# GRandLine: 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 GRandLine, the first adaptively secure randomness beacon protocol that overcomes all these limitations while preserving simplicity and optimal resilience in the synchronous network setting. We achieve our result in two steps. First, we design a novel distributed key generation (DKG) protocol GRand that runs in $O(\lambda n^2 \log n)$ bits of communication but, unlike most conventional DKG protocols, outputs both secret and public keys as group elements. Here, $\lambda$ 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 GRandLine and evaluate it using a network of up to 64 parties running in geographically distributed AWS instances. Our evaluation shows that GRandLine can produce about 2 beacon outputs per second in a network of 64 parties. We compare our protocol to the state-of-the-art randomness beacon protocols 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**.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
*CCS '24, October 14–18, 2024, Salt Lake City, UT, USA*
© 2024 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0636-3/24/10
https://doi.org/10.1145/3658644.3690287

## Keywords

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

**ACM Reference Format:**
Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. 2024. GRandLine: Adaptively Secure DKG and Randomness Beacon with (Log-)Quadratic Communication Complexity. In *Proceedings of the 2024 ACM SIGSAC Conference on Computer and Communications Security (CCS '24), October 14–18, 2024, Salt Lake City, UT, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3658644.3690287

## 1 Introduction

Distributed randomness plays a crucial role in many cryptographic and distributed system applications. A randomness beacon [18, 19, 34] 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 [53, 78] they are used to securely choose a leader or a committee among participating parties (e.g., through a verifiable random function [49]) 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 [39], 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 [53], 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 [32, 56]. 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 GRandLine which improves

upon the state-of-the-art by combining, for the first time, *all* of the following properties:[1]

- *Adaptive Security.* We prove GRandLine secure in the presence of an adaptive adversary. Many other protocols [49, 72, 76] are only proven statically secure.
- *One-Round Epoch.* Each epoch in GRandLine takes only a single asynchronous round of communication and is non-interactive. It only requires synchrony for its pre-processing phase. This sets GRandLine apart from all other protocols except Drand [67].
- *Communication-Efficient.* GRandLine has a communication cost of $O(\lambda n^2)$ bits per epoch, where $\lambda$ is the security parameter. This sets GRandLine apart from BRandPiper [19] and RandShare [76], 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 [34] and RandShare [76], 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 [52] and RandRunner [71], 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 [67], OptRand [18], and BRandPiper [19], 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 signature[2] 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 [32, 56] 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. [24]. For epoch $e \geq 1$, this works as follows:

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

In this manner, one obtains a simple and efficient randomness beacon protocol which is also used by many blockchain and consensus protocols [53, 78]. 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 [25]. 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 $\omega$. Unfortunately, even the most efficient DKG protocols [2, 74] 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 $r_i \leftarrow_{\$} \mathbb{Z}_p$ 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, 74], 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 [49], 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) [48]. 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 [75] 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. [51] 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 [40] (without assuming shared

[1] For the sake of clarity, we only compare to adaptively secure protocols in this list.
[2] By "locally verifiable" we mean that the final threshold signature does not have an efficient (independent of $n$) verification algorithm, but the partial signatures have.

**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. [24] | *async* | 1/3 | ✗ | ✓ | ✓ | 1 | $O(\lambda n^2)$ | Uniq. Th. Signature | *CRS* | $O(\lambda n^3)$ |
| RandHerd [76] | *async* | 1/3 | ✗ | ✓ | ✓ | ABA | $O(\lambda c^2 \log n)$ | PVSS & Th. Schnorr | *CRS* | $O(\lambda n^3)$ |
| Herb [30] | *sync* | 1/3 | ✗ | ✓ | ✗ | $t+1$ | $O(\lambda n^3)$ | Thresh. ElGamal | *CRS* | $O(\lambda n^3)$ |
| Drand [67] | *sync* | 1/2 | ✗ | ✓ | ✓* | 1 | $O(\lambda n^2)$ | Thresh. BLS | *CRS* | $O(\lambda n^3)$ |
| SPURT [34] | *part sync* | 1/3 | ✗ | ✓ | ✓ | 9 | $O(\lambda n^2)$ | PVSS & Pairing | *CRS* | ✗† |
| Algorand [49] | *part sync* | 1/3 | ✗ | $\Omega(t)$ | ✗ | BC | $O(\lambda cn)$ | VRF | *Seed* | $O(\lambda n^3)$ |
| HydRand [72] | *sync* | 1/3 | ✗ | $t+1$ | ✗ | 3 | $O(\lambda n^2)$ | PVSS | *Seed* | $O(\lambda n^3)$ |
| OptRand [18] | *sync* | 1/2 | ✗ | ✓ | ✓ | 11 | $O(\lambda n^2)$ | PVSS & Pairing | *q-SDH* | $O(\lambda n^3)$ |
| RandShare [76] | *async* | 1/3 | ✓ | ✓ | ✓ | ABA | $O(\lambda n^4)$ | VSS | *CRS* | ✗ |
| SPURT [13] | *part sync* | 1/3 | ✓ | ✓ | ✓ | 9 | $O(\lambda n^2)$ | PVSS & Pairing | *CRS & AGM* | ✗† |
| RandChain [52] | *sync* | 1/2 | ✓ | $O(\lambda)$ | ✓ | $\Delta_{\text{PoW}}$ | $O(\lambda n)$ | PoW & VDF | *CRS* | ✗◇ |
| RandRunner [71] | *sync* | 1/2 | ✓ | $t+1$ | ✗ | $\Delta_{\text{VDF}}$ | $O(\lambda n^2)$ | Trapdoor VDF | *Seed* | $O(\lambda n^3)$ |
| BRandPiper [19] | *sync* | 1/2 | ✓ | ✓ | ✗ | 11 | $O(\lambda f n^2)$ | VSS | *q-SDH* | $O(\lambda f n^3)$ |
| OptRand [13] | *sync* | 1/2 | ✓ | ✓ | ✓ | 11 | $O(\lambda n^2)$ | PVSS & Pairing | *q-SDH & AGM* | $O(\lambda n^3)$ |
| Drand [12] | *sync* | 1/2 | ✓ | ✓ | ✓* | 1 | $O(\lambda n^2)$ | Thresh. BLS | *CRS & AGM* | $O(\lambda n^3)$ |
| **GRandLine** | *sync* | 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 [66] and *AGM* denotes the algebraic group model [45]. **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.

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 protocol [60, 62] for Byzantine agreement is a well-known example for this. The standard, non-recursive Phase-King protocol [17] 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 [64]. 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 [64] 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 [13, 18]. Conversely, such honest contribution is guaranteed by an argument similar to [60, 62], 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 $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ 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 $\mathbf{C} := (C_1, \ldots, C_n) \in \mathbb{G}_1^n$, a vector of encryptions $\mathbf{E} := (E_1, \ldots, E_n) \in \mathbb{G}_2^n$, and some auxiliary data $\pi$ 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 [18] and SPURT [34]. We are taking a different route inspired by the work [51]. Specifically, we think of the commitments $C_1, \ldots, C_n$ as public key shares and of the decryptions $D_1, \ldots, D_n$ 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 $(PK_1, \ldots, PK_n)$ 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 $C_1, \ldots, C_n$.

**From One to Infinity: Towards a Simple Randomness Beacon.** Clearly, GRand is different from most DKG protocols found in the literature that output secret keys in a field rather than a group. However, a delightful key insight in [51] is that this setup is enough to generate a stream of one-round randomness values. The idea is inspired from the threshold BLS signature [20] and its application to randomness generation as introduced by Cachin et al. Specifically, each party $P_i$ non-interactively creates a threshold BLS signature share $\sigma_i := \mathsf{H}_1(m)^{sk_i}$ on the epoch number $m := e \in \mathbb{Z}_{\geq 1}$ with its secret key share $sk_i$ and multicasts (i.e., sends it to all parties) it. Upon receiving $t+1$ valid shares $\{\sigma_i\}_{i\in\mathcal{S}}$ (which is checked by a pairing equation $e(g, \sigma_i) = e(pk_i, \mathsf{H}_1(m))$ from $P_i$'s public key share $pk_i$), a party locally reconstructs the full signature $\sigma = \mathsf{H}_1(m)^{sk}$ by Lagrange interpolation in the exponent and derives the randomness beacon value as another hash $O_e = \mathsf{H}_2(\sigma)$. With our DKG protocol GRand that generates keys $(PK_i, SK_i)$ as group elements, the operation $\mathsf{H}_1(m)^{SK_i}$ is not possible. However, when we think of the operation of „raising the group element $\mathsf{H}_1(m)$ to a power of $sk_i$" as an abstract group action $sk_i \odot \mathsf{H}_1(m)$, we realize that the action[3] $SK_i \odot \mathsf{H}_1(m)$ defined as $e(\mathsf{H}_1(m), SK_i) \in \mathbb{G}_T$ is possible. Therefore, we let each party $P_i$ non-interactively create a share $\sigma_i$ as $e(\mathsf{H}_1(m), SK_i)$. And upon receiving $t+1$ valid such shares, each party can locally reconstruct the full signature by Lagrange interpolation in the exponent as $\sigma = e(\mathsf{H}_1(m), SK)$. Again, the randomness beacon value for epoch $e$ is then derived as another hash $O_e := \mathsf{H}_2(\sigma)$. Intuitively, since the secret key $SK$ is hidden from the adversary, the signature $\sigma$ 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* $\sigma_i$. Previously, it was possible to verify such a (signature) share by a pairing check, but now the beacon share $\sigma_i$ is an element in the target group $\mathbb{G}_T$ itself so that there is possibly no way to verify correctness of $\sigma_i = e(\mathsf{H}_1(m), SK_i)$. To resolve this issue, the authors in [51] augment the public keys and shares $\sigma_i$ 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.

[3]Note 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.

**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 $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random and sends an ElGamal encryption $\mathsf{cm}_i := (g^{\alpha_i}, h^{-\alpha_i} SK_i)$ of its secret key $SK_i$ to all parties. Correctness of its second component can 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 $\vartheta_i := (\mathsf{H}_1(m)^{\alpha_i}, \sigma_i)$ along with an efficient Chaum-Pedersen NIZK proof of discrete logarithm equality $\pi_i := \mathsf{Dleq}(g, g^{\alpha_i}, \mathsf{H}_1(m), \mathsf{H}_1(m)^{\alpha_i})$ to prove correctness of $\mathsf{H}_1(m)^{\alpha_i}$. Upon receiving such a tuple $(\vartheta_i, \pi_i)$, any party can verify the correctness of the beacon share $\sigma_i$ using a pairing equation that involves the element $\mathsf{cm}_{i,2}$. Having done this, each party can compute the randomness beacon value $O_e$ for epoch $e = m$ as described before. Intuitively, security is preserved because the elements $g^{\alpha_i}, \mathsf{H}_1(1)^{\alpha_i}, \mathsf{H}_1(2)^{\alpha_i}, \ldots$ do not reveal too much information about $\alpha_i$, 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 $\vartheta_i := (\mathsf{H}_1(m)^{\alpha_i}, \sigma_i)$ and a simple NIZK proof $\pi_i$ to the other parties. Further, $(PK_i, \mathsf{cm}_i)$ can be thought of as an updated public key share of $P_i$ which is three group elements, and verification of a beacon share $\sigma_i$ takes two pairing operations (the same as threshold BLS!) and verification of the NIZK proof $\pi_i$.

**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* in the algebraic group model. Crucially, 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 [12] who 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 [18, 34] discovered the

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

| Protocol | Network | Resil. | Adapt. | Commun. | Rounds | Field | Crypto. Prim. | Setup |
|---|---|---|---|---|---|---|---|---|
| Kokoris et al. [57] | *async* | 1/3 | ✓ | $O(\lambda n^4)$ | $O(n)$ | ✓ | AVSS | *CRS* |
| Abraham et al. [3, 46] | *async* | 1/3 | ✗ | $O(\lambda n^3)$ | $O(1)$ | ✗ | PVSS, Pairing | *CRS* |
| Das et al. [36, 37] | *async* | 1/3 | ✗ | $O(\lambda n^3)$ | $O(\log n)$ | ✓ | AVSS | *CRS* |
| Bingo [2] | *async* | 1/3 | ✓ | $O(\lambda n^3)$ | $O(1)$ | ✓ | AVSS, Pairing | *q-SDH, AGM* |
| HARTS [14] | *async* | 1/3 | ✓ | $O(\lambda n^3 \log n)$ | $O(1)$ | ✓ | AVSS | *CRS, AGM* |
| Shrestha et al. [74] | *sync* | 1/2 | ✗ | $O(\lambda n^3)$ | $O(n)$ | ✓ | VSS | *q-SDH* |
| Gurkan et al. [51] | *sync* | $\log n$ | ✗ | $O(\lambda n^3 \log n)$ | $O(n)$ | ✗ | PVSS, Pairing | *CRS* |
| NI-DKG$^\circ$ [28, 50, 55] | *sync* | 1/2 | ✗ | $O(\lambda n^4)^\diamond$ | $O(n)$ | ✓ | PVSS | *CRS* |
| Gennaro et al. [47] | *sync* | 1/2 | ✓ | $O(\lambda n^4)^\diamond$ | $O(n)$ | ✓ | VSS | *CRS, AGM* |
| Canetti et al. [25] | *sync* | 1/2 | ✓ | $O(\lambda n^4)^\diamond$ | $O(n)$ | ✓ | VSS, Erasures | *CRS* |
| Jarecki et al. [54] | *sync* | 1/2 | ✓ | $O(\lambda n^4)^\diamond$ | $O(n)$ | ✓ | PVSS | *CRS* |
| **GRand** [our work] | *sync* | 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 [47] was proven adaptively secure in the AGM [12]. **Comm.** denotes the communication cost in bits. $\diamond$ Most protocols [25, 47, 54] assume a broadcast channel, which we implement with the commonly-used Dolev-Strong broadcast protocol [41]. 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 [25] relies on secure erasure of secret states. **Setup** denotes the setup assumptions, including idealized models. $\circ$ The protocols [28, 50, 55] follow the common technique [54], 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).

power of aggregatable PVSS (APVSS) for simple and efficient randomness beacons, outperforming previous constructions for even statically secure randomness beacons. A follow-up work [13] then showed their adaptive security by giving an adaptive security proof for their (efficient) APVSS schemes. Inspired by their work [13], 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 [35, 44]. The first one [35] 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 [44] 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 that outputs secret keys as field elements. Further, none of these works consider adaptive adversaries.

## 1.2 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 [25, 47, 50, 51, 54, 74]. Among these, only the protocols [25, 47, 54] are proven adaptively secure. All of these synchronous DKG protocols with the exception of Shrestha et al. [74] 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 $\lambda$, which inevitably results in $\Omega(\lambda n^3)$ bits communication cost by a known lower bound for Byzantine broadcast [40] (without assuming shared randomness in the first place). Recent works [28, 55] follow the common paradigm [50, 54] 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 [50, 54]. 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, 14, 36, 37, 57]. 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 $\lambda$. To implement the broadcast channel, the adaptively secure Dolev-Strong broadcast protocol [41] is commonly used, which has $O(\lambda n^3)$ communication cost. However, recent works provide the following: (i) a statically secure broadcast protocol [10] with $O(\lambda n^2)$ bits communication cost (in the honest majority setting), and (ii) an adaptively secure broadcast protocol [44] (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 [40].

**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, 24, 30, 67, 76], parties generate a $(t, n)$-threshold key $(sk_1, \ldots, sk_n)$ by running a DKG protocol from which the randomness beacon value is derived as a unique threshold signature on some message (typically the hash of the current epoch number). In more detail, there are the following protocols. Cachin et al. [24] works in asynchrony, but does not specify the threshold signature nor the DKG protocol. Dfinity works in partial synchrony and uses the threshold BLS signature together with the DKG protocol of Groth [50] (it assumes a broadcast channel). Drand [67] works in synchrony and uses threshold BLS together with Gennaro et al.'s DKG [47]. The setup phase of these protocols have 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 [18, 19, 26, 34, 38, 52, 71, 72, 76]. Notable randomness beacons here are SPURT [34], BRandPiper [19], and OptRand [18]. The idea of this approach is to generate a new random value at each epoch by combining secret sharings from at least $t + 1$ parties. This ensures that the combined secret has contribution from at least one honest party that chose its secret uniformly at random so the randomness beacon value also inherits that property. Let us elaborate in more detail on some of the protocols. SPURT works in partial synchrony and has a communication cost of $O(\lambda n^2)$ bits per beacon output. It relies on a pairing-based PVSS scheme. BRandPiper is a protocol from the family of RandPiper protocols. It works in synchrony and achieves a communication cost of $O(\lambda f n^2)$ bits per beacon output, where $f \leq t$ is the actual number of faults in the system. It relies on an efficient VSS scheme and the RandPiper SMR protocol. Most of these protocols assume a setup phase that incurs cubic or higher communication cost. Further, protocols of this type have a computation-heavy epoch where most of the computation is carried by one single party (epoch leader). Finally, HashRand [15] is a recent randomness beacon in asynchrony that achieves adaptive security 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. 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.

*Specialized Tools.* The protocols in this category employ verifiable delay functions (VDFs) [21, 43] or proof-of-work (PoW) [63] in order to generate randomness [23, 27, 31, 43, 52, 71]. VDFs are functions that require a certain amount of time to compute but can be verified quickly. Solana uses VDFs in its proof-of-history consensus protocol to establish a global source of time and generate random values. While VDF-based protocols can offer strong security guarantees and quadratic communication 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 on some of these protocols. RandRunner [71] works in synchrony and uses a trapdoor VDF. Such a trapdoor VDF can generate unique function values efficiently with the knowledge of the trapdoor, but takes some high specified time $T$ otherwise. RandRunner achieves a communication cost of $O(\lambda n^2)$ bits per beacon output. However, it only achieves $(t + 1)$-unpredictability, since an adaptive adversary can simply corrupt the next $t$ leaders and thus learn the beacon values for the next $t$ epochs. RandChain [52] uses a combination of PoW, VFD, and Nakamoto Consensus, and achieves 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.

### 1.3 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 the full version of this paper [11], we cover additional preliminaries, present further figures related to GRand, and provide full proofs for our theorems.

## 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 $P_1, \ldots, P_n$ connected by pairwise authenticated channels, i.e., the receiver of a message is aware of the sender's identity.

**General Notation.** Let $\lambda$ denote the security parameter. Throughout the paper, we assume that global parameters $par := (\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, p, g, h, e)$ are fixed and known to all parties. Here, $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ is a type 3 asymmetric pairing of prime order $p$ cyclic groups with generators $g \in \mathbb{G}_1, h \in \mathbb{G}_2$. That means, there is no efficiently computable homomorphism from $\mathbb{G}_1$ to $\mathbb{G}_2$ and vice versa. For concrete choices, we will assume $\lambda = 128$ and that $\mathbb{G}_1, \mathbb{G}_2$ are instantiated with a 256-bit elliptic curve. We use $\mathbb{G}$ to denote a group specified by $par$. For two integers $a \leq b$, we define the set $[a, b] := \{a, \ldots, b\}$; if $a = 1$, we write this set as $[b]$, and if $a = 0$, we write it as $[\![b]\!]$. For an element $x$ in a set $S$, we write $x \leftarrow_\$ S$ to mean that $x$ was sampled from $S$ uniformly at random. All our algorithms may be randomized (unless stated otherwise) and written in uppercase letters. By $x \leftarrow \mathsf{A}(x_1, \ldots, x_n)$ we mean running algorithm A on inputs $(x_1, \ldots, x_n)$ and uniformly random coins and then assigning the output to $x$. If A has oracle access to some algorithm B during its execution, we write $x \leftarrow \mathsf{A}^{\mathsf{B}}(x_1, \ldots, x_n)$. We write $\mathbf{G}^{\mathsf{A}}$ to denote the output of the game $\mathbf{G}$ involving algorithm A. 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 $\Delta$. 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 $\Delta$. 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 [42], we treat signatures as information-theoretic objects with perfect unforgeability and perfect correctness.

**Idealized Models.** We assume the random oracle model (ROM) [16]. 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) [45]. 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 $\mathbf{z} = (z_1, \ldots, z_k)$ of integers such that $\zeta = \prod_{i \in [k]} g_i^{z_i}$, where $(g_1, \ldots, g_k) \in \mathbb{G}^k$ 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* [13] 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 $\mathsf{DL}_g$ an oracle that on input an element $\xi := g^z \in \mathbb{G}_1$ returns the discrete logarithm $z$ of $\xi$ 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 $k$-**COMDL**$^{\mathsf{A}}$ as follows:*

- Offline Phase. *Sample $(z_1, \ldots, z_k) \leftarrow_{\$} \mathbb{Z}_p^k$ 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 $(\xi_1, \ldots, \xi_k)$. In this phase,* A *gets access to the oracle $\mathsf{DL}_g$.*
- Winning Condition. *Let $(z'_1, \ldots, z'_k)$ denote the output of* A*. Return 1 if (i) $z'_i = z_i$ for $i \in [k]$, and (ii) $\mathsf{DL}_g$ 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 $(\varepsilon, T)$-*hard *if for all algorithms* A *running in time at most $T$, $\Pr[k\text{-}\mathbf{COMDL}^{\mathsf{A}} = 1] \leq \varepsilon$. Conversely, we say that an algorithm* A *$(\varepsilon, T)$-*solves *the co-one-more discrete logarithm problem of degree $k$ if it runs in time at most $T$ and $\Pr[k\text{-}\mathbf{COMDL}^{\mathsf{A}} = 1] > \varepsilon$.*

## 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 = (Keys, Enc, Dec, Dist, Agg, ConId, Ver, 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 $m$, $\Pr[\mathsf{Dec}_{sk_i}(\mathsf{Enc}_{pk_i}(m)) = m] = 1$.*
- Dist*: The randomized* secret sharing algorithm *takes as input a secret key $sk_i$ and public keys $pk_1, \ldots, pk_n$. It outputs a vector of encrypted shares $\mathbf{E} = (\mathsf{Enc}_{pk_1}(S_1), \ldots, \mathsf{Enc}_{pk_n}(S_n))$ and a proof $\pi$, where $S_1, \ldots, S_n$ 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 $(\mathbf{E}_1, \pi_1), \ldots, (\mathbf{E}_k, \pi_k)$, $k \in \mathbb{N}$. It outputs an* (aggregated) PVSS transcript $T := (\mathbf{E}, \pi)$.
- ConId*: The deterministic* contributor identifier algorithm *takes as input an (aggregated) PVSS transcript $T = (\mathbf{E}, \pi)$ and a public key $pk_i$. It outputs 1 (accept) or 0 (reject). In the first case, we refer to $P_i$ as a* contributor to $T$.[4]
- Ver*: The deterministic* verification algorithm *takes as input public keys $pk_1, \ldots, pk_n$, and an (aggregated) PVSS transcript $T = (\mathbf{E}, \pi)$. It outputs 1 (accept) or 0 (reject). In the first case, we call the transcript $T$* valid*; otherwise we call it* invalid.
- Rec*: The deterministic* reconstruction algorithm *takes as input $t+1$ shares $S_1, \ldots, S_{t+1}$. It outputs a secret $S \in \mathbb{G}$.*

**Discussion.** We defer formal definitions for correctness and secrecy of an APVSS scheme to the full version of this paper. Our definitions are based on the ones from [13]. Essentially, the only difference to their definitions is the following. Their aggregation algorithm takes exactly $t+1$ transcripts as input, in contrast to

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

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 [13] (slightly adapted). This notion captures malleability attacks and prohibits any $t$-bounded (i.e., corrupting at most $t$ parties) adversary from learning the secret of an aggregated transcript that has contribution from at least one honest party, even if the adversary is allowed to contribute to the aggregation itself.

**Linear Erasure and Error Correcting Codes.** We use standard $(q, b)$-Reed-Solomon (RS) codes [69]. 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 $\mathcal{Q} \subseteq \mathcal{P} = \{P_1, \ldots, P_n\}$ (called a *committee*) and $b$ being $\lceil q/2 \rceil$. In the special case $\mathcal{Q} = \mathcal{P}$, we have $(q, b) := (n, t+1)$. We defer formal definitions to the full version.

**Cryptographic Accumulator.** A cryptographic accumulator [65] 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 [22] built upon class groups of unknown order. Alternatively, we could use Merkle trees at the cost of $O(\log n)$ multiplicative overhead in the communication complexity. We defer formal definitions to the full version.

## 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 [59] 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 $\varepsilon$ which corresponds to the adversary's ability in breaking underlying cryptographic tools.

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

- Validity. $\Pi$ *is* $(t, \varepsilon)$-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. $\Pi$ *is* $(t, \varepsilon)$-consistent *if the following holds: every honest party that outputs a value outputs the same value $v$.*
- Termination. $\Pi$ *is* $(t, \varepsilon)$-terminating *if the following holds: every honest party terminates with an output value $v \in V$.*

*We say that $\Pi$ is a $(t, \varepsilon)$-*secure *Byzantine agreement protocol if it is $(t, \varepsilon)$-valid, $(t, \varepsilon)$-consistent, and $(t, \varepsilon)$-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 $\Sigma_i = (\Sigma_i[1], \Sigma_i[2], \ldots)$ with $\Sigma_i[\ell]$ being its beacon output at epoch $\ell \geq 1$. Initially, each value is set to $\perp$. We say that party $P_i$ outputs a beacon value in epoch $\ell$ 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 [19] and HydRand [72]) without actually having the power to bias it. We defer formal definitions to the full version.

# 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. [51], this is enough for applications such as efficient randomness beacons. We first formally define a DKG protocol.

Definition 4 (DKG Protocol). *Let $\Pi$ be a protocol executed by $n$ parties $P_1, \ldots, P_n$, where for all $i \in [n]$, $P_i$ outputs a* secret key share *$SK_i$, a vector of* public key shares *$(PK_1, \ldots, PK_n)$, and a* public key *$PK$. We define the following properties for $\Pi$ which each holds with probability at least $1 - \varepsilon$ in the presence of an adversary corrupting at most $t$ parties:*

- Consistency. $\Pi$ *is* $(t, \varepsilon)$-consistent *if the following holds: all honest parties output the same public key $PK$ and the same vector of public key shares $(PK_1, \ldots, PK_n)$.*
- Correctness. $\Pi$ *is* $(t, \varepsilon)$-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. $\Pi$ *is* $(t, \varepsilon, T)$-secret *if for all algorithms* A *that run in time at most* $T$*, its success probability in the following experiment is at most* $\varepsilon$*.*
  - 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* $\Pi$ *with* A *having full control over parties in* $C$*. Let* $y := PK \leftarrow \Pi$ *be the public key output by honest parties, and* $x := SK$ *the secret key.*
  - Winning Condition. *Let* $S^*$ *denote the output of* A*. Then,* A *is considered successful iff* $|C| \leq t$ *and* $S^* = x$*.*

*We say that* $\Pi$ *is a* $(t, \varepsilon, T)$-secure *DKG protocol if it is* $(t, \varepsilon)$*-consistent,* $(t, \varepsilon)$*-correct, and* $(t, \varepsilon, T)$*-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 the full version of this paper. Our APVSS scheme is similar to the one of Bhat et al. [18] which is essentially SCRAPE [26] 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. [13] 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 the full version. Essentially, this is merely a variant of the BA protocol of Momose and Ren [62]. 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. [9] 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 [9]. Further, we briefly discuss the security guarantees of the resulting BA protocol in the full version.

[5]When 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 the full version of this paper. The protocol design was introduced in [19, 64] 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 [19, 64], 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 [64].

## 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

Let $\mathcal{Q} \subseteq \mathcal{P}$ be a set of $q$ parties and let $b := \lceil q/2 \rceil$. Further, let $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ be a (deterministic) partition of $\mathcal{Q}$ 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 $\langle m \rangle := \mathsf{Encode}(m_1, \ldots, m_b)$ for a $(q, b)$-Reed-Solomon code $\mathsf{RS} = (\mathsf{Encode}, \mathsf{Decode})$ and where $(m_1, \ldots, m_b)$ 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 $P_i \in \mathcal{Q}_l$.

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

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

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

- **Transcript Generation.** Run GenAPVSS($\mathcal{P}$) among all parties in $\mathcal{P}$ and obtain a PVSS transcript $AT := \{\mathbf{C}, \mathbf{E}, \pi\}$ from the execution. *// This generates a common (aggregated) PVSS transcript $AT$ for all parties in $\mathcal{P}$.*
- **Key Derivation.** Compute the decryption $D_i := \mathsf{Dec}_{sk_i}(E_i)$. Terminate with output $(PK_1, \ldots, PK_n) := (C_1, \ldots, C_n)$ 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 $P_i$.**

committee $\mathcal{Q}_{1-i}$. All parties then terminate with the aggregation $AT := \mathsf{Agg}(T_1, T_2)$. In more detail, this works as follows.

Let $\mathcal{Q} = \mathcal{Q}_1 \cup \mathcal{Q}_2$ be a (deterministic) partition of $\mathcal{Q}$ into two disjoint sets called committees. For each $i \in [2]$, run the protocol GenAPVSS among parties in $\mathcal{Q}_i$ 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 $\mathcal{Q}_{1-i}$, each party sends only a much shorter accumulation value $z_i$ for $T_i$. To counteract malicious behavior, all parties in $\mathcal{Q}$ 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 $\mathcal{Q}_i$ do not know anything about the opposite committee $\mathcal{Q}_{1-i}$'s transcript $T_{1-i}$ other than the accumulation value $z_{1-i}$ and vice versa. Therefore, for each $i \in [2]$, parties in $\mathcal{Q}_i$ next broadcast their transcript $T_i$ to all parties in $\mathcal{Q}$ using the protocol Deliver on input $(\mathcal{Q}, T_i, z_i)$. 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 $\mathcal{Q}$ 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 $AT := \mathsf{Agg}(T_1, T_2)$ 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 $D_i := \mathsf{Dec}_{sk_i}(E_i)$ and terminates. The public key shares of GRand are defined as $(PK_1, \ldots, PK_n) := (C_1, \ldots, C_n)$ with the secret key shares being $(SK_1, \ldots, SK_n) := (D_1, \ldots, D_n)$. Using the specific APVSS scheme described in Figure **??**, 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.

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

- **Setup Phase.** Parties execute the DKG protocol GRand and obtain a vector of secret key shares $(SK_1, \ldots, SK_n) \in \mathbb{G}_2^n$, a vector of public key shares $(PK_1, \ldots, PK_n) \in \mathbb{G}_1^n$, 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 $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random and multicast (i.e., send to all parties) the commitment $\mathsf{cm}_i := (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Upon receiving $\mathsf{cm}_j = (\mathsf{cm}_{j,1}, \mathsf{cm}_{j,2})$ from party $P_j$, check if equality $e(PK_j, h) = e(\mathsf{cm}_{j,1}, h)e(g, \mathsf{cm}_{j,2})$ 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 $\mathsf{cm}_j$ it received from other parties.*
- **Beacon Epoch r.** Compute $\sigma_i := (g_r^{\alpha_i}, e(g_r, SK_i))$ along with $\pi_i := \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$, and multicast $(\sigma_i, \pi_i)$. Upon receiving $(\sigma_j, \pi_j)$ from party $P_j \in \mathcal{G}$, check if $\pi_j$ verifies using $\mathsf{cm}_{j,1}$ and $\sigma_{j,1}$. Further, check if $\sigma_{j,2} = e(g_r, \mathsf{cm}_{j,2})e(\sigma_{j,1}, h)$.
- **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 $\sigma$ in epoch $r$, output the beacon value as $\varrho_r := \mathsf{H}_2(\sigma) \in \{0,1\}^\lambda$. *// The beacon value is output as soon as $t+1$ valid tuples are received.*

**Figure 3: Description of the randomness beacon protocol GRandLine from the view of party $P_i$.**

## 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 proof of the following theorem, we refer to the full version.

Theorem 1. *If* APVSS *is $(t, \varepsilon_A, T_A, q_s)$-aggregated unpredictable, if* BA *is $(t, \varepsilon_B, T_B)$-secure, and if* AC *is $(n, \varepsilon_C, T_C)$-collision-resistant, then* GRand *(cf. Figures 1 and 2) is a $(t, \varepsilon, T)$-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,* GRand *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. [51] 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 GRandLine. For a formal description of the protocol, we refer to Figure 3 below. Let $\mathsf{H}_1$ and $\mathsf{H}_2$ be hash functions modeled as random oracle and denote $g_r := \mathsf{H}_1(r)$ for all $r \in \mathbb{N}$. Parties begin by executing GRand upon which every party $P_i$ obtains a public-secret key pair $(PK_i, SK_i) := (g^{f(i)}, h^{f(i)})$ for a hidden polynomial $f \in \mathbb{Z}_p[X]$ of degree $t$. The idea now is to use $\vartheta_i := e(g_r, SK_i) \in \mathbb{G}_T$ as a partial signature on the epoch number $r \in \mathbb{N}$, obtain the full signature $\vartheta := e(g_r, SK)$ via Lagrange interpolation in the exponent from enough shares, and derive the randomness beacon value as $\mathsf{H}_2(\vartheta) \in \{0,1\}^\lambda$. However, the problem with a naive implementation of this approach is that no party can verify the correctness of a received share $\vartheta_i = e(g_r, h)^{f(i)}$. In order to resolve this issue, we augment the signature shares with additional elements from which its correctness can be checked via pairing equations. For this, we follow an economical two-step approach. After DKG setup, each party $P_i$ locally samples an $\alpha_i \leftarrow_\$ \mathbb{Z}_p^*$ uniformly at random and multicasts (i.e., sends to all parties) $\mathsf{cm}_i = (g^{\alpha_i}, h^{-\alpha_i} SK_i)$. Correctness of its second component can be checked via a pairing equation. After this commitment phase, the actual randomness beacon starts. For epoch $r \geq 1$, each party computes $\sigma_i := (g_r^{\alpha_i}, \vartheta_i)$ along with a Chaum-Pedersen NIZK proof of discrete logarithm equality [29] as $\pi_i := \mathsf{Dleq}(g, g^{\alpha_i}, g_r, g_r^{\alpha_i})$ to prove correctness of $g_r^{\alpha_i}$. Upon receiving such a tuple, any party can verify the correctness of the partial signature $\vartheta_i$ 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 GRandLine. On an high level, consistency and availability follow from uniqueness of the signature $\vartheta := e(\mathsf{H}_1(r), SK)$ (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 $\vartheta$, and the final hash operation $\mathsf{H}_2(\vartheta)$ guarantees uniformity in the random oracle model. Essentially, these are the standard arguments for the transformation from unique threshold signatures to randomness beacons [24], but with the additional argument of soundness of NIZK proofs for correctness of the threshold signature. Using known techniques [12] to handle adaptive corruptions, we give a security reduction from the hardness of $n$-COMDL to the unforgeability of the threshold signature. For a proof of the following theorem, we refer to the full version of this paper.

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

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

*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 [18], BRandPiper [19], and Drand [67]. 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 [8] 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 [8]. We use a custom, optimized APVSS scheme implementation for our underlying cryptographic operations [6] and tokio [33] for networking. The implementation follows strictly the description in Figure 3 and it is publicly available at our GitHub repository [7].

*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 [61, 73] 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 [61], Drand [67] and OptRand [73]. 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 [34], 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 [15, 18, 19, 72] 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.

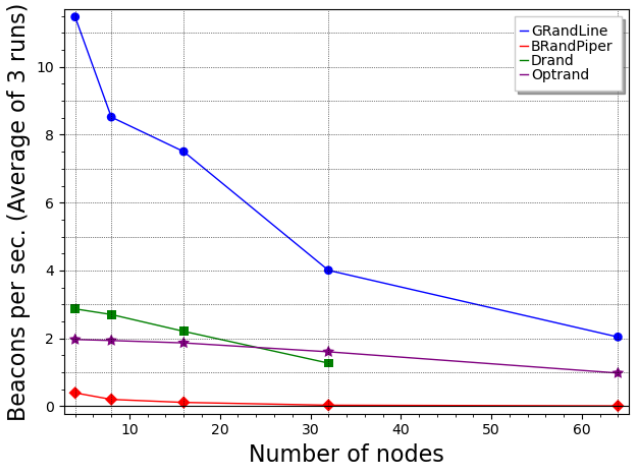



**Figure 4:** Performance graph for the randomness beacon protocols: OptRand, BRandPiper, Drand, and GRandLine.

**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. 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

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

| Protocol | Balanced | Rounds | Pairings | NIZKs (Ver) | NIZKs (Gen) | Exponents | Public Ver |
|---|---|---|---|---|---|---|---|
| Drand [67] | ✓ | 1 | $2n$ | 0 | 0 | $t+2$ | 2, 0 |
| OptRand [18] | ✗ | 11 | $4n+1$, $2n+1$ | $n$, $n^2+n$ | $n+1$ | $2n+t+2$ | $2n+3$, $3t+3$ |
| BRandPiper [19] | ✗ | 11 | $4n$, $2n$ | 0 | 0, $n^2$ | $n$, $n^2$ | $2n$, $t+1$ |
| **GRandLine** | ✓ | 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.

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 $O_e := \mathsf{H}(S_e)$. 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 [73] 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. [18] 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 [58]. Unfortunately, while this curve allows OptRand to benefit from more efficient modular operations, it is below acceptable security standards [70]. 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 $\mathbf{s}_e = (s_{e,1}, \ldots, s_{e,n}) \in \mathbb{Z}_p^n$ 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 $s_{e,i}, \ldots, s_{e-n+1,i}$ 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 $O_e := \mathsf{H}(R_e)$. 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 [61] 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 $\Delta$. A higher value for $\Delta$ leads to increased security but reduces the performance and vice versa. For BRandPiper, we first look for the smallest value for $\Delta$ 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 $11\Delta$. 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 $\sigma_e$ on the message $m := e$ and compute the beacon value $O_e$ for epoch $e$ as hash $O_e := \mathsf{H}(\sigma_e)$. Further, the protocol relies on an initial key setup realized through Pedersen's DKG protocol [68], which essentially runs $n$ parallel instances of Feldman's VSS with some additional verification steps. It has a communication cost of $O(\lambda n^4)$ 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 [19, 34]. In accordance with previous works [18, 19, 34], 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 throughput rates).

## 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 [77], this can be reduced to only six multi-exponentiations and two pairing evaluations.

## Acknowledgments

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. 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* (Santa Barbara, CA, USA). 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. 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] Anonymous. 2023. Cryptography for GRandLine. GitHub repository. https://github.com/DiPa0123/Optrand-PVSS

[7] Anonymous. 2023. Implementation of GRandLine. GitHub repository. https://github.com/DiPa0123/GRandLine

[8] arkworks contributors. 2022. `arkworks` *zkSNARK ecosystem.* https://arkworks.rs

[9] Thomas Attema, Ronald Cramer, and Matthieu Rambaud. 2021. Compressed $\Sigma$-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

[10] 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* (Santa Barbara, CA, USA). Springer-Verlag, Berlin, Heidelberg, 71–106. https://doi.org/10.1007/978-3-031-38557-5_3

[11] Renas Bacho, Christoph Lenzen, Julian Loss, Simon Ochsenreither, and Dimitrios Papachristoudis. 2023. GRandLine: Adaptively Secure DKG and Randomness Beacon with (Log-)Quadratic Communication Complexity. Cryptology ePrint Archive, Paper 2023/1887. https://eprint.iacr.org/2023/1887

[12] 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

[13] 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* (, Copenhagen, Denmark,) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 1791–1804. https://doi.org/10.1145/3576915.3623106

[14] 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. https://eprint.iacr.org/2024/280 https://eprint.iacr.org/2024/280.

[15] Akhil Bandarupalli, 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. https://eprint.iacr.org/2023/1755 https://eprint.iacr.org/2023/1755.

[16] 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

[17] 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

[18] 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

[19] 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

[20] 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

[21] 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

[22] 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

[23] Benedikt Bünz, Steven Goldfeder, and Joseph Bonneau. 2017. Proofs-of-delay and randomness beacons in Ethereum.

[24] 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

[25] 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

[26] 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

[27] 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

[28] 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* (Zurich, Switzerland). Springer-Verlag, Berlin, Heidelberg, 216–248. https://doi.org/10.1007/978-3-031-58740-5_8
[29] David Chaum and Torben P. Pedersen. 1993. Wallet Databases with Observers. 89–105. https://doi.org/10.1007/3-540-48071-4_7
[30] Alisa Cherniaeva, Ilia Shirobokov, and Omer Shlomovits. 2019. Homomorphic Encryption Random Beacon. Cryptology ePrint Archive, Report 2019/1320. https://eprint.iacr.org/2019/1320.
[31] Kevin Choi, Arasu Arun, Nirvan Tyagi, and Joseph Bonneau. 2023. Bicorn: An optimistically efficient distributed randomness beacon. Cryptology ePrint Archive, Report 2023/221. https://eprint.iacr.org/2023/221.
[32] 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
[33] Tokio contributors. 2023. Tokio library for networking in Rust. https://tokio.rs/
[34] 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
[35] Sourav Das, Benny Pinkas, Alin Tomescu, and Zhuolun Xiang. 2024. Distributed Randomness using Weighted VRFs. Cryptology ePrint Archive, Paper 2024/198. https://eprint.iacr.org/2024/198 https://eprint.iacr.org/2024/198.
[36] 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
[37] 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
[38] 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
[39] Roger Dingledine, Nick Mathewson, and Paul F. Syverson. 2004. Tor: The Second-Generation Onion Router. 303–320.
[40] 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
[41] 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 arXiv:https://doi.org/10.1137/0212045
[42] 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
[43] Justin Drake. 2018. Minimal VDF randomness beacon. (2018). https://ethresear.ch/t/minimal-vdf-randomness-beacon/3566
[44] 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. https://eprint.iacr.org/2024/168 https://eprint.iacr.org/2024/168.
[45] 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
[46] 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. https://eprint.iacr.org/2021/810.
[47] 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
[48] Craig Gentry, Shai Halevi, Bernardo Magri, Jesper Buus Nielsen, and Sophia Yakoubov. 2021. Random-Index PIR and Applications. 32–61. https://doi.org/10.1007/978-3-030-90456-2_2
[49] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. 2017. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. In *Proceedings of the 26th Symposium on Operating Systems Principles* (Shanghai, China) *(SOSP '17)*. Association for Computing Machinery, New York, NY, USA, 51–68. https://doi.org/10.1145/3132747.3132757
[50] Jens Groth. 2021. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339. https://eprint.iacr.org/2021/339.
[51] 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
[52] Runchao Han, Jiangshan Yu, and Haoyu Lin. 2020. RandChain: Decentralised Randomness Beacon from Sequential Proof-of-Work. Cryptology ePrint Archive, Report 2020/1033. https://eprint.iacr.org/2020/1033.
[53] Timo Hanke, Mahnush Movahedi, and Dominic Williams. 2018. DFINITY Technology Overview Series, Consensus System. arXiv:1805.04548 [cs.DC]
[54] 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
[55] 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. https://eprint.iacr.org/2023/451 https://eprint.iacr.org/2023/451.
[56] Alireza Kavousi, Zhipeng Wang, and Philipp Jovanovic. 2023. SoK: Public Randomness. Cryptology ePrint Archive, Paper 2023/1121. https://eprint.iacr.org/2023/1121 https://eprint.iacr.org/2023/1121.
[57] 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
[58] SCIPR Lab. 2021. C++ library for Finite Fields and Elliptic Curves. GitHub repository. https://github.com/scipr-lab/libff
[59] 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
[60] 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* (Salerno, Italy) *(PODC'22)*. Association for Computing Machinery, New York, NY, USA, 60–69. https://doi.org/10.1145/3519270.3538425
[61] Zhongtang Luo. 2022. Implementation for RandPiper. Github. https://github.com/zhtluo/randpiper-rs
[62] 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), Vol. 209)*, Seth Gilbert (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 32:1–32:16. https://doi.org/10.4230/LIPIcs.DISC.2021.32
[63] Satoshi Nakamoto. 2008. Bitcoin: A Peer-to-Peer Electronic Cash System. (2008). https://bitcoin.org/bitcoin.pdf
[64] 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), Vol. 179)*, Hagit Attiya (Ed.). Schloss Dagstuhl–Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 28:1–28:17. https://doi.org/10.4230/LIPIcs.DISC.2020.28
[65] 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.
[66] 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. https://eprint.iacr.org/2022/1592.
[67] Drand Organization. 2020. Drand - A Distributed Randomness Beacon Daemon. GitHub repository. https://github.com/drand/drand
[68] 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
[69] 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.
[70] Yumi Sakemi, Tetsutaro Kobayashi, Tsunekazu Saito, and Riad S. Wahby. 2022. Internet Research Task Force (IRTF) Draft for Pairing-Friendly Curves. https://datatracker.ietf.org/doc/draft-irtf-cfrg-pairing-friendly-curves/
[71] Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar R. Weippl. 2021. RandRunner: Distributed Randomness from Trapdoor VDFs with Strong Uniqueness.
[72] 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
[73] Nibesh Shrestha. 2022. Implementation for OptRand. Github. https://github.com/nibeshrestha/optrand/tree/crypto_dev
[74] Nibesh Shrestha, Adithya Bhat, Aniket Kate, and Kartik Nayak. 2021. Synchronous Distributed Key Generation without Broadcasts. Cryptology ePrint Archive, Report 2021/1635. https://eprint.iacr.org/2021/1635.
[75] Markus Stadler. 1996. Publicly Verifiable Secret Sharing. 190–199. https://doi.org/10.1007/3-540-68339-9_17
[76] 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
[77] 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
[78] 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