






GRandomLine: Adaptively Secure DKG and Randomness Beacon with (Log-)Quadratic Communication Complexity

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

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

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

Simon Ochsenreither 
s.ochsenreither@gmail.com
Vector Informatik GmbH
Stuttgart, Germany

Dimitrios Papachristoudis 
dimpapach87@gmail.com
Researcher
Saarbrücken, Germany

ABSTRACT

A *randomness beacon* is a source of continuous and publicly verifiable randomness which is of crucial importance for many applications. Existing works on randomness beacons suffer from at least one of the following drawbacks: (i) security only against static (i.e., non-adaptive) adversaries, (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 work, we introduce GRandomLine, 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 $O(\lambda n^2 \log n)$ bits of communication but, unlike most conventional DKG protocols, outputs both secret and public keys as group elements. Here, λ denotes the security parameter. 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 $O(\lambda n^2)$ bits of communication. We implement GRandomLine and evaluate it using a network of up to 64 parties running in geographically distributed AWS instances. Our evaluation shows that GRandomLine 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 OptRand (NDSS '23), BRandPiper (CCS '21), and Drand, in the same setting and observe that it vastly outperforms them.

CCS CONCEPTS

- **Security and privacy** → **Public key (asymmetric) techniques**;
- **Theory of computation** → **Cryptographic protocols**.

KEYWORDS

Adaptive Security, DKG, Randomness Beacon, Aggregatable PVSS, Pairing-Based Cryptography, Transparent Setup

1 INTRODUCTION

Distributed randomness plays a crucial role in many cryptographic and distributed system applications. A randomness beacon [15, 16, 31] 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 [50, 77] they are used to securely choose a leader or a committee among participating parties (e.g., through a verifiable random function [46]) 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 [36], 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 [50], 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 [29, 53]. However, existing randomness beacon protocols found in the literature suffer from at least one of the following drawbacks: (i) they achieve security only against static (i.e., non-adaptive) adversaries, (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 GRandomLine which improves upon the state-of-the-art by combining, for the first time, *all* of the following properties:¹

- *Adaptive Security*. We prove GRandomLine secure in the presence of an adaptive adversary. Many other protocols [46, 71, 75] are only proven statically secure.
- *One-Round Epoch*. Each epoch in GRandomLine takes only a single asynchronous round of communication and is non-interactive. It only requires synchrony for its pre-processing phase. This sets GRandomLine apart from all other protocols except Drand [64].
- *Communication-Efficient*. GRandomLine has a communication cost of $O(\lambda n^2)$ bits per epoch, where λ is the security parameter. This sets GRandomLine apart from BRandPiper [16] and

¹For the sake of clarity, we only compare to adaptively secure protocols in this list.

RandShare [75], which have (worst-case) cubic or higher communication cost per epoch.

- *Optimal Resilience.* GRandLine has optimal resilience threshold $t < n/2$ (i.e., corruption threshold) in the synchronous network. This sets GRandLine apart from SPURT [31] and RandShare [75], which tolerate only suboptimal $t < n/3$.
- *Lightweight Tools.* After its pre-processing phase, GRandLine only uses lightweight cryptography such as hash functions and pairings. Notably, it does not rely on tools such as PoW or VDF. This sets GRandLine apart from RandChain [49] and RandRunner [70], which rely on PoW and VDF, respectively.
- *Quadratic Pre-Processing.* GRandLine has a pre-processing phase with only $O(\lambda n^2 \log n)$ bits communication cost. This sets GRandLine apart from Drand [64], OptRand [15], and BRandPiper [16], which have cubic or higher communication cost for pre-processing.

We achieve our result in two steps. First, we design a novel distributed key generation (DKG) protocol GRand with $O(\lambda n^2 \log n)$ 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 simple construction that allows to use the keys output by GRand for a non-interactive and unique *locally verifiable* threshold signature² from which we naturally derive a one-round randomness beacon using a final hash operation. For a detailed comparison of existing work on randomness beacon and DKG protocols, we refer to Table 1 and Table 2, respectively. Further, recent work [29, 53] give an excellent systematization of knowledge (SoK) for the extensive literature on randomness beacons.

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. [21]. 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 $\{\sigma_i\}_{i \in S}$, a party locally reconstructs the full signature σ . The randomness beacon value is then computed as hash $O_e := H(\sigma)$.

In this manner, one obtains a simple and efficient randomness beacon protocol which is also used by many blockchain and consensus protocols [50, 77]. 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 [22]. 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

²By “locally verifiable” we mean that the final threshold signature does not have an efficient (independent of n) verification algorithm, but the partial signatures have.

most efficient DKG protocols [2, 73] incur a communication cost of $O(\lambda n^3)$ 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 with at most t of them being malicious is as follows. Each party P_i samples a random value $r_i \leftarrow_s \mathbb{Z}_p$ and shares it among all parties using a verifiable secret sharing (VSS) scheme where r_i lies on a 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 [5, 73], 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 $O(\lambda)$) perform the task of sharing a secret. This technique is commonly used in consensus protocols to boost its scalability [46], 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 resilience threshold of $t < n/4$, reliance on secure erasures of internal states, and inefficient primitives such as fully homomorphic encryption (FHE) [45]. Further, to sample the committee in the first place, these 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 [74] 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 required in regular VSS schemes, which greatly simplifies constructions based on PVSS schemes. Recently, Gurkan et al. [48] 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 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, given a known lower bound of $\Omega(n^2)$ communicated messages for Byzantine broadcast [37] (without assuming shared randomness in the first place), their protocol still has cubic communication cost due to the invocation of n instances of Byzantine broadcast. Additionally, their techniques only work against a static adversary that corrupts a mere $\log n$ parties maliciously. 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 on the returned value for the smaller input. This technique has also found application in distributed protocols in order to improve communication cost. The recursive Phase-King

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

Protocol	Network	Resil.	Adapt.	Unpred.	Resp.	Rounds	Commun.	Crypto. Primit.	Setup	Preproc.
Cachin et al. [21]	<i>async</i>	1/3	✗	✓	✓	1	$O(\lambda n^2)$	Uniq. Th. Signature	CRS	$O(\lambda n^3)$
RandHerd [75]	<i>async</i>	1/3	✗	✓	✓	ABA	$O(\lambda c^2 \log n)$	PVSS & Th. Schnorr	CRS	$O(\lambda n^3)$
Herb [27]	<i>sync</i>	1/3	✗	✓	✗	$t + 1$	$O(\lambda n^3)$	Thresh. ElGamal	CRS	$O(\lambda n^3)$
Drand [64]	<i>sync</i>	1/2	✗	✓	✓*	1	$O(\lambda n^2)$	Thresh. BLS	CRS	$O(\lambda n^3)$
SPURT [31]	<i>part sync</i>	1/3	✗	✓	✓	9	$O(\lambda n^2)$	PVSS & Pairing	CRS	✗ [†]
Algorand [46]	<i>part sync</i>	1/3	✗	$\Omega(t)$	✗	BC	$O(\lambda cn)$	VRF	Seed	$O(\lambda n^3)$
HydRand [71]	<i>sync</i>	1/3	✗	$t + 1$	✗	3	$O(\lambda n^2)$	PVSS	Seed	$O(\lambda n^3)$
OptRand [15]	<i>sync</i>	1/2	✗	✓	✓	11	$O(\lambda n^2)$	PVSS & Pairing	q -SDH	$O(\lambda n^3)$
RandShare [75]	<i>async</i>	1/3	✓	✓	✓	ABA	$O(\lambda n^4)$	VSS	CRS	✗
SPURT [10]	<i>part sync</i>	1/3	✓	✓	✓	9	$O(\lambda n^2)$	PVSS & Pairing	CRS & AGM	✗ [†]
RandChain [49]	<i>sync</i>	1/2	✓	$O(\lambda)$	✓	Δ_{PoW}	$O(\lambda n)$	PoW & VDF	CRS	✗ [◦]
RandRunner [70]	<i>sync</i>	1/2	✓	$t + 1$	✗	Δ_{VDF}	$O(\lambda n^2)$	Trapdoor VDF	Seed	$O(\lambda n^3)$
BRandPiper [16]	<i>sync</i>	1/2	✓	✓	✗	11	$O(\lambda f n^2)$	VSS	q -SDH	$O(\lambda f n^3)$
OptRand [10]	<i>sync</i>	1/2	✓	✓	✓	11	$O(\lambda n^2)$	PVSS & Pairing	q -SDH & AGM	$O(\lambda n^3)$
Drand [9]	<i>sync</i>	1/2	✓	✓	✓*	1	$O(\lambda n^2)$	Thresh. BLS	CRS & AGM	$O(\lambda n^3)$
GRandLine	<i>sync</i>	1/2	✓	✓	✓*	1	$O(\lambda n^2)$	PVSS & Pairing	CRS & AGM	$O(\lambda n^2 \log n)$

Resil. denotes the Byzantine resilience threshold. **Adapt.** denotes adaptive adversary. **Unpred.** denotes unpredictability. **Resp.** denotes responsiveness, i.e., progresses at the actual speed of the network.* In Drand and GRandLine, this is achieved only after the pre-processing phase. OptRand is responsive only when there are $t < n/4$ corrupt parties in the system. Asynchronous protocols are by default responsive. **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 $O(\log n)$ 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 [63] and AGM denotes the algebraic group model [42]. **Preproc.** denotes the communication cost for pre-processing. This can either be a DKG, an 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.

protocol [57, 59] for Byzantine agreement is a well-known example for this. The standard, non-recursive Phase-King protocol [14] runs over $t + 1$ phases with a different leader (called the king) in each phase. The protocol succeeds because at least one honest party is guaranteed to be a leader. Interestingly, the need to run over $t + 1$ phases can be avoided using recursion. In the recursive variant, the leader is replaced by one half of the entire system, whose value 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 from cubic to only quadratic.

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 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 for communication-efficient broadcast protocols [61]. On a high level, our protocol works as follows. We split the system of n parties 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 to transmit the transcript to all parties. For this step, we use the techniques developed in [61] 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 . Crucially, contribution from at least one honest party to the aggregate AT provides secrecy guarantees as discussed in [10, 15]. Conversely, such honest contribution is guaranteed by an argument similar to [57, 59], since at least one of the two halves has honest majority (relying on the fact that $t < n/2$).

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 (multiplicative) groups of prime order p with generators $g \in \mathbb{G}_1$ and $h \in \mathbb{G}_2$. We recall that a PVSS transcript generated by some dealing party P_* consists of a vector of commitments $\mathbf{C} := (C_1, \dots, C_n) \in \mathbb{G}_1^n$, a vector of encryptions $\mathbf{E} := (E_1, \dots, E_n) \in \mathbb{G}_2^n$, and some auxiliary data π which usually includes some proof. The commitments are computed 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 distinctive 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 aggregated transcript AT is used to obtain one-time randomness by direct reconstruction of the secret (and possibly subsequent hashing). This is also the case for several previous randomness beacon protocols such as OptRand [15] and SPURT [31]. We are taking a different route inspired by the work [48]. 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 standard Lagrange interpolation in the exponent from C_1, \dots, C_n .

From One to Infinity: Towards a Simple Randomness Beacon. Clearly, GRandom is different from most DKG protocols found in the literature that output secret keys in a field rather than a group. However, a delightful key insight in [48] is that this setup is enough to generate a stream of one-round randomness values. The idea is inspired from the threshold BLS signature [17] 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 $\sigma_i := H_1(m)^{sk_i}$ 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 $\{\sigma_i\}_{i \in S}$ (which is checked by a pairing equation $e(g, \sigma_i) = e(pk_i, H_1(m))$ from P_i 's public key share pk_i), a party locally reconstructs the full signature $\sigma = H_1(m)^{sk}$ by Lagrange interpolation in the exponent and derives the randomness beacon value as another hash $O_e = H_2(\sigma)$. With our DKG protocol GRandom that generates keys (PK_i, SK_i) as group elements, the operation $H_1(m)^{SK_i}$ is not possible. However, when we think of the operation of „raising the group element $H_1(m)$ to a power of sk_i “ as an abstract group action $sk_i \odot H_1(m)$, we realize that the action³ $SK_i \odot H_1(m)$ defined as $e(H_1(m), SK_i) \in \mathbb{G}_T$ is possible. Therefore, we let each party P_i non-interactively create a share σ_i as $e(H_1(m), SK_i)$. And upon receiving $t + 1$ valid such shares, each party can locally reconstruct the full signature by Lagrange interpolation in the exponent as $\sigma = e(H_1(m), SK)$. Again, the randomness beacon value for epoch e is then derived as another hash $O_e := H_2(\sigma)$. 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 $\sigma_i = e(H_1(m), SK_i)$. To resolve this issue, the authors in [48] augment the public keys and shares σ_i with additional elements inspired by Escala-Groth non-interactive zero-knowledge (NIZK) proofs. While their construction in the appendix is reasonably efficient, it still requires a lot of pairings and elements. Further, it lacks an adaptive security proof.

A Simple Two-Step Trick. In order to regain efficiency, we use the following two-step approach. After the DKG setup, each party P_i locally samples an element $\alpha_i \leftarrow_s \mathbb{Z}_p^*$ uniformly at random and sends an ElGamal encryption $cm_i := (g^{\alpha_i}, h^{-\alpha_i} SK_i)$ of its secret key SK_i to all parties. Correctness of its second component can

³Note that this does not define a group action in the mathematical sense. Here, we use the term *group action* only informally to convey the intuition.

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 asymptotic overhead. These additional elements allow parties to verify received beacon shares. Concretely, each party computes $\vartheta_i := (H_1(m)^{\alpha_i}, \sigma_i)$ along with an efficient Chaum-Pedersen NIZK proof of discrete logarithm equality $\pi_i := \text{Dleq}(g, g^{\alpha_i}, H_1(m), H_1(m)^{\alpha_i})$ to prove correctness of $H_1(m)^{\alpha_i}$. 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 $cm_{i,2}$. Having done this, each party can compute the randomness beacon value O_e for epoch $e = m$ as described before. Intuitively, security is preserved because the elements $g^{\alpha_i}, H_1(1)^{\alpha_i}, H_1(2)^{\alpha_i}, \dots$ do not reveal too much information about α_i , thus making it hard for the adversary to compute SK_i from the (randomized) element $h^{-\alpha_i} SK_i$. Overall, an epoch takes only a single round of communication in which each party P_i sends two group elements $\vartheta_i := (H_1(m)^{\alpha_i}, \sigma_i)$ 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 .

Adaptive Security. Our randomness beacon and DKG protocols are both secure under adaptive corruptions. In this model, the adversary can decide which parties to corrupt during the execution of the protocol based on its view of the execution. Particularly for distributed protocols, an adaptive adversary is a safer and more realistic assumption. The standard notion of security for DKG protocols entails a full simulation of the protocol without leaking any additional information (called *fully simulatable*). For our DKG protocol, we do not achieve this property but the weaker notion of *unpredictability* (cf. Definition 4). More importantly, we show that using our DKG protocol for subsequent distributed randomness generation suffices to obtain an adaptively secure randomness beacon protocol (in the random oracle model). In this sense, we follow the methodology of previous works [9] that have shown that weaker security notions for DKG are sufficient to obtain secure randomness beacons and threshold signatures. We emphasize that our randomness beacon is even more efficient than constructions that are only proven statically secure. The rationale behind that is the following. Only very recently, the works [15, 31] discovered the power of aggregatable PVSS (APVSS) for simple and efficient randomness beacons, outperforming previous constructions for even statically secure randomness beacons. A follow-up work [10] then showed their adaptive security by giving an adaptive security proof for their underlying (efficient) APVSS schemes. Inspired by the work [10], we show adaptive security of our APVSS-based randomness beacon construction in the algebraic group model.

Concurrent Work. Concurrent with or subsequent to our work, two other constructions for shared randomness generation have been proposed [32, 41]. The first one [32] focuses on the weighted setting for DKG and threshold verifiable unpredictable function (VUF), but the authors do not consider subcubic DKG protocols. The second one [41] focuses on communication-efficiency in DKG and achieves: (i) a DKG protocol with $O(\lambda n^{2.5} \log n)$ bits of communication cost that outputs secret keys as group elements, and (ii) a DKG protocol with $O(\lambda^2 n^{2.5} \log n)$ bits of communication cost

Table 2: Comparison table of representative distributed key generation (DKG) protocols.

Protocol	Network	Resil.	Adapt.	Commun.	Rounds	Field	Crypto. Primit.	Setup
Kokoris et al. [54]	<i>async</i>	1/3	✓	$O(\lambda n^4)$	$O(n)$	✓	AVSS	CRS
Abraham et al. [3, 43]	<i>async</i>	1/3	✗	$O(\lambda n^3)$	$O(1)$	✗	PVSS, Pairing	CRS
Das et al. [33, 34]	<i>async</i>	1/3	✗	$O(\lambda n^3)$	$O(\log n)$	✓	AVSS	CRS
Bingo [2]	<i>async</i>	1/3	✓	$O(\lambda n^3)$	$O(1)$	✓	AVSS, Pairing	q -SDH, AGM
HARTS [†] [11]	<i>async</i>	1/3	✓	$O(\lambda n^3 \log n)$	$O(1)$	✓	AVSS	CRS, AGM
Shrestha et al. [73]	<i>sync</i>	1/2	✗	$O(\lambda n^3)$	$O(n)$	✓	VSS	q -SDH
Gurkan et al. [48]	<i>sync</i>	$\log n$	✗	$O(\lambda n^3 \log n)$	$O(n)$	✗	PVSS, Pairing	CRS
NI-DKG [◊] [25, 47, 52]	<i>sync</i>	1/2	✗	$O(\lambda n^4)^\diamond$	$O(n)$	✓	PVSS	CRS
Gennaro et al. [44]	<i>sync</i>	1/2	✓	$O(\lambda n^4)^\diamond$	$O(n)$	✓	VSS	CRS, AGM
Canetti et al. [22]	<i>sync</i>	1/2	✓	$O(\lambda n^4)^\diamond$	$O(n)$	✓	VSS, Erasures	CRS
Jarecki et al. [51]	<i>sync</i>	1/2	✓	$O(\lambda n^4)^\diamond$	$O(n)$	✓	PVSS	CRS
GRand [our work]	<i>sync</i>	1/2	✓	$O(\lambda n^2 \log n)$	$O(n)$	✗	PVSS, Pairing	CRS, AGM

Resil. denotes the Byzantine resilience threshold. **Adapt.** denotes adaptive adversary. The protocol [44] was proven adaptively secure in the AGM [9]. **Comm.** denotes the communication cost in bits. \diamond Most protocols [22, 44, 51] assume a broadcast channel, which we implement with the commonly-used Dolev-Strong broadcast protocol [38]. However, we note that it is possible to achieve cubic communication cost by using an optimal broadcast protocol. **Rounds** denotes the number of (a)synchronous rounds to terminate. For asynchronous protocols, this is the expected number of rounds (as these protocols are randomized). **Field** denotes if the secret key is a field element or not (group element). **Crypto. Primit.** denotes the cryptographic primitives in usage. The protocol [22] relies on secure erasure of secret states. **Setup** denotes the setup assumptions, including idealized models. \circ The protocols [25, 47, 52] follow the common technique [51], where each party broadcasts a PVSS transcript for a field element, and primarily focus on the concrete computational efficiency of the PVSS scheme (which are rather inefficient compared to PVSS schemes for group elements). \dagger We note that HARTS [11] can generate a batch of up to $t + 1$ independent keys with cubic communication cost and thus has amortized quadratic communication cost per key.

that outputs secret keys as field elements. Further, none of these works consider adaptive adversaries.

1.2 Outline of the Paper

The rest of the paper is organized as follows. In Section 2, we define our model and relevant preliminaries, including cryptographic and consensus primitives. In Section 3, we present our new DKG protocol GRand. In Section 4, we present our one-round randomness beacon GRandLine on top of GRand. In Section 5, 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 cover additional preliminaries relevant for the paper. In Appendix C, we present figures for some of the building blocks of GRand. In Appendix D, we provide a security and complexity analysis for our randomness beacon and DKG protocols.

2 PRELIMINARIES AND MODEL

In this section, we fix the model and preliminaries for our paper. Throughout the paper, we consider a complete network \mathcal{P} of n parties P_1, \dots, P_n connected by pairwise authenticated channels, i.e., the receiver of a message is aware of the sender’s identity.

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, $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ 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 $[a, b] := \{a, \dots, b\}$; if $a = 1$, we write this set as $[b]$, and if $a = 0$, we write it as $\llbracket b \rrbracket$.

For an element x in a set S , we write $x \leftarrow_s S$ 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 $x \leftarrow A(x_1, \dots, x_n)$ we mean running algorithm A on inputs (x_1, \dots, 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 $x \leftarrow A^B(x_1, \dots, x_n)$. We write G^A to denote the output of the game G involving algorithm A . We measure the communication complexity of our distributed protocols in bits.

Network Model. We assume a *synchronous* network model, i.e., communication 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 consider an adversary who can corrupt up to $t = \lceil n/2 \rceil - 1$ parties maliciously and may cause them to deviate from the protocol arbitrarily. The adversary is *strongly adaptive* and can choose its corruptions at any time during the execution of the protocol. When it corrupts a party, it can delete or substitute any undelivered messages that this party previously sent while being correct. Note that once a party is corrupted, it remains corrupted. Further, we assume that the adversary is in full control over message delays, subject to the network delay Δ . In particular, the adversary is *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*.

Public Key Infrastructure. We assume that the parties have established a bulletin board public key infrastructure (PKI) before

the protocol execution. Concretely, 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 but sk_i is known only to P_i . 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, sik_i) 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 [39], we treat signatures as information-theoretic objects with perfect unforgeability and perfect correctness (cf. Appendix B).

Idealized Models. We assume the random oracle model (ROM) [13]. In this model, a hash function H is treated as an idealized random function to which the adversary gets oracle access. Further, we assume the algebraic group model (AGM) [42]. In this model, all algorithms are treated as algebraic (over a group \mathbb{G}): whenever an algorithm A outputs a group element $\zeta \in \mathbb{G}$, it additionally outputs a vector $\mathbf{z} = (z_1, \dots, z_k)$ of integers such that $\zeta = \prod_{i \in [k]} g_i^{z_i}$, where $(g_1, \dots, g_k) \in \mathbb{G}^k$ is the list of group elements A has received so far (either as input or as oracle responses).

Computational Assumptions. We rely on the *co-one-more discrete logarithm (co-OMDL) assumption* [10] for our security proofs. This is a generalization of the standard one-more discrete logarithm (OMDL) assumption to capture also type 3 bilinear groups as used widely in practice for efficiency reasons. Throughout the paper, we denote by DL_g an oracle that on input an element $\xi := g^z \in \mathbb{G}_1$ returns the discrete logarithm z of ξ to base g .

DEFINITION 1 (CO-OMDL PROBLEM). *Let $(\mathbb{G}_1, \mathbb{G}_2, p, g, h)$ be cyclic groups of prime order p as specified by par. For an algorithm A and $k \in \mathbb{N}$, we define the experiment $k\text{-COMDL}^A$ as follows:*

- **Offline Phase.** *Sample $(z_1, \dots, z_k) \leftarrow_s \mathbb{Z}_p^k$ uniformly at random and set $\xi_i := (g^{z_i}, h^{z_i}) \in \mathbb{G}_1 \times \mathbb{G}_2$ for all $i \in [k]$.*
- **Online Phase.** *Run A on input $(\mathbb{G}_1, \mathbb{G}_2, p, g, h)$ and (ξ_1, \dots, ξ_k) . In this phase, A gets access to the oracle DL_g .*
- **Winning Condition.** *Let (z'_1, \dots, z'_k) denote the output of A . Return 1 if (i) $z'_i = z_i$ for $i \in [k]$, and (ii) DL_g was queried at most $k - 1$ times during the online phase. Otherwise, return 0.*

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

2.1 Cryptographic Primitives

In this section, we define syntax and security notions for the cryptographic primitives used in the paper.

(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 2 (AGGREGATABLE PVSS SCHEME). *Let \mathbb{G} be a cyclic group of prime order p specified by par. A (t, n) -threshold aggregatable PVSS (APVSS) scheme over \mathbb{G} is a tuple of algorithms $APVSS = (\text{Keys}, \text{Enc}, \text{Dec}, \text{Dist}, \text{Agg}, \text{ConId}, \text{Ver}, \text{Rec})$ such that:*

- **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 , $\Pr[\text{Dec}_{sk_i}(\text{Enc}_{pk_i}(m)) = m] = 1$.*
- **Dist:** *The randomized secret sharing algorithm takes as input a secret key sk_i and public keys pk_1, \dots, pk_n . It outputs a vector of encrypted shares $\mathbf{E} = (\text{Enc}_{pk_1}(S_1), \dots, \text{Enc}_{pk_n}(S_n))$ and a proof π , where S_1, \dots, S_n are shares of a secret $S \in \mathbb{G}$. We refer to $T := (\mathbf{E}, \pi)$ as a PVSS transcript.*
- **Agg:** *The deterministic aggregation algorithm takes as input PVSS transcripts $(\mathbf{E}_1, \pi_1), \dots, (\mathbf{E}_k, \pi_k)$, $k \in \mathbb{N}$. 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 .⁴*
- **Ver:** *The deterministic verification algorithm takes as input public keys pk_1, \dots, 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; otherwise we call it invalid.*
- **Rec:** *The deterministic reconstruction algorithm takes as input $t + 1$ shares S_1, \dots, S_{t+1} . It outputs a secret $S \in \mathbb{G}$.*

Discussion. We defer formal definitions for correctness and secrecy of an APVSS scheme to Appendix B.1. Our definition above (and those in the appendix) are based on the ones from [10]. Essentially, the only difference to their definitions is the following. Their aggregation algorithm 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 [10] (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.

⁴We remark that ConId could return 1 on an invalid transcript.

Linear Erasure and Error Correcting Codes. We use standard (q, b) -Reed-Solomon (RS) codes [68]. This primitive allows to encode b data symbols into code words of q symbols (using the algorithm Encode) such that b elements of the code word suffice to recover the original data (using the algorithm Decode). In our DKG construction, we will use Reed-Solomon codes with varying (q, b) . Concretely, we use codes with q being the number of parties in some designated subset of parties $Q \subseteq \mathcal{P} = \{P_1, \dots, P_n\}$ (called a *committee*) and b being $\lceil q/2 \rceil$. In the special case $Q = \mathcal{P}$, we have $(q, b) := (n, t + 1)$. We defer formal definitions to Appendix B.1.

Cryptographic Accumulator. A cryptographic accumulator [62] allows to accumulate several elements from some set D into an accumulated value z (using the algorithm Eval). Further, for each element in D it allows to generate a compact proof of membership in D (using the algorithm Wit) called a witness. The standard security notion of collision-resistance requires that it is hard for an adversary to create invalid proofs of membership. An example of cryptographic accumulators are Merkle trees, where the root is the accumulation value and the authentication paths are membership proofs (i.e., witnesses) for the leaves. In this paper, we use an accumulator scheme with membership proofs and accumulation value each of size $O(\lambda)$. This can be implemented using the accumulator scheme of [19] built upon class groups of unknown order. Alternatively, we can use Merkle trees at the cost of $O(\log n)$ multiplicative overhead in the communication complexity. We defer formal definitions to Appendix B.1.

2.2 Consensus Primitives

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

Byzantine Agreement. A Byzantine agreement (BA) protocol [56] allows a set of parties, each holding an input $v_i \in V$ from a value set V with $|V| \geq 2$, to agree on a common output value $v \in V$ that was input from at least one honest party. In our definition, we also account for some probability of failure ε which corresponds to the adversary’s ability in breaking underlying cryptographic tools.

DEFINITION 3 (BYZANTINE AGREEMENT). *Let Π be a protocol executed by n parties P_1, \dots, P_n , where each party P_i holds an input value $v_i \in V$. We define the following properties for Π which each holds with probability at least $1 - \varepsilon$ in the presence of an adversary corrupting at most t parties:*

- **Validity.** Π is (t, ε) -valid if the following holds: if every honest party has the same input value v as input, then every honest party outputs this value v .
- **Consistency.** Π is (t, ε) -consistent if the following holds: every honest party that outputs a value outputs the same value v .
- **Termination.** Π is (t, ε) -terminating if the following holds: every honest party terminates with an output value $v \in V$.

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

Distributed Randomness Beacon. A randomness beacon is a distributed protocol that allows a set 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 $\Sigma_i = (\Sigma_i[1], \Sigma_i[2], \dots)$ 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 elaborate on these security notions. 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 (cf. GRandPiper [16] and Hydrand [71]) without actually having the power to bias it. We defer formal definitions to Appendix B.2.

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 DKG protocols that output secret keys in a field rather than a group. However, as demonstrated delightfully by Gurkan et al. [48], this is enough for applications such as efficient randomness beacons. We first formally define a DKG protocol.

DEFINITION 4 (DKG PROTOCOL). *Let Π be a protocol executed by n parties P_1, \dots, 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, \dots, PK_n) , and a public key PK . We define the following properties for Π which each holds with probability at least $1 - \varepsilon$ in the presence of an adversary corrupting at most t parties:*

- **Consistency.** Π is (t, ε) -consistent if the following holds: all honest parties output the same public key PK and the same vector of public key shares (PK_1, \dots, PK_n) .
- **Correctness.** Π is (t, ε) -correct if the following holds: there exists a deterministic algorithm Rec that on input any set of $t + 1$ secret key shares $\{SK_i\}_{i \in I}$ outputs the same unique secret key SK . Further, SK is a valid secret key for PK .
- **Secrecy.** Π is (t, ε, T) -secret if for all algorithms A that run in time at most T , its success probability in the following experiment is at most ε .
 - **Offline Phase.** Initialize a corruption index set $C := \emptyset$ and let $\mathcal{H} := [n] \setminus C$. 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 update $C := C \cup \{i\}$. Henceforth, A has full control over P_i .
 - **Online Phase.** Initiate an execution of Π with A having full control over parties in C . Let $y := PK \leftarrow \Pi$ be the public key output by honest parties, and $x := SK$.
 - **Winning Condition.** Let S^* denote the output of A . Then, A is considered successful iff $|C| \leq t$ and $S^* = x$.

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

3.1 Components of our DKG Protocol

In this section, we describe the building blocks that will be used in the construction of our DKG protocol GRand. Although we instantiate these building blocks with specific schemes, our construction of GRand in the next section is done in an abstract way such that these building blocks can also be instantiated with other such schemes.

Aggregatable PVSS Scheme. We will make use of an APVSS scheme in the construction of GRand. Concretely, we instantiate this with our APVSS scheme given in Figure 6 (cf. Appendix C). Our APVSS scheme is similar to the one of Bhat et al. [15] which is essentially SCRAPE [23] 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 only difference is that the reconstructed secret in our scheme is $S := h^\alpha$, whereas the one in their scheme is $S' := e(\hat{g}, h^\alpha)$ where $\hat{g} \in \mathbb{G}_1$ is an additional generator. This choice is motivated by their security analysis, which is a reduction from the co-decisional bilinear squaring (co-DBS) problem. 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 further generators in the source groups. Finally, aggregated unpredictability of our APVSS scheme follows directly from aggregated unpredictability of their APVSS scheme (cf. [10] for a proof of the latter), since any prediction $S^* := h^\alpha$ for the former gives a prediction $S'^* := e(\hat{g}, S^*)$ for the latter.

Byzantine Agreement Protocol. We will make use of a BA protocol in the construction of GRand. Concretely, we instantiate this with the BA protocol given in Appendix C.1. Essentially, this is merely a variant of the BA protocol of Momose and Ren [59]. Their protocol has optimal resilience $t < n/2$ and achieves a communication complexity of $O(\lambda n^2)$ bits assuming threshold signatures of size $O(\lambda)$ from a trusted setup. The threshold signatures are used to prove knowledge of a threshold of signatures from other parties on the same message. Instead, we implement these threshold signatures with the recent transparent-setup threshold signatures of Attema et al. [7] at the cost of multiplicative logarithmic overhead in the communication complexity.⁵ Importantly, this scheme has the following desirable features: (i) It does not require a trusted setup phase, i.e., all public parameters are random coins. (ii) The k -aggregation algorithm can be evaluated by any party with input at least k valid signatures from distinct signers, and it only takes the signatures and public values as input. (iii) It allows for any threshold $k \leq n$ which can be chosen by the aggregator at aggregation time independent of the setup phase. (iv) It is non-interactive, correct, and unforgeable against an adaptive adversary. Having said this, our resulting Byzantine agreement protocol has a communication complexity of $O(\lambda n^2 \log n)$ and terminates in a linear number of rounds. For more details on the transparent threshold signatures, we refer to the original work [7]. Further, we briefly discuss the security guarantees of the resulting BA protocol in Appendix C.1.

⁵When the network of parties is of small size $n \in O(\lambda)$, we can instead directly use an aggregated BLS signature augmented with the n -bit long vector of signers.

Deliver Protocol. We will make use of a protocol in the construction of GRand that allows parties to efficiently broadcast a long message. Concretely, we instantiate this with the protocol Deliver given in Figure 5 (cf. Appendix C). The protocol design was introduced in [16, 61] 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 Q . In contrast to [16, 61], we make use of Deliver for sets of varying sizes. We parameterize Deliver by a set Q of q parties among which it is executed. It is invoked by a party $P_i \in Q$ and takes as input a long message m , the accumulation value z for an encoding of m , and Q along with implicit parameters $q = |Q|$ 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 $P_j \in Q$. Upon receiving a valid triple of this type, P_j forwards it to all other parties in 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 Q when the sender P_i was honest. Finally, we note that correctness of Deliver is implied by collision-resistance of the underlying cryptographic accumulator scheme, since the only way to reconstruct a different message $m' \neq m$ is by receiving a witness for non-membership. For more details, we refer to [61].

3.2 Design of our DKG Protocol

In this section, we present our DKG protocol GRand. At its heart lies a recursive protocol GenAPVSS that allows parties to aggregate several PVSS transcripts with contribution from at least one honest party in an efficient manner. From such an aggregated PVSS transcript parties can locally derive their secret key shares without any further interaction between them.

Recursive PVSS Aggregation. We give an informal description of the protocol GenAPVSS (cf. Figure 1). We parameterize GenAPVSS by a set Q of q parties among which it is executed. Upon termination of the protocol, all parties output a common, single PVSS transcript AT which is obtained by aggregation of two transcripts. The high-level idea of the protocol is to split the system Q of all parties into two disjoint sets (called *committees*) Q_1, Q_2 of roughly equal size, let each committee Q_i , $i \in [2]$, run the protocol among themselves, and then broadcast the resulting PVSS transcript T_i to the other committee Q_{1-i} . All parties then terminate with the aggregation $AT := \text{Agg}(T_1, T_2)$. In more detail, this works as follows.

Let $Q = Q_1 \cup Q_2$ be a (deterministic) partition of Q into two disjoint sets called committees. For each $i \in [2]$, run the protocol GenAPVSS among parties in Q_i and let T_i denote the common output. Now, instead of directly sending the whole transcript T_i to all parties in the opposite committee Q_{1-i} , each party sends only a much shorter accumulation value z_i for T_i . To counteract malicious behavior, all parties in Q establish consensus on both accumulation values z_1, z_2 via two separate instances of a Byzantine agreement protocol BA. Note that at this stage, parties in Q_i do not know anything about the opposite committee Q_{1-i} 's transcript T_{1-i} other than the accumulation value z_{1-i} and vice versa. Therefore, for each $i \in [2]$, parties in Q_i next broadcast their transcript T_i to all parties

Let $Q \subseteq \mathcal{P}$ be a set of q parties and let $b := \lceil q/2 \rceil$. Further, let $Q = Q_1 \cup Q_2$ be a (deterministic) partition of Q into two disjoint subsets (each called a *committee*) of size $q_1 := \lceil q/2 \rceil$ and $q_2 := \lfloor q/2 \rfloor$, respectively. Hereafter, we use the notation $\langle m \rangle := \text{Encode}(m_1, \dots, m_b)$ for a (q, b) -Reed-Solomon code $RS = (\text{Encode}, \text{Decode})$ and where (m_1, \dots, m_b) is a deterministic partition of m . We describe the protocol from the view of party P_i and let $l \in \{1, 2\}$ be such that $P_i \in Q_l$.

- **Initialization.** Initialize empty lists C, T, Z of length 2 and set $Z[j] := \perp$ (default value) for $j \in \{1, 2\}$. // *Variables are defined.*
- **Recursive Execution.** Run $\text{GenAPVSS}(Q_l)$ among parties in Q_l and let T_l denote the output. If $|Q_l| = 1$, then obtain T_l by locally executing $T_l \leftarrow \text{Dist}(sk_i, (pk_1, \dots, pk_n))$. // *Both committees run the protocol recursively and output a PVSS transcript each.*
- **Accumulator Delivery.** Compute an accumulation value z_l for the encoding $\langle T_l \rangle$ and send z_l to all parties in Q . Upon receiving the same value z_j from $\lfloor q_j/2 \rfloor + 1$ distinct parties in Q_j (i.e., a majority set of parties), update $Z[j] := z_j$ for each $j \in \{1, 2\}$ at most once. // *Parties only accept the majority value z_j received from each committee Q_j .*
- **Accumulator Agreement.** For each $j \in \{1, 2\}$, run Byzantine agreement BA_j on input $Z[j]$ among parties in Q . Let z_j denote the output, and update $Z[j] := z_j$. // *Parties establish consensus on both accumulation values z_1, z_2 (one from each committee).*
- **Transcript Delivery.** If $Z[l] \neq \perp$, then invoke Deliver on input $(Q, T_l, Z[l])$ among parties in Q . Further, only participate in another instance of Deliver with respective accumulation value $z \neq \perp$ if $z \in Z[\cdot]$. Upon decoding a message T_j (with accumulation value $Z[j]$), update $T[j] := T_j$ for each $j \in \{1, 2\}$ at most once. // *Parties efficiently broadcast their T_j to all parties in Q .*
- **Committee Selection.** For each $j \in \{1, 2\}$, update $C[j] := \text{Ver}(T[j], (pk_1, \dots, pk_n)) \in \{0, 1\}$. For each $j \in \{1, 2\}$, run (binary) Byzantine agreement BA_j on input $C[j]$ among parties in Q . Let b_j denote the output bit, and update $C[j] := b_j$. // *Parties decide on which committee(s) have correctly delivered a valid PVSS transcript T_j with resp. accumulation value z_j .*
- **Transcript Agreement.** For each $j \in \{1, 2\}$ such that $C[j] = 1$, invoke Deliver on input $(Q, T[j], Z[j])$ among parties in Q . Upon decoding a message T_j (with respective index j) such that $C[j] = 1$ and $\text{Ver}(T_j, (pk_1, \dots, pk_n)) = 1$, update $T[j] := T_j$ at most once. // *This ensures that T_j reaches all honest parties in Q .*
- **Final Aggregation.** Compute the aggregation $AT := \text{Agg}(T[1], T[2])$ and output. // *Aggregate both transcripts and terminate.*

Figure 1: Description of our APVSS transcript generation protocol GenAPVSS for the set $Q \subseteq \mathcal{P}$ from the view of party P_i .

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 $\text{GenAPVSS}(\mathcal{P})$ among all parties in \mathcal{P} and obtain a PVSS transcript $AT := \{C, E, \pi\}$ from the execution. // *This generates a common (aggregated) PVSS transcript AT for all parties in \mathcal{P} .*
- **Key Derivation.** Compute the decryption $D_i := \text{Dec}_{sk_i}(E_i)$. 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 the PVSS transcript AT without further interaction. In particular, these key shares interpolate a degree- t polynomial $f \in \mathbb{Z}_p[X]$ in the exponent.*

Figure 2: Description of our DKG protocol GRand from the view of party P_i .

in Q using the protocol Deliver on input (Q, T_i, z_i) . This ensures efficient delivery of the large transcript T_i to all other parties. Since an adversarial-controlled committee could simply refuse to deliver its transcript to some of the honest parties, we introduce two further steps to maintain consistency. First, parties in Q decide on whether the previous step succeeded via two separate instances of binary Byzantine agreement, one to decide for each committee. Second, parties proceed with another invocation of Deliver to guarantee that all honest parties obtain the transcripts among $\{T_1, T_2\}$ for which the respective Byzantine agreement execution output 1. Parties conclude the protocol with aggregation of these transcripts and terminate with $AT := \text{Agg}(T_1, T_2)$ as output.

Our DKG Protocol. A formal description of the protocol GRand is given in Figure 2. 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 P_i computes its secret share $D_i := \text{Dec}_{sk_i}(E_i)$ 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 6, 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 polynomial of degree t encoded in the APVSS transcript AT . We note that even though P_i knows $h^{f(i)}$, it does not know $f(i)$ itself. In particular, this DKG protocol is different from many DKG protocols in the literature where the hidden polynomial itself is distributed among the parties.

3.3 Security and Complexity Analysis

In this section, we give a security and complexity analysis of our DKG protocol GRand . In the following, let APVSS be an aggregatable PVSS scheme, let BA be a Byzantine agreement protocol, and let AC be a cryptographic accumulator scheme. Then, assuming aggregated unpredictability of APVSS, security of BA, and collision-resistance of AC, this implies security of GRand . Further, it has log-quadratic communication complexity and linear round complexity when instantiated with our components from Section 3.1. For a full proof of the following theorem, we refer to Appendix D.1.

Let $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ be a pairing with generators $g \in \mathbb{G}_1, h \in \mathbb{G}_2$. Let $H_1 : \{0, 1\}^* \rightarrow \mathbb{G}_1$ and $H_2 : \mathbb{G}_T \rightarrow \{0, 1\}^\lambda$ be two cryptographic hash functions modeled as random oracle. Hereafter, let $g_r := H_1(r)$ for all $r \in \mathbb{N}$.

- **Setup Phase.** Parties execute the DKG protocol GRandom 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 an empty local set $\mathcal{G} := \emptyset$. Sample $\alpha_i \leftarrow_{\mathcal{S}} \mathbb{Z}_p^*$ uniformly at random and multicast (i.e., send to all parties) the commitment $\text{cm}_i := (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Upon receiving $\text{cm}_j = (\text{cm}_{j,1}, \text{cm}_{j,2})$ from party P_j , check if equality $e(PK_j, h) = e(\text{cm}_{j,1}, h)e(g, \text{cm}_{j,2})$ holds. Only if this equality holds, update the set $\mathcal{G} := \mathcal{G} \cup \{P_j\}$. // 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 $\pi_i := \text{Dleq}(g, g_r^{\alpha_i}, g_r, g_r^{\alpha_i})$, 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 $\text{cm}_{j,1}$ and $\sigma_{j,1}$. Further, check if $\sigma_{j,2} = e(g_r, \text{cm}_{j,2})e(\sigma_{j,1}, h)$.
- **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 without interaction.
- **Beacon Output.** Upon reconstruction of σ in epoch r , output the beacon value as $\varrho_r := H_2(\sigma) \in \{0, 1\}^\lambda$. // The beacon value is output as soon as $t + 1$ valid tuples are received.

Figure 3: Description of the randomness beacon protocol GRandomLine from the view of party P_i .

THEOREM 1. *If APVSS is $(t, \varepsilon_A, T_A, q_S)$ -aggregated unpredictable, if BA is (t, ε_B, T_B) -secure, and if AC is (n, ε_C, T_C) -collision-resistant, then GRandom (cf. Figures 1 and 2) is a (t, ε, T) -secure DKG protocol, where*

$$\varepsilon \leq 2n(\varepsilon_A + 2\varepsilon_B + n\varepsilon_C), \quad T \geq T_A + T_B + T_C + O(n^2).$$

Using the components in Section 3.1, GRandom has a communication complexity of $O(\lambda n^2 \log n)$ bits and terminates in $O(n)$ rounds.

4 DISTRIBUTED RANDOMNESS BEACON

In this section, we design an efficient and simple randomness beacon protocol. The construction is inspired by the threshold VUF design of Gurkan et al. [48] and can be thought of as an optimized version of their protocol that comes with an adaptive security proof. As such, our protocol is comparable with the threshold BLS signature.

4.1 Design of our Randomness Beacon

In this section, we present our randomness beacon GRandomLine. For a formal description of the protocol, we refer to Figure 3 below. Let H_1 and H_2 be hash functions modeled as random oracle and denote $g_r := H_1(r)$ for all $r \in \mathbb{N}$. Parties begin by executing GRandom 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 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 $H_2(\vartheta) \in \{0, 1\}^\lambda$. 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 $\alpha_i \leftarrow_{\mathcal{S}} \mathbb{Z}_p^*$ uniformly at random and multicasts (i.e., sends to all parties) $\text{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. 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 Chaum-Pedersen NIZK proof of discrete logarithm equality [26] as $\pi_i := \text{Dleq}(g, g_r^{\alpha_i}, g_r, g_r^{\alpha_i})$ 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

In this section, we give a security and complexity analysis of our randomness beacon protocol GRandomLine. On a high level, consistency and availability follow from uniqueness of the signature $\vartheta := e(H_1(r), SK)$ (per message r and public key PK) and soundness of the Chaum-Pedersen NIZK proof system for discrete logarithm equality. Unpredictability follows from unforgeability of ϑ , and the final hash operation $H_2(\vartheta)$ guarantees uniformity in the random oracle model. Essentially, these are the standard arguments for the transformation from unique threshold signatures to randomness beacons [21], but with the additional argument of soundness of NIZK proofs for correctness of the threshold signature. Using known techniques [9] to handle adaptive corruptions, we give a security reduction from the hardness of n -COMDL to the unforgeability of the threshold signature.

Intuitively, the adversary essentially has three options to successfully forge a signature $\vartheta^* = e(H_1(r^*), SK)$ on some message (i.e., epoch number) r^* . It either finds the secret key SK , the encryption secret α_i for an honest party's $i \in [n]$, or the discrete logarithm of the element $H_1(r^*)$. But this should be infeasible given secrecy of the underlying DKG protocol and the ElGamal encryption used for commitments $\text{cm}_i := (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Further, the output BLS signatures $H_1(1)^{\alpha_i}, H_1(2)^{\alpha_i}, \dots$ do not reveal additional information. We make this intuition sound by building a reduction that embeds the n -COMDL challenge ξ in either the PVSS transcript of some party (that remains honest at the end), in the ElGamal encryption

secrets $\alpha_1, \dots, \alpha_n$ of parties, or in the random oracle outputs $H_1(\cdot)$, a choice that remains hidden from the adversary. Leveraging the algebraic group model, we are able to solve ξ using the polynomial equations that come from the forgery and additional data output by the adversary. For a full proof of the following theorem, we refer to Appendix D.2.

THEOREM 2. *If n -COMDL is (ϵ_A, T_A) -hard in the AGM and BA is (t, ϵ_B, T_B) -secure, then GRandLine (cf. Figure 3) is a $(t, \epsilon, T, L, q_h, 1)$ -secure randomness beacon protocol in the AGM + ROM, where*

$$\epsilon \leq Ln \left(12\epsilon_A + 4n\epsilon_B + \frac{q_h^2 + 4q_h}{2p} \right), \quad T \geq T_A + T_B + O(Ln^2).$$

Further, GRandLine has a communication complexity of $O(\lambda n^2)$ bits per epoch, and each epoch takes one asynchronous round.

5 IMPLEMENTATION & EVALUATION

In this section, we evaluate the performance of our randomness beacon protocol GRandLine for various network sizes. Concretely, we evaluate its throughput (i.e., the number of beacon values output per second) and compare it to existing state-of-the-art randomness beacons in the same setting: OptRand [15], BRandPiper [16], and Drand [64]. 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 [6] 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 [6]. We use a custom, optimized APVSS scheme implementation for our underlying cryptographic operations [65] and tokio [30] for networking. The implementation follows strictly the description in Figure 3 and it is publicly available at our GitHub repository [66].

An important note: We emphasize that we have not implemented our DKG protocol and instead manually set up the keys of parties for our experiments. The same is also true for the other randomness beacon protocols of interest, BRandPiper and OptRand. Their publicly available implementations [58, 72] have the output generated from a potential pre-processing phase already configured. Without major modifications on their codes, it is not possible for us to run and evaluate their pre-processing phase. That being said, we likewise decided to not implement our DKG protocol.

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 [58], Drand [64] and OptRand [72]. 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 cryptographically heavy tools such as proof-of-work or (trapdoor) VDFs. For a comparison of the computation costs, we refer to Table 3.

5.3 Evaluation Results

Similar to previous work [31], we run each experiment three times for about 10 minutes each and took the average over these three runs. Additionally, we note that many other previous works [12, 15, 16, 71] do not specify the number of runs for their experiments (which could therefore potentially be only a single run). We report the throughput of GRandLine and the comparative randomness beacons as the number of beacon outputs per second in Figure 4.

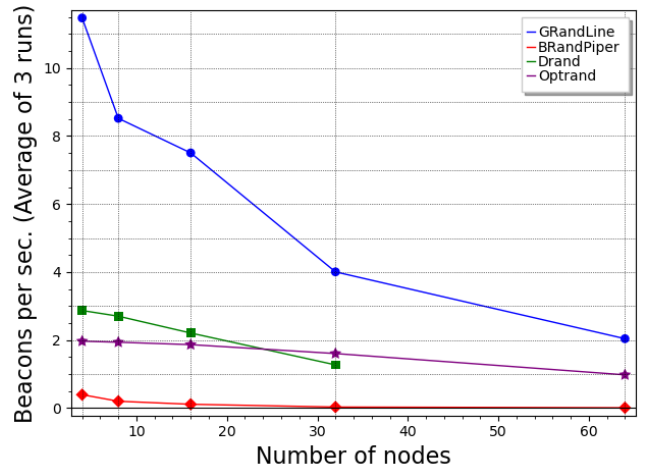


Figure 4: Performance graph for the randomness beacon protocols: OptRand, BRandPiper, Drand, and GRandLine.

GRandLine. Compared to all three baselines, GRandLine outputs beacons at significantly higher rates. In particular, our evaluation results show that with 4, 8, 16, 32, and 64 nodes, GRandLine generates on average 11, 8, 7, 4, and 2 beacons per second, respectively.

Table 3: Computation cost per epoch for non-leader, leader parties.

Protocol	Balanced	Rounds	Pairings	NIZKs (Ver)	NIZKs (Gen)	Exponents	Public Ver
Drand [64]	✓	1	$2n$	0	0	$t + 2$	2, 0
OptRand [15]	✗	11	$4n + 1, 2n + 1$	$n, n^2 + n$	$n + 1$	$2n + t + 2$	$2n + 3, 3t + 3$
BRandPiper [16]	✗	11	$4n, 2n$	0	$0, n^2$	n, n^2	$2n, t + 1$
G RandLine	✓	1	$2n + 1$	n	1	$t + 2$	$2t + 2, 4n + t + 1$

Balanced denotes balanced computation cost across all parties. **Rounds** denotes the number of rounds per epoch. **Pairings** denotes the number of pairing operations performed by each non-leader, leader party. **NIZKs (Ver)** denotes the number of NIZK verifications performed by each non-leader, leader party. OptRand and GRandLine employ Chaum-Pedersen proofs of discrete logarithm equality for the NIZKs. **NIZKs (Gen)** denotes the number of NIZKs generated by each non-leader, leader party. BRandPiper employs KZG proofs for the NIZKs whose verification uses pairings. **Exponents** denotes the number of group exponentiations performed by each non-leader, leader party. For Drand and GRandLine, this can be done using one multi-exponentiation and one regular exponentiation. For OptRand, this can be done using one multi-exponentiation and $2n + 1$ regular exponentiations. For BRandPiper, this can be done using one resp. n multi-exponentiation(s) by a non-leader resp. leader party. **Public Ver** denotes the number of pairings, group exponentiations performed to publicly verify an epoch beacon output. Here, we directly incorporate the NIZK verifications into the group exponentiation count.

The average time between generating two consecutive beacons is 87.19ms, 117.37ms, 133.29ms, 249.65ms, and 489.89ms, respectively. We recall that each experiment was run for about 10 minutes.

OptRand. The protocol proceeds through epochs of up to 11 rounds, where each epoch $e \geq 1$ is delegated by a different leader L_e chosen in a round-robin fashion. In each epoch $e \geq 1$, the leader L_e is instructed to collect and aggregate $t + 1$ valid PVSS transcripts that other parties send to it at the beginning of the epoch. Later, parties collectively reconstruct the secret S_e of the aggregated transcript and compute the beacon value O_e for epoch e as hash $O_e := H(S_e)$. Further, the protocol relies on an initial setup where parties start with agreed-upon buffers $\mathcal{B}(P_i)$ for all $i \in [n]$ that contain random PVSS transcripts each. In their public implementation [72] this phase is skipped and already configured. Finally, the protocol has two modes of operation: an optimistically-responsive mode and a non-optimistic mode. The former allows for responsive progress when there are $t < n/4$ actual corrupt parties in the system, which results in much higher throughput compared to the latter.

We test only against OptRand’s optimistically-responsive variant, since its non-optimistic variant performs comparably to BRandPiper (cf. [15] for a discussion on that). 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 compared to GRandLine where the computation cost is balanced across all nodes (cf. Table 3). Further, OptRand’s reference implementation is instantiated with the BN128 pairing-friendly curve from libff [55]. Unfortunately, while this curve allows OptRand to benefit from more efficient modular operations, it is below acceptable security standards [69]. Concretely, it provides only 100-bit security level. By instantiating GRandLine with a similar curve, specifically arkworks’ ark-bn254 crate [1], we estimate that our protocol is capable of generating roughly twice the number of beacons compared to what is shown in Figure 4.

BRandPiper. The protocol proceeds through epochs of 11 rounds, where each epoch $e \geq 1$ is delegated by a different leader L_e chosen in a round-robin fashion. In each epoch $e \geq 1$, the leader L_e shares n randomly chosen secrets $\mathbf{s}_e = (s_{e,1}, \dots, s_{e,n}) \in \mathbb{Z}_p^n$ among all parties via an efficient multi-secret VSS scheme. Later, parties collectively reconstruct the ephemeral randomness R_e as a sum of n secrets $s_{e,i}, \dots, s_{e-n+1,i}$ from n consecutive leaders (i.e., one secret from

each leader). The randomness beacon value O_e for epoch e is then computed as hash $O_e := H(R_e)$. Further, the protocol relies on an initial setup where parties start with agreed-upon buffers $\mathcal{B}(P_i)$ for all $i \in [n]$ that contain elements from a multi-secret VSS sharings. In their public implementation [58] this phase is skipped and already configured. Finally, the protocol is not responsive.

BRandPiper’s throughput depends heavily on an estimate for the synchronous network delay parameter Δ . A higher value for Δ leads to increased security but reduces the 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 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 and the use of multi-secret VSS. In particular, our results shown in Figure 4 confirm that BRandPiper’s performance is severely limited by the presence of a single slow node in the system.

Drand. The protocol uses the non-interactive and unique threshold BLS signature to generate the beacon value. Concretely, in each epoch $e \geq 1$, parties collectively reconstruct the full signature σ_e on the message $m := e$ and compute the beacon value O_e for epoch e as hash $O_e := H(\sigma_e)$. Further, the protocol relies on an initial key setup realized through Pedersen’s DKG protocol [67], which essentially runs n parallel instances of Feldman’s VSS with some additional verification steps. It has a communication cost of $\mathcal{O}(\lambda n^4)$ bits and outputs secret keys as field elements.

Although the threshold BLS signature allows for asynchronous communication, the actual deployment of Drand relies on a period parameter that determines the time after which a beacon value is output. Concretely, in each period of 30 seconds only a single beacon value is output. We note that we were only able to evaluate Drand’s throughput for up to 32 nodes, as in our experiments, Drand’s initialization step keeps failing for 64 or more nodes, even for large estimates of the network delay. The same issue was already reported in previous works [16, 31]. In accordance with previous works [15, 16, 31], we measure Drand’s throughput after its DKG phase by computing the time from the start of the epoch until the beacon is reconstructed. We observe that GRandLine outperforms Drand despite their similarity in computational cost (cf. Figure 3). We suspect there may be implementation inefficiencies in Drand that hindered its throughput. Another reason could be the choice of

programming language. Our protocol is implemented in Rust which is highly optimized for fast execution and has a better run-time performance due to the lack of garbage collection. On the other hand, Drand is implemented in Go-lang whose garbage collector can negatively influence consistency of performance (especially, at these high rates of throughput).

6 CONCLUSION

In this work, we presented a novel distributed key generation (DKG) protocol GRand and a novel distributed randomness beacon protocol GRandLine. Our DKG protocol GRand has a communication complexity of $O(\lambda n^2 \log n)$ bits while preserving optimal Byzantine resilience threshold $t < n/2$ in the synchronous network setting. This gives the first DKG protocol in any network setting that achieves subcubic communication complexity (cf. Table 2). Our randomness beacon protocol GRandLine has a communication complexity of $O(\lambda n^2)$ bits per epoch, where each epoch takes only a single asynchronous round of communication and is non-interactive (cf. Table 1). In each epoch, GRandLine employs only lightweight cryptography such as hash functions and pairings. Further, both our protocols are secure in the presence of a strongly adaptive adversary. Finally, we have implemented GRandLine in Rust (with manually configured key setup) and found that it vastly outperforms previous randomness beacons in the same setting.

While our DKG protocol has a low communication complexity, it only terminates after a linear number $O(n)$ of rounds. Especially in large-scale systems, a lower number of rounds is highly desirable, preferably independent of the number n of all parties. Therefore, an intriguing question is how to lower the number of rounds from linear $O(n)$ to expected constant $O(1)$ while preserving quadratic communication complexity. On the other hand, our randomness beacon has low communication and round complexity, but has slightly worse computational complexity compared to the randomness beacon derived from threshold BLS. Concretely, each party needs to compute two pairing evaluations along with a Chaum-Pedersen NIZK (for discrete logarithm equality) verification to verify other parties' beacon shares. In contrast, the threshold BLS scheme only requires one pairing evaluation for this step. Finally, our randomness beacon protocol requires $O(n)$ pairing evaluations and NIZK verifications for public verifiability of an epoch beacon value. This is where it significantly falls behind threshold BLS, but is comparable to the other protocols (cf. Table 3). However, using standard batch verification techniques [76], this can be reduced to only six multi-exponentiations and two pairing evaluations.

ACKNOWLEDGMENTS

We would like to thank Adithya Bhat, Aniket Kate, Kartik Nayak, and Nibesh Shrestha 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-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] 2023. Library implementation for the BN254 pairing-friendly elliptic curve. docs.rs. (2023). https://docs.rs/ark-bn254/0.4.0/ark_bn254/index.html
- [2] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, and Gilad Stern. 2023. 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*. Springer-Verlag, Berlin, Heidelberg, 39–70. https://doi.org/10.1007/978-3-031-38557-5_2
- [3] Ittai Abraham, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Reaching Consensus for Asynchronous Distributed Key Generation. In *40th ACM Symposium Annual on Principles of Distributed Computing*. Association for Computing Machinery, Portland, OR, USA, 363–373.
- [4] Ittai Abraham, Dahlia Malkhi, Kartik Nayak, and Ling Ren. 2018. Dfinity Consensus, Explored. Cryptology ePrint Archive, Report 2018/1153. (2018). <https://eprint.iacr.org/2018/1153>.
- [5] Nicolas Alhaddad, Mayank Varia, and Haibin Zhang. 2021. High-Threshold AVSS with Optimal Communication Complexity. 479–498. https://doi.org/10.1007/978-3-662-64331-0_25
- [6] arkworks contributors. 2022. arkworks zkSNARK ecosystem. (2022). <https://arkworks.rs>
- [7] Thomas Attema, Ronald Cramer, and Matthieu Rambaud. 2021. Compressed Σ -Protocols for Bilinear Group Arithmetic Circuits and Application to Logarithmic Transparent Threshold Signatures. 526–556. https://doi.org/10.1007/978-3-030-92068-5_18
- [8] Renas Bacho, Daniel Collins, Chen-Da Liu-Zhang, and Julian Loss. 2023. Network-Agnostic Security Comes (Almost) for Free in DKG and MPC. In *Advances in Cryptology – CRYPTO 2023: 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20–24, 2023, Proceedings, Part I*. Springer-Verlag, Berlin, Heidelberg, 71–106. https://doi.org/10.1007/978-3-031-38557-5_3
- [9] Renas Bacho and Julian Loss. 2022. On the Adaptive Security of the Threshold BLS Signature Scheme. 193–207. <https://doi.org/10.1145/3548606.3560656>
- [10] Renas Bacho and Julian Loss. 2023. Adaptively Secure (Aggregatable) PVSS and Application to Distributed Randomness Beacons. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security (CCS '23)*. Association for Computing Machinery, New York, NY, USA, 1791–1804. <https://doi.org/10.1145/3576915.3623106>
- [11] Renas Bacho, Julian Loss, Gilad Stern, and Benedikt Wagner. 2024. HARTS: High-Threshold, Adaptively Secure, and Robust Threshold Schnorr Signatures. Cryptology ePrint Archive, Paper 2024/280. (2024). <https://eprint.iacr.org/2024/280>
- [12] Akhil Bandrupalli, Adithya Bhat, Saurabh Bagchi, Aniket Kate, and Michael Reiter. 2023. HashRand: Efficient Asynchronous Random Beacon without Threshold Cryptographic Setup. Cryptology ePrint Archive, Paper 2023/1755. (2023). <https://eprint.iacr.org/2023/1755>
- [13] Mihir Bellare and Phillip Rogaway. 1993. Random Oracles are Practical: A Paradigm for Designing Efficient Protocols. 62–73. <https://doi.org/10.1145/168588.168596>
- [14] Piotr Berman, Juan A. Garay, and Kenneth J. Perry. 1989. Towards Optimal Distributed Consensus (Extended Abstract). 410–415. <https://doi.org/10.1109/SFCS.1989.63511>
- [15] Adithya Bhat, Nibesh Shrestha, Aniket Kate, and Kartik Nayak. 2023. OptRand: Optimistically Responsive Reconfigurable Distributed Randomness. *Proceedings 2023 Network and Distributed System Security Symposium* (2023). <https://api.semanticscholar.org/CorpusID:257499606>
- [16] Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. 2021. RandPiper - Reconfiguration-Friendly Random Beacons with Quadratic Communication. 3502–3524. <https://doi.org/10.1145/3460120.3484574>
- [17] Alexandra Boldyreva. 2003. Threshold Signatures, Multisignatures and Blind Signatures Based on the Gap-Diffie-Hellman-Group Signature Scheme. 31–46. https://doi.org/10.1007/3-540-36288-6_3
- [18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. 2018. Verifiable Delay Functions. 757–788. https://doi.org/10.1007/978-3-319-96884-1_25
- [19] Dan Boneh, Benedikt Bünz, and Ben Fisch. 2019. Batching Techniques for Accumulators with Applications to IOPs and Stateless Blockchains. 561–586. https://doi.org/10.1007/978-3-030-26948-7_20
- [20] Benedikt Bünz, Steven Goldfeder, and Joseph Bonneau. 2017. Proofs-of-delay and randomness beacons in Ethereum.
- [21] Christian Cachin, Klaus Kursawe, and Victor Shoup. 2005. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. 18, 3 (July 2005), 219–246. <https://doi.org/10.1007/s00145-005-0318-0>
- [22] Ran Canetti, Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 1999. Adaptive Security for Threshold Cryptosystems. 98–115. https://doi.org/10.1007/3-540-48405-1_7

- [23] Ignacio Cascudo and Bernardo David. 2017. SCRAPE: Scalable Randomness Attested by Public Entities. 537–556. https://doi.org/10.1007/978-3-319-61204-1_27
- [24] Ignacio Cascudo and Bernardo David. 2020. ALBATROSS: Publicly Attestable BATched Randomness Based On Secret Sharing. 311–341. https://doi.org/10.1007/978-3-030-64840-4_11
- [25] Ignacio Cascudo and Bernardo David. 2024. Publicly Verifiable Secret Sharing Over lass Groups and Applications to DKG and YOSO. In *Advances in Cryptology – EUROCRYPT 2024: 43rd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zurich, Switzerland, May 26–30, 2024, Proceedings, Part V*. Springer-Verlag, Berlin, Heidelberg, 216–248. https://doi.org/10.1007/978-3-031-58740-5_8
- [26] David Chaum and Torben P. Pedersen. 1993. Wallet Databases with Observers. 89–105. https://doi.org/10.1007/3-540-48071-4_7
- [27] Alisa Cherniaeva, Iliia Shirobokov, and Omer Shlomovits. 2019. Homomorphic Encryption Random Beacon. Cryptology ePrint Archive, Report 2019/1320. (2019). <https://eprint.iacr.org/2019/1320>.
- [28] Kevin Choi, Arasu Arun, Nirvan Tyagi, and Joseph Bonneau. 2023. Bicorn: An optimistically efficient distributed randomness beacon. Cryptology ePrint Archive, Report 2023/221. (2023). <https://eprint.iacr.org/2023/221>.
- [29] Kevin Choi, Aathira Manoj, and Joseph Bonneau. 2023. SoK: Distributed Randomness Beacons. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21–25, 2023*. IEEE, 75–92. <https://doi.org/10.1109/SP46215.2023.10179419>
- [30] Tokio contributors. 2023. Tokio library for networking in Rust. (2023). <https://tokio.rs/>
- [31] Sourav Das, Vinith Krishnan, Irene Miriam Isaac, and Ling Ren. 2022. Spurt: Scalable Distributed Randomness Beacon with Transparent Setup. 2502–2517. <https://doi.org/10.1109/SP46214.2022.9833580>
- [32] Sourav Das, Benny Pinkas, Alin Tomescu, and Zhuolun Xiang. 2024. Distributed Randomness using Weighted VRFs. Cryptology ePrint Archive, Paper 2024/198. (2024). <https://eprint.iacr.org/2024/198> <https://eprint.iacr.org/2024/198>.
- [33] Sourav Das, Zhuolun Xiang, Lefteris Kokoris-Kogias, and Ling Ren. 2023. Practical Asynchronous High-threshold Distributed Key Generation and Distributed Polynomial Sampling. In *32nd USENIX Security Symposium (USENIX Security 23)*. USENIX Association, Anaheim, CA, 5359–5376. <https://www.usenix.org/conference/usenixsecurity23/presentation/das>
- [34] Sourav Das, Thomas Yurek, Zhuolun Xiang, Andrew K. Miller, Lefteris Kokoris-Kogias, and Ling Ren. 2022. Practical Asynchronous Distributed Key Generation. 2518–2534. <https://doi.org/10.1109/SP46214.2022.9833584>
- [35] Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. 2018. Ouroboros Praos: An Adaptively-Secure, Semi-synchronous Proof-of-Stake Blockchain. 66–98. https://doi.org/10.1007/978-3-319-78375-8_3
- [36] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. 303–320.
- [37] Danny Dolev and Rüdiger Reischuk. 1985. Bounds on information exchange for Byzantine agreement. *J. ACM* 32, 1 (jan 1985), 191–204. <https://doi.org/10.1145/2455.214112>
- [38] D. Dolev and H. R. Strong. 1983. Authenticated Algorithms for Byzantine Agreement. *SIAM J. Comput.* 12, 4 (1983), 656–666. <https://doi.org/10.1137/0212045> <https://arxiv.org/abs/https://doi.org/10.1137/0212045>
- [39] D. Dolev and A. Yao. 1983. On the security of public key protocols. *IEEE Transactions on Information Theory* 29, 2 (1983), 198–208. <https://doi.org/10.1109/TIT.1983.1056650>
- [40] Justin Drake. 2018. Minimal VDF randomness beacon. (2018). <https://ethresear.ch/t/minimal-vdf-randomness-beacon/3566>
- [41] Hanwen Feng, Zhenliang Lu, and Qiang Tang. 2024. Breaking the Cubic Barrier: Distributed Key and Randomness Generation through Deterministic Sharding. Cryptology ePrint Archive, Paper 2024/168. (2024). <https://eprint.iacr.org/2024/168> <https://eprint.iacr.org/2024/168>.
- [42] Georg Fuchsbauer, Eike Kiltz, and Julian Loss. 2018. The Algebraic Group Model and its Applications. 33–62. https://doi.org/10.1007/978-3-319-96881-0_2
- [43] Yingzi Gao, Yuan Lu, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. 2021. Efficient Asynchronous Byzantine Agreement without Private Setups. Cryptology ePrint Archive, Report 2021/810. (2021). <https://eprint.iacr.org/2021/810>
- [44] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. 2007. Secure Distributed Key Generation for Discrete-Log Based Cryptosystems. 20, 1 (Jan. 2007), 51–83. <https://doi.org/10.1007/s00145-006-0347-3>
- [45] Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakubov. 2021. Random-Index PIR and Applications. 32–61. https://doi.org/10.1007/978-3-030-90456-2_2
- [46] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 36th Symposium on Operating Systems Principles (SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 51–68. <https://doi.org/10.1145/3132747.3132757>
- [47] Jens Groth. 2021. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339. (2021). <https://eprint.iacr.org/2021/339>.
- [48] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. 2021. Aggregatable Distributed Key Generation. 147–176. https://doi.org/10.1007/978-3-030-77870-5_6
- [49] Runchao Han, Jiangshan Yu, and Haoyu Lin. 2020. RandChain: Decentralised Randomness Beacon from Sequential Proof-of-Work. Cryptology ePrint Archive, Report 2020/1033. (2020). <https://eprint.iacr.org/2020/1033>.
- [50] Timo Hanke, Mahnush Movahedi, and Dominic Williams. 2018. DFINITY Technology Overview Series, Consensus System. (2018). [arXiv:cs.DC/1805.04548](https://arxiv.org/abs/1805.04548)
- [51] Stanislaw Jarecki and Anna Lysyanskaya. 2000. Adaptively Secure Threshold Cryptography: Introducing Concurrency, Removing Erasures. 221–242. https://doi.org/10.1007/3-540-45539-6_16
- [52] Aniket Kate, Easwar Vivek Mangipudi, Pratyay Mukherjee, Hamza Saleem, and Sri Aravinda Krishnan Thyagarajan. 2023. Non-interactive VSS using Class Groups and Application to DKG. Cryptology ePrint Archive, Paper 2023/451. (2023). <https://eprint.iacr.org/2023/451> <https://eprint.iacr.org/2023/451>.
- [53] Alireza Kavousi, Zhipeng Wang, and Philipp Jovanovic. 2023. SoK: Public Randomness. Cryptology ePrint Archive, Paper 2023/1121. (2023). <https://eprint.iacr.org/2023/1121> <https://eprint.iacr.org/2023/1121>.
- [54] Eleftherios Kokoris-Kogias, Dahlia Malkhi, and Alexander Spiegelman. 2020. Asynchronous Distributed Key Generation for Computationally-Secure Randomness, Consensus, and Threshold Signatures. 1751–1767. <https://doi.org/10.1145/3372297.3423364>
- [55] SCIPR Lab. 2021. C++ library for Finite Fields and Elliptic Curves. GitHub repository. (2021). <https://github.com/scipr-lab/libff>
- [56] Leslie Lamport, Robert Shostak, and Marshall Pease. 1982. The Byzantine Generals Problem. *ACM Trans. Program. Lang. Syst.* 4, 3 (jul 1982), 382–401. <https://doi.org/10.1145/357172.357176>
- [57] Christoph Lenzen and Sahar Sheikholeslami. 2022. A Recursive Early-Stopping Phase King Protocol. In *Proceedings of the 2022 ACM Symposium on Principles of Distributed Computing (PODC'22)*. Association for Computing Machinery, New York, NY, USA, 60–69. <https://doi.org/10.1145/3519270.3538425>
- [58] Zhongtang Luo. 2022. Implementation for RandPiper. Github. (2022). <https://github.com/zhtluo/randpiper-rs>
- [59] Atsuki Momose and Ling Ren. 2021. Optimal Communication Complexity of Authenticated Byzantine Agreement. In *35th International Symposium on Distributed Computing (DISC 2021) (Leibniz International Proceedings in Informatics (LIPIcs))*, Seth Gilbert (Ed.), Vol. 209. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 32:1–32:16. <https://doi.org/10.4230/LIPIcs.DISC.2021.32>
- [60] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. (2008). <https://bitcoin.org/bitcoin.pdf>
- [61] Kartik Nayak, Ling Ren, Elaine Shi, Nitin H. Vaidya, and Zhuolun Xiang. 2020. Improved Extension Protocols for Byzantine Broadcast and Agreement. In *34th International Symposium on Distributed Computing (DISC 2020) (Leibniz International Proceedings in Informatics (LIPIcs))*, Hagit Attiya (Ed.), Vol. 179. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 28:1–28:17. <https://doi.org/10.4230/LIPIcs.DISC.2020.28>
- [62] Lan Nguyen. 2005. Accumulators from Bilinear Pairings and Applications. In *Topics in Cryptology – CT-RSA 2005*, Alfred Menezes (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 275–292.
- [63] Valeria Nikolaenko, Sam Ragsdale, Joseph Bonneau, and Dan Boneh. 2022. Powers-of-Tau to the People: Decentralizing Setup Ceremonies. Cryptology ePrint Archive, Report 2022/1592. (2022). <https://eprint.iacr.org/2022/1592>.
- [64] Drand Organization. 2020. Drand - A Distributed Randomness Beacon Daemon. GitHub repository. (2020). <https://github.com/drand/drand>
- [65] Dimitrios Papachristoudis. 2023. Cryptography for GRandLine. GitHub repository. (2023). <https://github.com/DiPa0123/Optrand-PVSS>
- [66] Dimitrios Papachristoudis. 2023. Implementation of GRandLine. GitHub repository. (2023). <https://github.com/DiPa0123/GRandLine>
- [67] Torben P. Pedersen. 1992. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. 129–140. https://doi.org/10.1007/3-540-46766-1_9
- [68] Irving S Reed and Gustave Solomon. 1960. Polynomial codes over certain finite fields. *Journal of the society for industrial and applied mathematics* 8, 2 (1960), 300–304.
- [69] Yumi Sakemi, Tetsutaro Kobayashi, Tsunekazu Saito, and Riad S. Wahby. 2022. Internet Research Task Force (IRTF) Draft for Pairing-Friendly Curves. (Nov. 2022). <https://datatracker.ietf.org/doc/draft-irtf-cfrg-pairing-friendly-curves/>
- [70] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar R. Weippl. 2021. RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness.
- [71] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar R. Weippl. 2020. HydRand: Efficient Continuous Distributed Randomness. 73–89. <https://doi.org/10.1109/SP40000.2020.00003>
- [72] Nibesh Shrestha. 2022. Implementation for OptRand. Github. (2022). https://github.com/nibeshrestha/optrand/tree/crypto_dev
- [73] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. 2021. Synchronous Distributed Key Generation without Broadcasts. Cryptology ePrint Archive,

- Report 2021/1635. (2021). <https://eprint.iacr.org/2021/1635>.
- [74] Markus Stadler. 1996. Publicly Verifiable Secret Sharing. 190–199. https://doi.org/10.1007/3-540-68339-9_17
- [75] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. 2017. Scalable Bias-Resistant Distributed Randomness. 444–460. <https://doi.org/10.1109/SP.2017.45>
- [76] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan-Gueta, and Srinivas Devadas. 2020. Towards Scalable Threshold Cryptosystems. 877–893. <https://doi.org/10.1109/SP40000.2020.00059>
- [77] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan-Gueta, and Ittai Abraham. 2019. HotStuff: BFT Consensus with Linearity and Responsiveness. 347–356. <https://doi.org/10.1145/3293611.3331591>

A MORE ON RELATED WORK

In this section, we provide a detailed discussion on existing work of randomness beacon and distributed key generation protocols.

Distributed Key Generation. Most of the DKG protocols found in the literature are in synchrony [22, 44, 47, 48, 51, 73]. Among these, only the protocols [22, 44, 51] are proven adaptively secure. All of these synchronous DKG protocols with the exception of Shrestha et al. [73] assume the existence of a broadcast channel, which is invoked $O(n)$ times among all parties. More importantly, these DKG protocols require each party to broadcast a message of size at least λ , which inevitably results in $\Omega(\lambda n^3)$ bits communication cost by a known lower bound for Byzantine broadcast [37] (without assuming shared randomness in the first place). Recent works [25, 52] follow the common paradigm [47, 51] of letting each party broadcast a PVSS transcript for a field element from which each party can locally compute its share of the secret key. Crucially, these works primarily focus on the concrete computational efficiency of the underlying PVSS (or non-interactive VSS) scheme. By working over class groups, these schemes gain an efficiency advantage compared to the previous works [47, 51]. Yet, these PVSS schemes are still much less efficient compared to conventional PVSS schemes for group elements. DKG protocols in asynchrony have only gained attention very recently by the works in [2, 3, 11, 33, 34, 54]. All these constructions have cubic or higher communication cost, and the one of Abraham et al. [2] relies on a powers-of-tau setup.

Broadcast Instantiation. Most synchronous DKG protocols assume the existence of a broadcast channel, which is invoked $O(n)$ times. Essentially, the broadcast channel is used for the commitment (or delivery) phase and followed by a complaint phase. The PVSS-based constructions circumvent the need of the complaint phase, but in general at the cost of higher computational cost. More importantly, the commitment phase alone involves n -times access to the broadcast channel on messages of size at least λ . To implement the broadcast channel, the adaptively secure Dolev-Strong broadcast protocol [38] is commonly used, which has $O(\lambda n^3)$ communication cost. However, recent works provide the following: (i) a statically secure broadcast protocol [8] with $O(\lambda n^2)$ bits communication cost (in the honest majority setting), and (ii) an adaptively secure broadcast protocol [41] (and this work) with $O(\lambda n^2 \log n)$ communication cost by employing transparent threshold signatures of size $O(\lambda \log n)$. However, without assuming shared randomness, all these DKG protocols inevitably have at least cubic communication cost by the known lower bound for Byzantine broadcast [37].

Randomness Beacons. We categorize existing randomness beacons according to their assumptions and reliance on cryptographic

tools. Essentially, there are two types of designs: the first uses threshold cryptography, while the second relies on specialized tools such as proof-of-work (PoW) or VDF.

Threshold Cryptography. The protocol of this type employ threshold cryptography in order to generate randomness. For this, there are two approaches. In the first one [4, 21, 27, 64, 75], 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). The setup phase of these protocols has a communication cost of $O(\lambda n^3)$ 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 cost of $O(\lambda n^2)$ per beacon output within optimal one round. The second approach works through (P)VSS [15, 16, 23, 31, 35, 49, 70, 71, 75]. Notable randomness beacons here are SPURT [31], BRand-Piper [16], and OptRand [15]. 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. Still, most of these protocols assume a setup phase that when actually implemented incurs cubic or higher communication cost. Further, they have a computation-heavy epoch where most of the computation is carried by one single party (the epoch leader). Finally, HashRand [12] is a recent randomness beacon in asynchrony that works without the use of a threshold cryptographic setup. It is based on a (small) committee selection from which the secret shares of an AVSS scheme are reconstructed. The committee requires the presence of only one honest party for security. Their techniques rely on secure erasures of secret states and without that their protocol has a communication cost of $O(\lambda n^3 \log n)$ per epoch. Further, it has post-quantum security.

Specialized Tools. The protocols in this category employ verifiable delay functions (VDFs) [18, 40] or proof-of-work (PoW) [60] in order to generate randomness [20, 24, 28, 40, 49, 70]. 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 have good communication cost, 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 briefly. RandRunner [70] 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 has a communication cost of $O(\lambda n^2)$ bits per beacon output. However, it only achieves $(t + 1)$ -unpredictability, since an adaptive adversary can corrupt the next t leaders and thus learn the beacon values for the next t epochs. RandChain [49] uses a combination of PoW, VDF, and Nakamoto Consensus, and has a communication cost of $O(\lambda n)$ bits per beacon output. One crucial drawback is that the beacon output is only guaranteed to be 1/5-fair. Further, it suffers from blockchain-related attacks.

B ADDITIONAL PRELIMINARIES

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

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.

DEFINITION 5 (REED-SOLOMON CODE.). A Reed-Solomon code is a tuple of deterministic algorithms $\Sigma = (\text{Encode}, \text{Decode})$ with the following properties:

- **Encode:** The deterministic encoding algorithm takes as input b data symbols (m_1, \dots, m_b) . It outputs a code word (s_1, \dots, 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, \dots, s_q) of length q . It outputs b data symbols (m_1, \dots, m_b) . This algorithm tolerates up to c errors and d erasures in a code word (s_1, \dots, 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 .

DEFINITION 6 (CRYPTOGRAPHIC ACCUMULATOR). A cryptographic accumulator scheme is a tuple of probabilistic polynomial-time algorithms $\Sigma = (\text{Gen}, \text{Eval}, \text{Wit}, \text{Ver})$ 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 $D = \{d_1, \dots, d_n\}$ of elements. It outputs an accumulation value z for 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 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.

We continue with the standard security notion of a collision-resistant accumulator scheme. Intuitively, it states that it is hard for an adversary to create invalid proofs of membership.

DEFINITION 7 (COLLISION-RESISTANT ACCUMULATOR). Let $\Sigma = (\text{Gen}, \text{Eval}, \text{Wit}, \text{Ver})$ be a cryptographic accumulator scheme. For an algorithm A and $n \in \mathbb{N}$, define the experiment $\text{CR}_{\Sigma, n}^A$ as follows:

- (1) **Offline Phase.** Run the accumulator key generation algorithm to get $ak \leftarrow \text{Gen}(\lambda, n)$.
- (2) **Online Phase.** Run A on input par and (n, ak) . At some point in the experiment, A returns a tuple $(\{d_1, \dots, d_n\}, d', w')$.

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

We say that Σ is (ϵ, T, n) -collision-resistant if for all algorithms A that run in time at most T and all $k \in [n]$, $\Pr[\text{CR}_{\Sigma, k}^A = 1] \leq \epsilon$. Conversely, we say that A (ϵ, T, n) -breaks collision-resistance of Σ if it runs in time at most T and $\Pr[\text{CR}_{\Sigma, k}^A = 1] > \epsilon$ for some $k \in [n]$.

Digital Signature Scheme. A digital signature scheme provides a user with a verification-signing key pair (vk, sik) , 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.

DEFINITION 8 (DIGITAL SIGNATURE SCHEME). A digital signature scheme is a tuple of algorithms $\text{DS} = (\text{SKey}, \text{Sign}, \text{Ver})$ 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 sik_i .
- **Sign:** The possibly randomized signing algorithm takes as input a signing key sik_i and a message m . It outputs a signature σ . We also write $(m)_i$ to denote the message-signature pair (m, σ) where $\sigma \leftarrow \text{Sign}(sik_i, m)$.
- **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 cannot create a signature on a new message, even after obtaining many signatures on messages of its choice.

DEFINITION 9 (UNFORGEABILITY UNDER CHOSEN MESSAGE ATTACK). Let $\text{DS} = (\text{SKey}, \text{Sign}, \text{Ver})$ be a digital signature scheme. For an algorithm A , define the unforgeability under chosen message experiment $\text{UF-CMA}_{\text{DS}}^A$ as follows:

- **Offline Phase.** Run **SKey** on input (par, i) to obtain a key pair (vk_i, sik_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 answers queries of the following type: When A submits a message m , return $\sigma \leftarrow \text{Sign}(sik_i, m)$ and update $\mathcal{M} := \mathcal{M} \cup \{m\}$.
- **Output Determination.** When A outputs a message m^* and a signature σ^* , do the following. If $\text{Ver}(vk_i, m^*, \sigma^*) = 1$ 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, $\Pr[\text{UF-CMA}_{\text{DS}}^A = 1] \leq \epsilon$. 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 $\Pr[\text{UF-CMA}_{\text{DS}}^A = 1] > \epsilon$.

Aggregatable PVSS Scheme. We define the security notions for an APVSS schemes (cf. Definition 2). We start with correctness and public verifiability of (aggregated) transcripts.

- *Correctness.* We say that APVSS is *correct* if for all key pairs $(pk_1, sk_1), \dots, (pk_n, sk_n) \in \text{Keys}(par)$ and all $i \in [n]$,

$$\Pr[\text{Ver}((pk_j)_{j \in [n]}, T) = 1 \wedge \text{Conld}(pk_i, T) = 1] = 1,$$

where the probability is taken over all transcripts T output by $\text{Dist}(sk_i, (pk_i)_{i \in [n]})$.

- *Public Verifiability.* We say that APVSS is *publicly verifiable* if for all key pairs $(pk_1, sk_1), \dots, (pk_n, sk_n) \in \text{Keys}(par)$ and all (\mathbf{E}, π) such that $\text{Ver}((pk_1, \dots, pk_n), (\mathbf{E}, \pi)) = 1$, there exists a unique $S \in \hat{\mathbb{G}}$ such that

$$\text{Rec}(\{\text{Dec}_{sk_i}(\mathbf{E}_i)\}_{i \in \mathcal{I}}) = S \quad \forall \mathcal{I} \subset [n], |\mathcal{I}| = t + 1.$$

We require that the secret reconstructed from an aggregated transcript $T = \text{Agg}(T_1, \dots, T_k)$ corresponds to the sum of the secrets S_i that can be reconstructed from the T_i individually. Further, we require that the set of contributors to T consists of the contributors to the single transcripts T_i . Formally, we define this as follows.

DEFINITION 10 (CORRECTNESS OF AGGREGATION). Let APVSS = (Keys, Enc, Dec, Dist, Agg, Conld, Ver, Rec) be a publicly verifiable APVSS scheme over $\hat{\mathbb{G}}$. We say that APVSS is *correctly aggregatable* if for all $(pk_1, sk_1), \dots, (pk_n, sk_n) \in \text{Keys}(par)$, 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 $\text{Ver}((pk_1, \dots, pk_n), (\mathbf{E}_i, \pi_i)) = 1$, then for all $\mathcal{I} \subset [n]$, $|\mathcal{I}| = t + 1$, the (aggregated) transcript $(\mathbf{E}', \pi') := \text{Agg}((\mathbf{E}_1, \pi_1), \dots, (\mathbf{E}_k, \pi_k))$ satisfies

$$\text{Rec}(\{\text{Dec}_{sk_i}(\mathbf{E}'_i)\}_{i \in \mathcal{I}}) = \prod_{j \in [k]} \text{Rec}(\{\text{Dec}_{sk_i}(\mathbf{E}_{j,i})\}_{i \in \mathcal{I}}),$$

where we write $\mathbf{E}_j = (\mathbf{E}_{j,1}, \dots, \mathbf{E}_{j,n})$. Additionally, we require that $\text{Conld}(pk_i, T) = 1$ for an $i \in [n]$ if and only if there is an $j \in [k]$ such that $\text{Conld}(pk_i, T_j) = 1$.

We recall the security notion for APVSS schemes called *aggregated unpredictability* as introduced in [10]. Intuitively, it 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.

DEFINITION 11 (AGGREGATED UNPREDICTABILITY OF APVSS). Let APVSS = (Keys, Enc, Dec, Dist, Agg, Conld, Ver, Rec) be a publicly verifiable APVSS scheme over $\hat{\mathbb{G}}$. For an algorithm A , we define the aggregated unpredictability experiment $\text{AggPred}_{\text{APVSS}, t}^A$ as follows:

- *Offline Phase.* Initialize $\mathcal{T} := \emptyset$. For all $i \in [n]$, run Keys on input (par, i) to generate keys $(pk_i, sk_i) \leftarrow \text{Keys}(par, i)$. On input par and $\{pk_i\}_{i \in [n]}$, A returns an index set $C \subset [n]$ of initially corrupted parties along with updated public keys $\{\hat{pk}_i\}_{i \in C}$. Set $pk_i := \hat{pk}_i$ for all $i \in C$.
- *Corruption Queries.* At any point of the experiment, A may corrupt a party by submitting an index $i \in [n] \setminus C$. In this case, return the secret key sk_i and set $C := C \cup \{i\}$.
- *Transcript Queries.* At any point of the experiment, A gets access to an oracle of the following type: When A submits a request $(\text{givePVSS}, i)$ for an $i \in [n] \setminus C$, do the following. On behalf of dealer P_i , run Dist on input sk_i and pk_1, \dots, 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}', π') and an element $S^* \in \hat{\mathbb{G}}$, proceed as follows:

- Return 1 if $|C| \leq t$, $\text{Ver}((pk_1, \dots, pk_n), (\mathbf{E}', \pi')) = 1$, $S^* = \text{Rec}(\{\text{Dec}_{sk_i}(\mathbf{E}'_i)\}_{i \in [t+1]})$, and there is an index $i \in [n] \setminus C$ such that $\text{Conld}((\mathbf{E}', \pi'), pk_i) = 1$.
- 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, $\Pr[\text{AggPred}_{\text{APVSS}, t}^A = 1] \leq \epsilon$. Conversely, we say that $A(t, \epsilon, T, q_s)$ -breaks aggregated unpredictability of APVSS if it runs in time at most T , makes at most q_s transcript queries, and $\Pr[\text{AggPred}_{\text{APVSS}, t}^A = 1] > \epsilon$.

B.2 Consensus Primitives

Distributed Randomness Beacon. We provide a formal definition of a distributed randomness beacon and corresponding security notions. We follow the definition given in [10].

DEFINITION 12 (d -SECURE RANDOMNESS BEACON). Let \mathcal{RB} be an epoch-based protocol executed by n parties P_1, \dots, 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 $\sigma_e \in \{0, 1\}^\lambda$ in epoch $e \in [L]$, then all honest parties output σ_e in epoch e .
- *Availability.* \mathcal{RB} is (t, L) -available if the following holds whenever at most t parties are corrupted: for each $e \in [L]$, every honest party outputs a value $\sigma_e \in \{0, 1\}^\lambda$ in epoch e .
- *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 $\Sigma_{A,L}$ 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

$$\left| \Pr_{\sigma \leftarrow \Sigma_{A,L}} [D(\sigma) = 1] - \Pr_{u \leftarrow U_L} [D(u) = 1] \right| \leq \epsilon,$$
 where U_L denotes the uniform distribution over the L -fold Cartesian product of $\{0, 1\}^\lambda$ with itself.
- *d -Unpredictability.* \mathcal{RB} is $(t, \epsilon, T, L, q_h, d)$ -unpredictable if it is (t, L) -available, (t, L) -consistent, and for all $e \in [L]$ 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 $(t, \epsilon, T, L, q_h, d)$ -secure randomness beacon protocol if it is (t, ϵ, T, L) -bias-resistant, $(t, \epsilon, T, L, q_h, d)$ -unpredictable, (t, L) -available, and (t, L) -consistent.

DEFINITION 13 (d -UNPREDICTABILITY FOR \mathcal{RB}). Let \mathcal{RB} be an epoch-based protocol as defined above. For an algorithm A and $e \in [L]$, we define the experiment $d\text{-Unpred}_{\mathcal{RB}, t}^{A, e}$ as follows:

- *Offline Phase.* For all $i \in [n]$, generate keys as $(pk_i, sk_i) \leftarrow \text{Keys}(par, i)$. On input par and $\{pk_i\}_{i \in [n]}$, A returns an index set $C \subset [n]$ of initially corrupted parties along with updated public keys $\{pk_i\}_{i \in C}$. Set $pk_i := \hat{pk}_i$ for all $i \in C$. Initiate an execution of \mathcal{RB} with A controlling parties in 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 $H[m] = \perp$. In this case, set $H[m] \leftarrow \{0, 1\}^\lambda$. Return $H[m]$.

- **Corruption Queries.** *At any point of the experiment, A may corrupt a party P_i by submitting an index $i \in [n] \setminus C$. In this case, return the internal state of P_i and set $C := C \cup \{i\}$. Henceforth, A fully controls P_i .*
- **Online Phase.** *Run \mathcal{RB} with A. When A outputs a tuple (σ'_ℓ, ℓ) for an $\ell > e$, the experiment ends with output 0 in case there is an honest party that has output a value σ_{e+1} for epoch $e+1$. Continue the execution of \mathcal{RB} for another $\ell - e$ epochs.*
- **Winning Condition.** *Return 1 if $|C| \leq t$, $\ell \geq e + d$, $L \geq \ell$, and $\sigma'_\ell = \sigma_\ell$. Otherwise, return 0.*

C ADDITIONAL FIGURES

In this section, we provide figures for components of our DKG protocol. In Figure 5, we give a description of the deliver function whose purpose is to efficiently broadcast a long message to all parties in some designated set. In Figure 6, we give a description of the aggregatable PVSS scheme which allows for public verifiability of a sharing with the additional feature of secure aggregation of several sharings. For the description, we denote by \mathcal{LC} the linear vector space over \mathbb{Z}_p of length n and dimension $t + 1$ defined as

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

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

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

where $\mu_i := \prod_{j \in [n] \setminus \{i\}} 1/(i - j)$. Further, we denote by $\langle m \rangle_i$ the tuple consisting of message m and a signature σ_i of P_i on m .

C.1 Byzantine Agreement Protocol

In this section, we present the Byzantine agreement protocol designed by Momose and Ren [59] in which we implement the threshold signatures with the ones of Attema et al. [7]. On a high level, the protocol recursively calls itself two times on each half and uses threshold signatures (with variable thresholds) to prove knowledge of a threshold of signatures from different parties on the same message. We emphasize that the transparent threshold signatures [7] we use are not unique and thus not suitable for randomness beacons. We provide a formal description of the BA protocol in Figure 7. As a building block, it uses the protocol in Figure 8. For more details, we refer directly to the original work [59].

Transparent Threshold Signatures. In the following, we elaborate on the setup of the transparent threshold signatures of Attema et al. [7]. Concretely, it requires a plain PKI and additional $\mathcal{O}(n)$ random group elements for a vector commitment scheme which can for example be derived from random oracles. Essentially, the latter defines the public parameters of the commitment scheme, which is an unstructured public random string. In the recursive Byzantine agreement protocol, threshold signatures are used within sets of parties that are determined at the onset of the protocol execution (as the partition of the set of parties is deterministic). Therefore, the setup can be run for each such set of parties specified by the protocol, once the set of all parties along with their PKI keys is fixed. By the recursion, the total number of such sets is $1 + 2 + 4 + \dots + n \leq 2n$ with exponentially decreasing sizes. Thus, the total number of additional random group elements required for setup is still $\mathcal{O}(n)$ with essentially no overhead in the running time.

Security. The authors show that their BA protocol [59] is secure against an adaptive adversary, assuming perfect security of the threshold signature. Conversely, if the threshold signature has only computational security, we can build a straightforward reduction from the security of the Byzantine agreement protocol to the security of the threshold signature. For this, observe that a total of $\mathcal{O}(n \log n)$ threshold signatures are exchanged in an execution of the BA protocol, and any violation of security in the BA protocol is directly incurred by a threshold signature. Therefore, a guess with success probability more than $1/n^2$ suffices to build an efficient reduction against the security of the threshold signature. In particular, the security of the threshold signature of Attema et al. implies the security of our Byzantine agreement protocol.

D SECURITY PROOFS FOR DKG AND RANDOMNESS BEACON

In this section, we provide the security and complexity proofs for the theorems given in the main body of the paper.

D.1 Proofs for our DKG Protocol

Here, we give a proof of Theorem 1. We split our proof into two parts. First, we show security of GRand in the sense of Definition 4, assuming security of the underlying components. Second, we compute the communication and round complexity of GRand when using our concrete components. In this context, we note that security of GRand with our concrete components follows from the first part, since the security of these components was already discussed in previous sections of this paper.

PROOF. Before we do the analysis, we briefly recall the high-level idea of the DKG protocol design. For the following discussion, we assume for the sake of presentation that the number of all parties n is a power of two $n := 2^k$, $k \in \mathbb{N}_k$, and recall that $\mathcal{P} := \{P_1, \dots, P_n\}$. For the protocol description, we follow the direct down-top approach. First, each party P_{2i-1} , $i \in [n/2]$, generates a random (t, n) -threshold PVSS transcript T_{2i-1} by itself, which encodes a degree- t polynomial $f_{2i-1} \in \mathbb{Z}_p[X]$ in the exponent. Then, each party P_{2i-1} interacts with its neighbor P_{2i} with the goal to exchange their transcripts and aggregate them to a single (aggregated) transcript $AT_i := \text{Agg}(T_{2i-1}, T_{2i})$. Then, we split \mathcal{P} into disjoint sets of neighboring parties (that we call *committees*) $\mathcal{Q}_i := \{P_{2i-1}, P_{2i}\}$ for all $i \in [n/2]$, so that

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

We consider committee \mathcal{Q}_1 and its interaction with \mathcal{Q}_2 , but note that each committee \mathcal{Q}_j , $j \in [n/2]$, executes the same instructions. Let AT_1 and AT_2 denote the transcripts established by committees \mathcal{Q}_1 and \mathcal{Q}_2 , respectively. First, each party $P_i \in \mathcal{Q}_1$ generates an accumulation value z_1 for AT_1 and sends it to all parties in $\mathcal{Q} := \mathcal{Q}_1 \cup \mathcal{Q}_2$. Then, parties in \mathcal{Q} collectively execute two instances of Byzantine agreement BA in order to have consensus on the values z_1, z_2 . Following this, each party $P_i \in \mathcal{Q}_1$ broadcasts AT_1 to all parties in \mathcal{Q} using Deliver on input (AT_1, z_1) . In the next step, parties in \mathcal{Q} collectively execute two instances of (binary) Byzantine agreement BA in order to decide which committee(s) delivered its transcript correctly. Finally, each party $P_i \in \mathcal{Q}$ broadcasts AT_j , $j \in \{1, 2\}$, to all parties in \mathcal{Q} using Deliver on input (AT_j, z_j) in case the committee

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

- **Round 1.** Split m into b data symbols (m_1, \dots, m_b) as per a predefined policy. Run Encode on input (m_1, \dots, m_b) to obtain q code words (s_1, \dots, s_q) . For all $j \in [q]$, compute a witness $w_j \leftarrow \text{Wit}(ak, z, s_j)$ and send $\langle s_j, w_j, z \rangle_i$ to party $P_j \in Q$.
- **Round 2.** Any party $P_j \in Q$ 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 Q . // *Valid here means that the tuple (s_j, w_j, z) verifies according to $\text{Ver}(ak, z, w_j, s_j) = 1$.*
- **Local Output.** Upon receiving b valid code words for z , run Decode on these code words to obtain m .

Figure 5: Description of the Deliver protocol on input (Q, m, z) invoked by party P_i .

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, \dots, pk_n . It outputs a PVSS transcript $T_L := \{C, E, \pi\}$ as follows.

- (1) Sample a polynomial $f(X) = \alpha + \alpha_1 X + \dots + \alpha_t X^t \in \mathbb{Z}_p[X]$ 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 Schnorr 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 $\{C_i, E_i, \pi_i\}_{i \in [k]}$ and outputs a transcript $T := \{C, E, \pi\}$. Let μ_1, \dots, μ_{t+1} denote the Lagrange coefficients for the set $\{1, \dots, t+1\}$ at the point $x = 0$.

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

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

- (5) Parse π as (π_1, \dots, π_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, \dots, pk_n and a transcript $T = \{C, E, \pi\}$. It outputs 1 (accept) or 0 (reject). Let \mathcal{LC} be the linear space over \mathbb{Z}_p as defined above in (\star) and let \mathcal{LC}^\perp be its dual space.

- (6) Sample $(v_1, \dots, v_n) \leftarrow_s \mathcal{LC}^\perp$ and check if $C_1^{v_1} \cdot \dots \cdot C_n^{v_n} = 1$. For all $i \in [n]$, check if $e(g, E_i) = e(C_i, pk_i)$.
- (7) Parse π as (π_1, \dots, π_k) and check if $\zeta_1 \cdot \dots \cdot \zeta_k = C_0$ (obtained by Lagrange interpolation in the exponent from 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 $E = (E_1, \dots, E_n)$) 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 6: Description of the algorithms of our aggregatable PVSS scheme.

Q_j was determined to be successful in the previous step. This ensures that the transcripts reach all honest parties. Parties conclude by locally aggregating $AT := \text{Agg}(AT_1, AT_2)$ the valid PVSS transcripts and progress to the next level of iteration. This high-level idea applies to all levels $\ell \in [\log n]$ of the iteration/recursion: the two neighboring committees Q_{2i-1} and Q_{2i} of size $q/2 := 2^{\ell-1}$ each interact with each other to exchange their newly generated PVSS transcripts from the previous level and finally aggregate them. At the end of the protocol, all parties obtain the same, aggregated PVSS transcript AT from which they directly derive the key shares without any further interaction.

Security Analysis. Let A be an adversary that (t, ϵ, T) -breaks security of GRand. Using this adversary, we build a reduction against the aggregated unforgeability of the underlying aggregatable PVSS scheme APVSS. For this, we split our proof into two parts. First, we provide a simulation of the aggregated unforgeability experiment

to A via a sequence of games. In particular, we interpolate between some games using reductions against the security of the Byzantine agreement protocol BA and the security of the cryptographic accumulator scheme AC. Second, we bound A 's winning probability in the final game by providing an efficient reduction against the aggregated unforgeability of APVSS. We consider the following sequence of games with A as adversary. Throughout, we denote by $C \subset [n]$ the set of corrupt parties and by $\mathcal{H} := [n] \setminus C$ the set of honest parties.

GAME G_0 : This is the real game. In particular, the game samples system parameters par and initializes a corruption set $C := \emptyset$ and updates $\mathcal{H} := [n] \setminus C$ throughout the game. Then, the game runs A on input par with access to a corruption oracle. Whenever A decides to corrupt a party $P_i \in \mathcal{H}$, the game faithfully returns the internal state of party P_i to A and updates $C := C \cup \{i\}$. Henceforth,

Let $Q \subseteq \mathcal{P}$ be a set of q parties and let $b := \lfloor q/2 \rfloor + 1$. Partition Q into two disjoint subsets $Q = Q_1 \cup Q_2$ (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 $P_i \in Q_l$ and fix some small constant $const \in \mathbb{Z}_{\geq 1}$.

- **Graded Consensus.** Execute $\text{GBA}(Q, v_i)$ (cf. Figure 8) among all parties in Q . Let (v_i, g_i) denote the output.
- **First Recursion.** If $l = 1$, execute $\text{BA}(Q_1, v_i)$ among parties in Q_1 . Let v denote the output. In case $|Q_1| \leq const$, parties can execute any Byzantine agreement protocol to obtain v .
- **Proposal Phase.** If $l = 1$, send $\langle \text{propose}, v \rangle_i$ to all parties in Q . Upon receiving the same proposal value v from $\lfloor q_1/2 \rfloor + 1$ different parties in Q_1 (a majority) and if $g_i = 0$, update $v_i = v$.
- **Graded Consensus.** Execute $\text{GBA}(Q, v_i)$ among all parties in Q . Let (v_i, g_i) denote the output.
- **Second Recursion.** If $l = 2$, execute $\text{BA}(Q_2, v_i)$ among parties in Q_2 . Let v denote the output. In case $|Q_2| \leq const$, parties can execute any Byzantine agreement protocol to obtain v .
- **Proposal Phase.** If $l = 2$, send $\langle \text{propose}, v \rangle_i$ to all parties in Q .
- **Output Generation.** Upon receiving the same proposal value v from $\lfloor q_2/2 \rfloor + 1$ different parties in Q_2 (a majority) and if $g_i = 0$, update $v_i = v$. Terminate with output v_i .

Figure 7: Recursive Byzantine agreement protocol BA described from the view of party P_i on input v_i .

Let $Q \subseteq \mathcal{P}$ 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 threshold signature on m with threshold b , which is a proof of knowledge of b signatures on m from different parties.

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

Figure 8: Graded Byzantine agreement protocol GBA described from the view of party P_i on input (Q, v_i) .

A gets full control over P_i . Further, all honest parties follow the protocol instructions for the DKG protocol as specified in Figure 1. In particular, at the beginning of the protocol, each party P_i honestly samples a polynomial $f_i \in \mathbb{Z}_p[X]$ of degree t and computes a PVSS transcript T_i . After the protocol execution, each party P_i outputs a transcript AT and derives the public key PK , the vector of public key shares, and its secret key share SK_i . At the end of the game, A outputs a secret S^* . It wins the game if $|C| \leq t$ and $S^* = SK$. Clearly, A's advantage in winning the game is given by

$$\Pr[G_0 = 1] = \epsilon.$$

GAME G_1 : This game is identical to the previous game, except that we add an abort condition. The idea of this hybrid is to rule out failure of the protocol Deliver. Namely, whenever an instance of Deliver fails to output the correct message, the game aborts. At each level $r \in [\log n]$ of the recursion, there are $2n \cdot 2^{r-1}$ instances of Deliver. Summing these up over all levels, we obtain

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

It is easy to see that Deliver only fails when an invalid proof of membership is received. As such, the probability of failure of Deliver is directly given by the probability of finding a collision for the cryptographic accumulator scheme AC underlying Deliver. As this probability is given by ϵ_C , we can bound the winning probability of this game by

$$\Pr[G_1 = 1] \geq \Pr[G_0 = 1] - 2n^2 \epsilon_C.$$

GAME G_2 : This game is identical to the previous game, except that we add another abort condition. The idea of this hybrid is to rule out failure of the consensus protocol BA. 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 recursion, there are $4 \cdot 2^{r-1}$ instances of the protocol (each of the 2^{r-1} committees executes two instances of BA for accumulation value agreement and two instances of BA for committee selection). Summing these 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 by

$$\Pr[G_2 = 1] \geq \Pr[G_1 = 1] - 4n \epsilon_B.$$

GAME G_3 : This game is identical to the previous game, except that we add another abort condition. So far we have rule out failure of the distributed protocols BA and Deliver. From Lemma 1, it follows that the protocol execution then established the same aggregated transcript AT for all parties that has contribution from at least one honest party. As such, the idea of this hybrid is to guess this special party $P^* \in [n]$ that contributes to the aggregate AT and that remains honest until the end of the game. Concretely, at the beginning of the game, the game makes a random guess by sampling $i^* \leftarrow_s [n]$ and executes the game as in G_2 . At the end of the game,

the game aborts if $P_{i^*} \neq P^*$ or $P_{i^*} \notin \mathcal{H}$. Since the choice of i^* remains information-theoretically hidden from A 's view, we can bound the winning probability of this game by

$$\Pr[G_3 = 1] \geq \Pr[G_2 = 1]/(2n).$$

Note that the factor n comes from the condition $P_{i^*} = P^*$, while the factor 2 comes from the condition $P_{i^*} \in \mathcal{H}$. It remains to bound the probability that the final game G_3 outputs 1. For that, we build an efficient reduction R against the aggregated unforgeability of APVSS. The design should be straightforward.

Building a reduction. At the beginning of the aggregated unforgeability experiment, R submits a request (givePVSS, i^*) and obtains a PVSS transcript T_{i^*} . Then it simulates the game G_3 to A by sampling random degree- t polynomials $f_i \in \mathbb{Z}_p[X]$ for all $i \in \mathcal{H} \setminus \{i^*\}$ and computing corresponding PVSS transcripts T_i . For party P_{i^*} , however, it uses T_{i^*} . Whenever A decides to corrupt a party $P_i \in \mathcal{H}$, the reduction R simply forwards this query to its own challenger for the aggregated unforgeability experiment and returns the output to the adversary A . In this manner, R can correctly answer all corruption queries of A . At the end of the simulation to A , the adversary A outputs a secret x^* to R . We assume that the adversary outputs a correct forgery x^* , so that x^* is the secret of the final aggregated transcript AT which has contribution from P_{i^*} . Now, the reduction R outputs the tuple (AT, T_{i^*}, x^*) to the challenger of the aggregated unforgeability experiment. In particular, the winning conditions are satisfied: (i) $|C| \leq t$ is clear, since the same holds true for the adversary A . (ii) $\text{Ver}((pk_1, \dots, pk_n), AT) = 1$ is clear, since the DKG protocol execution succeeded by assumption. (iii) The existence of an index $i \in \mathcal{H}$ such that $\text{Conld}(AT, pk_i) = 1$ is clear, since i^* is this index. Having said this, it follows immediately that we can bound the winning probability of the final game by

$$\Pr[G_3 = 1] \leq \epsilon_A.$$

Overall, we obtain the final bound

$$\epsilon_A \geq \frac{1}{2n} (\epsilon - 2n^2 \epsilon_c - 4n \epsilon_B) \iff \epsilon \leq 2n (\epsilon_A + 2\epsilon_B + n\epsilon_C).$$

We proceed with the proof for the statement that the DKG protocol is complete assuming no failure of the distributed protocols BA and Deliver. Here, complete means that all honest parties at the end of the protocol execution output the same transcript AT that has contribution from at least one honest party.

LEMMA 1. *If the protocols BA and Deliver are perfectly secure, then GRand (cf. Figures 1 and 2) is a complete DKG protocol in the sense that all honest parties output the same transcript AT with contribution from at least one honest party at the end of the protocol.*

PROOF. We will show this lemma using an iterative argument. Concretely, we show that the recursive protocol GenAPVSS executed among a set of parties $Q := Q_1 \cup Q_2$ succeeds under the assumption that Q has honest majority. By this we mean that after termination, all honest parties output a common transcript AT that has contribution from at least one subcommittee among $\{Q_1, Q_2\}$ with honest majority itself (the existence of such a subcommittee is clear, otherwise Q itself would not have honest majority). From this, it then follows that GRand is complete, since at least one of the two committees $\mathcal{P} = \mathcal{P}_1 \cup \mathcal{P}_2$ from the first recursion step has honest majority (as we assume $t < n/2$), say \mathcal{P}_1 for example. Then, the

same argument also applies to \mathcal{P}_1 and its splitting $\mathcal{P}_1 = \mathcal{P}_{1,1} \cup \mathcal{P}_{1,2}$, and so on. Since at the bottom level, there is a pair of neighboring parties that form a committee, we know that both their initially sampled PVSS transcript will be included in all subsequent levels and thus also in the final aggregate AT .

We start with the proof. Consider a committee $Q = Q_1 \cup Q_2$ with honest majority among which the protocol GenAPVSS is executed. It follows that one of the two committees Q_1, Q_2 also has honest majority, and we assume w.l.o.g. Q_1 to be this committee. Further, we assume that parties in Q_1 have already established a common PVSS transcript T_1 . We will show that at the end, all parties in Q obtain a common PVSS transcript AT that has contribution T_1 . In the first step of the protocol, all parties in Q_1 generate an accumulation value z_1 for T_1 and then send it to all parties in Q . Since Q_1 has honest majority and we assume that all messages are signed by a party before it sends it, all parties in Q will set their local accumulation value list as $Z[1] := \{z_1\}$ and have agreement on it (without possibly knowing this). Then, parties run two instances of the Byzantine agreement protocol, which by consistency establishes the same accumulation values z_1, z_2 for all parties in Q . By validity of BA, we know that z_1 is the correct accumulation value for T_1 , which was sent by an honest majority of parties in Q . In the next step, each party in Q_1 broadcasts T_1 to all parties in Q using Deliver. Since this protocol 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 Q , all parties in Q will reconstruct the correct 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 Q_2 . It could be that corrupt parties in Q_2 (which can form the majority in that committee) send a valid transcript T_2 to only some honest parties in 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 $2BA_1, 2BA_2$ 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 BA_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 Q were able to reconstruct the full transcript T_1 coming from the (honest) committee Q_1 , we know that all honest parties input 1 into BA_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 Q using the Deliver protocol. If the output of BA_i was 0, then parties simply ignore any transcript T_i delivered by any party. If the output of BA_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 BA_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 BA_2 has also output 1, then likewise all honest parties will have the same transcript T_2 at the end of this step. If BA_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 $T_2 = \emptyset$ simply. Having said all this, we have shown that in case the committee 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 Q_1 and Q_2) where at least one of them is a valid and true PVSS transcript $T_i \neq \emptyset$. As a result, after the local aggregation step of $\{T_i\}_{i \in [2]}$, all honest parties obtain the same transcript $AT := \text{Agg}(T_1, T_2)$. 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. \square

Communication Complexity. In the following, we measure the communication complexity of GRand. We begin by focusing on 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 Q_1 and Q_2 each of size $2^{\ell-1}$ that merge into Q which is of size $q := 2^\ell$. We count the communicated bits among honest parties in the committee 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 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., $O(\lambda q^3)$ 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 Q and its children committees Q_1, 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 Q multicasts an accumulation value of size $O(\lambda)$ to all other parties in Q . Thus, this step takes a total communication of $O(\lambda q^2)$ bits, as the number of parties in 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 $O(\lambda q^2 \log q)$ bits. 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 $O(\lambda n)$. 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 $q \cdot O(\lambda n/q \cdot q) = O(\lambda nq)$ 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 $O(\lambda nq)$. 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 obtain $O(\lambda nq + \lambda q^2 \log q)$ bits for the total communication complexity of honest parties in the committee Q of size q . Since there are n/q such parent committees,

we get a communication complexity of $n/q \cdot O(\lambda nq + \lambda q^2 \log q) = O(\lambda n^2 + \lambda nq \log q)$ bits for that particular level. By summing over all levels for $\ell \in [\log n]$ with $q = 2^\ell$, we obtain a total communication complexity of

$$\begin{aligned} \sum_{\ell=1}^{\log n} O(\lambda n^2 + \lambda nq \log q) &= O(\lambda n^2 \log n) + \sum_{\ell=1}^{\log n} O(\lambda n 2^\ell \ell) \\ &\leq O(\lambda n^2 \log n) + \sum_{\ell=1}^{\log n} O(\lambda n 2^\ell \log n) \leq O(\lambda n^2 \log n) \end{aligned}$$

bits, as claimed. This concludes our discussion on the communication complexity of GRand.

Round Complexity. We proceed with the round complexity of 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.1. 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 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 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. \square

D.2 Proofs for our Randomness Beacon

Here, we give a proof for Theorem 2. Since the claim on the communication and round complexity of GRandLine is trivial to verify, we will only focus on the security analysis.

PROOF. In the following, we prove 1-unpredictability and bias-resistance of our randomness beacon 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 threshold 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 GRandLine follows. Having said that, let A be an algebraic algorithm that $(t, \varepsilon, T, L, q_h, 1)$ -breaks unpredictability of 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 $\varrho_r := H_2(\sigma)$ 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 $C \subset [n]$ be the dynamically changing set of corrupt parties and let $\mathcal{H} := [n] \setminus C$ be the set of honest parties. Initially, we have $C = \emptyset$. We consider the following sequence of games with A as adversary.

GAME 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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ 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 $(pk_i, sk_i) \leftarrow \text{Keys}(par, i)$ 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 $C := C \cup \{i\}$. 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 $\alpha_i \leftarrow_s \mathbb{Z}_p^*$ uniformly at random and publishing $cm_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Answer random oracle queries r_i to H_1 by sampling $\gamma_i \leftarrow_s \mathbb{Z}_p$ and returning $H_1[r_i] := g^{\gamma_i} \in \mathbb{G}_1$. Answer random oracle queries s_i to H_2 by sampling $\varrho_i \leftarrow_s \{0, 1\}^\lambda$ 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 $\pi_i := \text{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ 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 $\{e(g_r, SK_i)\}_S$ 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 $\varrho_\ell := H_2(e(g_\ell, SK))$. Clearly, A 's advantage in winning the game is per definition given by

$$\Pr[G_0 = 1] = \varepsilon.$$

GAME G_1 : This game is identical to the previous game, except that we add an abort condition. Before the execution of the game, sample a guess $\ell^* \leftarrow_s [L]$ 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 by

$$\Pr[G_1 = 1] \geq \Pr[G_0 = 1]/L.$$

GAME 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 $\gamma_i \leftarrow_s \mathbb{Z}_p$ 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 $\xi_{1,1} \leftarrow_s \mathbb{G}_1$ is some randomly sampled group element. Clearly, this game is indistinguishable from the previous one so that there is no change in the winning probability, i.e.,

$$\Pr[G_2 = 1] = \Pr[G_1 = 1].$$

GAME 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 $AT \leftarrow \text{GenAPVSS}$ 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, make a guess by sampling $i^* \leftarrow_s [n]$. 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 by

$$\Pr[G_3 = 1] \geq \Pr[G_2 = 1]/n.$$

GAME 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 BA to agree on which PVSS transcripts of the children committees to aggregate). Summing these 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 by

$$\Pr[G_4 = 1] \geq \Pr[G_3 = 1] - 4n\varepsilon_B.$$

GAME 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 its PVSS transcripts, the game aborts. As the NIZK proof of our PVSS scheme is a regular Schnorr proof with 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 by

$$\Pr[G_5 = 1] \geq \Pr[G_4 = 1] - \frac{q_h}{p}.$$

GAME 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 is a standard Chaum-Pedersen proof with 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 by

$$\Pr[G_6 = 1] \geq \Pr[G_5 = 1] - \frac{q_h}{p}.$$

GAME G7: 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 $H_1 : \{0, 1\}^* \rightarrow \mathbb{G}_1$, 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 by

$$\Pr[G_7 = 1] \geq \Pr[G_6 = 1] - \frac{q_h^2}{2p}.$$

As A is an algebraic adversary, at the end of the game it returns the forgery (ϱ_ℓ, ℓ) where $\varrho_\ell = H_2(e(g_\ell, SK))$ 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)

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

of elements in \mathbb{Z}_p such that $e(g_\ell, SK)$ equals

$$\stackrel{!}{=} e(g, h)^{a_0} \cdot \prod_{i=1}^n e(g, Y_i)^{a_{i,1}} \cdot e(g, pk_i)^{a_{i,2}} \cdot e(g, cm_{i,2})^{a_{i,3}} \\ \cdot \prod_{i=1}^n e(C_i, h)^{b_{i,1}} \cdot e(cm_{i,1}, h)^{b_{i,2}} \\ \cdot \prod_{i=1}^{q_h} e(h_{1,i}, h)^{b_{i,3}} \cdot \prod_{i=1}^{q_s} \prod_{j=1}^n e(\sigma_{i,j}, h)^{r_{i,j}} \\ \cdot \prod_{i,j=1}^n e(C_i, Y_j)^{c_{i,j,1}} \cdot e(C_i, pk_j)^{c_{i,j,2}} \cdot e(C_i, cm_{j,2})^{c_{i,j,3}} \\ \cdot \prod_{i,j=1}^n e(cm_{i,1}, Y_j)^{d_{i,j,1}} \cdot \prod_{i=1}^{q_h} \prod_{j=1}^n e(h_{1,i}, Y_j)^{d_{i,j,2}} \\ \cdot \prod_{i=1}^{q_s} \prod_{j,k=1}^n e(\sigma_{i,j}, Y_k)^{s_{i,j,k}} \cdot \prod_{i,j=1}^n e(cm_{i,1}, pk_j)^{e_{i,j,1}} \\ \cdot \prod_{i=1}^{q_h} \prod_{j=1}^n e(h_{1,i}, pk_j)^{e_{i,j,2}} \cdot \prod_{i=1}^{q_s} \prod_{j,k=1}^n e(\sigma_{i,j}, pk_k)^{t_{i,j,k}} \\ \cdot \prod_{i,j=1}^n e(cm_{i,1}, cm_{j,2})^{f_{i,j,1}} \cdot \prod_{i=1}^{q_h} \prod_{j=1}^n e(h_{1,i}, cm_{j,2})^{f_{i,j,2}} \\ \cdot \prod_{i=1}^{q_s} \prod_{j,k=1}^n e(\sigma_{i,j}, cm_{k,2})^{u_{i,j,k}} \cdot \prod_{i=1}^{q_s} \prod_{j=1}^n e(g_i, SK_j)^{v_{i,j}}. \quad (1)$$

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, \dots, C_n and encrypted shares Y_1, \dots, Y_n of

the aggregated transcript output by GenAPVSS (which has contribution from the designated party P^*), the public keys pk_1, \dots, pk_n of parties (these constitute the setup phase), the auxiliary commitments cm_1, \dots, cm_n (these constitute the commitment phase), the answers to hash queries $h_{1,i} := H_1(m_i), i \in [q_h]$, 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 $g_i := H_1(i)$ 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 $[q_h] \setminus \{\ell\}$ (recall that ℓ is the index where the forgery happens). Then the above equation over \mathbb{G}_T to base $e(g, h)$ yields

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

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 (\spadesuit)

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

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 2. *Let 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:*

$$\begin{aligned} \Pr[n\text{-COMDL}^{A_1} = 1] &= \Pr[G_7^A = 1 \wedge \neg E_1], \\ \Pr[n\text{-COMDL}^{A_2} = 1] &= \Pr[G_7^A = 1 \wedge E_1 \wedge \neg E_2], \\ \Pr[n\text{-COMDL}^{A_3} = 1] &= \Pr[G_7^A = 1 \wedge E_1 \wedge E_2 \wedge \neg E_3], \\ \Pr[n\text{-COMDL}^{A_4} = 1] &= \frac{1}{2} \Pr[G_7^A = 1 \wedge \dots \wedge E_3 \wedge \neg E_4], \\ \Pr[n\text{-COMDL}^{A_5} = 1] &= \frac{1}{2} \Pr[G_7^A = 1 \wedge \dots \wedge E_4 \wedge \neg E_5], \\ \Pr[n\text{-COMDL}^{A_6} = 1] &= \Pr[G_7^A = 1 \wedge E_1 \wedge \dots \wedge E_5]. \end{aligned}$$

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

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 G_7 to the adversary A.

SIMULATOR $Sim_1(\xi, par)$: On input ξ , Sim_1 queries the discrete logarithm oracle DL_g on ξ_2, \dots, ξ_n and gets (z_2, \dots, z_n) . It generates the public-secret key pairs of honest parties by sampling $sk_i \leftarrow_s \mathbb{Z}_p$ 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 $\alpha_i \leftarrow_s \mathbb{Z}_p^*$ uniformly at random for all $i \in \mathcal{H}$ and publishes $cm_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Further, for all $i \neq \ell$ it answers random oracle queries r_i to H by sampling $\gamma_i \leftarrow_s \mathbb{Z}_p$ 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 $\pi_i = \text{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ 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 $g_\ell = H_1(\ell)$ is generated (on which the forgery is produced).

SIMULATOR $Sim_2(\xi, par)$: On input ξ , Sim_2 generates the public-secret key pairs of honest parties by sampling $sk_i \leftarrow_s \mathbb{Z}_p$ 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 $cm_i = (\xi_{i,1}, \xi_{i,2}^{-1} SK_i)$ and publishes it. Further, for all i it answers random oracle queries r_i to H by sampling $\gamma_i \leftarrow_s \mathbb{Z}_p$ 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 $\pi_i = \text{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ 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 $\alpha_i := DL_g(cm_i)$ 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 $Sim_3(\xi, par)$: 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 $\alpha_i \leftarrow_s \mathbb{Z}_p^*$ uniformly at random for all $i \in \mathcal{H}$ and publishes $cm_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Further, for all i it answers random oracle queries r_i to H by sampling $\gamma_i \leftarrow_s \mathbb{Z}_p$ 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 $\pi_i = \text{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ 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 $\text{Sim}_4(\xi, \text{par})$: On input ξ , Sim_4 queries the discrete logarithm oracle DL_g on ξ_{t+2}, \dots, ξ_n and gets (z_{t+2}, \dots, z_n) . It generates the public-secret key pairs of honest parties by sampling $sk_i \leftarrow_s \mathbb{Z}_p$ 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 $f_{i^*} = d_0 + d_1X + \dots + d_tX^t \in \mathbb{Z}_p[X]$ 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 ξ_1, \dots, ξ_{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 $\alpha_i \leftarrow_s \mathbb{Z}_p^*$ uniformly at random for all $i \in \mathcal{H}$ and publishes $\text{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Further, for all i it answers random oracle queries r_i to H by sampling $\gamma_i \leftarrow_s \mathbb{Z}_p$ 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 $\pi_i = \text{Dleq}(g, g_r^{\alpha_i}, g_r, g_r^{\alpha_i})$ 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 G_7) into a valid solution to the COMDL instance ξ with high probability, conditioned on some event happening. Recall equation (\spadesuit) defined as $B\gamma_\ell = A$ for the appropriate terms A and B . We now describe the algorithms A_i that simulate the game G_7 to the adversary.

Algorithm $A_1(\xi, \text{par})$: 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 $g_\ell = H_1(\ell)$ is generated on which the forgery is produced. Suppose that A_1 wins the game G_7 and that event $\neg E_1$ happens, i.e., $B \neq 0$. Then equation (\spadesuit) is a non-trivial equation of degree one in $\gamma_\ell = \text{DL}_g(\xi_1)$ (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

$$\Pr[n\text{-COMDL}^{A_1} = 1] = \Pr[G_7^A = 1 \wedge \neg E_1].$$

The bound on the running time of A_1 is obvious.

Algorithm $A_2(\xi, \text{par})$: 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 $\text{cm}_i = (\xi_{i,1}, \xi_{i,2}^{-1} SK_i)$ 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 $\alpha_i = \text{DL}_g(\xi_i)$. Suppose that A_2 wins the game 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 $F := f_{i,1,2}\alpha_1 + \dots + f_{i,n,2}\alpha_n$ and from $B = 0$ get the identity

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

and so all terms on the right-hand side of the equation are independent from the variables $\alpha_1, \dots, \alpha_n$ 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 $C = [t]$ and $\mathcal{H} = [t+1, n]$. As a result, there are linear polynomials $F_i \in \mathbb{Z}_p[X]$ for all $i \in C$ such that $\alpha_i = F_i(\alpha_{t+1}, \dots, \alpha_n) = F_i(z_{t+1}, \dots, z_n)$ that only depend on the outputs by the honest parties. Therefore, the defining equation for F can be transformed into

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

for some appropriately defined coefficients $f_0, f_{t+1}, \dots, f_n \in \mathbb{Z}_p$. By assumption we suppose that A_2 wins the game 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 (\heartsuit) can be concretely computed by A_2 , in combination with the above equation for F linear in the variables $\alpha_{t+1}, \dots, \alpha_n$ the algorithm A_2 proceeds as follows. It computes all the values $\alpha_j = \text{DL}_g(\xi_j)$ 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 $\alpha_1, \dots, \alpha_n$ and thus efficiently output the discrete logarithm values of the COMDL challenge ξ . Overall, we obtain

$$\Pr[n\text{-COMDL}^{A_2} = 1] = \Pr[G_7^A = 1 \wedge E_1 \wedge \neg E_2].$$

The bound on the running time of A_2 is obvious.

Algorithm $A_3(\xi, \text{par})$: 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 G_7 and the event $E_1 \wedge E_2 \wedge \neg E_3$ happens. Recall the E_1 defining equation (\diamond)

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

Again, let $F := f_{\ell,1,2}\alpha_1 + \dots + f_{\ell,n,2}\alpha_n$ as before. In contrast to the previous paragraph, now the values α_i for all $i \in C$ (here, $C \subset [n]$) 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, \dots, 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 $\text{cm}_{i,1} = g^{\alpha_i} \in \mathbb{G}_1$ 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 $i \in C$ 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. 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 $f(0), f(1), \dots, f(n)$ are known to A_3 and independent from the variables z_1, \dots, z_n . Overall, the only variables in the above equation (\blacklozenge) that depend on z_1, \dots, z_n are the secret keys sk_1, \dots, sk_n . We simplify/update the equation as

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

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 $i \in C$ (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 $C = [t]$ and $\mathcal{H} = [t+1, n]$. Thus, let us write

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

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

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

where $[\dots]$ 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}, \dots, z_n . We define the corresponding coefficients of z_j for $j \in \mathcal{H}$ as e_j , that is $e_j := \hat{e}_j + \sum_{i \in C} \lambda_{j,i}$. By assumption we suppose that A_3 wins the game 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 $z_j = \text{DL}_g(\xi_j)$ 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, \dots, sk_n and thus efficiently output the discrete logarithm values of the COMDL challenge ξ that it received. Overall, we obtain

$$\Pr[n\text{-COMDL}^{A_3} = 1] = \Pr[G_7^A = 1 \wedge E_1 \wedge E_2 \wedge \neg E_3].$$

The bound on the running time of A_3 is obvious.

Algorithm $A_4(\xi, \text{par})$: 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 $f_{i^*} = d_0 + d_1X + \dots + d_tX^t \in \mathbb{Z}_p[X]$ 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 ξ_1, \dots, ξ_{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 $AT \leftarrow \text{GenAPVSS}$ 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 of 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 [10]. 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 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

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

From event E_3 we know that $e_j = 0$ for all $j \in \mathcal{H}$ where $e_j = \hat{e}_j + \sum_{i \in C} \lambda_{j,i} = \hat{e}_j$ (observe that now $\lambda_{j,i} = 0$ for all $i \in C$ 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

$$-\frac{e_{\ell,j,2}}{d_{\ell,j,2}} = z_1 + z_2j + \dots + z_{t+1}j^t \quad (\heartsuit)$$

in the variables z_1, \dots, z_{t+1} (recall that the reduction A_4 chooses the polynomial f_{i^*} such that its coefficients are the discrete logarithm values of ξ_1, \dots, ξ_{t+1}). Further, by calling its discrete logarithm oracle DL_g on inputs ξ_2, \dots, ξ_{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, \dots, z_{t+1} in $d_{\ell,j,2}f(j)sk_j + e_{\ell,j,2}sk_j$ are the $f(j)$ for $j \in C$. Since $|C| \leq t$, the algorithm A_4 proceeds as follows. It calls its discrete logarithm oracle DL_g on inputs $g^{f(j)}$ for all $j \in C$, thus obtaining t new linearly independent equations $f(j) = z_1 + z_2j + \dots + z_{t+1}j^t$ (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

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

which is equivalent to

$$\begin{aligned} \sum_{j=1}^n [-f_{\ell,j,2}\alpha_j + f_{\ell,j,2}f(j)] &= f(0) + D \\ \iff \sum_{j=1}^n [f(j) - \alpha_j] f_{\ell,j,2} &= f(0) + D \end{aligned} \quad (2)$$

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 $j \in C$. On the other hand, when the adversary A outputs its commitments $cm_{i,2}$ in the second group \mathbb{G}_2 it outputs them as a (known) linear combination of the elements $h, pk_1, \dots, pk_n, cm_{t+1,2}, \dots, cm_{n,2}$, and Y_1, \dots, Y_n . As a result, this gives an identity for all $i \in C$ as follows (4)

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

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

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

where we define the symbols

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

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 $cm_i = (\xi_{i,1}, \xi_{i,2}^{-1}SK_i)$ for all $i \in \mathcal{H}$ and thus $\alpha_i = DL_g(\xi_i)$. 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 $\alpha_j = DL_g(\xi_j)$ 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 $\alpha_1, \dots, \alpha_n$ 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

$$\Pr[n\text{-COMDL}^{A_4} = 1] = \frac{1}{2} \Pr[G_7^A = 1 \wedge \dots \wedge E_3 \wedge \neg E_4].$$

The bound on the running time of A_4 is obvious.

Algorithm $A_5(\xi, par)$: 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

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

Together with event E_4 , we have as before

$$f(0) + D = r_0 + \sum_{j=1}^n r_jsk_j + \sum_{j=1}^n t_jf(j)sk_j$$

where by event E_3 now

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

Taking these together, we get

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

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

$$\Pr[n\text{-COMDL}^{A_5} = 1] = \frac{1}{2} \Pr[G_7^A = 1 \wedge \dots \wedge E_4 \wedge \neg E_5].$$

The bound on the running time of A_5 is obvious.

Algorithm $A_6(\xi, par)$: 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 $f_{i^*} = d_0 + d_1X + \dots + d_tX^t \in \mathbb{Z}_p[X]$ 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 ξ_1, \dots, ξ_{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 G_7 and that the event $E_1 \wedge \dots \wedge E_5$ happens. Then the above equation (\diamond) simplifies into the following

$$\begin{aligned} f(0) - (r_0 + b_{\ell,3}) &= \sum_{j \in C} r'_j s k_j + \sum_{j \in C} t'_j f(j) s k_j \\ \iff f(0) &= r_0 + b_{\ell,3} + \sum_{j \in C} r'_j s k_j + \sum_{j \in C} t'_j f(j) s k_j. \end{aligned}$$

Having computed that, the only unknown terms dependent on z_1, \dots, z_{t+1} on the right-hand side of this equation are the $f(j)$ for $j \in C$. Since $|C| \leq t$, the algorithm A_6 proceeds as follows. It calls its discrete logarithm oracle DL_g on inputs $g^{f(j)}$ for all $j \in C$, thus obtaining t new linearly independent equations $f(j) = z_1 + z_2 j + \dots + z_{t+1} j^t$ (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

$$\Pr[n\text{-COMDL}^{A_6} = 1] = \Pr[G_7^\Delta = 1 \wedge \dots \wedge E_4 \wedge E_5].$$

The bound on the running time of A_6 is obvious. \square

To end the proof, consider algorithm B playing in $n\text{-COMDL}$ as follows: B samples $i^* \leftarrow_s [6]$ 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

$$\begin{aligned} \Pr[n\text{-COMDL}^B = 1] &= \frac{1}{6} \sum_{i=1}^6 \Pr[n\text{-COMDL}^{A_i} = 1] \\ &\geq \frac{1}{12} \cdot \Pr[G_7^\Delta = 1] \\ &\geq \frac{1}{12} \left(\frac{\varepsilon}{Ln} - 4n\varepsilon_B - \frac{2q_h}{p} - \frac{q_h^2}{2p} \right). \end{aligned}$$

By simple rearrangement of this identity, we conclude the proof of Theorem 2 with the final bound

$$\varepsilon \leq Ln \left(12\varepsilon_A + 4n\varepsilon_B + \frac{q_h^2 + 4q_h}{2p} \right).$$

\square