of the form $S = h^{f(0)} \in \mathbb{G}_2$.

Figure 1: Description of the algorithms of our aggregatable PVSS scheme.

Let $\mathcal{Q} \subseteq \mathcal{P}$ be a set of q parties and let $b := \lceil q/2 \rceil$ be the decoding threshold for a (q, b)-Reed-Solomon code RS = (Encode, Decode). Further, let AC = (Gen, Eval, Wit, Ver) be an accumulator scheme, and let $z \leftarrow$ Eval$(ak,$ Encode$(m))$ be the accumulation value for a predefined encoding of message m.

- **Round 1.** Split m into b data symbols $(m_1, \ldots, m_b)$ as per a predefined policy. Run Encode on input $(m_1, \ldots, m_b)$ to obtain q code words $(s_1, \ldots, s_q)$. For all $j \in [q]$, compute a witness $w_j \leftarrow$ Wit(ak, z, s_j) and send $\langle s_j, w_j, z \rangle_i$ to party $P_j \in \mathcal{Q}$.
- **Round 2.** Any party $P_j \in \mathcal{Q}$ does the following. Upon receiving the *first* valid code word $\langle s_j, w_j, z \rangle_*$ for z, forward this code word to all parties in $\mathcal{Q}$. *// This step happens for party P_j only if it knows z.*
- **Local Output.** Upon receiving b valid code words for z, run Decode on these code words to obtain m.

Figure 2: Description of the Deliver protocol on input $(\mathcal{Q}, m, z)$ invoked by party P_i.

simply refuse to deliver its transcript, parties in $\mathcal{Q}$ need to decide on whether the previous step succeeded. For this, all parties run two instances of BA in parallel. If the output of a particular instance is 1, then at least one honest party thinks that the previous step succeeded and it has obtained the respective transcript. The fourth step proceeds with another invocation of Deliver to ensure that each honest party in the whole system $\mathcal{Q}$ obtains the transcripts $\{T_1, T_2\}$ for which the respective instance of BA output 1 (for the other, parties can set $T_i := \perp$). Finally, each party aggregates both transcripts and outputs $AT \leftarrow$ Agg(T_1, T_2). Aggregation of $(\perp, \perp)$ outputs $\perp$. Overall, the protocol outputs a transcript AT among all parties. We provide a formal description of GenAPVSS in Figure 3 below.

3.2. Design of our DKG Protocol

In the following, we describe our DKG protocol GRand. Essentially, it generates an aggregated PVSS transcript through an execution of GenAPVSS from which parties can locally derive their secret key shares without any further communication among them.

Our DKG Protocol. A formal description of the protocol GRand is given in Figure 4. The protocol consists of two simple steps. First, parties in $\mathcal{P}$ execute the protocol GenAPVSS to establish consensus on an aggregated PVSS transcript $AT := \{C_j, E_j, \pi\}_{j \in [n]}$ (which has contribution from at least one honest party by design and thus is secure from the adversary). Then, each party

Let $\mathcal{Q} \subseteq \mathcal{P}$ be a set of q parties and let $b := \lceil q/2 \rceil$. Partition $\mathcal{Q}$ into two disjoint subsets $\mathcal{Q} = \mathcal{Q}_1 \sqcup \mathcal{Q}_2$ (each called a *committee*) of size $q_1 := \lceil q/2 \rceil$ and $q_2 := \lfloor q/2 \rfloor$, respectively. Let $l \in \{1, 2\}$ be such that $P_i \in \mathcal{Q}_l$. We use the notation $m^\dagger := \mathsf{Encode}(m_1, \dots, m_b)$ for a (q, b)-Reed-Solomon code $\mathsf{RS} = (\mathsf{Encode}, \mathsf{Decode})$ and where $(m_1, \dots, m_b)$ is a partition of m as per a predefined policy.

- **Initialization Phase.** Let $\perp$ denote a default value. Initialize variables $v_1, v_2 := \perp$ and sets $W_1, W_2 := \varnothing$.
- **Recursion Phase.** Run $\mathsf{GenAPVSS}(\mathcal{Q}_l)$ among all parties in $\mathcal{Q}_l$ and let T_l denote the output. If $|\mathcal{Q}_l| = 1$, obtain T_l by executing $T_l \leftarrow \mathsf{Dist}(sk_i, (pk_1, \dots, pk_n))$. *// Each committee recursively executes the protocol to obtain a PVSS transcript each. Both committees do this step in parallel (and not sequential).*
- **Accumulator Proposal.** Generate an accumulation value z_l for T_l by executing $z_l \leftarrow \mathsf{Eval}(ak, T_l^\dagger)$ and send $\langle \mathsf{accum}, z_l \rangle_i$ to all parties in $\mathcal{Q}$. Upon receiving the *same* value $\tilde{z}_j$ (of type accum, with valid signature) from $\lfloor q_j/2 \rfloor + 1$ distinct parties in $\mathcal{Q}_j$ (a majority set of parties), update the variable $v_j := \tilde{z}_j$ for each $j \in \{1, 2\}$ at most once. *// Parties in the larger system $\mathcal{Q}$ only accept the majority value z_j received from each committee $\mathcal{Q}_j$.*
- **Accumulator Agreement.** For each $j \in \{1, 2\}$, run an instance BA_j of Byzantine agreement on input v_j (with implicit index) among all parties in $\mathcal{Q}$. Update v_j as the output of BA_j. *// Parties run both Byzantine agreement instances in parallel for $j \in \{1, 2\}$. This step is done to have consensus on the accumulation value z_j assigned to each committee $\mathcal{Q}_j$.*
- **Transcript Delivery.** Let $Qual := \{j \in [2] \mid v_j \neq \perp\}$. Only if $z_l = v_l$ and $l \in Qual$, propagate T_l among all parties in $\mathcal{Q}$ by executing $\mathsf{Deliver}(\mathcal{Q}, T_l, z_l)$. Upon decoding a *valid* transcript $\tilde{T}_j$ (with accumulation value v_j) for which $j \in Qual$, update the respective set $W_j := \{\tilde{T}_j\}$ at most once. *// Parties that have the correct T_j for the agreed-upon accumulation value v_j, deliver it to all parties in the whole system $\mathcal{Q}$.*
- **Committee Selection.** For each $j \in Qual$, run an instance BA_j of Byzantine agreement on input $|W_j| \in \{0, 1\}$ (input is 0 if W_j is empty, and 1 otherwise) among all parties in $\mathcal{Q}$. Let $b_j \in \{0, 1\}$ be the output bit, and update $Qual := Qual \cap \{j \mid b_j = 1\}$. *// Parties decide on whether a (qualified) committee $\mathcal{Q}_j$ has correctly delivered its initial transcript T_j.*
- **Transcript Agreement.** For each $j \in Qual$, do as follows: only if $W_j \neq \varnothing$, propagate $\tilde{T}_j \in W_j$ among all parties in $\mathcal{Q}$ by executing $\mathsf{Deliver}(\mathcal{Q}, \tilde{T}_j, v_j)$. On the other hand, upon decoding a transcript $\tilde{T}_j$ for which $j \in Qual$ and $W_j = \varnothing$, update the respective set $W_j := \{\tilde{T}_j\}$ at most once. *// Since a committee is selected from the previous step only if (!) at least one honest party has T_j, this step ensures that T_j reaches all honest parties in $\mathcal{Q}$.*
- **Final Aggregation.** For each $j \in \{1, 2\}$, do as follows: only if $W_j = \varnothing$, update $W_j := \perp$ as default value. Generate a PVSS transcript AT by executing $AT \leftarrow \mathsf{Agg}(W_1, W_2)$ and output AT. *// Parties aggregate both transcripts and terminate.*

Figure 3: Description of the recursive PVSS transcript generation protocol $\mathsf{GenAPVSS}$ for the set $\mathcal{Q} \subseteq \mathcal{P}$ from the view of party $P_i \in \mathcal{Q}$.

P_i computes its secret share $D_i := \mathsf{Dec}_{sk_i}(E_i)$ and terminates. The public key shares of GRand are defined as $(PK_1, \dots, PK_n) := (C_1, \dots, C_n)$ with the secret key shares being $(SK_1, \dots, SK_n) := (D_1, \dots, D_n)$. Using the specific APVSS scheme described in Figure 1, the public key shares of GRand are $PK_i = g^{f(i)}$ and the secret key shares are $SK_i = h^{f(i)}$, where $f \in \mathbb{Z}_p[X]$ is the hidden degree-t polynomial encoded in the APVSS transcript AT. We note that even though P_i knows $h^{f(i)}$, it has (almost) no knowledge on $f(i)$. In particular, this DKG protocol is very different from most DKG protocols in the literature where the hidden polynomial itself is distributed among the parties.

In the following theorem, we show that GRand is a secure DKG with almost quadratic communication complexity and linear round complexity. The security of the protocol follows from the security of the underlying Byzantine agreement protocol and the aggregated unpredictability of the APVSS scheme. Further, we derive the communication and round complexity from recursive formulas. We provide a formal proof for the following theorem in Appendix D.1.

Theorem 1. *If* APVSS *is* $(t, \varepsilon_A, T_A, q_s, q_h)$*-aggregated unpredictable in the ROM and* BA *is* t*-secure, then* GRand *(cf. Figures 3 and 4) is a* (t, ε, T)*-secure DKG protocol, where* $\varepsilon \leq \varepsilon_A + (q_s + q_h)/p$ *and* $T \geq T_A + \mathcal{O}(n^2)$*. Further,* GRand *has a communication complexity of* $\mathcal{O}(\lambda n^2 \log n)$ *bits and terminates in* $\mathcal{O}(n)$ *rounds.*

4. Our 1-Round Randomness Beacon

In this section, we design our efficient and simple randomness beacon protocol $\mathsf{GRandLine}$.

4.1. Construction of GRandLine

In the following, we give an informal description of our randomness beacon protocol. A formal description of $\mathsf{GRandLine}$ is given in Figure 5. Let H_1 and H_2 be hash functions modeled as random oracle and denote $g_r := \mathsf{H}_1(r)$ for all $r \in \mathbb{N}$. Parties begin by executing GRand upon which every party P_i obtains a public-secret key pair $(PK_i, SK_i) := (g^{f(i)}, h^{f(i)})$ for a hidden polynomial $f \in \mathbb{Z}_p[X]$ of degree t. The idea now is to use $\vartheta_i := e(g_r, SK_i) \in \mathbb{G}_T$ as a partial

Let $\mathcal{P} = \{P_1, \dots, P_n\}$. The protocol outputs a vector of secret key shares $(SK_1, \dots, SK_n) \in \mathbb{G}_2^n$ where SK_j is known only to P_j, a vector of public key shares $(PK_1, \dots, PK_n) \in \mathbb{G}_1^n$, and a public key $PK \in \mathbb{G}_1$.

- **Transcript Generation.** Run $\mathsf{GenAPVSS}(\mathcal{P})$ among all parties in $\mathcal{P}$ and obtain a PVSS transcript $AT := \{\mathbf{C}, \mathbf{E}, \pi\}$ from the execution. *// This step generates an agreed-upon PVSS transcript AT for all parties in $\mathcal{P}$.*
- **Key Derivation.** Compute your secret share D_i by executing $D_i \leftarrow \mathsf{Dec}_{sk_i}(E_i)$. Terminate with output $(PK_1, \dots, PK_n) := (C_1, \dots, C_n)$ and $SK_i := D_i$. *// Parties derive their secret key shares directly from AT without further communication. These key shares interpolate a degree-t polynomial $f \in \mathbb{Z}_p[X]$ in the exponent.*

Figure 4: Description of the DKG protocol GRand from the view of party P_i.

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ be a pairing with generators $g \in \mathbb{G}_1, h \in \mathbb{G}_2$. Let $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}_1$ and $\mathsf{H}_2 : \mathbb{G}_T \to \{0,1\}^\lambda$ be two cryptographic hash functions modeled as random oracle. Hereafter, let $g_r := \mathsf{H}_1(r)$ for all $r \in \mathbb{N}$.

- **Setup Phase.** Parties execute the DKG protocol GRand and obtain a vector of secret key shares $(SK_1, \dots, SK_n) \in \mathbb{G}_2^n$, a vector of public key shares $(PK_1, \dots, PK_n) \in \mathbb{G}_1^n$, and a public key $PK \in \mathbb{G}_1$. *// This serves as setup for the randomness beacon which starts following a one-time, one-round commitment phase.*
- **Commitment Phase.** Initialize a local set $\mathcal{G} := \varnothing$. Sample $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random and multicast (send to all parties) $\mathsf{cm}_i := (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Upon receiving $\mathsf{cm}_j := (\mathsf{cm}_{j,1}, \mathsf{cm}_{j,2})$ from party P_j, check if $e(PK_j, h) = e(\mathsf{cm}_{j,1}, h) \cdot e(g, \mathsf{cm}_{j,2})$. Only if the check verifies, update $\mathcal{G} := \mathcal{G} \cup \{P_j\}$. *// This step is done only once, and each party stores the commitments cm_j it received from other parties.*
- **Beacon Epoch** r**.** Compute $\sigma_i := (g_r^{\alpha_i}, e(g_r, SK_i))$ along with $\pi_i := \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$, and multicast (σ_i, π_i). Upon receiving (σ_j, π_j) from party $P_j \in \mathcal{G}$, check if π_j verifies using $\mathsf{cm}_{j,1}$ and $\sigma_{j,1}$. Further, check if $\sigma_{j,2} = e(g_r, \mathsf{cm}_{j,2}) \cdot e(\sigma_{j,1}, h)$.
- **Reconstruction Phase.** Upon receiving $t+1$ valid tuples $\{(\sigma_j, \pi_j)\}_{j \in \mathcal{S}}$ from distinct parties in $\mathcal{G}$, compute $\sigma := e(g_r, SK)$ by Lagrange interpolation in the exponent from $\{\sigma_{j,2} = e(g_r, SK_j)\}_{j \in \mathcal{S}}$. *// Only local computation.*
- **Beacon Output.** Upon reconstruction of σ in epoch r, output the beacon value $\varrho_r := \mathsf{H}_2(\sigma) \in \{0,1\}^\lambda$. *// The beacon value is output responsively as soon as $t+1$ valid tuples are received.*

Figure 5: Description of the randomness beacon protocol $\mathsf{GRandLine}$ from the view of party P_i.

signature on the epoch number $r \in \mathbb{N}$, obtain the full signature $\vartheta := e(g_r, SK)$ via Lagrange interpolation in the exponent from enough shares, and derive the randomness beacon value as $\mathsf{H}_2(\vartheta) \in \{0,1\}^\lambda$. However, the problem with a naive implementation of this approach is that no party can verify the correctness of a received share $\vartheta_i = e(g_r, h)^{f(i)}$. In order to resolve this issue, we augment the signature shares with additional elements from which its correctness can be checked via pairing equations. For this, we follow an economical two-step approach. After DKG setup, each party P_i locally samples an $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random and multicasts (i.e., sends to all parties) $\mathsf{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Correctness of its second component can be checked via a pairing equation. After this commitment phase, the actual randomness beacon starts. For epoch $r \geq 1$, each party computes $\sigma_i := (g_r^{\alpha_i}, \vartheta_i)$ along with a NIZK proof of discrete logarithm equality $\pi_i := \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ to prove correctness of $g_r^{\alpha_i}$. Upon receiving such a tuple, any party can verify the correctness of the partial signature ϑ_i using a pairing equation. Having done this, each party can compute the randomness beacon value for epoch r as described above. Overall, partial signatures consist of two group elements along with a simple proof of discrete logarithm equality. And verification of a share takes a single pairing equation with two pairing operations (as for the regular BLS signature!).

4.2. Security and Complexity Analysis of GRandLine in the AGM & ROM

In the following, we give a formal security analysis for our randomness beacon protocol $\mathsf{GRandLine}$. On an intuitive level, security follows from the uniqueness and unforgeability of our locally-verifiable threshold signature. In more detail, we give a security reduction from the hardness of n-COMDL to the unforgeability of the threshold signature. Further, we provide a complexity analysis of $\mathsf{GRandLine}$. For the full analysis, we refer to Appendix D.2.

Theorem 2. *If* n**-COMDL** *is* (ε_A, T_A)*-hard in the AGM and* BA *is* t*-secure, then* $\mathsf{GRandLine}$ *(cf. Figure 5) is a* $(t, \varepsilon, T, L, q_h, 1)$*-secure randomness beacon protocol in the AGM + ROM, where*

$$\varepsilon \leq Ln\left(12\varepsilon_A + (q_h^2 + q_h)/p\right), \quad T \geq T_A + \mathcal{O}(Ln^2).$$

$\mathsf{GRandLine}$ *has a communication complexity of* $\mathcal{O}(\lambda n^2)$ *bits per epoch, is responsive, and each epoch takes one round.*

5. Implementation & Evaluation

In this section, we evaluate the performance of our protocol at various system sizes. We evaluate $\mathsf{GRandLine}$'s throughput, i.e., the number of beacon values emitted per second, and compare it to existing state-of-the-art randomness beacons: OptRand [3], BRandPiper [1], Drand [11]. Although we developed our code to be agnostic to the choice of a pairing-friendly curve, we have used BLS12-381 for our instantiation. In particular, we have used the implementation of bls12-381 by arkworks [39] for primitive elliptic curve operations.

5.1. Implementation Details

We have implemented our prototype of GRandLine using the Rust programming language and the arkworks ecosystem [39]. We use a custom, optimized APVSS scheme implementation for our underlying cryptographic operations [40], ed25519-dalek [41] for constant-time signing of NIZK proofs and tokio [42] for networking. The implementation follows strictly the description in Figure 5 and it is publicly available for review at our Github repository [43].

Instantiation. We instantiate pairings with the BLS12-381 pairing-friendly family of elliptic curves. For efficiency, we use in our implementation $\mathbb{G}_1$ as the group for encrypted shares of the underlying APVSS scheme and $\mathbb{G}_2$ as the group for commitment shares. For our group generators $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$, we use fixed generators of unknown exponent. We simulate our protocol's setup phase by precomputing and using config files with the PVSS public keys and BLS12-381 public keys for Schnorr digital signatures.

5.2. Experimental Setup

We demonstrate the efficiency of GRandLine by evaluating our implementation with a varying total number of nodes, i.e., 4, 8, 16, 32, and 64. All experiments were conducted over Amazon EC2 where each replica was executed on a t3.medium instance. Each instance has 2 vCPUs, all cores sustained a Turbo CPU clock speed of up to 3.1GHz. The machines have up to 5 Gbps bandwidth, 4GB memory and run Ubuntu 22.04 LTS.

Network. To simulate execution over the Internet and to ensure comparability with other proposals, all of our experiments were conducted over Amazon EC2 where each replica was executed on a t3.medium instance across 8 regions: N. Virginia (us-east-1), Ohio (us-east-2), N. California (us-west-1), Oregon (us-west-2), Stockholm (eu-north-1), Frankfurt (eu-central-1), Tokyo (ap-northeast-1), Sydney (ap-southeast-2). For any choice of total number of nodes, we distribute the nodes evenly across all eight regions. For our runs with 4 nodes we used the following regions: N. Virginia (us-east-1), N. California (us-west-1), Frankfurt (eu-central-1), Tokyo (ap-northeast-1). In all cases, we create an overlay network among nodes where all nodes are pairwise connected in a complete graph.

Baselines. We compare the performance of our implementation to three state-of-art publicly available implementations: BRandPiper [44], Drand [11] and OptRand [45]. Our choice is motivated by the fact that all of these schemes are adaptively secure, have optimal resilience threshold $t < n/2$, and do not use heavy tools such as proof-of-work or (trapdoor) VDFs.

5.3. Evaluation Results

Each experiment is run 3 times and each data point represents the average. We report the throughput of GRandLine as the number of beacon outputs per second in Figure 6.

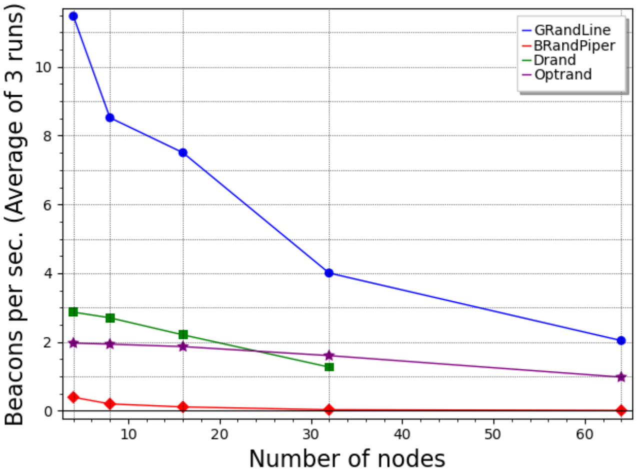

Figure 6: Performance of randomness beacon protocols: Drand, BRandPiper and OptRand in contrast to GRandLine.

GRandLine. Compared to all three baselines, GRandLine emits beacons at *significantly higher rates*. In particular, our evaluation results illustrate that with 4, 8, 16, 32, and 64 nodes, GRandLine on average can generate 11, 8, 7, 4, and 2 beacons per second, respectively. The average time between generating two consecutive beacons is: 87.19ms, 117.37ms, 133.29ms, 249.65ms, and 489.89ms, respectively.

OptRand. We test against OptRand's optimistically responsive variant. Moreover, we do not consider OptRand's non-optimistic variant, as it performs comparably to BRandPiper (cf. [3]). OptRand is leader-based, has many rounds of communication per epoch, and the epoch leader has to carry most of the computation which leads to a bottleneck for higher values of n. This results in significantly lower throughput per second across the board. Moreover, OptRand's reference implementation is instantiated with the BN128 pairing-friendly curve from libff [46]. Unfortunately, while this curve allows OptRand to benefit from more efficient modular operations, it is below acceptable security standards [47]. By instantiating GRandLine with arkworks' ark-bn254 crate [48], we estimate that our protocol is capable of generating roughly twice the number of beacons compared to what is shown in Figure 6. Our experiments for OptRand yield slightly worse performance than what is reported in their paper [3], but this could be attributed to running tests on different AWS regions.

BRandPiper. BRandPiper assumes a synchronous network. Their throughput is tied to a monotonically non-decreasing function $F(\Delta)$ of the maximum estimated network delay parameter Δ. A higher estimate for Δ leads to increased security but harms performance and vice versa. For BRandPiper, we first look for the smallest value for Δ that does not break their implementation and then measure throughput with this value for the delay parameter. Like OptRand's non-optimistic, synchronous variant, BRandPiper outputs beacons every 11Δ. BRandPiper suffers from increased overheads incurred by both synchronization as well as cryptographic operations due to its round-robin leader election nature. In particular, our results illustrated in Figure 6 confirm that BRandPiper's throughput is severely limited by the presence of a single slow node in the system.

Drand. Similarly, Drand assumes a synchronous network but relies on a *period parameter*. In each period, a single beacon is emitted and the actual deployment of Drand sets this parameter to 30 seconds. As such, Drand is also limited by the speed of a conservatively chosen estimated network delay. We note that we were only able to evaluate Drand's throughput for up to 32 nodes, as in our experiments, Drand's DKG initialization step keeps failing for 64 or more nodes, even for large estimates of the network delay. Similarly to BRandPiper, we took our time measurements after the initial setup was done because there is a small startup delay for setting up the network.

Acknowledgement

We would like to thank Adithya Bhat, Nibesh Shrestha, and Kartik Nayak for helpful discussions on the implementation of OptRand. We would also like to thank Zubayr Khalid for helping with our experiments. This work is funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) – 507237585, and by the European Union, ERC-2023-STG, Project ID: 101116713. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

[1] A. Bhat, N. Shrestha, Z. Luo, A. Kate, and K. Nayak, "RandPiper - reconfiguration-friendly random beacons with quadratic communication," in *ACM CCS 2021*, G. Vigna and E. Shi, Eds. ACM Press, Nov. 2021, pp. 3502–3524.

[2] S. Das, V. Krishnan, I. M. Isaac, and L. Ren, "Spurt: Scalable distributed randomness beacon with transparent setup," in *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2022, pp. 2502–2517.

[3] A. Bhat, N. Shrestha, A. Kate, and K. Nayak, "Optrand: Optimistically responsive reconfigurable distributed randomness," *Proceedings 2023 Network and Distributed System Security Symposium*, 2023. [Online]. Available: https://api.semanticscholar.org/CorpusID:257499606

[4] M. Yin, D. Malkhi, M. K. Reiter, G. Golan-Gueta, and I. Abraham, "HotStuff: BFT consensus with linearity and responsiveness," in *38th ACM PODC*, P. Robinson and F. Ellen, Eds. ACM, Jul. / Aug. 2019, pp. 347–356.

[5] T. Hanke, M. Movahedi, and D. Williams, "Dfinity technology overview series, consensus system," 2018.

[6] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, ser. SOSP '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 51–68. [Online]. Available: https://doi.org/10.1145/3132747.3132757

[7] R. Dingledine, N. Mathewson, and P. F. Syverson, "Tor: The second-generation onion router," in *USENIX Security 2004*, M. Blaze, Ed. USENIX Association, Aug. 2004, pp. 303–320.

[8] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan, and N. Christin, "An empirical analysis of traceability in the monero blockchain," 2018.

[9] K. Choi, A. Manoj, and J. Bonneau, "Sok: Distributed randomness beacons," in *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2023, pp. 75–92. [Online]. Available: https://doi.org/10.1109/SP46215.2023.10179419

[10] A. Kavousi, Z. Wang, and P. Jovanovic, "Sok: Public randomness," Cryptology ePrint Archive, Paper 2023/1121, 2023, https://eprint.iacr.org/2023/1121. [Online]. Available: https://eprint.iacr.org/2023/1121

[11] "Drand - a distributed randomness beacon," 2020. [Online]. Available: https://github.com/drand/drand

[12] E. Syta, P. Jovanovic, E. Kokoris-Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2017, pp. 444–460.

[13] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. R. Weippl, "RandRunner: Distributed randomness from trapdoor VDFs with strong uniqueness," in *NDSS 2021*. The Internet Society, Feb. 2021.

[14] R. Han, J. Yu, and H. Lin, "RandChain: Decentralised randomness beacon from sequential proof-of-work," Cryptology ePrint Archive, Report 2020/1033, 2020, https://eprint.iacr.org/2020/1033.

[15] C. Cachin, K. Kursawe, and V. Shoup, "Random oracles in Constantinople: Practical asynchronous byzantine agreement using cryptography," *Journal of Cryptology*, vol. 18, no. 3, pp. 219–246, Jul. 2005.

[16] A. Cherniaeva, I. Shirobokov, and O. Shlomovits, "Homomorphic encryption random beacon," Cryptology ePrint Archive, Report 2019/1320, 2019, https://eprint.iacr.org/2019/1320.

[17] P. Schindler, A. Judmayer, N. Stifter, and E. R. Weippl, "HydRand: Efficient continuous distributed randomness," in *2020 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2020, pp. 73–89.

[18] R. Bacho and J. Loss, "Adaptively secure (aggregatable) pvss and application to distributed randomness beacons," in *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 1791–1804. [Online]. Available: https://doi.org/10.1145/3576915.3623106

[19] ——, "On the adaptive security of the threshold BLS signature scheme," in *ACM CCS 2022*, H. Yin, A. Stavrou, C. Cremers, and E. Shi, Eds. ACM Press, Nov. 2022, pp. 193–207.

[20] V. Nikolaenko, S. Ragsdale, J. Bonneau, and D. Boneh, "Powers-of-tau to the people: Decentralizing setup ceremonies," Cryptology ePrint Archive, Report 2022/1592, 2022, https://eprint.iacr.org/2022/1592.

[21] R. Canetti, R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Adaptive security for threshold cryptosystems," in *CRYPTO'99*, ser. LNCS, M. J. Wiener, Ed., vol. 1666. Springer, Heidelberg, Aug. 1999, pp. 98–115.

[22] N. Shrestha, A. Bhat, A. Kate, and K. Nayak, "Synchronous distributed key generation without broadcasts," Cryptology ePrint Archive, Report 2021/1635, 2021, https://eprint.iacr.org/2021/1635.

[23] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, and G. Stern, "Bingo: Adaptivity and asynchrony in verifiable secret sharing and distributed key generation," in *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2023, p. 39–70. [Online]. Available: https://doi.org/10.1007/978-3-031-38557-5_2

[24] N. Alhaddad, M. Varia, and H. Zhang, "High-threshold AVSS with optimal communication complexity," in *FC 2021, Part II*, ser. LNCS, N. Borisov and C. Díaz, Eds., vol. 12675. Springer, Heidelberg, Mar. 2021, pp. 479–498.

[25] C. Gentry, S. Halevi, B. Magri, J. B. Nielsen, and S. Yakoubov, "Random-index PIR and applications," in *TCC 2021, Part III*, ser. LNCS, K. Nissim and B. Waters, Eds., vol. 13044. Springer, Heidelberg, Nov. 2021, pp. 32–61.

[26] M. Stadler, "Publicly verifiable secret sharing," in *EUROCRYPT'96*, ser. LNCS, U. M. Maurer, Ed., vol. 1070. Springer, Heidelberg, May 1996, pp. 190–199.

[27] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu, "Aggregatable distributed key generation," in *EUROCRYPT 2021, Part I*, ser. LNCS, A. Canteaut and F.-X. Standaert, Eds., vol. 12696. Springer, Heidelberg, Oct. 2021, pp. 147–176.

[28] P. Berman, J. A. Garay, and K. J. Perry, "Towards optimal distributed consensus (extended abstract)," in *30th FOCS*. IEEE Computer Society Press, Oct. / Nov. 1989, pp. 410–415.

[29] C. Lenzen and S. Sheikholeslami, "A recursive early-stopping phase king protocol," in *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing*, ser. PODC'22. New York, NY, USA: Association for Computing Machinery, 2022, p. 60–69. [Online]. Available: https://doi.org/10.1145/3519270.3538425

[30] K. Nayak, L. Ren, E. Shi, N. H. Vaidya, and Z. Xiang, "Improved Extension Protocols for Byzantine Broadcast and Agreement," in *34th International Symposium on Distributed Computing (DISC 2020)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), H. Attiya, Ed., vol. 179. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020, pp. 28:1–28:17. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2020/13106

[31] A. Boldyreva, "Threshold signatures, multisignatures and blind signatures based on the gap-Diffie-Hellman-group signature scheme," in *PKC 2003*, ser. LNCS, Y. Desmedt, Ed., vol. 2567. Springer, Heidelberg, Jan. 2003, pp. 31–46.

[32] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Transactions on Information Theory*, vol. 29, no. 2, pp. 198–208, 1983.

[33] G. Fuchsbauer, E. Kiltz, and J. Loss, "The algebraic group model and its applications," in *CRYPTO 2018, Part II*, ser. LNCS, H. Shacham and A. Boldyreva, Eds., vol. 10992. Springer, Heidelberg, Aug. 2018, pp. 33–62.

[34] V. Shoup, "Lower bounds for discrete logarithms and related problems," in *EUROCRYPT'97*, ser. LNCS, W. Fumy, Ed., vol. 1233. Springer, Heidelberg, May 1997, pp. 256–266.

[35] D. Boneh, B. Bünz, and B. Fisch, "Batching techniques for accumulators with applications to IOPs and stateless blockchains," in *CRYPTO 2019, Part I*, ser. LNCS, A. Boldyreva and D. Micciancio, Eds., vol. 11692. Springer, Heidelberg, Aug. 2019, pp. 561–586.

[36] A. Momose and L. Ren, "Optimal Communication Complexity of Authenticated Byzantine Agreement," in *35th International Symposium on Distributed Computing (DISC 2021)*, ser. Leibniz International Proceedings in Informatics (LIPIcs), S. Gilbert, Ed., vol. 209. Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2021, pp. 32:1–32:16. [Online]. Available: https://drops.dagstuhl.de/opus/volltexte/2021/14834

[37] "Halo2 - zero-knowledge succinct non-interactive argument of knowledge (zk-snark)," 2023. [Online]. Available: https://github.com/scroll-tech/halo2-snark-aggregator

[38] B. Bünz, B. Fisch, and A. Szepieniec, "Transparent SNARKs from DARK compilers," in *EUROCRYPT 2020, Part I*, ser. LNCS, A. Canteaut and Y. Ishai, Eds., vol. 12105. Springer, Heidelberg, May 2020, pp. 677–706.

[39] "arkworks - a rust ecosystem for zksnark programming." 2022. [Online]. Available: https://github.com/arkworks-rs

[40] "Cryptography for grandline," Github, 2023. [Online]. Available: https://github.com/AnonymousUser-2023/Optrand-APVSS

[41] "ed25519 signing and verification in rust," GitHub, 2022. [Online]. Available: https://github.com/dalek-cryptography/ed25519-dalek

[42] "Tokio library for networking." 2023. [Online]. Available: https://tokio.rs/

[43] "Implementation of grandline," Github, 2023. [Online]. Available: https://github.com/AnonymousUser-2023/GRandLine_reference

[44] Z. Luo, "Implementation for randpiper," Github, 2022. [Online]. Available: https://github.com/zhtluo/randpiper-rs

[45] N. Shrestha, "Implementation for optrand," Github, 2022. [Online]. Available: https://github.com/nibeshrestha/optrand/tree/crypto_dev

[46] "C++ library for finite fields and elliptic curves," Github, 2021. [Online]. Available: https://github.com/scipr-lab/libff

[47] Y. Sakemi, T. Kobayashi, T. Saito, and R. S. Wahby, "Internet research task force (irtf) draft for pairing-friendly curves," Nov. 2022. [Online]. Available: https://datatracker.ietf.org/doc/draft-irtf-cfrg-pairing-friendly-curves/

[48] "Library implementation for the bn254 curve," docs.rs, 2023. [Online]. Available: https://docs.rs/ark-bn254/0.4.0/ark_bn254/index.html

[49] I. Abraham, D. Malkhi, K. Nayak, and L. Ren, "Dfinity consensus, explored," Cryptology ePrint Archive, Report 2018/1153, 2018, https://eprint.iacr.org/2018/1153.

[50] J. Groth, "Non-interactive distributed key generation and key resharing," Cryptology ePrint Archive, Report 2021/339, 2021, https://eprint.iacr.org/2021/339.

[51] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," *Journal of Cryptology*, vol. 20, no. 1, pp. 51–83, Jan. 2007.

[52] D. R. Stinson and R. Strobl, "Provably secure distributed Schnorr signatures and a (t, n) threshold scheme for implicit certificates," in *ACISP 01*, ser. LNCS, V. Varadharajan and Y. Mu, Eds., vol. 2119. Springer, Heidelberg, Jul. 2001, pp. 417–434.

[53] I. Cascudo and B. David, "SCRAPE: Scalable randomness attested by public entities," in *ACNS 17*, ser. LNCS, D. Gollmann, A. Miyaji, and H. Kikuchi, Eds., vol. 10355. Springer, Heidelberg, Jul. 2017, pp. 537–556.

[54] B. David, P. Gazi, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *EUROCRYPT 2018, Part II*, ser. LNCS, J. B. Nielsen and V. Rijmen, Eds., vol. 10821. Springer, Heidelberg, Apr. / May 2018, pp. 66–98.

[55] A. Bandarupalli, A. Bhat, S. Bagchi, A. Kate, and M. Reiter, "Hashrand: Efficient asynchronous random beacon without threshold cryptographic setup," Cryptology ePrint Archive, Paper 2023/1755, 2023, https://eprint.iacr.org/2023/1755. [Online]. Available: https://eprint.iacr.org/2023/1755

[56] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *CRYPTO 2018, Part I*, ser. LNCS, H. Shacham and A. Boldyreva, Eds., vol. 10991. Springer, Heidelberg, Aug. 2018, pp. 757–788.

[57] J. Drake, "Minimal vdf randomness beacon," 2018. [Online]. Available: https://ethresear.ch/t/minimal-vdf-randomness-beacon/3566

[58] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[59] I. Cascudo and B. David, "ALBATROSS: Publicly AttestabLe BATched Randomness based On Secret Sharing," in *ASIACRYPT 2020, Part III*, ser. LNCS, S. Moriai and H. Wang, Eds., vol. 12493. Springer, Heidelberg, Dec. 2020, pp. 311–341.

[60] K. Choi, A. Arun, N. Tyagi, and J. Bonneau, "Bicorn: An optimistically efficient distributed randomness beacon," Cryptology ePrint Archive, Report 2023/221, 2023, https://eprint.iacr.org/2023/221.

[61] B. Bünz, S. Goldfeder, and J. Bonneau, "Proofs-of-delay and randomness beacons in ethereum," 2017.

[62] S. Jarecki and A. Lysyanskaya, "Adaptively secure threshold cryptography: Introducing concurrency, removing erasures," in *EUROCRYPT 2000*, ser. LNCS, B. Preneel, Ed., vol. 1807. Springer, Heidelberg, May 2000, pp. 221–242.

[63] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "ETHDKG: Distributed key generation with Ethereum smart contracts," Cryptology ePrint Archive, Report 2019/985, 2019, https://eprint.iacr.org/2019/985.

[64] E. Kokoris-Kogias, D. Malkhi, and A. Spiegelman, "Asynchronous distributed key generation for computationally-secure randomness, consensus, and threshold signatures," in *ACM CCS 2020*, J. Ligatti, X. Ou, J. Katz, and G. Vigna, Eds. ACM Press, Nov. 2020, pp. 1751–1767.

[65] I. Abraham, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu, "Reaching consensus for asynchronous distributed key generation," in *40th ACM Symposium Annual on Principles of Distributed Computing*. Association for Computing Machinery, Portland, OR, USA, 2021, pp. 363–373.

[66] S. Das, Z. Xiang, L. Kokoris-Kogias, and L. Ren, "Practical asynchronous high-threshold distributed key generation and distributed polynomial sampling," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 5359–5376. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/das

[67] S. Das, T. Yurek, Z. Xiang, A. K. Miller, L. Kokoris-Kogias, and L. Ren, "Practical asynchronous distributed key generation," in *2022 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, May 2022, pp. 2518–2534.

[68] J. Groth, "On the size of pairing-based non-interactive arguments," in *EUROCRYPT 2016, Part II*, ser. LNCS, M. Fischlin and J.-S. Coron, Eds., vol. 9666. Springer, Heidelberg, May 2016, pp. 305–326.

[69] S. Das, P. Camacho, Z. Xiang, J. Nieto, B. Bunz, and L. Ren, "Threshold signatures from inner product argument: Succinct, weighted, and multi-threshold," Cryptology ePrint Archive, Paper 2023/598, 2023, https://eprint.iacr.org/2023/598. [Online]. Available: https://eprint.iacr.org/2023/598

Appendix A. Related Work

In this section, we provide a detailed discussion on existing works for randomness beacon and distributed key generation protocols. For an excellent discussion of existing randomness beacons, we refer to [9], [10].

Randomness Beacons. We categorize randomness beacons found in the literature and practical applications according to their assumptions and reliance on cryptographic tools.
Threshold Secret Sharing. The protocol of this type employ threshold cryptography in order to generate randomness. For this, there are two approaches. In the first one [11], [15], [12], [49], [16], parties generate a (t, n)-threshold key $(sk_1, \dots, sk_n)$ by running a DKG protocol from which the randomness beacon value is derived as a unique threshold signature on some message (typically the hash of the current epoch number). In more detail, there are the following protocols. Cachin et al. [15] works in asynchrony, but does not specify the threshold signature nor the DKG protocol. Dfinity works in partial synchrony and uses the threshold BLS signature together with the non-interactive DKG protocol of Groth [50] (it assumes a broadcast channel). Drand [11] works in synchrony and uses threshold BLS together with Gennaro et al.'s DKG [51]. Herb works in synchrony and uses the threshold ElGamal signature, and RandHerd works in asynchrony and uses the threshold Schnorr signature [52]. The setup phase of these protocols have a communication complexity of $\mathcal{O}(\lambda n^3)$ or higher due to the use of a DKG protocol for field elements. On the other hand, once the setup phase has terminated, these protocols achieve an improved communication complexity of $\mathcal{O}(\lambda n^2)$ per beacon output within optimal one round. The second approach works through (P)VSS [12], [14], [13], [1], [3], [17], [2], [53], [54]. Notable randomness beacons here are SPURT [2], BRandPiper [1], and OptRand [3]. The idea of this approach is to generate a new random value at each epoch by combining secret sharings from at least $t + 1$ parties. This ensures that the combined secret has contribution from at least one honest party that chose its secret uniformly at random so the randomness beacon value also inherits that property. Let us elaborate in more detail on some of the protocols. SPURT works in partial synchrony and has a communication complexity of $\mathcal{O}(\lambda n^2)$ bits per beacon output. It relies on a pairing-based PVSS scheme. BRandPiper is a protocol from the family of RandPiper protocols. It works in synchrony and achieves a communication complexity of $\mathcal{O}(\lambda f n^2)$ bits per beacon output, where $f \leq t$ is the actual number of faults in the system. It relies on an efficient VSS scheme and the RandPiper SMR protocol. In BRandPiper, the leader of an epoch shares n secrets at once and for the beacon output parties reconstruct a random value accumulating secrets from $t + 1$ different (previous) leader parties. Most of these protocols assume a setup phase that when actually implemented incurs cubic or higher communication cost. Further, protocols of this type have a computation-heavy epoch where most of the computation is carried by one single party (epoch leader). Some protocols here have a communication complexity of $\mathcal{O}(\lambda n^2)$ per epoch, while others have cubic or higher. Finally, HashRand [55] is a recent randomness beacon in asynchrony that achieves adaptive security without the use of a threshold cryptographic setup. Their randomness beacon is based on a (small) committee selection from which the secret shares of a AVSS scheme are reconstructed. However, their technique relies on secure erasures of secret states (which are hard to implement in practice) and without that their protocol has a communication complexity of $\mathcal{O}(\lambda n^3 \log n)$ per epoch. Further, the technique of committee-selection is only meaningful when n is much larger than λ. One advantage of their protocol is that it provides post-quantum security.

Specialized Tools. The protocols in this category employ verifiable delay functions (VDFs) [56], [57] or Proof-of-Work (PoW) [58] in order to generate randomness [14], [59], [13], [60], [61], [57]. VDFs are functions that require a certain amount of time to compute but can be verified quickly. Solana uses VDFs in its Proof-of-History consensus protocol to establish a global source of time and generate random values. While VDF-based protocols can offer strong security guarantees and quadratic communication complexity, they require specialized hardware to compute the VDFs efficiently, which might not be accessible to all participants. The same applies to PoW-based protocols that rely on the assumption that the adversary has less computational hash power than the honest parties. In general, these primitives are computationally expensive tools with specialized hardware and are highly energy-consuming. We elaborate on some of these protocols. RandRunner [13] works in synchrony and uses a trapdoor VDF. Such a trapdoor VDF can generate unique function values efficiently with the knowledge of the trapdoor, but takes some high specified time T otherwise. RandRunner achieves a communication complexity of $\mathcal{O}(\lambda n^2)$ bits per beacon output. However, it only achieves $(t+1)$-unpredictability, since an adaptive adversary can simply corrupt the next t leaders and thus learn the beacon values for the next t epochs. RandChain [14] uses a combination of PoW, VFD, and Nakamoto Consensus, and achieves a communication complexity of $\mathcal{O}(\lambda n)$ bits per beacon output. One crucial drawback is that the beacon output is only guaranteed to be $1/5$-fair. And it also suffers from problems concerning blockchain-oriented attacks.

Distributed Key Generation. Most of the DKG protocols found in the literature are in synchrony [21], [50], [27], [62], [63], [22]. Among these protocols, only the ones of Canetti et al. [21] and Jarecki and Lysyanskaya [62] provide adaptive security, but at the cost of heavy tools such as non-committing encryption or reliable erasure of secret states. All of these synchronous protocols with the exception of Shrestha et al. [22] assume the existence of an one-round broadcast channel. When instantiating the broadcast channel with the adaptively secure Byzantine broadcast protocol in our work (Byzantine agreement and Byzantine broadcast are equivalent in the honest majority setting), all these protocols have cubic or higher communication complexity. Crucially, none of them achieves subcubic communication complexity. DKG protocols in asynchrony have only gained attention very recently by the works in [64], [23], [65], [66], [67]. All these constructions have cubic or higher communication complexity, and the one of Abraham et al. [23] relies on a universal powers-of-tau setup in order to obtain termination in expected constant rounds.

Appendix B. Formal Definitions and Security Notions

In this section, we provide formal definitions and security notions for additional primitives used in the main body. This is an extension of the primitives defined in Section 2.

B.1. Cryptographic Primitives

Digital Signature Scheme. A digital signature scheme provides a user with a verification-signing key pair (vk, dk), where the signing key is only known to the user but the verification key is public. The signing key allows the user to sign any message of its choice, while any third party that knows vk can verify that the message was indeed signed by that particular user. We formally define this as follows.

Definition 10 (Digital Signature Scheme)**.** *A digital signature scheme is a tuple of algorithms* $\mathsf{DS} = (\mathsf{SKey}, \mathsf{Sign}, \mathsf{Ver})$ *with the following properties:*

- SKey*: The randomized* key generation algorithm *takes as input system parameters par and an identity index $i \in [n]$. It outputs a verification key vk_i and a signing key dk_i.*
- Sign*: The possibly randomized* signing algorithm *takes as input a signing key dk_i and a message m. It outputs a signature σ. We also write $\langle m \rangle_i$ to denote the message-signature pair (m, σ) where $\sigma \leftarrow$* $\mathsf{Sign}(dk_i, m)$*.*
- Ver*: The deterministic* verification algorithm *takes as input a verification key vk_i, a message m, and a signature σ. It outputs 1 (accept) or 0 (reject). In the first case we call the signature σ* valid *(relative to vk_i); otherwise we call it* invalid*.*

For a secure digital signature scheme, we require that an adversary (not knowing the signing key) cannot create a signature on a new message, even after obtaining many signatures on messages of its choice.

Definition 11 (Unforgeability Under Chosen Message Attack)**.** *Let* $\mathsf{DS} = (\mathsf{SKey}, \mathsf{Sign}, \mathsf{Ver})$ *be a digital signature scheme. For an algorithm* A*, define the unforgeability under chosen message experiment* $\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\mathsf{DS}}$ *as follows:*

- Offline Phase. *Run* SKey *on input (par, i) to obtain a key pair (vk_i, dk_i). Run* A *on input (par, vk_i). Initialize $\mathcal{M} := \emptyset$.*
- Signing Oracle Queries. *At any point of the experiment,* A *gets access to an oracle that answer queries of the following type: When* A *submits a message m, return $\sigma \leftarrow$* $\mathsf{Sign}(dk_i, m)$ *and update $\mathcal{M} := \mathcal{M} \cup \{m\}$.*
- Output Determination. *When* A *outputs a message m^* and a signature σ^*, do the following. If* $\mathsf{Ver}(vk_i, m^*, \sigma^*) = 1$ *and $m^* \notin \mathcal{M}$, return 1. Otherwise, return 0.*

We say that DS *is (ε, T, q_s)-*unforgeable under chosen message attacks (UF-CMA) *if for all algorithms* A *that run in time at most T and make at most q_s signing oracle queries,* $\Pr[\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\mathsf{DS}} = 1] \leq \varepsilon$*. Conversely, we say that* A (ε, T, q_s)-breaks unforgeability *of* DS *under chosen message attacks if it runs in time at most T, makes at most q_s signing oracle queries, and* $\Pr[\mathbf{UF\text{-}CMA}^{\mathsf{A}}_{\mathsf{DS}} = 1] > \varepsilon$*.*

Cryptographic Accumulator. We continue with the notion of a collision-resistant accumulator scheme, which is the notion of security we require for an accumulator scheme.

Table 2: Comparison table of existing distributed key generation protocols.

Protocol	Network	Resil.	Adapt.	Commun.	Rounds	Field	Crypto. Prim.	Setup
Kokoris et al. [64]	*async*	1/3	✓	$\mathcal{O}(\lambda n^4)$	$\mathcal{O}(n)$	✓	AVSS	*CRS*
Abraham et al. [65]	*async*	1/3	✗	$\mathcal{O}(\lambda n^3 \log n)$	$\mathcal{O}(1)$	✗	PVSS, Pairing	*CRS*
Bingo [23]	*async*	1/3	✓	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(1)$	✓	AVSS, Pairing	*q-SDH, AGM*
Das et al. [67], [66]	*async*	1/3	✗	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(\log n)$	✓	AVSS	*CRS*
Shrestha et al. [22]	*sync*	1/2	✗	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(n)$	✓	VSS	*q-SDH*
Gennaro et al. [51]	*sync*	1/2	✗	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(n)$	✓	VSS	*CRS*
Gurkan et al. [27]	*sync*	$\log n$	✗	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(n)$	✗	PVSS, Pairing	*CRS*
Canetti et al. [21]	*sync*	1/2	✓	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(n)$	✓	VSS, Erasures	*CRS*
Jarecki et al. [21]	*sync*	1/2	✓	$\mathcal{O}(\lambda n^3)$	$\mathcal{O}(n)$	✓	PVSS, NC Enc.	*CRS*
GRand [our work]	*sync*	1/2	✓	$\mathcal{O}(\lambda n^2 \log n)$	$\mathcal{O}(n)$	✗	PVSS, Pairing, SNARK†	*CRS, AGM*

Resil. denotes the Byzantine resilience threshold. **Adapt.** denotes adaptive adversary. **Unpred.** denotes unpredictability. **Comm.** denotes the communication cost in bits. **Rounds** denotes the number of (a)synchronous rounds to terminate. For asynchronous protocols, this is the expected number of rounds (since these protocols are randomized). **Field** denotes if the secret key is a field element or not. **Crypto. Primit.** denotes the cryptographic primitives in usage. Canetti et al. relies on secure erasure of secret states, while Jarecki-Lysyanskaya uses non-committing encryption. **Setup** denotes the setup assumption. † GRand only requires SNARKs when n is much larger than λ.

Definition 12 (Collision-Resistant Accumulator)**.** *Let* $\Sigma = (\mathsf{Gen}, \mathsf{Eval}, \mathsf{Wit}, \mathsf{Ver})$ *be a cryptographic accumulator scheme. For set size* n *and an algorithm* A*, define the* collision-resistance *experiment* $\mathbf{CR}^{\mathsf{A}}_{\Sigma}(\lambda, n)$ *as follows:*

(1) Offline Phase. *Run the accumulator key generation algorithm to get* $ak \leftarrow \mathsf{Gen}(\lambda, n)$*.*

(2) Online Phase. *On input par and* (n, ak)*,* A *returns a tuple* $(\{d_1, \dots, d_n\}, d', w')$*.*

(3) Output Determination. *Compute the accumulation value* $z \leftarrow \mathsf{Eval}(ak, \{d_1, \dots, d_n\})$*. Return 1 if* $\mathsf{Ver}(ak, z, w', d') = 1$ *and* $d' \notin \{d_1, \dots, d'\}$*. Return 0 otherwise.*

We say that Σ *is collision-resistant or secure if for all PPT algorithms* A *and any* n*,* $\Pr[\mathbf{CR}^{\mathsf{A}}_{\Sigma}(\lambda, n) = 1] \leq \mathsf{negl}(\lambda)$*.*

Aggregatable PVSS Scheme. In the following, we define the correctness and security notions for an APVSS schemes. We start with correctness and public verifiability of (aggregated) transcripts as follows:

- *Correctness.* We say that APVSS is *correct* if for all key pairs $(pk_1, sk_1), \dots, (pk_n, sk_n) \in \mathsf{Keys}(par)$ and all $i \in [n]$,
$$\Pr[\mathsf{Ver}((pk_j)_{j \in [n]}, T) = 1 \wedge \mathsf{ConId}(pk_i, T) = 1] = 1,$$
where the probability is taken over all transcripts T output by $\mathsf{Dist}(sk_i, (pk_i)_{i \in [n]})$.
- *Public Verifiability.* We say that APVSS is *publicly verifiable* if for all key pairs $(pk_1, sk_1), \dots, (pk_n, sk_n) \in \mathsf{Keys}(par)$ and all $(\mathbf{E}, \pi)$ such that $\mathsf{Ver}((pk_1, \dots, pk_n), (\mathbf{E}, \pi)) = 1$, there exists a unique $S \in \hat{\mathbb{G}}$ such that
$$\mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\mathbf{E}_i)\}_{i \in \mathcal{I}}) = S \quad \forall \mathcal{I} \subset [n],\ |\mathcal{I}| = t + 1.$$

We would also like to guarantee that the secret reconstructed from an aggregated transcript $T = \mathsf{Agg}(T_1, \dots, T_k)$ corresponds to the sum of the secrets S_i that can be reconstructed from T_i. Additionally, we would like to ensure that the set of contributors to T consists of the contributors to the single transcripts T_i. These properties are captured in the following definitions.

Definition 13 (Correctness of Aggregation)**.** *Let* $\mathsf{APVSS} = (\mathsf{Keys}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Dist}, \mathsf{Agg}, \mathsf{ConId}, \mathsf{Ver}, \mathsf{Rec})$ *be a publicly verifiable APVSS scheme over* $\hat{\mathbb{G}}$*. We say that* APVSS *is* correctly aggregatable *if for all* $(pk_1, sk_1), \dots, (pk_n, sk_n) \in \mathsf{Keys}(par)$*, all* $k \in \mathbb{N}$*, and all transcripts* $(\mathbf{E}_1, \pi_1), \dots, (\mathbf{E}_k, \pi_k)$ *the following is true. If for all* $i \in [k]$*, we have* $\mathsf{Ver}((pk_1, \dots, pk_n), (\mathbf{E}_i, \pi_i)) = 1$*, then for all* $\mathcal{I} \subset [n]$*,* $|\mathcal{I}| = t+1$*, the (aggregated) transcript* $(\mathbf{E}', \pi') := \mathsf{Agg}((\mathbf{E}_1, \pi_1), \dots, (\mathbf{E}_k, \pi_k))$ *satisfies*
$$\mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\mathbf{E}'_i)\}_{i \in \mathcal{I}}) = \prod_{j \in [k]} \mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\mathbf{E}_{j,i})\}_{i \in \mathcal{I}}),$$
where we write $\mathbf{E}_j = (\mathbf{E}_{j,1}, \dots, \mathbf{E}_{j,n})$*. Additionally, we require that* $\mathsf{ConId}(pk_i, T) = 1$ *for an* $i \in [n]$ *if and only if there is an* $j \in [k]$ *such that* $\mathsf{ConId}(pk_i, T_j) = 1$*.*

We recall the newly introduced security notion for APVSS schemes called *aggregated unpredictability* as defined in [18]. This notion captures malleability attacks and prohibits any *t-bounded* (i.e., corrupting at most t parties) adversary from learning the secret of an aggregated transcript that has contribution from at least one honest party (even if the adversary is allowed to contribute to the aggregation itself).

Definition 14 (Aggregated Unpredictability of APVSS Scheme)**.** *Let* $\mathsf{APVSS} = (\mathsf{Keys}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Dist}, \mathsf{Agg}, \mathsf{ConId}, \mathsf{Ver}, \mathsf{Rec})$ *be a publicly verifiable APVSS scheme over* $\hat{\mathbb{G}}$*. For an algorithm* A*, we define the* aggregated unpredictability *experiment* $\mathbf{AggPred}^{\mathsf{A}}_{\mathsf{APVSS},t}$ *as follows:*

- Offline Phase. *Initialize* $\mathcal{T} := \emptyset$*. For all* $i \in [n]$*, run* Keys *on input* (par, i) *to generate keys* $(pk_i, sk_i) \leftarrow \mathsf{Keys}(par, i)$*. On input par and* $\{pk_i\}_{i \in [n]}$*,* A *returns an index set* $\mathcal{C} \subset [n]$ *of initially corrupted parties along with updated public keys* $\{\hat{pk}_i\}_{i \in \mathcal{C}}$*. Set* $pk_i := \hat{pk}_i$ *for all* $i \in \mathcal{C}$*.*

- Corruption Queries. *At any point of the experiment,* A *may corrupt a party by submitting an index* $i \in [n] \setminus \mathcal{C}$*. In this case, return the secret key* sk_i *and set* $\mathcal{C} := \mathcal{C} \cup \{i\}$.
- Transcript Queries. *At any point of the experiment,* A *gets access to an oracle of the following type: When* A *submits a request* $(\mathsf{givePVSS}, i)$ *for an* $i \in [n] \setminus \mathcal{C}$*, do the following. On behalf of dealer* P_i*, run* Dist *on input* sk_i *and* $pk_1, \ldots, pk_n$*. Return the output* $T = (\mathbf{E}, \pi)$ *and set* $\mathcal{T} := \mathcal{T} \cup \{(T, i)\}$.
- Output Determination. *When* A *outputs an aggregated transcript* $(\mathbf{E}', \pi')$ *and an element* $S^* \in \hat{\mathbb{G}}$*, proceed as follows:*
  - *Return 1 if* $|\mathcal{C}| \leq t$*,* $\mathsf{Ver}((pk_1, \ldots, pk_n), (\mathbf{E}', \pi')) = 1$*,* $S^* = \mathsf{Rec}(\{\mathsf{Dec}_{sk_i}(\mathbf{E}'_i)\}_{i \in [t+1]})$*, and there is an index* $i \in [n] \setminus \mathcal{C}$ *such that* $\mathsf{ConId}((\mathbf{E}', \pi'), pk_i) = 1$.
  - *Return 0 otherwise.*

We say that APVSS *is* (t, ε, T, q_s)-aggregated unpredictable *if for all algorithms* A *that run in time at most* T *and make at most* q_s *transcript queries,* $\Pr[\mathbf{AggPred}^{\mathsf{A}}_{\mathsf{APVSS},t} = 1] \leq \varepsilon$*. Conversely, we say that* A (t, ε, T, q_s)-breaks aggregated unpredictability *of* APVSS *if it runs in time at most* T*, makes* $\leq q_s$ *transcript queries, and* $\Pr[\mathbf{AggPred}^{\mathsf{A}}_{\mathsf{APVSS},t} = 1] > \varepsilon$.

Succinct Non-Interactive Argument of Knowledge. We continue with the notion of a succinct non-interactive argument of knowledge (SNARK) as defined by Groth [68]. A non-interactive argument enables a prover Prove to convince a verifier Ver that a statement x is in an NP language $\mathcal{L}$ such that no interaction is required. The prover outputs only one message, called an argument, which convinces the verifier of the truth of the statement. When the argument is of small size and low verification complexity, the system is called succinct. We formally define this as follows.

Definition 15 (Non-Interactive Argument System)**.** *Let* $\mathcal{R}$ *be a relation generator that on input the security parameter* λ *outputs an efficiently decidable binary relation* R*. For pairs* $(x, w) \in \mathcal{R}$*, we call* x *the statement and* w *the witness. The relation generator may also output some auxiliary information* z *which is given to the adversary. An efficient publicly verifiable non-interactive argument of knowledge for* R *is a tuple of probabilistic polynomial-time algorithms* $\Sigma = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver})$ *with the following properties:*

- Setup*: This is a randomized parameter generation algorithm that takes as input a polynomial-time decidable binary relation* R*. It outputs public parameters* par*. All other algorithms implicitly take* par *as input.*
- Prove*: This is a randomized prover algorithm that takes as input a polynomial-time decidable binary relation* R*, a statement* x*, and a witness* w*. It outputs an argument* π*.*
- Ver*: This is a deterministic verification algorithm that takes as input a statement* x*, and an argument* π*. It outputs 1 (accept) if the argument is valid and 0 (reject) otherwise.*

We continue with the standard security notions for a non-interactive argument system: perfect completeness, knowledge-soundness, and succinctness.

Definition 16 (Perfect Completeness)**.** *Let* $\Sigma = (\mathsf{Setup}, \mathsf{Prove}, \mathsf{Ver})$ *be a non-interactive argument system as defined above. We say that* Σ *has perfect completeness if for all* $(x, w) \in R$*,* $par \leftarrow \mathsf{Setup}(R)$*,* $\pi \leftarrow \mathsf{Prove}(R, par, x, w)$*, it holds* $\mathsf{Ver}(par, x, \pi) = 1$.

Definition 17 (Knowledge-Soundness)**.** *Let* Σ *be as above. We say that* Σ *has knowledge-soundness if the following holds. For any probabilistic polynomial-time (PPT) prover* A*, there exists a PPT extractor* $\mathcal{X}$ *such that when* A *outputs a valid argument* π *for a statement* x*, then* $\mathcal{X}$ *can compute a witness* w *with* $(x, w) \in R$*. Here, the extractor gets full access to the prover's state, including any random coins.*

Definition 18 (Succinctness)**.** *Let* Σ *be a knowledge-sound non-interactive argument system as defined above. We say that* Σ *is succinct if the proof size is linear in* λ.

B.2. Consensus Primitives

We begin with the notion of graded Byzantine agreement (also called *graded consensus*), which is a weak form of Byzantine agreement.

Graded Byzantine Agreement. A graded Byzantine agreement (GBA) protocols is a weak form of Byzantine agreement whose purpose is to test for pre-existing agreement among the honest parties. In a graded Byzantine agreement protocol each party outputs a value v with a grade bit g that reflects the party's confidence in its output value. We formally define a graded Byzantine agreement protocol as follows.

Definition 19 (Graded Byzantine Agreement)**.** *Let* Π *be a protocol executed by parties* $P_1, \ldots, P_n$*, where each party* P_i *begins holding an input* $v_i \in V$*. Each party* P_i *terminates upon outputting a tuple* (v, g) *where* $g \in \{0, 1\}$ *is a grade. We define the following security and correctness properties for* Π*:*

- Graded Validity. Π *is* t*-graded valid if the following holds whenever at most t parties are corrupted: if every honest party has the same value* v *as input, then every honest party outputs* $(v, 1)$*.*
- Graded Consistency. Π *is* t*-graded consistent if whenever at most t parties are corrupted: if an honest party outputs* $(v, 1)$*, then all honest parties output* $(v, \cdot)$*.*
- Termination. Π *is* t*-terminating if whenever at most* t *parties are corrupted: every honest party terminates with a graded output value* (v, g) *where* $v \in V$ *and* $g \in \{0, 1\}$*.*

If Π *is* t*-graded valid,* t*-graded consistent, and* t*-terminating, we say it is* t*-secure.*

Let $\mathcal{Q} \subseteq \mathcal{P}$ be a set of q parties among which the protocol is executed and let $b := \lfloor q/2 \rfloor + 1$. We denote by $\pi_b(m)$ a compact certificate on m with threshold b, i.e., a proof of knowledge of b signatures on m from different parties.

- **Echo Phase.** Initialize sets $W, C_1, C_2 := \varnothing$, grade $g := 0$, and variable $\mathsf{sent} := 0$. Send $\langle \mathsf{echo}, v_i \rangle_i$ to all parties.
- **Forward Phase.** Upon receiving b valid echo messages $\langle \mathsf{echo}, v \rangle_j$ on the same value v from different parties, update $W := W \cup \{(v, \pi_b(\mathsf{echo}, v))\}$. Once $W \neq \varnothing$, send $\langle v, \pi_b(v) \rangle_i \in W$ to all parties and update $\mathsf{sent} = 1$.
- **First Vote Phase.** If $\mathsf{sent} = 1$ and did not receive a valid tuple $(\tilde{v}, \pi_b(\tilde{v}))$ for a different value $\tilde{v} \neq v$, send $\langle \mathsf{vote}_1, v \rangle_i$ to all parties. Upon receiving b valid votes $\langle \mathsf{vote}_1, w \rangle_j$ on the same value w from different parties, update $C_1 := C_1 \cup \{(w, \pi_b(\mathsf{vote}_1, w))\}$.
- **Second Vote Phase.** Once $C_1 \neq \varnothing$, send $\langle \mathsf{vote}_2, w \rangle_i$ along with $\langle w, \pi_b(\mathsf{vote}_1, w) \rangle_i \in C_1$ to all parties.
- **Output Generation.** Upon receiving b valid votes $\langle \mathsf{vote}_2, u \rangle_j$ on the same value u from different parties, update $C_2 := C_2 \cup \{(u, \pi_b(\mathsf{vote}_2, u))\}$. If $C_1 \neq \varnothing$, set $v_i := \tilde{v} \in C_1$. If further $\tilde{v} \in C_2$, set $g := 1$. Terminate with (v_i, g).

Figure 7: Graded Byzantine agreement protocol GBA described from the view of party P_i on input $(\mathcal{Q}, v_i)$.

Let $\mathcal{Q} \subseteq \mathcal{P}$ be a set of q parties and let $b := \lfloor q/2 \rfloor + 1$. Partition $\mathcal{Q}$ into two disjoint subsets $\mathcal{Q} = \mathcal{Q}_1 \sqcup \mathcal{Q}_2$ (each called a *committee*) of size $q_1 := \lceil q/2 \rceil$ and $q_2 := \lfloor q/2 \rfloor$, respectively. Let $l \in \{1, 2\}$ be such that $P_i \in \mathcal{Q}_l$ and fix some small constant $const \in \mathbb{Z}_{\geq 1}$.

- **Graded Consensus.** Execute $\mathsf{GBA}(\mathcal{Q}, v_i)$ among all parties in $\mathcal{Q}$. Let (v_i, g_i) denote the output.
- **First Recursion.** If $l = 1$, execute $\mathsf{BA}(\mathcal{Q}_1, v_i)$ among parties in $\mathcal{Q}_1$. Let v denote the output. In case $|\mathcal{Q}_1| \leq const$, parties can execute any Byzantine agreement protocol to obtain v.
- **Proposal Phase.** If $l = 1$, send $\langle \mathsf{propose}, v \rangle_i$ to all parties in $\mathcal{Q}$. Upon receiving the same proposal value v from $\lfloor q_1/2 \rfloor + 1$ different parties in $\mathcal{Q}_1$ (a majority) and if $g_i = 0$, update $v_i = v$.
- **Graded Consensus.** Execute $\mathsf{GBA}(\mathcal{Q}, v_i)$ among all parties in $\mathcal{Q}$. Let (v_i, g_i) denote the output.
- **Second Recursion.** If $l = 2$, execute $\mathsf{BA}(\mathcal{Q}_2, v_i)$ among parties in $\mathcal{Q}_2$. Let v denote the output. In case $|\mathcal{Q}_2| \leq const$, parties can execute any Byzantine agreement protocol to obtain v.
- **Proposal Phase.** If $l = 2$, send $\langle \mathsf{propose}, v \rangle_i$ to all parties in $\mathcal{Q}$.
- **Output Generation.** Upon receiving the same proposal value v from $\lfloor q_2/2 \rfloor + 1$ different parties in $\mathcal{Q}_2$ (a majority) and if $g_i = 0$, update $v_i = v$. Terminate with output v_i.

Figure 8: Recursive Byzantine agreement protocol BA described from the view of party P_i on input v_i.

Appendix C. Byzantine Agreement Protocol

In this section, we explain how to modify the Byzantine agreement protocol from Momose and Ren [36]. Their protocol has a communication cost of $\mathcal{O}(\lambda n^2)$ bits and relies on threshold key setups. Concretely, the protocol recursively calls itself two times on each half and uses a threshold key setup for each half. The threshold keys are are used (at each level of the recursion) in order to provide a compact proof (i.e., a *compact certificate*) for the statement that the prover knows a threshold of signatures from different parties on the same message. We observe that a threshold key is a more powerful primitive and not needed for this functionality. Instead, we implement the compact certificates with succinct non-interactive arguments of knowledge (SNARKs) that do not require trusted setup [37], [38]. In particular, the communication cost of the Byzantine agreement protocol stays $\mathcal{O}(\lambda n^2)$. In the following, we will define the relation that specifies these SNARKs. For simplicity, we will define a relation for each level of the recursion. We note that one could implement the certificates also with the recent inner product argument based threshold signatures from Das et al. [69]. Their scheme requires a universal powers-of-tau setup and plain PKI. It generates constant-sized threshold signatures and supports arbitrary thresholds. Further, the authors provide an adaptive security proof of their scheme in the AGM. We emphasize that their threshold signature is not unique and thereby not suitable for randomness beacons.

Compact Certificates. In the following, we define the relation that specifies the SNARK used by parties in order for them to prove knowledge of a threshold of signatures on the same message they have received. We define the relation relative to a hash function Hash and a tuple of numbers (ℓ, i) where $\ell \in [\![\log n]\!], i \in [2^\ell]$ that specifies a set $\mathcal{Q}(\ell, i) \subseteq [n]$ of parties with public keys $pk_{i_1}, \ldots, pk_{i_q}$ where $q := |\mathcal{Q}(\ell, i)|$. A tuple (x, w) is in the relation $R(\ell, i)$ if the following holds:

- The statement x is of the form (r, z) where $r := \lfloor q/2 \rfloor + 1$ and z is a hash value.
- The witness w is of the form $(\mathcal{S}, m)$ where $\mathcal{S}$ is a list of r signatures and m is a message such that (i) for all $k \in [r]$, $\mathcal{S}(k)$ is a signature on m that verifies with respect to pk_{i_k}, and (ii) $z = \mathsf{Hash}(pk_{i_1}, \ldots, pk_{i_q})$.

In our Byzantine agreement protocol, the protocol will call itself on each half of the whole system of n parties $\{P_1, \ldots, P_n\}$. And this will define a particular set of parties $\mathcal{Q}(\ell, i)$ for each pair (ℓ, i) such that $\ell \in [\![\log n]\!]$ specifies the level of the recursion and $i \in [2^\ell]$ the i-th committee among all 2^ℓ committees at that level. That means for each such ℓ, we have a disjoint union $\mathcal{Q} = \bigsqcup_{i \le 2^\ell} \mathcal{Q}(\ell, i)$. The above relation then simply says that a prover has knowledge of a majority $\lfloor |\mathcal{Q}(\ell, i)|/2 \rfloor + 1$ of signatures from different parties in the committee $\mathcal{Q}(\ell, i)$ on the same message m.

Graded Consensus Protocol. A formal description of the graded consensus protocol GBA is given in Figure 7. The protocol is parameterized by a subset $\mathcal{Q} \subseteq \mathcal{P}$ of $q := |\mathcal{Q}|$ parties among which the protocol is executed. Each party P_i has an input value v_i and sets its grade bit $g_i := 0$. First, every party sends its signed value v_i to all parties in $\mathcal{Q}$. After receiving the same value v from $b := \lfloor q/2 \rfloor + 1$ distinct parties (if at all), a party builds an echo certificate using a (q, b)-threshold signature and sends it to all parties. Following this, a party votes for v via a signed (vote_1, v) message if it has received an echo certificate for v and for no other value. Parties repeat the last two steps applied to vote_1 messages (instead of echo messages) to prevent the formation of vote_1 certificates on different values. Once a party receives a vote_1 certificate on v, it sends the certificate along with a signed (vote_2, v) message to all parties. For output generation, a party sets $v_i := v$ if it received a vote_1 certificate on v and $g_i := 1$ if it additionally received b vote_2 messages on v from different parties.

Byzantine Agreement Protocol. The protocol recursively calls itself two times on each half $\mathcal{Q}_1$ and $\mathcal{Q}_2$. Here, the set of all parties $\mathcal{Q} = \{P_1, \ldots, P_n\}$ is partitioned into a disjoint union $\mathcal{Q}_1 \sqcup \mathcal{Q}_2$ of two *halves/committees* of roughly equal size. The idea of the protocol is that each half calls BA itself and then emulates a single party that proposes a value (each committee basically emulates a king as defined in the familiar Phase-King protocol). At the lowest level of the recursion when committees fall below a predefined size $const$, parties can run any Byzantine agreement protocol since efficiency is no concern for constant-sized committees. We briefly elaborate on the structure of the Phase-King protocol. First, all parties execute an instance of the graded consensus protocol GBA. Then, a designated party called king (in our protocol it is emulated by a committee), whose role is to establish agreement among honest parties in case they are split between different values, proposes its own value to all parties. Now a party adopts the king's proposed value only if its own grade bit is $g_i = 1$; the grade determines if a party sticks to its value output by the graded consensus or if it adopts the value proposed by the king. In our protocol, a party only considers a proposed value received from a majority of the committee's parties. A simple observation shows that honest parties reach agreement once an honest king proposes. Thus, since at least one of the two committees that we consider has an honest majority, the protocol terminates with all honest parties agreeing on the same value v.

Appendix D. Security and Complexity Analysis of DKG and Randomness Beacon

In this section, we provide a security and complexity analysis of our protocols, that includes the DKG protocol GRand (cf. Figure 4) which establishes threshold public-secret key pairs for parties (where both the public and the secret keys are group elements) and the subsequent randomness beacon protocol GRandLine (cf. Figure 5). In order to have a more thorough overview of the proofs, we proceed as follows. In the first subsection, we give a security analysis of GRand along with a communication and round complexity analysis. Here, the security analysis shows correctness and secrecy in the presence of an adversary that corrupts up to $t < n/2$ parties. For this, we show that at the end of the protocol each party holds a secret key share along with public key shares of all parties. Further, we show that the final aggregated transcript AT (from which the keys are derived locally) has contribution from at least one honest party. Having done this, it follows immediately that the DKG protocol is secret as long as the underlying APVSS scheme is aggregated unpredictable (the secrecy definition is in line with aggregated unpredictability of the APVSS). In the second subsection, we then provide a full security and complexity analysis of our randomness beacon GRandLine that has GRand as a pre-processing phase.

D.1. Analysis of our DKG GRand

Here, we provide a full proof of Theorem 1. We divide our proof as follows. First, we show that all honest parties terminate the recursive part GenAPVSS of the protocol with the same valid (aggregated) PVSS transcript AT that has contribution from at least one honest party. From this, the correctness properties of the DKG protocol directly follow, as the rest involves only local computation without any further interaction. Concretely, each party P_i derives its secret key share SK_i from the transcript $AT := \{C_j, E_j, \pi\}_{j \in [n]}$ as the decrypted share $SK_i = \mathsf{Dec}_{sk_i}(E_i)$ and the public key shares $(PK_1, \ldots, PK_n)$ are simply the commitments $(C_1, \ldots, C_n)$. Second, we provide a communication and round complexity of the recursive part GenAPVSS of the protocol. Concretely, we derive the communication and round complexity through recursive formulas that we establish from the analysis of a recursion level of GenAPVSS.

Proof. We begin with the proof. Before we delve into the analysis, we recall the protocol. To understand the recursive protocol better, we give an explanation of it that starts at the very beginning. For the subsequent discussion, we assume for the sake of presentation that the number n of parties in the system is a power of two $n = 2^k$. Recall the notation $\mathcal{P} = \{P_1, \ldots, P_n\}$ for the whole system of parties. At the beginning of the protocol, each party $P_i \in \mathcal{P}$ locally generates a random (t, n)-threshold PVSS transcript via $T_i \leftarrow \mathsf{Dist}(sk_i, (pk_1, \ldots, pk_n))$. Essentially, this transcript encodes a random degree-t polynomial $f_i \in \mathbb{Z}_p[X]$ in the exponent and comes with a signature (concretely, a signed NIZK proof θ of knowledge of $f_i(0)$) that bounds the transcript to party P_i). We split the whole system $\mathcal{P}$ of

n parties into disjoint sets of neighboring parties (that we call *committees*) as $\mathcal{Q}_i := \{P_{2i-1}, P_{2i}\}$ for all $i \in [n/2]$. Concretely, that means that

$$\mathcal{Q}_1 = \{P_1, P_2\},\ \mathcal{Q}_2 = \{P_3, P_4\},\ \ldots,\ \mathcal{Q}_{n/2} = \{P_{n-1}, P_n\}.$$

The goal now is for the parties in each committee to interact with each other (but only within its own committee) in order to exchange their (local) PVSS transcripts and finally aggregate them into a single (aggregated) transcript. For now, consider a single committee $\mathcal{Q}_i$ (each committee executes the same instructions). Each party first generates an accumulation value for its PVSS transcript and sends it to the other party in the same committee. Then both parties execute two instances of Byzantine agreement in order to have agreement on these two accumulation values that we label z_{2i-1} and z_{2i}. Having done this, each party sends its local PVSS transcript to the other party by usage of the efficient $\mathsf{Deliver}$ algorithm (since they have already agreed on the accumulation value for each transcript, this step makes sense at this point). Afterwards, parties in the committee execute two instances of binary Byzantine agreement in order to determine if the other party has delivered its PVSS transcript correctly. Finally, parties locally aggregate the two PVSS transcripts (in case everything was done correctly) and obtain a single (aggregated) transcript that both have. In this way, both parties have established a new PVSS transcript and can progress to the next stage of the iteration. This idea generalizes to higher levels $\ell \in [\log n]$ of the iteration also in which two (neighboring) committees $\mathcal{Q}_{2i-1}$ and $\mathcal{Q}_{2i}$ of size $q/2 = 2^{\ell-1}$ each interact with each other to exchange their previously generated PVSS transcripts from one level lower. To argue on the security of our protocol GRand, we will need to show that at the end of the recursive protocol $\mathsf{GenAPVSS}$ all honest parties have obtained the same (aggregated) PVSS transcript AT from which they then locally derive their own secret key shares and all public key shares of GRand.

Since we assume $t < n/2$ for our protocols, we know that at least one of the two committees $\mathcal{Q}_1$ and $\mathcal{Q}_2$ from the first step of the recursion is honest (i.e., has an honest majority of parties). Let us assume without loss of generality that the first committee $\mathcal{Q}_1$ is honest. Further, we assume that parties in $\mathcal{Q}_1$ have already generated a PVSS transcript T_1 and agreed on it. Once we show that after the interaction between parties in the *parent committee* $\mathcal{Q} := \mathcal{Q}_1 \cup \mathcal{Q}_2$ (which is the whole system of n parties in this case, but we will simply keep the general notation and label it by $\mathcal{Q}$, as the whole discussion then also applies to any set $\mathcal{Q}$ with honest majority) that all honest parties have obtained two transcripts $\{T_1, T_2\}$, where possibly $T_2 = \bot$ and T_i corresponds to committee $\mathcal{Q}_i$ for $i \in [2]$, then it will follow by simple induction (or an iterative argument) that the final transcript AT output by $\mathsf{GenAPVSS}$ is an aggregation of several single PVSS transcripts (generated by individual parties at the bottom level of the recursion) from which at least one was generated by an honest party. The reason is that in an honest committee $\mathcal{Q}_1$, there was at least one honest *child committee* that it was formed from and at the bottom parties sampled their transcripts locally. So let parties in committee $\mathcal{Q}_1$ have their transcript T_1 and for parties in committee $\mathcal{Q}_2$ we do not assume anything (since it could have majority corrupt parties and the adversary could let different honest parties in $\mathcal{Q}_2$ output anything). In the first step of the protocol, all parties in $\mathcal{Q}_1$ generate an accumulation value z_1 for a deterministic and predefined encoding of their transcript T_1 and then multicast it to all parties in the parent committee $\mathcal{Q}$. Since $\mathcal{Q}_1$ has honest majority and we assume that all messages are signed by a party before it sends it, all parties in $\mathcal{Q}_1$ will set their local accumulation buffer V_1 for committee $\mathcal{Q}_1$ to $V_1 := \{z_1\}$. For the accumulation buffer V_2 for committee $\mathcal{Q}_2$ we do not assume anything and all honest parties could have a different value set V_2 (as the majority corrupt parties in $\mathcal{Q}_2$ could send different accumulation values to different honest parties in $\mathcal{Q}$). Since there is possible disagreement on the received accumulation values V_1 and V_2 among the parties and honest parties do not know which of the committees is honest (if at all), two instances $\mathsf{BA}_1, \mathsf{BA}_2$ of the Byzantine agreement protocol BA are executed among the parties on input the received accumulation values (note that the input/output $\bot$ is possible). Since there is an honest majority of honest parties in the system $\mathcal{Q}_1 \cup \mathcal{Q}_2 = \mathcal{Q}$, the guarantees of the Byzantine agreement protocol tell us that all honest parties in $\mathcal{Q}$ output the same values $\tilde{z}_1$ and $\tilde{z}_2$ from the executions of $\mathsf{BA}_1, \mathsf{BA}_2$ (agreement) and further that $\tilde{z}_1 = z_1$ is the accumulation value for committee $\mathcal{Q}_1$ that all honest parties in $\mathcal{Q}$ did input into BA_1 (validity). To avoid cumbersome notation, we simply use the labels z_1 and z_2 for the output values. In the next step, each party in $\mathcal{Q}_1$ sends its transcript T_1 to all other parties in $\mathcal{Q}$ using the Deliver protocol. Since the deliver protocol at this stage uses Reed-Solomon codes with a reconstruction threshold of $q/2 + 1$ (majority threshold) and there are at least $q/2+1$ honest parties in $\mathcal{Q}$ (by assumption that $\mathcal{Q}$ has honest majority), all honest parties in the parent committee $\mathcal{Q}$ will be able to reconstruct the transcript T_1 after this step. On the other hand, we cannot say anything regarding a hypothetical transcript T_2 with accumulation value z_2 coming from the other committee $\mathcal{Q}_2$. It could be that corrupt parties in $\mathcal{Q}_2$ (which can form the majority in that committee) send a valid transcript T_2 to only some honest parties in $\mathcal{Q}$ or even none so that not all honest parties might be able to reconstruct the full message. Therefore, in the next step, parties execute two instances $2\mathsf{BA}_1, 2\mathsf{BA}_2$ of binary Byzantine agreement on input 1 if it was able to reconstruct a transcript T_i with accumulation value z_i and on input 0 otherwise. The security guarantees of Byzantine agreement now have the following implications. If the output for say B_i is 1, then there must have been at least one honest party that provided the input 1 into the protocol. In particular, that honest party was able to reconstruct a transcript T_i with accumulation value z_i from the previous step. If the output for B_i is 0, then we cannot say anything further (as it could be that some honest parties have input 0 and others 1). But since all honest parties in $\mathcal{Q}$ were able to reconstruct the full transcript T_1 coming from the (honest) committee $\mathcal{Q}_1$, we know that all honest parties input 1 into B_1 and therefore by validity of Byzantine agreement they all output 1. In the final step of the protocol, all parties that have a transcript T_i with accumulation value z_i send it to all other parties in $\mathcal{Q}$ using the Deliver protocol. If the output of B_i was 0, then parties simply ignore any transcript T_i delivered by any party. If the output of B_i was 1, then all honest parties will reconstruct T_i, as there was at least one honest party that invoked the deliver protocol on T_i by the properties of BA as already clarified. As B_1 has output 1, we know that all honest parties will have the same transcript T_1 at the end of this step. Further, if B_2 has also output 1, then likewise all honest parties will have the same transcript T_2 at the end of this step. If B_2 has output 0, then we know that all honest parties will simply ignore any transcript T_2 delivered by any party and thus all honest parties will agree on $T_2 = \varnothing$ simply. Having said all this, we have shown that in case the committee $\mathcal{Q}$ has honest majority, then all honest parties will end up with the same transcripts T_1 and T_2 (coming from the children committees $\mathcal{Q}_1$ and $\mathcal{Q}_2$) where at least one of them is a valid and true PVSS transcript $T_i \neq \varnothing$. As a result, after the local aggregation step of $\{T_i\}_{i\in[2]}$, all honest parties obtain the same transcript $AT := \mathsf{Agg}(T_1, T_2)$. This concludes our discussion on our initial goal to show that all honest parties terminate GenAPVSS with the same aggregated transcript that has at least one honest contribution. Termination of the protocol is clear, as all building blocks are deterministic and run in a finite number of rounds. Finally, secrecy of the protocol by definition follows from the aggregated unpredictability of the underlying APVSS. By a typical game hops argument in which the reduction aborts when the adversary forges a NIZK proof (with statistical soundness) with a total of at most $q_s + q_h$ tries, we directly find the bound $\varepsilon \leq \varepsilon_A + (q_s + q_h)/p$. The bound on the running time is also clear. Therefore, it follows that GRand is a (t, ε, T)-secure DKG protocol with the given bounds. That concludes the discussion on the security of the DKG protocol based on the aggregated unpredictability of the underlying APVSS.

In the following, we measure the communication and round complexity of GRand. We begin with the communication complexity by focusing one particular level of the recursive protocol GenAPVSS and then sum over the total number of $\log n$ levels. For this, let us consider the level $\ell \in [\log n]$ of the recursion tree ($\ell = 1$ denotes the bottom level in which parties form committees of size 2) with $\mathcal{Q}_1$ and $\mathcal{Q}_2$ each of size $2^{\ell-1}$ that merge into $\mathcal{Q}$ which is of size $q := 2^{\ell}$. We count the communicated bits among honest parties in the committee $\mathcal{Q}$ and then multiply this with the number n/q of parent committees at that particular level ℓ. However, in contrast to the previous analysis we do not now assume that $\mathcal{Q}$ has honest majority. The reason for this is that even though our protocol is deterministic, it could be possible that in a corrupt majority committee the honest parties communicate much more bits (e.g., $\mathcal{O}(\lambda q^3)$ bits) than in an honest majority committee which would have devastating consequences for the overall communication complexity. Therefore, we do not make any assumptions on $\mathcal{Q}$ and its children committees $\mathcal{Q}_1, \mathcal{Q}_2$. We begin with the analysis assuming the worst-case scenario in which both committees are corrupt and all honest parties in both committees start each with a different transcript. In the first stage, each party in $\mathcal{Q}$ multicasts an accumulation value of size $\mathcal{O}(\lambda)$ to all other parties in $\mathcal{Q}$. Thus, this step takes a total communication of $\mathcal{O}(\lambda q^2)$ bits, as the number of parties in $\mathcal{Q}$ is q. In the next stage, the parties execute two instances of the Byzantine agreement protocol BA which itself has a communication complexity of $\mathcal{O}(\lambda q^2)$ bits (cf. Appendix C). Here, we assume the scenario where the adversary lets the honest parties agree all on a different accumulation value which is the one from its own local PVSS transcript. Afterwards, parties invoke the deliver protocol on their PVSS transcript which is of size $\mathcal{O}(\lambda n)$. By definition of the deliver, each party splits its transcript into chunks of size $O(\lambda n/q)$ and sends its chunk to one particular party. Therefore, the total communication complexity of this step is $q \cdot \mathcal{O}(\lambda n/q \cdot q) = \mathcal{O}(\lambda nq)$ bits. In the next step of the deliver protocol, each party multicasts a chunk it received from a different party only in case the augmented accumulation value corresponds to its own accumulation value and it does so only once in total. Hence, this step also incurs the same number of communicated bits which is $\mathcal{O}(\lambda nq)$. Following this, the next two steps of the recursive protocol are identically to the preceding two steps: two instances of (binary) Byzantine agreement followed by an invocation of the deliver protocol. As a result, we can bound the total communication complexity of honest parties in the committee $\mathcal{Q}$ of size q by $\mathcal{O}(\lambda nq)$ bits because $q \leq n$. Since there are n/q such parent committees at each level, we get a communication complexity of $n/q \cdot \mathcal{O}(\lambda nq) = \mathcal{O}(\lambda n^2)$ bits for one particular level. Since there are exactly $\log n$ levels of the recursion tree, we obtain the total communication complexity of $\mathcal{O}(\lambda n^2 \log n)$ bits. This concludes our discussion on the communication complexity of GRand. We proceed with the round complexity.

In the following, we compute the round complexity of our protocol GRand which is the same as the one of the recursive protocol GenAPVSS. For this, we first give a formula for the round complexity of our Byzantine agreement protocol in Appendix C. Denote by $r(n)$ the round complexity of the protocol that consists of two sequential executions of the four-round graded Byzantine agreement protocol GBA with two rounds of proposal phases and two sequential executions of the Byzantine agreement protocol recursively on committees of size $n/2$. From this observation, we easily derive the recursive formula

$$\begin{aligned} r(n) &= (4 + r(n/2) + 1) + (4 + r(n/2) + 1) \\ &= 2r(n/2) + 10, \end{aligned}$$

where at the lowest level of the recursion we have $r(1) = 0$, since a Byzantine agreement protocol involving a single party is trivial. It can easily be seen that $r(n) = 10n - 10$ is the correct solution for this recursive formula. We use this now to establish a formula for the round complexity of our recursive protocol $\mathsf{GenAPVSS}$. The protocol consists of the following steps: two parallel executions of the protocol itself with committees of size $n/2$, a one-round accumulator proposal step, two parallel executions of Byzantine agreement protocol, an invocation of the two-round deliver protocol, again two parallel executions of Byzantine agreement, and finally again an invocation of the two-round deliver protocol. From this observation, we derive for the round complexity $R(n)$ of $\mathsf{GenAPVSS}$ the following recursive formula

$$\begin{aligned} R(n) &= (R(n/2) + 1) + (r(n) + 2) + (r(n) + 2) \\ &= R(n/2) + 2r(n) + 5 \\ &= R(n/2) + 20n - 15, \end{aligned}$$

where at the lowest level of the recursion we trivially have $R(1) = 0$. Again this can be solved using standard techniques, which gives us the solution $R(n) = 40n - 15\log n - 2 - 40$. This concludes our security and complexity analysis of GRand. □

D.2. Analysis of Randomness Beacon GRandLine

Here, we provide a full proof of Theorem 2.

Proof. In the following, we prove 1-unpredictability and bias-resistance of our randomness beacon $\mathsf{GRandLine}$. We do this by showing that it is hard for an algebraic adversary to output a future randomness beacon value that is valid. Since our randomness beacon values are derived as a unique (deterministic) threshold signature from distributed keys output by GRand, it is enough to show that the adversary cannot produce a forged signature for a future epoch. Since we hash the final epoch signature through a random oracle H_2, the 1-unpredictability and bias-resistance of $\mathsf{GRandLine}$ follows. Having said that, let A be an algebraic algorithm that $(t, \varepsilon, T, L, q_h, 1)$-breaks unpredictability of $\mathsf{GRandLine}$, and let $\xi = (\xi_1, \dots, \xi_n) \in (\mathbb{G}_1 \times \mathbb{G}_2)^n$ be the co-one-more discrete logarithm challenge of degree n with corresponding oracle DL_g where $\xi_i = (\xi_{i,1}, \xi_{i,2}) = (g^{z_i}, h^{z_i})$ for all $i \in [n]$. Without loss of generality, we assume that A queries the random oracle H_2 before producing its prediction $\varrho_r := \mathsf{H}_2(\sigma)$ for some round $r \in [L]$. Further, we assume that all parties are honest prior to the execution of the protocol. It is straightforward how to adjust the proof to the general case. Hereafter, let $\mathcal{C} \subset [n]$ be the dynamically changing set of corrupt parties and $\mathcal{H} := [n] \setminus \mathcal{C}$ the set of honest parties. Initially, we have $\mathcal{C} = \emptyset$. We consider the following sequence of games with A as adversary.

GAME $\mathbf{G}_0$: This is the real game. Generate the system parameters $par = (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, e)$ where $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is an asymmetric type 3 pairing of prime order p cyclic groups with generators $g \in \mathbb{G}_1, , h \in \mathbb{G}_2$. For all indices $i \in [n]$, generate the key pairs as $(pk_i, sk_i) \leftarrow \mathsf{Keys}(par, i)$ such that $pk_i = h^{sk_i}$. Whenever A decides to corrupt a party P_i, return the internal state of that party and set $\mathcal{C} := \mathcal{C} \cup \{i\}$. Thereafter, A gets full control over P_i. Execute the DKG protocol GRand on behalf of the honest parties and let $(PK_1, \dots, PK_n)$ and $(SK_1, \dots, SK_n)$ be the vector of public and secret key shares, respectively. For all $i \in \mathcal{H}$, execute the commitment phase by sampling $\alpha_i \leftarrow_{\$} \mathbb{Z}_p^*$ uniformly at random and publishing $\mathsf{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Answer random oracle queries r_i to H_1 by sampling $\gamma_i \leftarrow_{\$} \mathbb{Z}_p$ and returning $H_1[r_i] := g^{\gamma_i} \in \mathbb{G}_1$. Answer random oracle queries s_i to H_2 by sampling $\varrho_i \leftarrow_{\$} \{0,1\}^\lambda$ and returning $H_2[s_i] := \varrho_i$. For all $i \in \mathcal{H}$ and epoch numbers $r \geq 1$, compute the beacon share σ_i of party P_i as $(g_r^{\alpha_i}, e(g_r, SK_i))$ along with a proof of discrete logarithm equality $\pi_i := \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ certifying the correctness of $g_r^{\alpha_i}$ and publish it. Output the beacon value $\varrho_r := e(g_r, SK)$ for epoch r (either by Lagrange interpolation in the exponent from beacon shares $\{e(g_r, SK_i)\}_{\mathcal{S}}$ or directly from the knowledge of the secret key SK). At any point of the game, say in epoch r, A outputs a prediction (ϱ_ℓ^*, ℓ) for an epoch $\ell \in [L]$. The adversary wins the game if $\ell > r$ and $\varrho_\ell^* = \varrho_\ell$ where $\varrho_\ell := \mathsf{H}_2(e(g_\ell, SK))$. Clearly, A's advantage in winning the game is per definition given by

$$\Pr[\mathbf{G}_0 = 1] = \varepsilon.$$

GAME $\mathbf{G}_1$: This game is identical to the previous game, except that we add an abort condition. Before the execution of the game, sample $\ell^* \leftarrow_{\$} [L]$ uniformly at random. Then, execute the game as before and abort the game if $\ell \neq \ell^*$. Since the choice of ℓ^* remains hidden from A and does not affect the subsequent execution of the game, we bound the winning probability of this game as

$$\Pr[\mathbf{G}_1 = 1] \geq \Pr[\mathbf{G}_0 = 1]/L.$$

GAME $\mathbf{G}_2$: This game is identical to the previous game, except that we reprogram the random oracle H_1 on input ℓ differently (note that $\ell = \ell^*$ is the epoch for which A provides a prediction, i.e., a forgery for the underlying threshold signature). To this end, program H_1 on input r_i as follows. For $r_i \neq \ell$, sample $\gamma_i \leftarrow_{\$} \mathbb{Z}_p$ uniformly at random and return $H_1[r_i] := g^{\gamma_i}$. For $r_i = \ell$, however, return $H_1[\ell] := \xi_{1,1}$ where $\xi = (\xi_1, \dots, \xi_n)$ is the COMDL challenge. Clearly, this game is perfectly indistinguishable from the previous one so that there is no change in the winning probability of the game, i.e.

$$\Pr[\mathbf{G}_2 = 1] = \Pr[\mathbf{G}_1 = 1].$$

GAME $\mathbf{G}_3$: This game is identical to the previous game, except that we add another abort condition. The idea of this hybrid is to guess a party $P_{i^*} \in [n]$ that contributes to the keys generated from GRand and that remains honest until the end of the game. Note that the keys output by GRand are directly derived from the final APVSS transcript $AT \leftarrow \mathsf{GenAPVSS}$ which is just an aggregation of several initially sampled PVSS transcripts with contribution from at least one honest party P^*. Before the execution of the game, sample $i^* \leftarrow_{\$} [n]$ uniformly at random. Then, execute the game as before and let $P^* \in \mathcal{H}$ be the honest party whose initial PVSS transcript is included in the final aggregated transcript AT during the execution of GRand. At the end, abort the game if $P_{i^*} \neq P^*$. Since the choice of i^* remains information-theoretically hidden from A's view, we bound the winning probability of this game as

$$\Pr[\mathbf{G}_3 = 1] \geq \Pr[\mathbf{G}_2 = 1]/n.$$

GAME $\mathbf{G}_4$: This game is identical to the previous game, except that we add another abort condition. Namely, whenever an instance of the Byzantine agreement protocol BA fails to establish consensus, the game aborts. At each level $r \in [\log n]$ of the recursive setup phase, there are $4 \cdot 2^{r-1}$ instances of the consensus protocol (each of the 2^{r-1} committees executes two instances of BA to agree on accumulation values and two instances of $\mathsf{2BA}$ to agree on which PVSS transcripts of the children committees to aggregate). Summing them up over all levels, we obtain

$$\sum_{r=1}^{\log n} 4 \cdot 2^{r-1} = 4 \cdot \sum_{r=1}^{\log n} 2^{r-1} \leq 4n.$$

As each instance of BA fails with probability ε_B, we can bound the winning probability of this game as

$$\Pr[\mathbf{G}_4 = 1] \geq \Pr[\mathbf{G}_3 = 1] - 4n\varepsilon_B.$$

GAME $\mathbf{G}_5$: This game is identical to the previous game, except that we add another abort condition. Namely, whenever the adversary can forge a NIZK proof of knowledge of discrete logarithm θ for one of his PVSS transcripts, the game aborts. As the NIZK proof of our PVSS scheme has statistical soundness, we may bound the soundness error simply by $1/p$. Since the adversary makes a total of q_h random oracle queries, we can bound the winning probability of this game as

$$\Pr[\mathbf{G}_5 = 1] \geq \Pr[\mathbf{G}_4 = 1] - \frac{q_h}{p}.$$

GAME $\mathbf{G}_6$: This game is identical to the previous game, except that we add another abort condition. Namely, whenever the adversary can forge a NIZK proof of discrete logarithm equality π for one of his partial signatures during a randomness beacon epoch, the game aborts. As the NIZK proof of discrete logarithm equality has statistical soundness, again we may bound the soundness error simply by $1/p$. Since the adversary makes a total of q_h random oracle queries, we can bound the winning probability of this game as

$$\Pr[\mathbf{G}_6 = 1] \geq \Pr[\mathbf{G}_5 = 1] - \frac{q_h}{p}.$$

GAME $\mathbf{G}_7$: This game is identical to the previous game, except that we add another abort condition. Namely, whenever the adversary finds a collision among the random oracle queries to the hash function $\mathsf{H}_1 : \{0,1\}^* \to \mathbb{G}_1$, the game aborts. As the adversary has a total of q_h tries and can run the birthday paradox algorithm, we can bound the winning probability of this game as

$$\Pr[\mathbf{G}_7 = 1] \geq \Pr[\mathbf{G}_6 = 1] - \frac{q_h^2}{2p}.$$

As A is an algebraic adversary, at the end of the game it returns the forgery (ϱ_ℓ, ℓ) where $\varrho_\ell = \mathsf{H}_2(e(g_\ell, SK))$ together with an algebraic representation (w.l.o.g. we assume that the adversary queries the random oracle on $e(g_\ell, SK)$ before outputting ρ_ℓ)

$$\Big(a_0, \{a_{i,1}, \ldots, b_{i,2}\}_{i=1}^{n}, \{b_{i,3}\}_{i=1}^{q_h}, \{r_{i,1}, \ldots, r_{i,n}\}_{i=1}^{q_s},$$
$$\{c_{i,j,1}, \ldots, c_{i,j,3}\}_{i,j=1}^{n}, \{d_{i,1,2}, \ldots, d_{i,n,2}\}_{i=1}^{q_h},$$
$$\{e_{i,1,2}, \ldots, e_{i,n,2}\}_{i=1}^{q_h}, \{f_{i,1,2}, \ldots, f_{i,n,2}\}_{i=1}^{q_h},$$
$$\{d_{i,j,1}, e_{i,j,1}, f_{i,j,1}\}_{i,j=1}^{n}, \{s_{i,1,1}, \ldots, s_{i,n,n}\}_{i=1}^{q_s},$$
$$\{t_{i,1,1}, \ldots, t_{i,n,n}\}_{i=1}^{q_s}, \{u_{i,1,1}, \ldots, u_{i,n,n}\}_{i=1}^{q_s},$$
$$\{v_{i,1}, \ldots, v_{i,n}\}_{i=1}^{q_s}\Big)$$

of elements in $\mathbb{Z}_p$ such that

$$\begin{aligned}
&e(g_\ell, SK) \\
&\overset{!}{=} e(g,h)^{a_0} \cdot \prod_{i=1}^{n} e(g, Y_i)^{a_{i,1}} \cdot e(g, pk_i)^{a_{i,2}} \cdot e(g, \mathsf{cm}_{i,2})^{a_{i,3}} \\
&\cdot \prod_{i=1}^{n} e(C_i, h)^{b_{i,1}} \cdot e(\mathsf{cm}_{i,1}, h)^{b_{i,2}} \\
&\cdot \prod_{i=1}^{q_h} e(h_{1,i}, h)^{b_{i,3}} \cdot \prod_{i=1}^{q_s} \prod_{j=1}^{n} e(\sigma_{i,j}, h)^{r_{i,j}} \\
&\cdot \prod_{i,j=1}^{n} e(C_i, Y_j)^{c_{i,j,1}} \cdot e(C_i, pk_j)^{c_{i,j,2}} \cdot e(C_i, \mathsf{cm}_{j,2})^{c_{i,j,3}} \\
&\cdot \prod_{i,j=1}^{n} e(\mathsf{cm}_{i,1}, Y_j)^{d_{i,j,1}} \cdot \prod_{i=1}^{q_h} \prod_{j=1}^{n} e(h_{1,i}, Y_j)^{d_{i,j,2}} \\
&\cdot \prod_{i=1}^{q_s} \prod_{j,k=1}^{n} e(\sigma_{i,j}, Y_k)^{s_{i,j,k}} \cdot \prod_{i,j=1}^{n} e(\mathsf{cm}_{i,1}, pk_j)^{e_{i,j,1}} \\
&\cdot \prod_{i=1}^{q_h} \prod_{j=1}^{n} e(h_{1,i}, pk_j)^{e_{i,j,2}} \cdot \prod_{i=1}^{q_s} \prod_{j,k=1}^{n} e(\sigma_{i,j}, pk_k)^{t_{i,j,k}} \\
&\cdot \prod_{i,j=1}^{n} e(\mathsf{cm}_{i,1}, \mathsf{cm}_{j,2})^{f_{i,j,1}} \cdot \prod_{i=1}^{q_h} \prod_{j=1}^{n} e(h_{1,i}, \mathsf{cm}_{j,2})^{f_{i,j,2}} \\
&\cdot \prod_{i=1}^{q_s} \prod_{j,k=1}^{n} e(\sigma_{i,j}, \mathsf{cm}_{k,2})^{u_{i,j,k}} \cdot \prod_{i=1}^{q_s} \prod_{j=1}^{n} e(g_i, SK_j)^{v_{i,j}}. \quad (1)
\end{aligned}$$

Here, the representation is split (from left to right) into powers of pairing evaluations on combinations of the generators g, h, the polynomial commitments $C_1, \ldots, C_n$ and encrypted shares $Y_1, \ldots, Y_n$ of the aggregated transcript output by $\mathsf{GenAPVSS}$ (which has contribution from the designated party P^*), the public keys $pk_1, \ldots, pk_n$ of parties (these constitute the setup phase), the auxiliary commitments $\mathsf{cm}_1, \ldots, \mathsf{cm}_n$ (these constitute the commitment phase), the answers to hash queries $h_{1,i} := \mathsf{H}_1(m_i), i \in [q_h]$, returned by the random oracle, and the beacon value shares (seen as partial signatures of the underlying threshold signature scheme) $e(g_i, SK_j)$, where $i \in [q_s]$ and $j \in [n]$, along with the correctness shares $\sigma_{i,j} := g_i^{\alpha_j}$ (recall that $g_i := \mathsf{H}_1(i)$ by definition). Here, q_s is defined as the current epoch in which the adversary outputs its prediction and thus $q_s < \ell$ by assumption (as the protocol is deterministic after the setup phase and parties do output the beacon values in sequence). Further, we assume w.l.o.g. that $m_i = i$ for all $i \in [q_s]$. In the following, we let Q_h denote the set $[q_h] \backslash \{\ell\}$ (recall that ℓ is the index where the forgery happens). Then the above equation over $\mathbb{G}_T$ to base $e(g, h)$ yields

$$\begin{aligned}
\gamma_\ell f(0) &\overset{!}{=} a_0 + \sum_{i=1}^{n} a_{i,1} f(i) sk_i + a_{i,2} sk_i + a_{i,3}(-\alpha_i + f(i)) \\
&+ \sum_{i=1}^{n} b_{i,1} f(i) + b_{i,2}\alpha_i + \sum_{i\in Q_h} b_{i,3}\gamma_i + \sum_{i=1}^{q_s}\sum_{j=1}^{n} r_{i,j}\gamma_i\alpha_j \\
&+ \sum_{i,j=1}^{n} c_{i,j,1} f(i) f(j) sk_j + c_{i,j,2} f(i) sk_j \\
&+ c_{i,j,3} f(i)(-\alpha_j + f(j)) \\
&+ \sum_{i,j=1}^{n} d_{i,j,1}\alpha_i f(j) sk_j + \sum_{i\in Q_h}\sum_{j=1}^{n} d_{i,j,2}\gamma_i f(j) sk_j \\
&+ \sum_{i=1}^{q_s}\sum_{j,k=1}^{n} s_{i,j,k}\gamma_i\alpha_j f(k) sk_k + \sum_{i,j=1}^{n} e_{i,j,1}\alpha_i sk_j \\
&+ \sum_{i\in Q_h}\sum_{j=1}^{n} e_{i,j,2}\gamma_i sk_j + \sum_{i=1}^{q_s}\sum_{j,k=1}^{n} t_{i,j,k}\gamma_i\alpha_j sk_k \\
&+ \sum_{i,j=1}^{n} f_{i,j,1}\alpha_i(-\alpha_j + f(j)) + \sum_{i\in Q_h}\sum_{j=1}^{n} f_{i,j,2}\gamma_i(-\alpha_j + f(j)) \\
&+ \sum_{i=1}^{q_s}\sum_{j,k=1}^{n} u_{i,j,k}\gamma_i\alpha_j(-\alpha_k + f(k)) + \sum_{i=1}^{q_s}\sum_{j=1}^{n} v_{i,j}\gamma_i f(j) \\
&+ \gamma_\ell\Big(b_{\ell,3} + \sum_{j=1}^{n} d_{\ell,j,2} f(j) sk_j + \sum_{j=1}^{n} e_{\ell,j,2} sk_j \\
&+ \sum_{j=1}^{n} f_{\ell,j,2}(-\alpha_j + f(j))\Big).
\end{aligned}$$

Note that we have split the terms into those that contain the ℓ-th term and those that do not. As a result, the terms on the right-hand side of the equation other than the last one are independent from the variable γ_ℓ. By rewriting, we get the simplified identity (♠)

$$\gamma_\ell\left(b_{\ell,3} - f(0) + \sum_{j=1}^{n}\left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j + f_{\ell,j,2}(\dots)\right]\right) = A,$$

where A is the negative/minus of the appropriate rest term. Let us write this equation as $\gamma_\ell \cdot B = A$ for the appropriate B. We consider the following five events:

- E_1 defined by the identity $B = 0$.
- E_2 defined by: there is no index $i \in \mathcal{H}$ such that $f_i \neq 0$ that are known polynomials in $f_{\ell,1,2}, \dots, f_{\ell,n,2}$.
- E_3 defined by: there is no index $i \in \mathcal{H}$ such that $e_i \neq 0$ that are known polynomials in the coefficients output by A.
- E_4 defined by: there is no index $i \in \mathcal{H}$ such that $d_{\ell,i,2} \neq 0$ and $f_{\ell,i,2} + s_i \neq 0$ for known coefficients s_i.
- E_5 defined by: there is no index $i \in \mathcal{H}$ s.t. $r_i' \neq 0$ and $t_i' \neq 0$ that are known polynomials in the coefficients output by A.

With this, we obtain the following technical lemma.

Lemma 1. *Let $\mathbf{G}_7$ and events E_i for $i \in [5]$ be defined as above. Then there exist (algebraic) algorithms A_j for $j \in [6]$ playing in game n-**COMDL** that run in time at most T such that:*

$$\begin{aligned}
&\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_1} = 1] = \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \neg E_1], \\
&\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_2} = 1] = \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge E_1 \wedge \neg E_2], \\
&\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_3} = 1] = \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge \neg E_3], \\
&\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_4} = 1] = \frac{1}{2}\Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \ldots \wedge E_3 \wedge \neg E_4], \\
&\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_5} = 1] = \frac{1}{2}\Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \ldots \wedge E_4 \wedge \neg E_5] \\
&\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_6} = 1] = \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge E_1 \wedge \ldots \wedge E_5].
\end{aligned}$$

Moreover, $T \leq T' + \mathcal{O}(Ln^2)$.

Proof. Let $\xi = (\xi_1, \dots, \xi_n) \in \mathbb{G}^n$ with $\xi_i = (g^{z_i}, h^{z_i})$ for $i \in [n]$ be the COMDL instance of degree n. Algorithms A_i for $i \in [6]$ have access to a (perfect) discrete logarithm oracle DL_g in $\mathbb{G}_1$ (to base g) which they can query at most $n-1$ times. When we say algorithm A_i queries the discrete logarithm oracle on ξ_j, we mean that it queries DL_g on the first component of ξ_j which is a group element in $\mathbb{G}_1$. Before we start with the description of the algorithms A_i, we describe four different algorithms Sim_i for $i \in [4]$ that give a perfect simulation of the game $\mathbf{G}_7$ to the adversary A.

SIMULATOR $\mathsf{Sim}_1(\xi, par)$: On input ξ, Sim_1 queries the discrete logarithm oracle DL_g on $\xi_2, \dots, \xi_n$ and gets $(z_2, \dots, z_n)$. It generates the public-secret key pairs of honest parties by sampling $sk_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and publishes $pk_i := h^{sk_i}$. Sim_1 executes the DKG protocol GRand on behalf of the honest parties by sampling degree-t polynomials $f_i \in \mathbb{Z}_p[X]$ uniformly at random for all $i \in \mathcal{H}$. At any point of the simulation, Sim_1 answers corruption queries by returning the internal state of the respective party faithfully. After this setup phase, it executes the commitment phase by sampling $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random for all $i \in \mathcal{H}$ and publishes $\mathsf{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Further, for all $i \neq \ell$ it answers random oracle queries r_i to H by sampling $\gamma_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and returning $H_1[r_i] := g^{\gamma_i}$. For $r_i = \ell$, however, it returns $H_1[\ell] := \xi_1$. It answers random oracle queries to H_2 by lazy sampling as usual. After the setup phase, the sequential randomness beacon phase begins. For all epochs $r < \ell$, Sim_1 computes the beacon share $\sigma_i = (g_r^{\alpha_i}, e(g_r, SK_i))$ along with the proof of discrete logarithm equality $\pi_i = \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ for honest party P_i by simple computation from the knowledge of α_i and SK_i. It is clear that this simulation is perfect, since the simulator follows all steps of the protocol instructions faithfully and only changes the way how the random group element $g_\ell = \mathsf{H}_1(\ell)$ is generated (on which the forgery is produced).

SIMULATOR $\mathsf{Sim}_2(\xi, par)$: On input ξ, Sim_2 generates the public-secret key pairs of honest parties by sampling $sk_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and publishes $pk_i := h^{sk_i}$. Sim_2 executes the DKG protocol GRand on behalf of the honest parties by sampling degree-t polynomials $f_i \in \mathbb{Z}_p[X]$ uniformly at random for all $i \in \mathcal{H}$. After this setup phase, it executes the commitment phase as follows. For all $i \in \mathcal{H}$, it generates the commitment cm_i of party P_i as $\mathsf{cm}_i := (\xi_{i,1}, \xi_{i,2}^{-1} SK_i)$ and publishes it. Further, for all i it answers random oracle queries r_i to H by sampling $\gamma_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and returning $H_1[r_i] := g^{\gamma_i}$. It does so also for the random oracle query $H_1[\ell]$. It answers random oracle queries to H_2 by lazy sampling as usual. After the setup phase, the sequential randomness beacon phase begins. For all epochs $r < \ell$, Sim_2 computes the beacon share $\sigma_i = (g_r^{\alpha_i}, e(g_r, SK_i))$ along with the proof of discrete logarithm equality $\pi_i = \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ for honest party P_i by the identity $g_r^{\alpha_i} = \xi_{i,1}^{\gamma_i}$, by computation $e(g_r, SK_i)$ from the knowledge of SK_i, and π_i by honest-verifier zero-knowledge (HVZK) simulation. At any point of the simulation, Sim_2 answers corruption queries $i \in \mathcal{H}$ by calling its discrete logarithm oracle DL_g on input ξ_i to obtain $\alpha_i := \mathsf{DL}_g(\mathsf{cm}_i)$ and returning α_i along with other internal data such as sk_i (note that α_i is the only value for party P_i that was not generated honestly). Note that this is different from the previous simulator in that it was able to return the full internal state without calling its discrete logarithm oracle. It is clear that this simulation is perfect, since the simulator only changes the way how the commitments cm_i for $i \in \mathcal{H}$ are generated (indistinguishable from an honest generation) and can output beacon shares via random oracle programming and HVZK simulation.

SIMULATOR $\mathsf{Sim}_3(\xi, par)$: On input ξ, Sim_3 generates the public-secret key pairs of honest parties as $pk_i := \xi_{i,2}$ and publishes them. This implicitly fixes the secret keys as $sk_i = z_i$ which is the discrete logarithm value of ξ_i. Sim_3 executes the DKG protocol GRand on behalf of the honest parties by sampling degree-t polynomials $f_i \in \mathbb{Z}_p[X]$ uniformly at random for all $i \in \mathcal{H}$. After this setup phase, it executes the commitment phase by sampling $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random for all $i \in \mathcal{H}$ and publishes $\mathsf{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Further, for all i it answers random oracle queries r_i to H by sampling $\gamma_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and returning $H_1[r_i] := g^{\gamma_i}$. It does so also for the random oracle query $H_1[\ell]$. It answers random oracle queries to H_2 by lazy sampling as usual. After the setup phase, the sequential randomness beacon phase begins. For all epochs $r < \ell$, Sim_3 computes the beacon share $\sigma_i = (g_r^{\alpha_i}, e(g_r, SK_i))$ along with the proof of discrete logarithm equality $\pi_i = \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ for honest party P_i as follows: generation of $g_r^{\alpha_i}$ and π_i are by simple computation from the knowledge of α_i, and generation of $e(g_r, SK_i)$ is done via the identity

$$e(g_r, SK_i) = e(g, SK_i)^{\gamma_i} = e(g^{f(i)}, h)^{\gamma_i} = e(PK_i, h)^{\gamma_i}.$$

At any point of the simulation, Sim_3 answers corruption queries $i \in \mathcal{H}$ by calling its discrete logarithm oracle DL_g on input ξ_i to obtain sk_i and returning sk_i along with other internal data such as α_i (note that sk_i is the only value for party P_i that was not generated honestly). It is clear that this simulation is perfect, since the simulator follows all steps of the protocol instructions faithfully and only changes the bulletin board keys are generated.

SIMULATOR $\mathsf{Sim}_4(\xi, par)$: On input ξ, Sim_4 queries the discrete logarithm oracle DL_g on $\xi_{t+2}, \ldots, \xi_n$ and gets $(z_{t+2}, \ldots, z_n)$. It generates the public-secret key pairs of honest parties by sampling $sk_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and publishes $pk_i := h^{sk_i}$. Sim_4 executes the DKG protocol GRand on behalf of the honest parties by sampling degree-t polynomials $f_i \in \mathbb{Z}_p[X]$ uniformly at random for all $i \in \mathcal{H} \setminus \{P^*\}$. For the designated party P^* (that contributes to the final aggregated PVSS transcript and remains honest), however, it generates the degree-t polynomial $f_{i^*} = d_0 + d_1 X + \ldots + d_t X^t \in \mathbb{Z}_p[X]$ such that $g^{d_j} = \xi_{j+1}$ for all $j \in [\![t]\!]$ (i.e., the $t + 1$ coefficients of the polynomial are given by the discrete logarithm values of $\xi_1, \ldots, \xi_{t+1}$). From this, it can generate the commitments and encrypted shares of party P^*'s PVSS transcript by Lagrange interpolation in the exponent and knowledge of the secret keys sk_j of all parties. In this context, it is important to note the following crucial and subtle observation: since Sim_4 generates the public keys pk_i of honest parties faithfully and the adversary A is algebraic, it outputs the (updated) public keys of corrupt parties as a linear combination of known values which enables Sim_4 to compute the respective secret keys from this linear combination along with the algebraic representation. At any point of the simulation, Sim_4 answers corruption queries by returning the internal state of the respective party faithfully. Note that by assumption P^* remains honest and for any other party the simulator follows the protocol instructions honestly so that it can return the internal state of the respective party without any discrete logarithm query. After this setup phase, it executes the commitment phase by sampling $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random for all $i \in \mathcal{H}$ and publishes $\mathsf{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Further, for all i it answers random oracle queries r_i to H by sampling $\gamma_i \leftarrow_\$ \mathbb{Z}_p$ uniformly at random and returning $H_1[r_i] := g^{\gamma_i}$. It does so also for the random oracle query $H_1[\ell]$. It answers random oracle queries to H_2 by lazy sampling as usual. After the setup phase, the sequential randomness beacon phase begins. For all epochs $r < \ell$, Sim_4 computes the beacon share $\sigma_i = (g_r^{\alpha_i}, e(g_r, SK_i))$ along with the proof of discrete logarithm equality $\pi_i = \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ for honest party P_i by simple computation from the knowledge of α_i and SK_i. It is clear that this simulation is perfect, since the simulator follows all steps of the protocol instructions faithfully for all parties except P^* and only changes the way the designated party generates its PVSS transcript which is indistinguishable from an honestly generated transcript.

Having defined the above simulators Sim_i for $i \in [4]$, we can now describe the algorithms A_i and especially how they convert the forgery output by A (winning game $\mathbf{G}_7$) into a valid solution to the COMDL instance ξ with high probability, conditioned on some event happening. Recall equation (♠) defined as $B\gamma_\ell = A$ for the appropriate terms A and B. We now describe the algorithms A_i that simulate the game $\mathbf{G}_7$ to the adversary.

Algorithm $\mathsf{A}_1(\xi, par)$: Algorithm A_1 works identical as the simulator Sim_1. In particular, the simulation is perfect and the only change is the way how the random group element $g_\ell = \mathsf{H}_1(\ell)$ is generated on which the forgery is produced. Suppose that A_1 wins the game $\mathbf{G}_7$ and that event $\neg E_1$ happens, i.e., $B \neq 0$. Then equation (♠) is a non-trivial equation of degree one in $\gamma_\ell = \mathsf{DL}_g(\xi_1)$ (as clarified before, the terms A and B are completely independent from the variable γ_ℓ). By simple algebra, this allows A_1 to solve for $\gamma_\ell = A \cdot B^{-1}$ and thus efficiently output the discrete logarithm values of the COMDL challenge ξ. Overall, we obtain

$$\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_1} = 1] = \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \neg E_1].$$

The bound on the running time of A_1 is obvious.

Algorithm $\mathsf{A}_2(\xi, par)$: Algorithm A_2 works identical as the simulator Sim_2. In particular, the simulation is perfect and the only change is the way how the commitments cm_i are generated. In this case, we have $\mathsf{cm}_i = (\xi_{i,1}, \xi_{i,2}^{-1}SK_i)$ for all $i \in \mathcal{H}$ (here, $\mathcal{H} \subset [n]$ is the set of all honest parties up to the point in which parties output these commitments) and thus $\alpha_i = \mathsf{DL}_g(\xi_i)$. Suppose that A_2 wins the game $\mathbf{G}_7$ and that event $E_1 \wedge \neg E_2$ happens, i.e., $B = 0$ and also there is an index $i \in \mathcal{H}$ such that $f_i \neq 0$ (we will define these very soon). We let $F := f_{\ell,1,2}\alpha_1 + \ldots + f_{\ell,n,2}\alpha_n$ and from $B = 0$ get the identity

$$F = b_{\ell,3} - f(0) + \sum_{j=1}^{n} \left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j + f_{\ell,j,2} f(j) \right], \tag{♥}$$

and so all terms on the right-hand side of the equation are independent from the variables $\alpha_1, \ldots, \alpha_n$ and can also be concretely computed by A_2 as they are known. We consider the defining equation of F in more detail now. Since the adversary A is algebraic, it outputs its commitments cm_i as (known) linear combinations of the commitments of the honest parties, since all previously generated elements output by the simulator Sim_2 are generated honestly and thus independent from the unknown ξ. For simplicity, we assume for the remainder of this paragraph that $\mathcal{C} = [t]$ and $\mathcal{H} = [t+1, n]$. As a result, there are linear polynomials $F_i \in \mathbb{Z}_p[X]$ for all $i \in \mathcal{C}$ such that $\alpha_i = F_i(\alpha_{t+1}, \ldots, \alpha_n) = F_i(z_{t+1}, \ldots, z_n)$ that only depend on the outputs by the honest parties. Therefore, the defining equation for F can be transformed into

$$F = f_0 + f_{t+1}\alpha_{t+1} + f_{t+2}\alpha_{t+2} + \ldots + f_n\alpha_n$$

for some appropriately defined coefficients $f_0, f_{t+1}, \ldots, f_n \in \mathbb{Z}_p$. By assumption we suppose that A_2 wins the game $\mathbf{G}_7$ and that event $\neg E_2$ happens, that means there is an index $i \in \mathcal{H}$ such that $f_i \neq 0$. Since the value of F by equation (♥) can be concretely computed by A_2, in combination with the above equation for F linear in the variables $\alpha_{t+1}, \ldots, \alpha_n$ the algorithm A_2 proceeds as follows. It computes all the values $\alpha_j = \mathsf{DL}_g(\xi_j)$ by calling its discrete logarithm oracle on input ξ_j for $j \in \mathcal{H} \setminus \{i\}$ and then solves for the value α_i in the above linear equation for F. By simple algebra, this allows A_2 to solve for $\alpha_1, \ldots, \alpha_n$ and thus efficiently output the discrete logarithm values of the COMDL challenge ξ. Overall, we obtain

$$\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_2} = 1] = \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge E_1 \wedge \neg E_2].$$

The bound on the running time of A_2 is obvious.

Algorithm $\mathsf{A}_3(\xi, par)$: Algorithm A_3 works identical as the simulator Sim_3. In particular, the simulation is perfect and the only change is the way the bulletin board keys are generated. In this case, we have $pk_i = \xi_{i,2}$ for all $i \in \mathcal{H}$ and thus implicitly $sk_i = z_i$ which is the discrete logarithm value of ξ_i. Suppose that A_3 wins the game $\mathbf{G}_7$ and the event $E_1 \wedge E_2 \wedge \neg E_3$ happens. Recall the E_1 defining equation (♦)

$$\sum_{j=1}^{n} \left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j - f_{\ell,j,2}\alpha_j + f_{\ell,j,2} f(j) \right] \\ = f(0) - b_{\ell,3}.$$

Again, let $F := f_{\ell,1,2}\alpha_1 + \ldots + f_{\ell,n,2}\alpha_n$ as before. In contrast to the previous paragraph, now the values α_i for all $i \in \mathcal{C}$ (here, $\mathcal{C} \subset [n]$ is the set of all corrupt parties up to the point in which parties output their commitments cm_i) are known and independent of $z_1, \ldots, z_n$ that are the discrete logarithm values of ξ_i. The reason for this is that since the adversary A is algebraic, it outputs the commitments $\mathsf{cm}_{i,1} = g^{\alpha_i} \in \mathbb{G}_1$ with an algebraic representation of elements in the prime field $\mathbb{Z}_p$. However, as the simulator Sim_3 (that defines the algorithm A_3) only uses the elements $\xi_{i,2} \in \mathbb{G}_2$ in the second source group for its simulation of the protocol, A is agnostic of the elements $\xi_{i,1} \in \mathbb{G}_1$ and therefore has to explicitly provide knowledge of the α_i for $i \in \mathcal{C}$ through the algebraic representation. This argument relies on the fact that the underlying pairing is of type 3 and there is no efficient way for A to transform elements from one source group to the other source group.[8] The same argument also applies to the evaluations of the hidden degree-t polynomial $f(X) \in \mathbb{Z}_p[X]$, as the adversary outputs the commitments of its chosen polynomials in the first source group $\mathbb{G}_1$ and therefore independent of the $\xi_{i,2}$ elements. As a result, the values $f(0), f(1), \ldots, f(n)$ are known to A_3 and independent from the variables $z_1, \ldots, z_n$.

[8] When working with a pairing of type 1 or 2, this argument has to be adjusted slightly. Informally, it would involve another reduction/algorithm that solves for the discrete logarithm of h to base g, since that would be the only way the algebraic adversary can output an element to base g from given elements to base h.

Overall, the only variables in the above equation ($\blacklozenge$) that depend on $z_1, \ldots, z_n$ are the secret keys $sk_1, \ldots, sk_n$. We simplify/update the equation as

$$\sum_{j=1}^{n} \hat{e}_j sk_j = f(0) - b_{\ell,3} + \sum_{j=1}^{n} \left[f_{\ell,j,2}\alpha_j - f_{\ell,j,2} f(j) \right], \quad (\blacklozenge)$$

where $\hat{e}_j = d_{\ell,j,2} f(j) + e_{\ell,j,2}$ for all $j \in [n]$. Since A is algebraic, it outputs its (updated) keys pk_i for $i \in \mathcal{C}$ (up to the point in which parties publish their keys on the bulletin board) as linear combinations of the honest parties' keys. For simplicity, we assume that $\mathcal{C} = [t]$ and $\mathcal{H} = [t+1, n]$. Thus, let us write

$$sk_i = \lambda_{0,i} + \lambda_{t+1,i} z_{t+1} + \ldots + \lambda_{n,i} z_n$$

for all $i \in \mathcal{C}$ and some coefficients $\lambda_{0,i}, \lambda_{t+1,i}, \ldots, \lambda_{n,i} \in \mathbb{Z}_p$, since $sk_i = z_i$ for $i \in \mathcal{H}$. If we plug in these identities into ($\blacklozenge$), we obtain

$$\begin{aligned} & \sum_{j \in \mathcal{C}} \hat{e}_j \lambda_{0,j} + \sum_{j \in \mathcal{H}} z_j \left(\hat{e}_j + \sum_{i \in \mathcal{C}} \lambda_{j,i} \right) = [\ldots] \\ \iff & \sum_{j \in \mathcal{C}} \hat{e}_j \lambda_{0,j} + \sum_{j \in \mathcal{H}} z_j \left(\hat{e}_j + \lambda_{j,1} + \ldots \lambda_{j,t} \right) = [\ldots], \end{aligned}$$

where $[\ldots]$ is simply identical to the right-hand side of ($\blacklozenge$). From this equation we see that it defines a polynomial of degree one in the variables $z_{t+1}, \ldots, z_n$. We define the corresponding coefficients of z_j for $j \in \mathcal{H}$ as e_j, that is $e_j := \hat{e}_j + \sum_{i \in \mathcal{C}} \lambda_{j,i}$. By assumption we suppose that A_3 wins the game $\mathbf{G}_7$ and that event $\neg E_3$ happens, that means there is an index $i \in \mathcal{H}$ such that $e_i \neq 0$. Having said that, algorithm A_3 proceeds as follows. It computes all the values $z_j = \mathsf{DL}_g(\xi_j)$ by calling its discrete logarithm oracle on input ξ_j for $j \in \mathcal{H} \setminus \{i\}$ and then solves for the remaining variable z_i in the above linear equation. By simple algebra, this allows A_3 to solve for $sk_1, \ldots, sk_n$ and thus efficiently output the discrete logarithm values of the COMDL challenge ξ that it received. Overall, we obtain

$$\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_3} = 1] = \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge E_1 \wedge E_2 \wedge \neg E_3].$$

The bound on the running time of A_3 is obvious.

Algorithm $\mathsf{A}_4(\xi, par)$: Algorithm A_4 works identical as the simulator Sim_4. In particular, the simulation is perfect and the only change is the way the final aggregated PVSS transcript is formed. In this case, the simulator generates the degree-t polynomial $f_{i^*} = d_0 + d_1 X + \ldots + d_t X^t \in \mathbb{Z}_p[X]$ such that $g^{d_j} = \xi_{j+1}$ for all $j \in [\![t]\!]$ (i.e., the $t+1$ coefficients of the polynomial are given by the discrete logarithm values of $\xi_1, \ldots, \xi_{t+1}$). Everything else is generated honestly (in particular, the bulletin board keys and the PVSS transcripts of other parties). Without loss of generality, we assume that the adversary chooses all PVSS transcripts that contribute to the final aggregated transcript $AT \leftarrow \mathsf{GenAPVSS}$ output from the DKG execution except the one from party P^*. Further, we assume for simplicity that the final transcript only consists of two single PVSS transcript (i.e., the contribution vector $b \in \{0,1\}^n$ is of weight two $|b| = 2$). The general case in which there is more than one PVSS transcript chosen by A works analogously (intuitively, it does not make a difference if A outputs its PVSS transcripts separately or aggregated). Now there are two cases that can happen: (i) A chooses its PVSS transcript dependent from the transcript T^* of the honest party P^*, and (ii) A generates its PVSS transcript honestly and independent from T^*. However, since parties augment the PVSS transcript with a (non-interactive) proof od knowledge θ, the idea is that the reduction can extract the evaluation $f_i(0)$ of the polynomial $f_i \in \mathbb{Z}_p[X]$ chosen by A (since the proof of knowledge comes with an algebraic representation of the respective hash query for the challenge in the Fiat-Shamir heuristic) and thereby solve for $f_{i^*}(0)$ in case A chose f_i dependent on f_{i^*}. This allows the reduction to obtain the discrete logarithm value $z_1 = f_{i^*}(0)$ and thus solve the COMDL challenge. This intuition is made formal and explicit in the security reduction of [18]. We omit it here and directly assume that the adversary chooses its polynomial f_i honestly and independent of the honest party's P^* polynomial f_{i^*}. On the other hand, the adversary has also the possibility to choose its commitments cm_i dependent of the elements output by the transcript T^* (or equivalently the aggregated transcript AT), since the transcript includes terms in both source groups $\mathbb{G}_1$ and $\mathbb{G}_2$ to base g and h, respectively. The bulletin board keys, however, remain independent from the transcript or challenge ξ, as already clarified in the description of Sim_4. We will take these facts into consideration in our following analysis. Suppose that A_4 wins the game $\mathbf{G}_7$ and that the event $E_1 \wedge E_2 \wedge E_3 \wedge \neg E_4$ happens. From event E_1 we recall the equation

$$\begin{aligned} & \sum_{j=1}^{n} \left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j - f_{\ell,j,2}\alpha_j + f_{\ell,j,2} f(j) \right] \\ & = f(0) - b_{\ell,3}. \end{aligned}$$

From event E_3 we know that $e_j = 0$ for all $j \in \mathcal{H}$ where $e_j = \hat{e}_j + \sum_{i \in \mathcal{C}} \lambda_{j,i} = \hat{e}_j$ (observe that now $\lambda_{j,i} = 0$ for all $i \in \mathcal{C}$ and $j \in \mathcal{H}$ as the secret keys are independent of the challenge ξ). It follows that $0 = \hat{e}_j = d_{\ell,j,2} f(j) + e_{\ell,j,2}$. The easy case now is if there is an $j \in \mathcal{H}$ such that $d_{\ell,j,2} \neq 0$, as this allows us to rewrite $f(j) = -e_{\ell,j,2}/d_{\ell,j,2}$ and obtain a non-trivial equation

$$-\frac{e_{\ell,j,2}}{d_{\ell,j,2}} = z_1 + z_2 j + \ldots + z_{t+1} j^t \qquad (\heartsuit)$$

in the variables $z_1, \ldots, z_{t+1}$ (recall that the reduction A_4 chooses the polynomial f_{i*} such that its coefficients are the discrete logarithm values of $\xi_1, \ldots, \xi_{t+1}$). Further, by calling its discrete logarithm oracle DL_g on inputs $\xi_2, \ldots, \xi_{t+1}$, it can solve for z_1 in the above equation ($\heartsuit$) and thus efficiently output the discrete logarithm values of the COMDL challenge ξ. This case in the definition of event $\neg E_4$ is settled and thus for the remainder of this paragraph we assume $d_{\ell,j,2} = 0$ for all $j \in \mathcal{H}$. Having said that, the only unknown terms dependent on $z_1, \ldots, z_{t+1}$ in $d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j$ are the $f(j)$ for $j \in \mathcal{C}$. Since $|\mathcal{C}| \leq t$, the algorithm A_4 proceeds as follows. It calls its discrete logarithm oracle DL_g

on inputs $g^{f(j)}$ for all $j \in \mathcal{C}$, thus obtaining t new linearly independent equations $f(j) = z_1 + z_2 j + \ldots + z_{t+1} j^t$ (the equations written in matrix form give a Vandermonde matrix which is well-known to have full rank) with known values $f(j)$. In particular, we can rewrite the above equation from event E_1 as

$$\sum_{j=1}^{n} \left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j - f_{\ell,j,2}\alpha_j + f_{\ell,j,2} f(j) \right]$$
$$= f(0) - b_{\ell,3}$$

which is equivalent to

$$\iff \sum_{j=1}^{n} \left[-f_{\ell,j,2}\alpha_j + f_{\ell,j,2} f(j) \right] = f(0) + D$$
$$\iff \sum_{j=1}^{n} \left[f(j) - \alpha_j \right] f_{\ell,j,2} = f(0) + D, \qquad (2)$$

where $D \in \mathbb{Z}_p$ is the appropriate (known) rest term. Note that the terms $d_{\ell,j,2} f(j) sk_j$ vanish for $j \in \mathcal{H}$, the values $e_{\ell,j,2} sk_j$ and $b_{\ell,3}$ are known, and finally the terms $d_{\ell,j,2} f(j) sk_j$ are also now known for $j \in \mathcal{C}$. On the other hand, when the adversary A outputs its commitments $\mathsf{cm}_{i,2}$ in the second group $\mathbb{G}_2$ it outputs them as a (known) linear combination of the elements $h, pk_1, \ldots, pk_n$, $\mathsf{cm}_{t+1,2}, \ldots, \mathsf{cm}_{n,2}$, and $Y_1, \ldots, Y_n$. As a result, this gives an identity for all $i \in \mathcal{C}$ as follows

$$f(i) - \alpha_i$$
$$= r_{0,i} + \sum_{j=1}^{n} r_{j,i} sk_j + \sum_{j\in\mathcal{H}} s_{j,i} \left(f(j) - \alpha_j \right) + \sum_{j=1}^{n} t_{j,i} f(j) sk_j, \qquad (4)$$

where the $r_{j,i}, s_{j,i}, t_{j,i} \in \mathbb{Z}_p$ are known coefficients. We take the sum of $f_{\ell,i,2}(f(i) - \alpha_i)$ over all $i \in [n]$ and also use the equations given by (4) and the one given by (2) above. This results in

$$\begin{aligned} f(0) + D &= \sum_{i=1}^{n} f_{\ell,i,2}(f(i) - \alpha_i) \\ &= \sum_{i\in\mathcal{H}} f_{\ell,i,2}(f(i) - \alpha_i) + r_0 + \sum_{j=1}^{n} r_j sk_j \\ &\quad + \sum_{j\in\mathcal{H}} s_j(f(j) - \alpha_j) + \sum_{j=1}^{n} t_j f(j) sk_j \\ &= r_0 + \sum_{j=1}^{n} r_j sk_j + \sum_{j=1}^{n} t_j f(j) sk_j \\ &\quad + \sum_{j\in\mathcal{H}} (f_{\ell,j,2} + s_j)(f(j) - \alpha_j), \qquad (\clubsuit) \end{aligned}$$

where we define

$$r_0 := \sum_{i\in\mathcal{C}} f_{\ell,i,2} r_{0,i}, \qquad r_j := \sum_{i\in\mathcal{C}} f_{\ell,i,2} r_{j,i},$$
$$s_j := \sum_{i\in\mathcal{C}} f_{\ell,i,2} s_{j,i}, \qquad t_j = \sum_{i\in\mathcal{C}} f_{\ell,i,2} t_{j,i}.$$

Now the other case of $\neg E_4$ applies, which means that there is an index $i \in \mathcal{H}$ such that $f_{\ell,i,2} + s_i \neq 0$. In that case, we could let A_4 instead work identical as simulator Sim_2. In particular, the simulation is perfect and the only change is the way how the commitments cm_i are generated. In this case, we have $\mathsf{cm}_i = (\xi_{i,1}, \xi_{i,2}^{-1} SK_i)$ for all $i \in \mathcal{H}$ and thus $\alpha_i = \mathsf{DL}_g(\xi_i)$. Analogously, we can derive the same (general) equation ($\clubsuit$), where D only depends on $f(j)$ and sk_j for $j \in [n]$ and is independent of α_j for $j \in \mathcal{H}$. As a result, this equation gives a linear polynomial in the variables $\{\alpha_j\}_{j\in\mathcal{H}}$ and there is a non-zero coefficient $f_{\ell,i,2} + s_i \neq 0$ of α_i for that particular i given by event $\neg E_4$ happening. As for algorithm A_2, the algorithm A_4 proceeds as follows. It computes all the values $\alpha_j = \mathsf{DL}_g(\xi_j)$ by calling its discrete logarithm oracle on input ξ_j for $j \in \mathcal{H} \setminus \{i\}$ and then solves for the value α_i given the above linear equation ($\clubsuit$). By simple algebra, this allows A_4 to solve for $\alpha_1, \ldots, \alpha_n$ and thus efficiently output the discrete logarithm values of the COMDL challenge ξ. Finally, we let algorithm A_4 choose the simulation strategies Sim_2 and Sim_4 each with probability $1/2$ at the beginning of its execution. Since, this choice remains completely hidden from the adversary A's view, we obtain

$$\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_4} = 1] = \frac{1}{2} \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \ldots \wedge E_3 \wedge \neg E_4].$$

The bound on the running time of A_4 is obvious.

Algorithm $\mathsf{A}_5(\xi, par)$: Before we proceed with the description of algorithm A_5, we give a brief summary of the identities we have so far. From event E_1 in its very plain form, we have the equation

$$\sum_{j=1}^{n} \left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j - f_{\ell,j,2}\alpha_j + f_{\ell,j,2} f(j) \right]$$
$$= f(0) - b_{\ell,3}.$$

Together with event E_4, we have as before

$$f(0) + D = r_0 + \sum_{j=1}^{n} r_j sk_j + \sum_{j=1}^{n} t_j f(j) sk_j,$$

where by event E_3 now

$$D = -\left(b_{\ell,3} + \sum_{j\in\mathcal{C}} \left[d_{\ell,j,2} f(j) sk_j + e_{\ell,j,2} sk_j \right] \right).$$

Taking these together, we get

$$f(0) - (r_0 + b_{\ell,3}) = \sum_{j=1}^{n} r'_j sk_j + \sum_{j=1}^{n} t'_j f(j) sk_j \qquad (\diamond)$$

for appropriate (known) coefficients r'_j and t'_j. This equation has the same form as the one that algorithm A_3 started with. From this observation, we let algorithm A_5 choose the simulation strategies Sim_3 and Sim_4 each with probability $1/2$ at the beginning of its execution. This choice remains completely hidden from the adversary A's view. Therefore, the same calculations as for A_3 and A_4 with our new coefficients and just adjusted event $\neg E_5$ (which is basically just an adaption of events $\neg E_3$ and the first case of event $\neg E_4$), A_5 can derive a solution for the COMDL challenge ξ with probability $1/2$. We omit the whole calculation here again and simply proceed with the next event. Overall, we obtain

$$\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_5} = 1] = \frac{1}{2}\Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \ldots \wedge E_4 \wedge \neg E_5].$$

The bound on the running time of A_5 is obvious.

Algorithm $\mathsf{A}_6(\xi, par)$: Algorithm A_6 works identical as the simulator Sim_4. In particular, the simulation is perfect and the only change is the way the final aggregated PVSS transcript is formed. In this case, the simulator generates the degree-t polynomial $f_{i^*} = d_0 + d_1 X + \ldots + d_t X^t \in \mathbb{Z}_p[X]$ such that $g^{d_j} = \xi_{j+1}$ for all $j \in [\![t]\!]$ (i.e., the $t+1$ coefficients of the polynomial are given by the discrete logarithm values of $\xi_1, \ldots, \xi_{t+1}$). Everything else is generated honestly (in particular, the bulletin board keys and the PVSS transcripts of other parties). As clarified before, we make the assumptions as in the fourth algorithm description A_4. That is, we assume that the adversary chooses its polynomial independent of the honest party P^*'s transcript and thus we may also assume directly that the aggregated transcript hides the polynomial $f = f_{i^*} \in \mathbb{Z}_p[X]$ (i.e., we ignore the shift caused by the adversary's polynomials). We suppose that A_6 wins the game $\mathbf{G}_7$ and that the event $E_1 \wedge \ldots \wedge E_5$ happens. Then the above equation ($\Diamond$) simplifies into the following

$$f(0) - (r_0 + b_{\ell,3}) = \sum_{j \in \mathcal{C}}^{n} r'_j sk_j + \sum_{j \in \mathcal{C}}^{n} t'_j f(j) sk_j$$

$$\iff f(0) = r_0 + b_{\ell,3} + \sum_{j \in \mathcal{C}}^{n} r'_j sk_j + \sum_{j \in \mathcal{C}}^{n} t'_j f(j) sk_j.$$

Having computed that, the only unknown terms dependent on $z_1, \ldots, z_{t+1}$ on the right-hand side of this equation are the $f(j)$ for $j \in \mathcal{C}$. Since $|\mathcal{C}| \leq t$, the algorithm A_6 proceeds as follows. It calls its discrete logarithm oracle DL_g on inputs $g^{f(j)}$ for all $j \in \mathcal{C}$, thus obtaining t new linearly independent equations $f(j) = z_1 + z_2 j + \ldots + z_{t+1} j^t$ (the equations written in matrix form give a Vandermonde matrix which is known to have full rank) with known values $f(j)$. Finally, the above equation allows A_6 to compute $f(0) = z_1$ and thus obtain in total $t+1$ points on the polynomial $f \in \mathbb{Z}_p[X]$ of degree t. Hence, it can efficiently solve for the discrete logarithm values of the COMDL challenge ξ. Overall, we obtain

$$\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_6} = 1] = \Pr[\mathbf{G}_7^{\mathsf{A}} = 1 \wedge \ldots \wedge E_4 \wedge E_5].$$

The bound on the running time of A_6 is obvious. □

To end the proof, consider algorithm B playing in n-**COMDL** as follows: B samples $i^* \leftarrow_{\$} [6]$ and then internally emulates A_{i^*}. Clearly, B is an algebraic algorithm running in time at most T (the running time of A_i, $1 \leq i \leq 6$). An application of the law of total probability yields the following

$$\begin{aligned}
&\Pr[n\text{-}\mathbf{COMDL}^{\mathsf{B}} = 1] \\
&= \frac{1}{6}\sum_{i=1}^{6} \Pr[n\text{-}\mathbf{COMDL}^{\mathsf{A}_i} = 1] \\
&\geq \frac{1}{12} \cdot \Pr[\mathbf{G}_7^{\mathsf{A}} = 1] \\
&\geq \frac{1}{12}\left(\frac{\varepsilon}{Ln} - 4n\varepsilon_B - \frac{q_h}{p} - \frac{q_h^2}{2p}\right).
\end{aligned}$$

As we assume perfect security of the underlying Byzantine agreement protocol, we obtain $\varepsilon_B = 0$ and thus finally

$$\varepsilon \leq Ln\left(12\varepsilon_A + \frac{q_h^2 + q_h}{2p}\right).$$

This concludes the proof of Theorem 2. □