Mt. Random: Multi-Tiered Randomness Beacons

Ignacio Cascudo[1]*, Bernardo David[2]**, Omer Shlomovits[3], and Denis Varlakov[4]

[1] IMDEA Software Institute, Madrid, Spain. ignacio.cascudo@imdea.org
[2] IT University of Copenhagen, Copenhagen, Denmark. bernardo@bmdavid.com
[3] No affiliation, Israel. omer.shlomovits@gmail.com
[4] No affiliation, Israel. denis@varlakov.me

Abstract. Many decentralized applications require a common source of randomness that cannot be biased or predicted by any single party. Randomness beacons provide such a functionality, allowing parties to periodically obtain fresh random outputs and verify that they are computed correctly. In this work, we propose Mt. Random, a multi-tiered randomness beacon that combines Publicly Verifiable Secret Sharing (PVSS) and (Threshold) Verifiable Random Function (VRF) techniques in order to provide efficiency/randomness quality trade-offs with security under the standard DDH assumption (in the random oracle model) using only a bulletin board as setup (a requirement for the vast majority of beacons). Each tier provides a constant stream of random outputs offering progressive efficiency vs. quality trade-offs: true uniform randomness is refreshed less frequently than pseudorandomness, which in turn is refreshed less frequently than (bounded) biased randomness. This wide span of efficiency/quality allows for applications to consume random outputs from an optimal point in this trade-off spectrum. In order to achieve these results, we construct two new building blocks of independent interest: GULL, a PVSS-based beacon that preprocesses a large batch of random outputs but allows for gradual release of smaller "sub-batches", which is a first in the literature of randomness beacons; and a *publicly verifiable* and *unbiasable* protocol for Distributed Key Generation protocol (DKG), which is significantly more efficient than most of previous DKGs secure under standard assumptions and closely matches the efficiency of the currently most efficient *biasable* DKG protocol. We showcase the efficiency of our novel building blocks and of the Mt. Random beacon via benchmarks made with a prototype implementation.

* Ignacio Cascudo was supported by the Spanish Government under the project SecuRing (ref. PID2019-110873RJ-I00/MCIN/AEI/10.13039/501100011033), by the Madrid Government as part of the program S2018/TCS-4339 (BLOQUES-CM) co-funded by EIE Funds of the European Union, and by a research grant from Nomadic Labs and the Tezos Foundation.

** Bernardo David was supported by the Concordium Foundation and by the Independent Research Fund Denmark (IRFD) grants number 9040-00399B (TrA^{2}C), 9131-00075B (PUMA) and 0165-00079B.

1 Introduction

Randomness is essential for constructing provably secure cryptographic primitives and protocols. For several applications, it does not suffice that parties simply have a local source of randomness, but we require instead a randomness beacon that can periodically provide the same fresh unbiased and unpredictable random values to all parties. This is particularly important in decentralized applications such as Proof-of-stake blockchains (*e.g.* [25, 15, 11]), sharding protocols [36] and smart contracts that need randomness. In such settings, it is desirable to implement a random beacon as a protocol among the mutually distrustful participants of the corresponding system without trusting any single party. Moreover, such protocols must have guaranteed output delivery, publicly verifiable outputs.

Notice that a simple coin tossing protocol where parties commit to local randomness and then output the sum of the opened values is not sufficient, as parties can bias the output with a selective abort strategy, where they open or not their commitments depending on their view so far. Hence, several alternatives for constructing randomness beacons have been proposed based on publicly verifiable secret sharing (PVSS) [25, 8, 9, 34, 32, 4], verifiable random functions (VRF) [11, 15, 14, 22, 19, 35], verifiable delay functions (VDF) [5, 37, 3, 2, 31] and homomorphic encryption [12]. Moreover, achieving fairness against rational adversaries via financial punishments has been proposed by the RANDAO project [29].

Constructions using plain VRFs require very little computation and communication, but are open to the aforementioned selective abort bias. Since they rely on the computation of a VRF by a party who has a certain secret key, an adversarial party can always bias the final output by choosing whether to reveal or not their own VRF output under its secret key (see [15]). Threshold (also called distributed) VRFs, or TVRFs, solve this by always allowing a large enough set of parties (*e.g.* a majority of parties) to compute the verifiable random function, after a setup that consists on a distributed key generation protocol. However, current TVRF-based random beacons protocols simply apply the TVRF to the previous output of the beacon, defining the new beacon output as some fixed function of the new TVRF output. This approach requires a fixed initial seed to which the TVRF is applied in the first round, and since the entropy of such seed is of course finite, the unpredictability guarantees of the process will on the long run necessarily deteriorate. To the best of our knowledge there is no analysis of how exactly this plays out.

Finally, PVSS-based beacons such as SCRAPE [8] and ALBATROSS [9] enhance the commit-and-open strategy mentioned above by having parties commit to their inputs via PVSS. This renders the selective abort strategy useless, since unopened secrets can always be reconstructed by an honest majority of parties. On the downside, such protocols require more communication and computation. Although ALBATROSS [9] allows parties to generate a large batch of outputs with little overhead, these outputs are all revealed at once, instead of gradually providing fresh random values, as in TVRF-based protocols.

There are a number of recent works on constructing randomness beacons from time based primitives, such as Time Lock Puzzles (TLP) [30, 6, 24, 18] and

Verifiable Delay Functions (VDF) [5, 28, 37, 16]. Such protocols [5, 3, 2] require a structured common reference string as setup and achieve communication complexity linear in the number of parties but are based on sequential computation assumptions, *e.g.* [30] and [16]. These assumptions are arguably less understood than classical assumptions such as the Decisional Diffie-Hellman (DDH) assumption, upon which PVSS and (T)VRF-based beacons can be constructed. Hence, we focus on the latter, basing all our constructions on DDH. However, time-based primitives can potentially be used to instantiate Tier 2 of our beacon.

1.1 Our Contributions

In this work, we aim to combine the PVSS and (threshold) VRF approaches to obtain a best-of-both-worlds "multi-tiered" randomness beacon construction. Moreover, as a key part of Mt. Random's construction, we design a novel protocol for publicly verifiable and unbiasable distributed key generation. Finally we also present GULL (Gradually UnLeashed aLbatross), a new PVSS-based beacon that generates a large batch of random outputs like ALBATROSS but allows for gradually releasing of smaller "sub-batches" of outputs. All of our constructions are publicly verifiable and proven secure against malicious adversaries under a single standard assumption, *i.e.* Decisional Diffie Hellman (DDH).

Mt. Random: A multi-tiered randomness beacon More precisely, Mt. Random is a protocol where VRF, TVRF and PVSS-based random beacons are run as independent tiers executed in parallel. Each tier offers a different trade-off between complexity and randomness quality. By using the outputs of each tier as seeds for the next one, we aim at constructing a flexible architecture for randomness beacons that achieves good concrete efficiency without sacrificing security guarantees. Moreover, our approach allows for higher level protocols to choose what tier to use when obtaining randomness, according to the best complexity vs. randomnness quality trade-off for each application. At a glance, Mt. Random is constructed as follows:

– **Setup - Distributed Key Generation (DKG):** All Tiers use a Public Ledger for communication in order to achieve public verifiability and Tiers 1 and 2 use our novel DKG protocol to obtain threshold key pairs.
– **Tier 1 - Uniform Randomness via PVSS with GULL:** This tier gives $O(n)$ individual batches of $O(n)$ uniformly random outputs with communication and computational complexities of $O(n^2)$ for n parties.
– **Tier 2 - Uniform Pseudorandomness via TVRFs:** This tier uses a fresh output from Tier 1 as a seed when a new one is needed, providing 1 uniformly pseudorandom output per execution with communication and computational complexities linear in the number of parties running the tier.
– **Tier 3 - Bounded-Biased Pseudorandomness via VRFs:** This tier uses a fresh output from Tier 2 as a nonce when a new one is needed. Communication and computational complexities depend on output bias, *i.e.* the lower the complexity the higher the adversarial bias on the output.

Publicly Verifiable Distributed Key Generation We introduce a new publicly verifiable distributed key generation (DKG) protocol that can provide both the keys needed for the threshold encryption used in GULL (Tier 1) and for the TVRF (Tier 2). The security of our DKG scheme is based solely on DDH (in the random oracle model) and it guarantees that the public key cannot be biased by a rushing adversary (unlike in some other alternatives). In terms of communication and computation, our protocol is more efficient than previous unbiasable DKG schemes and essentially as efficient as the best biasable schemes.

GULL (Gradually UnLeashed aLbatross) We introduce GULL, a PVSS-based random beacon that generates $O(n)$ individual sub-batches of $O(n)$ random outputs each, where n is the number of parties in the protocol. Differently from the previous work ALBATROSS, all sub-batches remain initially hidden and can be opened at different points of time. Opening one sub-batch gives no information about the yet unopened ones. While GULL is marginally slower than ALBATROSS if a full batch of random outputs is required, it is significantly more efficient when a sequence of fresh unpredictable outputs are required: GULL allows for preprocessing a large amount of sub-batches of uniformly random outputs that can be gradually revealed at a low cost.

1.2 Technical Overview

Besides the Mt. Random architecture described above, our main technical contributions are novel protocols for PVSS-based randomness generation (GULL, used for implementing Tier 1) and for distributed key generation (used by both Tiers 1 and 2). We describe our main novel techniques.

Distributed Key Generation Departing from SCRAPE and ALBATROSS, we construct a DKG secure under the DDH-assumption and compatible with threshold El Gamal and TVRFs. Our goal is to establish a common public key $\mathsf{tpk} = g^{\mathsf{tsk}}$ and partial public keys of the form $\mathsf{tpk}_i = g^{\mathsf{tsk}_i}$, where each party P_i receives secret keys tsk_i that are Shamir shares for tsk. The aforementioned PVSS-based beacons output a group element g^r, that can be set as tpk. This is essentially constructed by having each party deal a secret g^s by constructing Shamir shares σ_i of s and encrypting g^{σ_i} under P_i's public key. Then g^r is defined by aggregating the g^s that were shared correctly (i.e. $g^r = \prod g^{s^{(a)}}$ where $s^{(a)}$ is correctly shared by P_a). In the process of reconstructing g^r, parties obtain partial public keys tpk_i by decrypting the shares they have received and aggregating them (i.e. $\mathsf{tpk}_i = \prod g^{\sigma_i^{(a)}}$), while proving the validity of the process.

However, parties do not obtain the corresponding partial secret keys sk_i from the PVSSs, as parties only obtain values g^{σ_i} but not σ_i. In our DKG, we modify the secret sharing phase described above, by having dealers also send a ciphertext containing the Shamir share σ_i which can only be decrypted by learning the corresponding "group share" g^{σ_i}. This guarantees that only party P_i can reconstruct σ_i. However, we then need to deal with the case where P_i detects that the group share and Shamir share are inconsistent with each other. In comparison to Fouque-Stern DKG, where the use of Paillier encryption allows the dealer to construct an elegant, but expensive, proof of the fact that the two values are the same, here this is not possible. Instead, we resort to a dispute resolution where the complaining party P_i reveals the received Shamir share and either P_i or the dealer of that share is disqualified. Revealing this share does not harm privacy since one of the two parties will be disqualified and that share will never be used.

One technical novelty we introduce with respect to ALBATROSS, which we will also exploit later in GULL, lies in the order of operations: in ALBATROSS parties first decrypt and reveal each g^{σ_i}, jointly reconstruct the secrets g^s of each dealer, and these opened secrets are aggregated to create the final output g^r; here, instead, parties aggregate their encrypted shares first and only reveal (and prove correctness of) the aggregated shares, which become the partial keys public tpk_i; then tpk is reconstructed from the tpk_i. This change of order can be done because the operations involved are linearly homomorphic.

GULL While the ALBATROSS construction provides a large uniformly random output, one problem is that the whole output is reconstructed by the participants at once. In Mt. Random and other applications, it is instead desirable that parts of this output are released gradually, while the rest of the output is kept hidden. We depart from ALBATROSS to construct GULL, a random beacon that accomplishes this. In ALBATROSS, the output consists of $\ell \cdot \ell'$ group elements, that we can group as ℓ' blocks of ℓ elements each; all these blocks are released at once. In GULL, parties execute the beginning of the protocol as in ALBATROSS (until the whole output is fixed), but then can release every block of ℓ outputs independently. Every block release needs little communication and computation and blocks not yet released are unpredictable given the opened ones.

In order to do this, after parties publish their encrypted shares and correctness proofs, and the set $\mathcal{Q}$ of dealers who have shared their secret correctly has been set, every party aggregates their shares received from the different dealers as a first step, then decrypts their aggregated shares, but rather than communicating them, they re-encrypt them under threshold El Gamal encryption, proving this encryption is correct and consistent with the rest of the protocol. We introduce and analyse a protocol π_{EG} for this proof. At this point, there are ℓ' blocks of ℓ secrets shared, where for each block the vector of ℓ secrets is shared via packed Shamir secret sharing, and at least $t+\ell$ receivers have encrypted their share with threshold El Gamal. Due to linearity of both the El Gamal scheme and the secret sharing reconstruction, the encryptions of the secrets in a block can be computed from any set of $t + \ell$ encrypted shares. Now a large enough subset of parties can decrypt each coordinate of the secret. Unfortunately, decrypting one of these coordinates may give information about the other secret coordinates *in the same block*, so we release all of the block of ℓ coordinates at once. However, the remaining unopened blocks of ℓ coordinates are still secret and uniformly distributed in the view of any subset of t parties.

1.3 Related Works

Scheme	Comp. (Exp/Enc/Dec)	Comm. (bits)	Rounds	Bias Resist.	Assump.
Pedersen [27]	$nt + 5n + t + 1$	$(2n^2 + tn + n)k_q$	$1 + 2$	No	DDH
Gennaro et al. [20]	$2nt + 11n + 3t + 3$	$(4n^2 + 2tn + 2n)k_q$	$2 + 3$	Yes	DDH
Fouque-Stern [17]	$(nt + 5n + t + 1)$ Exp. $+4n$ Enc$+n$ Dec	$(2n^2 + tn + n)k_q$ $+2n^2 k_h + 3n^2 k_N$	1	No	DDH +DCR
F-S [17] in terms of Exp. and k_q	$nt + 18005n + t + 1$	$(28n^2 + tn + n)k_q$	1	No	DDH +DCR
Our Result	$9n + t + 2$	$(2n^2 + tn + 5n)k_q$	$2 + 2$	Yes	DDH

Table 1: Comparison of DKG schemes where n is the total number of parties, t is the number of corrupted parties, k_q is the number of bits of an element of $\mathbb{G}_q$ or $\mathbb{Z}_q$, k_N is the number of bits of the Paillier cryptosystem modulus N and k_h is the output length of a hash function. Exp, Enc, Dec stand for operation of $\mathbb{G}$ (*i.e.* exponentiation), Paillier encryption and Paillier decryption, respectively. We consider that Pedersen and Gennaro *et al.* have private messages encryted under El Gamal. For typical parameters $k_q = 256, k_N = 2048$, we have $k_N = 8k_q$, Enc=3600 Exp and Dec=4880 Exp.

Distributed Key Generation While most distributed key generation (DKG) protocols employ secret sharing similarly to the one we introduce, the key differences lie on how parties prove the correctness of their shares and their consistency with the public information they post. Possibly the best known is Pedersen's protocol [27], where parties use a Feldman's VSS to do this, resulting in a protocol with at least 1 round of interaction, and 2 additional rounds if there are disputes. Gennaro et al. [20] observed that malicious parties can bias the public key generated by Pedersen's DKG and fixed this problem by introducing a new round of interaction and a new round of dispute resolution. Fouque and Stern [17] proposed a publicly verifiable one-round DKG based on the Paillier cryptosystem that still allows the adversary to bias public keys. A recent work by Gurkan et al. [21] introduces a publicly verifiable DKG with communication complexity of $O(n)$ based on the notion of aggregation via gossip. However, this protocol is based on pairing assumptions, stronger than our DDH assumption, and also outputs group elements as secret keys (rather than elements in $\mathbb{Z}_q$), *i.e.* the output is *incompatible with threshold El Gamal encryption.* It would be very interesting to achieve the type of output keys we need with gossip techniques.

In Table 1, we compare DKG protocols in terms of computation, communication, number of rounds (fixed rounds+dispute resolution rounds), assumptions and biasability of the global public key. Since Pedersen's and Gennaro et al.'s protocols need private communication between parties, we assume that this is done through the public ledger using El Gamal encryption. For comparing to Fouque-Stern, we estimated that Paillier encryption and decryption are equivalent to 3600 and 4880 group operations over a DDH-hard group, respectively, at the 128-bit security level, on a Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz using the RELIC library [1]. Our protocol requires almost the same communication as Pedersen's, differing only in lower order terms, and less communication than Gennaro et al. and Fouque-Stern, especially when compared with the latter, since k_N is larger than k_q (we can currently assume $k_q = 256$, $k_N = 2048$). On the other hand, Pedersen and of course Fouque-Stern have better round complexity, at the cost of allowing bias on the public key.

	Computation	Comm. Ledger	Output Size	Bias Resist.	Comp. Assumption	Setup Assumption
GRandPiper [4]	$O(n)$	$O(n^2)$	1	Yes	q-SDH	SCRS
HydRand [32]	$O(n)$	$O(n^2)$	1	Yes	DDH	PKI, RO
ALBATROSS [9]	$O(n^2 \log n)$	$O(n^2)$	$O(n^2)$	Yes	DDH	PKI, RO
GULL (generation)	$O(n^2)$	$O(n^2)$	$O(n^2)$	Yes	DDH	PKI, RO
GULL (opening)	$O(n^2)$	$O(n^2)$	$O(n)$	Yes	DDH	PKI, RO
DRAND [35]	$O(n)$	$O(n)$	1	Yes	Gap-DH	DKG, RO, Many URS
Ouroboros Praos [15]	$O(n)$	$O(n)$	1	No	CDH	PKI, RO, URS
Original VDF [5]	VDF	$O(n)$	1	Yes	VDF	SCRS
RandRunner [31]	VDF	$O(n)$	1	Yes	VDF	URS, RO
Mt. Random-Tier 1	$O(n^2)$	$O(n^2)$	$O(n)$	Yes	DDH	PKI, RO
Mt. Random-Tier 2	$O(n)$	$O(n)$	1	Yes	DDH	PKI, RO, Tier 1
Mt. Random-Tier 3	$O(n)$	$O(n)$	1	No	DDH	PKI, RO, Tier 2

Table 2: Comparison of random beacon protocols according to computation in terms of modular exponentiations, communication in terms of bits posted on public ledger, bias resistance, computational assumption and setup assumption. n=number of parties, PKI=Public Key Infrastructure, RO=Random Oracle, SCRS=Structured Common Reference String, URS=Uniform Random String, VDF=computational cost of VDF evaluation. In GULL (generation), "Output size" means that a batch of $O(n^2)$ *encrypted* random elements are prepared but not opened until the Opening phase.

Randomness Beacons Mt. Random is the first random beacon of its kind, *i.e.* combining different sub-protocols generating fresh random outputs each with a different randomness quality vs. efficiency trade-off. Previous works have focused on improving one specific approach to randomness generation instead of combining multiple approaches. On the other hand, in introducing the novel multi-tiered architecture of Mt. Random the main hurdle was supporting protocols compatible with all Tiers and based on a single assumption (DDH): the new DKG protocol used for Tiers 1 and 2 and the new Tier 1 beacon GULL that allows for efficiently gradually releasing large batches of random outputs. Hence, when comparing with previous works, we focus on comparing each previous work with a specific tier of Mt. Random. Besides communication/computational/round complexity, we also take into consideration the supporting protocols required as setup by previous works and the security assumptions they are based in, since providing a cohesive suite of protocols based on a single assumption is a key feature of Mt. Random. A comparison is presented in Table 2.

In comparing Mt. Random with previous works it is useful to observe the following correspondences between its Tiers and each previous protocol: Tier 1 consists of executing GULL's generation phase and then successively opening individual sub-batches of outputs; Tier 2 consists of running a DDH-based version of DRAND using seeds from Tier 1; Tier 3 consists of running Ouroboros Praos using nonces from Tier 2. Tier 1 is has a little overhead with respect to the state-of-the-art beacon ALBATROSS in case $O(n^2)$ outputs are needed at once, but has an advantage within Mt. Random's architecture where sub-batches of fresh $O(n)$ outputs are frequently required. In this case, ALBATROSS would have to be executed n times to obtain the same amount of fresh randomness batches, resulting in complexity much hire than that of Tier 1 using GULL. Tiers 2 and 3 naturally have the same complexity as DRAND and Ouroboros Praos. However, instead of relying on stronger assumptions, an external DKG phase and an external source of fresh seeds as is the case of DRAND, Tier 2 is fully based on DDH and receives keys and fresh seeds from our DKG protocol and Tier 1, which are also based on DDH. Hence, the main advantage of Mt. Random is providing a self-contained infrastructure for executing its upper Tiers based on a single well studied assumption without sacrificing efficiency, which is made possible by our novel DKG and GULL protocols respectively executed for generating threshold keys and running Tier 1.

2 Preliminaries

For integers $m \leq n$, $[m, n]$ denotes the set $\{m, m+1, \ldots, n\}$. We let $[n] = [1, n]$. Our protocols take place in a cyclic group $\mathbb{G}$ of prime order q. We denote by $\mathbb{Z}_q$ the finite field of q elements (for the same prime q), consisting of the integers modulo q, and note that we can speak of g^a for $g \in \mathbb{G}, a \in \mathbb{Z}_q$ and this respects the rule $g^a \cdot g^b = g^{a+b}$ where the sum is in $\mathbb{Z}_q$. Finally $\mathbb{Z}_q[X]_{\leq d}$ denotes the set of polynomials in $\mathbb{Z}_q[X]$ with degree at most d.

2.1 Adversarial and Communication Models

We consider security against a malicious static adversary, which may arbitrarily deviate from the protocol but chooses what parties to corrupt before the execution starts[5]. The adversary will corrupt at most $(n - \ell)/2$ parties, for some integer $\ell > 0$, which we think of as a small fraction of n. For simplicity, we assume access to an authenticated bulletin board. Once a party posts a message

[5] This is the model assumed by most random beacon constructions with the only exception (to the best of our knowledge) of [15], and proving security against adaptive adversary would require expensive techniques such as non-committing encryption.

to the bulletin board, it becomes immutable and immediately available to all other parties, who can also verify the authenticity of the message (*i.e.* that it was indeed posted by a given party). Such a bulletin board could be substituted by a blockchain based public ledger, a public key infrastructure and digital signatures; however, modeling the corner cases that arise in this scenario introduces a number of technicalities that are not the main focus of this work. We assume synchronous communication: messages sent (or posted to the bulletin board) in a round are guaranteed to be received by all honest parties by the next round. Our protocols can be extended to the partially synchronous setting (*i.e.* with adversarially controlled but finite communication delays) via standard techniques (*e.g.* waiting for a majority of valid shares to be received [9, 3, 2]).

2.2 Packed Shamir Secret Sharing

Secret sharing allows to distribute a secret among n parties $P_1, \ldots, P_n$ by delivering a share to each party, so that only certain subsets of these parties can later reconstruct it by pooling together their received shares.

In (t, ℓ)-packed Shamir secret sharing over $\mathbb{Z}_q$ (with q prime), the secret is a vector $(s_0, \ldots, s_{\ell-1}) \in \mathbb{Z}_q^\ell$, and order to share this secret, the dealer chooses $f \in \mathbb{Z}_q[X]_{\leq t+\ell-1}$ with $f(-j) = s_j$ for $j \in [0, \ell - 1]$, and sends the evaluation $\sigma_i := f(i)$ as share to party P_i. Here we are assuming that $q \geq n + \ell$, so that the evaluation points $-j$ for the secrets and i for the shares are disjoint. The more classical Shamir secret sharing scheme is the case $\ell = 1$. The (t, ℓ)-packed Shamir secret sharing satisfies t-privacy, meaning that the secret vector is distributed independently from any set of t shares, and $t + \ell$-reconstruction, i.e., the secret vector can be recovered from any set A of $t + \ell$ shares. For the latter, Lagrange interpolation can be used: each secret coordinate can be reconstructed as a linear combination of the shares in the set. Namely, $s_j = \sum_{i \in A} L_{i,A}(-j) \cdot \sigma_i$ for the Lagrange polynomials $L_{i,A}(X) = \prod_{k \in A, k \neq i} \frac{X-k}{i-k}$.

2.3 Non-interactive zero knowledge proofs

In a zero knowledge proof of knowledge a prover wants to convince a verifier that she knows a piece of information (witness) that makes certain statement true without revealing anything about this witness. Non-interactive proofs carry out this with a single message from the prover. We consider proofs for public verifiers, meaning anyone can verify the proof. We need non-interactive zero-knowledge proofs of knowledge for two types of statements in a cyclic group of prime order q: discrete logarithm equality (DLEQ) proofs [10] and low-degree exponent interpolation (LDEI) [9]. In fact, DLEQ proofs can be seen as a special case of LDEI proofs. Both can realized from Sigma-protocol techniques.

In a LDEI proof, the statement is given by a vector of elements of the group, and the prover claims that their discrete logarithms with respect to a given vector of public generators are evaluations of a polynomial of degree lower than certain bound. If we set this bound to be 0 (i.e. the only accepted polynomials are constants) then we have a DLEQ proof: we are proving that the discrete logarithm of the given elements with respect to the generators are all equal. A non-interactive LDEI proof of knowledge (in the random oracle model) of the interpolating polynomial was presented in [9] and is given in Figure 1.

Low-degree exponent interpolation (LDEI) ZKPoK
$\pi_{LDEI}((g_i)_{i=1}^m, (x_i)_{i=1}^m, d)$.

Setup: Finite group $\mathbb{G}$ of prime order q, fixed pairwise distinct elements $\alpha_1, \ldots, \alpha_m$ in $\mathbb{Z}_q$, a random oracle $H : \mathbb{G}^{3m} \to \mathbb{Z}_q$.
Public inputs: $g_1, \ldots, g_m, x_1, \ldots, x_m \in \mathbb{G}$, $d \in \mathbb{Z}$, $d \geq 0$.
Statement: The prover knows $w(X) \in \mathbb{Z}_q[X]_{\leq d}$ with $x_i = g_i^{w(\alpha_i)}$ $\forall i \in [m]$.
Protocol:
- The prover samples $u(X) \leftarrow \mathbb{Z}_q[X]_{\leq d}$ and computes $a_i = g_i^{u(\alpha_i)}$ for all $i \in [m]$, $e = H(g_1, \ldots, g_m, x_1, \ldots, x_m, a_1, \ldots, a_m)$, and $z(X) = u(X) - e \cdot w(X)$. The proof is (e, z).
- The verifier computes $a_i = g_i^{z(\alpha_i)} x_i^e$ $\forall i \in [m]$ and accepts iff $\deg z \leq d$ and $e = H(g_1, \ldots, g_m, x_1, \ldots, x_m, a_1, \ldots, a_m)$.

Fig. 1: LDEI zero knowledge proof of knowledge π_{LDEI} from [9]. $\pi_{DLEQ}((g_i)_{i=1}^m, (x_i)_{i=1}^m)$ will denote the case $d = 0$.

2.4 Publicly Verifiable Secret Sharing (PVSS)

A publicly verifiable secret sharing scheme allows any external party to verify the correct sharing and reconstruction of a secret, with the help of zero knowledge proofs posted respectively by the dealer and the reconstructing parties. We will base our constructions upon techniques from SCRAPE [8] and the subsequent modifications in ALBATROSS [9]. These schemes in turn follow the blueprint of Schoenmakers' PVSS [33]. The PVSS in ALBATROSS can be seen as a generalization of SCRAPE that allows for a flexible trade-off where the dealer can share a vector of ℓ group elements, while at most $t \leq (n - \ell)/2$ parties can be corrupted if we want both t-privacy and $(n - t)$-reconstruction, which will be necessary later. In contrast, the parameters in SCRAPE (and in Schoenmakers' PVSS) would correspond to the case $\ell = 1$. The amortized computation and communication per secret shared in ALBATROSS becomes much better as ℓ grows.

If a party correctly PVSSs a secret, meaning that the (publicly verifiable) proof of correct sharing is valid, then this party is committed to the secret in a way that the commitment can be opened by a large enough set of honest share-receivers, regardless of whether the dealer wants to open it. This allows to construct a commit-and-open random beacon that does not present the problem that parties may decide on opening their commitments depending on other opened commitments they have seen. More precisely, in a PVSS-based random beacon each party PVSSs a random secret and everyone can compute the set $\mathcal{Q}$ of parties that have correctly shared; all the secrets dealt by parties in $\mathcal{Q}$ are then opened and the final output is obtained by applying a randomness extractor to these opened secrets, which guarantees that the result is uniformly random and independent from the inputs of any set of t parties. The randomness extractor in ALBATROSS is built from a t-resilient matrix, which we define next.

Definition 1. *A matrix $M \in \mathbb{Z}_q^{r \times m}$ is t-resilient if for any $A = \{i_1, ..., i_t\} \subseteq [m]$ of size t, $M\mathbf{v}$ is independent from the coordinates of $\mathbf{v}$ indexed by A, i.e. for any $(y_1, \ldots, y_t) \in \mathbb{Z}_q^t$, the distribution of $M\mathbf{v}$ when conditioned to $v_{i_1} = y_1, \ldots, v_{i_t} = y_t$ and $(v_j)_{j \notin A}$ being uniform in $\mathbb{Z}_q^{m-t}$, is uniform in $\mathbb{Z}_q^r$.*

A t-resilient matrix with the parameters above needs to satisfy $r \leq m - t$. An optimal choice (i.e. $r = m - t$) is to let M be a transpose of a Vandermonde matrix (we are assuming $q \geq m$). For computation efficiency reasons, [9] chooses M with entries $M_{ij} = \alpha^{i \cdot j}$ for some $\alpha \in \mathbb{Z}_q$ of order larger than $r \cdot m$.

In our case, $m = n - t$ (as we can only guarantee $n - t$ parties have correctly shared their inputs) and thus $\ell' = n - 2t$ is the maximum size of the output of the t-resilient function. In [9], parameters were set such that $\ell' = \ell$. In this paper, we keep ℓ and $\ell' = n - 2t$ as two separate parameters.

For reference, we include a description of ALBATROSS using our notation in Appendix A.

2.5 Verifiable Random Functions (VRFs)

A verifiable random function (VRF) [26] is a pseudorandom function that can be evaluated by the owner of a secret key, who at the same time gets a proof or correct evaluation that can be verified with the corresponding public key. A VRF scheme consists of three algorithms (λ is a security parameter):

- $\mathsf{KeyGen}(1^\lambda)$: outputs a public and secret key pair $(\mathsf{pk}, \mathsf{sk})$.
- $\mathsf{Eval}(\mathsf{sk}, x)$ is a deterministic algorithm which outputs a pair (y, π) where y is the output of the function and π is a proof.
- $\mathsf{Verify}(\mathsf{pk}, x, y, \pi)$ is a probabilistic algorithm that outputs 0 or 1 (respectively meaning "reject" or "accept" the proof).

It has been observed in [15] that the standard VRF security definition is not sufficient in the randomness beacon setting. Pseudorandomness only holds if the key pair has been honestly generated (*i.e.* by KeyGen) but when it is generated maliciously the adversary can bias VRF outputs. In VRF-based beacons (*e.g.* Figure 3), the adversary can generate its own key pairs maliciously, and hence we require the VRF to be unpredictable under malicious key generation as defined in [15]. A VRF scheme $(\mathsf{KeyGen}(1^\lambda), \mathsf{Eval}(\mathsf{sk}, x), \mathsf{Verify}(\mathsf{pk}, x, y, \pi))$ with unpredictability under malicious key generation is secure if it holds that:

- (complete provability): for every $(\mathsf{pk}, \mathsf{sk})$ generated by KeyGen, and every x, if $(y, \pi) = \mathsf{Eval}(\mathsf{sk}, x)$ then $\mathsf{Verify}(\mathsf{pk}, x, y, \pi) = 1$ with overwhelming probability;

VRF from Ouroboros Praos

Setup: Let G be a cyclic group of prime order q, with generator g. Let $H : \{0,1\}^* \to \{0,1\}^{\ell_{VRF}}$ and $H' : \{0,1\}^* \to G$ be random oracles. In addition we implicitely need a random oracle $H^* : \{0,1\}^* \to \mathbb{Z}_q$ for the DLEQ proof.
Commands:
- $\mathsf{KeyGen}(1^\lambda)$ chooses a uniformly random $\mathsf{sk} \in \mathbb{Z}_q$, sets $\mathsf{pk} = g^{\mathsf{sk}}$ and outputs $(\mathsf{pk}, \mathsf{sk})$
- $\mathsf{Eval}(\mathsf{sk}, x)$ sets $y = H(x, u)$ where $u = H'(x)^{\mathsf{sk}}$. It moreover defines $\pi = (u, \pi_{DLEQ}((g, H'(x)), (\mathsf{pk}, u)))$, the latter being the proof that $g^k = \mathsf{pk}$ and $H'(x)^k = u$ for a common k, in this case $k = \mathsf{sk}$. It outputs (y, π).
- $\mathsf{Verify}(\mathsf{pk}, x, y, \pi)$ parses $\pi = (u, \pi')$, checks that π' is a correct DLEQ proof for $(g, H'(x)), (\mathsf{pk}, u))$ and checks $y = H(x, u)$. It accepts if all these checks pass.

Fig. 2: VRF with unpredictability under malicious key generation [15].

- (unique provability): for every x, any $y_1 \neq y_2$, and any proofs π_1, π_2, then at least one of $\mathsf{Verify}(\mathsf{pk}, x, y_1, \pi_1)$ or $\mathsf{Verify}(\mathsf{pk}, x, y_2, \pi_2)$ output 0 with overwhelming probability.
- (pseudorandomness): no PPT adversary can distinguish between $\mathsf{Eval}(\mathsf{sk}, x)$ and a uniformly random string, even when having chosen x, after seeing pk.
- (unpredictability under malicious key generation) no PPT adversary who generated $(\mathsf{pk}, \mathsf{sk})$ arbitrarily can distinguish between $\mathsf{Eval}(\mathsf{sk}, x)$ and a uniformly random string for an unknown uniformly random x.

VRF-based beacon

Setup: An initial seed σ_0, and a random oracle $H : \{0,1\}^{\ell_{VRF}} \to \{0,1\}^m$.
Beacon:
1. Each P_i executes $\mathsf{KeyGen}(1^\lambda)$ of the VRF obtaining $(\mathsf{pk}_i, \mathsf{sk}_i)$ and publishes pk_i.
2. At round $r = 1, 2, \ldots$: Let $m_r = r || \sigma_{r-1}$.
   (a) Every party P_i computes and publishes $(\sigma_r^i, \pi^i) = \mathsf{Eval}(\mathsf{sk}_i, m_r)$.
   (b) Each party defines I to be the set of i for which $\mathsf{Verify}(\mathsf{pk}_i, m_r, \sigma_r^i, \pi^i)$ passes and computes $\sigma_r = \bigoplus_{i \in I} \sigma_r^i$. The output of this round is $w_r = H(\sigma_r)$.

Fig. 3: VRF-based beacon from [15].

We describe in Figure 2 the VRF with unpredictability under malicious key generation from [15]. A randomness beacon based on this VRF was presented in [15], and is described in Figure 3. The beacon uses an initial seed σ_0 which may come from a CRS or, as will happen in our multi-tiered beacon, as an output from some protocol. The beacon proceeds iteratively as follows: At each round r each party has a key-pair for a VRF and evaluates the VRF on the seed σ_{r-1}

obtaining σ_r^i. The parties compute σ_r as the XOR of the correctly computed σ_r^i (which the can check using the verification procedure and the public keys). The output of that round is the hash $H(\sigma_r)$, while σ_r is used as seed for the next round. Note that malicious parties may bias the result by waiting until honest parties publish their evaluations of the VRF and then deciding whether they publish theirs.

2.6 Threshold Verifiable Random Functions (TVRFs)

Analogously to the case of signatures, one can define a distributed notion of verifiable random functions, where each party can compute a partial evaluation, and any $t+1$ valid partial evaluations can be combined to obtain the global evaluation of the VRF. Following [19] we define a TVRF as the tuple of algorithms below, where as usual t denotes the corruption threshold:

- $\mathsf{DistKeyGen}(1^\lambda)$: outputs secret keys $\mathsf{tsk}_i, i \in [n]$, corresponding public partial keys tpk_i and global public key tpk.
- $\mathsf{PartialEval}(x, \mathsf{tsk}_i, \mathsf{tpk}_i)$ is a deterministic algorithm which outputs a pair $m_i = (y_i, \pi_i)$ where y_i is a partial evaluation and π_i is a proof.
- $\mathsf{Combine}(\mathsf{tpk}, \{\mathsf{tpk}_i\}, x, A, (m_i)_{i\in A})$, where $A \subseteq [n]$ with $|A| \geq t+1$, outputs either (y, π) consisting of global evaluation y and a global proof π, or $\perp$.
- $\mathsf{Verify}(\mathsf{tpk}, x, y, \pi)$ is a probabilistic algorithm that outputs 0 or 1 (respectively meaning "reject" or "accept" the proof).

A TVRF has the following properties:

- Consistency: Given any x, when we apply $\mathsf{Combine}$ to any $\geq t+1$ correct partial evaluations $(m_i)_{i\in A}$, we obtain the same y.
- Robustness: If $\mathsf{Combine}$ outputs a pair (y, π), then $\mathsf{Verify}(\mathsf{tpk}, x, y, \pi) = 1$
- Uniqueness: for every x, for any $y_1 \neq y_2$, and any proofs π_1, π_2, then at least one of $\mathsf{Verify}(\mathsf{tpk}, x, y_1, \pi_1)$ or $\mathsf{Verify}(\mathsf{tpk}, x, y_2, \pi_2)$ output 0 with overwhelming probability.
- Pseudorandomness: roughly, the adversary correcting t parties cannot distinguish the output of the function from a uniformly random value, even when chosing the input.

We describe in Figure 4 a DDH-based threshold VRF inspired by a threshold Boneh-Lynn-Shacham (BLS) signatures from in [19]. Notice that the original DRAND/Dfinity TVRF uses actual pairing based threshold BLS signatures in order to achieve compact proofs. Both this construction and the improved GLOW TVRF construction are proven secure in [19] and could serve as a building block for the DRAND/Dfinity beacon. However, we present the DDH based version for the sake of simplicity and for making it clear that all Mt. Random building blocks can be instantiated from DDH in the ROM. Note that we do not make the instantiation of $\mathsf{DistKeyGen}$ explicit, as we both introduced our own scheme in Section 3 and discussed a number of alternatives in the Introduction.

The DRAND/Definity beacon we alluded to above is given in Figure 5. Notice that, in the threshold scenario, the pseudorandomness property of the standard

DDH-based threshold VRF (DDH-DVRF in [19])

Setup: Let G be a cyclic group of prime order q, with generator g. Let $H : \{0,1\}^* \to G$ a random oracle. In addition we implicitly need a random oracle $H^* : \{0,1\}^* \to \mathbb{Z}_q$ for the DLEQ proof.

Commands:

- $\mathsf{DistKeyGen}(1^\lambda)$ The distributed key generation creates $\mathsf{tsk}_i \in \mathbb{Z}_q$ such that $(\mathsf{tsk}_i)_{i=1}^n$ is a valid Shamir sharing of some secret $\mathsf{tsk} \in \mathbb{Z}_q$. It outputs public $\mathsf{tpk}_i = g^{\mathsf{tsk}_i}$ and $\mathsf{tpk} = g^{\mathsf{tsk}}$, and privately tsk_i only to party P_i, for $i \in [n]$.
- $\mathsf{PartialEval}(x, \mathsf{tsk}_i, \mathsf{tpk}_i)$: y_i is computed by P_i as $y_i = H(x)^{\mathsf{tsk}_i}$. In addition compute $\pi_i = \pi_{DLEQ}((g, H(x)), (\mathsf{tpk}_i, y_i))$.
- $\mathsf{Combine}(\mathsf{pk}, \{\mathsf{tpk}_i\}, x, A, (y_i, \pi_i)_{i \in A})$: A subset $A' \subseteq A$ is selected such that A' has cardinality $t+1$ and π_i is accepted for $i \in A'$. Then $y = \prod_{i \in A'} y_i^{L_{i,A'}(0)}$ and $\pi = (y_i, \pi_i)_{i \in A'}$
- $\mathsf{Verify}(\mathsf{tpk}, x, y, \pi)$: Parse $\pi = (y_i, \pi_i)_{i \in A'}$, verify all π_i for $i \in A'$, and check whether $y = \prod_{i \in A'} y_i^{L_{i,A'}(0)}$. Output 1 if all checks pass, otherwise output 0.

Fig. 4: DDH-based threshold VRF (DDH-DVRF in [19]).

definition is sufficient to guarantee that VRF outputs are unbiased because distributed key generation guarantees that keys are correctly generated.

The DRAND/Dfinity beacon

We assume $t \leq (n-1)/2$, so there are at least $t+1$ honest parties. We fix an initial seed σ_0 and $H' : \mathbb{G} \to \{0,1\}^*$ a hash function.

1. Parties invoke $\mathsf{DistKeyGen}$ from the TVRF to obtain the keys $(\mathsf{tsk}, \mathsf{tsk}_i, \mathsf{tpk}_i)$.
2. At round $r = 1, 2, \ldots$: Let $m_r = r || \sigma_{r-1}$.
   (a) P_i computes and broadcasts $(y_i, \pi_i) = \mathsf{PartialEval}(m_r, \mathsf{tsk}_i, \mathsf{tpk}_i)$.
   (b) Each party applies locally $\mathsf{Combine}(\mathsf{pk}, \{\mathsf{tpk}_i\}_{i \in [n]}, m_r, [n], ((y_i, \pi_i))_{i \in [n]})$ obtaining values (y, π).
   (c) We define $\sigma_r = y$ (for use in the next round). The output of round r is $z = H'(\sigma_r)$.

Note that at each step, a public verifier can attest the correctness of the computation by running $\mathsf{Verify}(\mathsf{tpk}, x, y, \pi)$.

Fig. 5: The DRAND/Dfinity beacon.

2.7 Threshold Encryption

A threshold encryption scheme allows to encrypt a message towards a group of receivers, such that the message can be decrypted by any $t+1$ of them, but not less. A threshold encryption scheme is composed by the following algorithms:

- $\mathsf{DistKeyGen}(1^\lambda)$: outputs secret keys $\mathsf{tsk}_i, i \in [n]$, corresponding public partial keys tpk_i and a global public key tpk.
- $\mathsf{Enc}(\mathsf{tpk}, m)$ outputs a ciphertext E.
- $\mathsf{LocalDec}(\mathsf{tpk}_i, \mathsf{tsk}_i, E)$ outputs a partial decrypted message x_i.
- $\mathsf{GlobalDec}(\mathsf{tpk}, I, \{\mathsf{tpk}_i\}_{i\in I}, \{x_i\}_{i\in I}, E)$, where $I \subseteq [n]$ with $|I| \geq t+1$, outputs a decrypted message m' or an error $\perp$.

We describe informally below the properties we want from a threshold encryption scheme, following the work of [13], which we refer to for formal definitions. These are attained by the threshold version of El Gamal encryption that we show in Figure 6.

- Completeness: If the keys have been honestly generated with DistKeyGen, a message m honestly encrypted, and a set I of at least $t+1$ honest parties have computed correct partial decryptions x_i of the corresponding cyphertexts with their keys, then GlobalDec, taking that cyphertext and the public keys and partial decryptions of I, will output m
- Robustness: Given as inputs 2 subsets I and J of at least $t+1$ parties and their corresponding partial decryptions of a same cyphertext, if GlobalDec does not reject then it outputs the same message on both inputs with overwhelming probability.
- IND-CPA against static corruption: We assume the adversary corrupts a set A of at most t parties at the beginning of the protocol. The scheme is IND-CPA secure if the adversary cannot guess (with success probability non-negligibly larger than 1/2) the plaintext corresponding to a given cyphertext, even if this a cyphertext encrypts a message from a set of 2 possible messages that the adversary has chosen, and given of course that the adversary knows all the public keys and the secret keys corresponding to A.

3 Distributed Key Generation via PVSS

We introduce a DKG protocol $\pi_{DDH-DKG}$ in Figure 7 that is publicly verifiable and guarantees the adversary cannot bias the global public key. Moreover, the $\pi_{DDH-DKG}$ can be easily extended to generate more than one threshold key pair (Remark 2) and to refresh existing secret key shares (Remark 1). We formally analyse the security of $\pi_{DDH-DKG}$ in the real/ideal simulation paradigm with sequential composition. This paradigm is commonly used to analyse cryptographic protocol security and provides strong security guarantees, namely that several instances of the protocol can be executed in sequence while preserving their security. More details about this model can be found in [7]. Concretely, we will prove that $\pi_{DDH-DKG}$ securely implements the functionality $\mathcal{F}_{DDH-DKG}$ in Figure 8.

Theorem 1. *Under the DDH assumption and assuming an authenticated bulletin board, $\pi_{DDH-DKG}$ securely realizes $\mathcal{F}_{DDH-DKG}$ in the random oracle model against a malicious static PPT adversary $\mathcal{A}$ corrupting $t \leq \frac{n-1}{2}$ parties.*

Threshold El Gamal encryption scheme.

Setup: Let G be a cyclic group of prime order q, with generator g.
Commands:

- $\mathsf{DistKeyGen}(1^\lambda)$: The distributed key generation creates $\mathsf{tsk}_i \in \mathbb{Z}_q$ such that $(\mathsf{tsk}_i)_{i=1}^n$ is a valid Shamir sharing of some secret $\mathsf{tsk} \in \mathbb{Z}_q$. It outputs public $\mathsf{tpk}_i = g^{\mathsf{tsk}_i}$ and $\mathsf{tpk} = g^{\mathsf{tsk}}$, and privately tsk_i only to party P_i, for $i \in [n]$.
- $\mathsf{Enc}(\mathsf{tpk}, m)$: To encrypt a message $m \in \mathbb{G}$, sample r uniformly at random in $\mathbb{Z}_q$, and output $E = (g^r, \mathsf{tpk}^r \cdot m) := (c, d) \in \mathbb{G}^2$
- $\mathsf{LocalDec}(\mathsf{tpk}_i, \mathsf{tsk}_i, E)$ outputs $x_i = (y_i, \pi_i)$ where $y_i = c^{\mathsf{tsk}_i}$ and $\pi_i = \pi_{DLEQ}((g, c), (\mathsf{tpk}_i, y_i))$.
- $\mathsf{GlobalDec}(\mathsf{tpk}, I, \{\mathsf{tpk}_i\}_{i\in I}, \{x_i\}_{i\in I}, c)$ outputs $\perp$ if no more than t DLEQ proofs $\pi_i, i \in I$ pass. Otherwise, it takes a subset $I' \subseteq I$ of cardinality exactly $t+1$ such that $\pi_{i\in I'}$ are all correct, and computes

$$m' = d \cdot (\prod_{i\in I'} y_i^{-L_{i,I'}(0)})$$

Fig. 6: Threshold El Gamal encryption scheme

Proof. In order to prove this theorem, we construct a simulator $\mathcal{S}$ that interacts with the adversary $\mathcal{A}$ and with functionality $\mathcal{F}_{DDH-DKG}$ in such a way that view of $\mathcal{A}$ in a real execution of $\pi_{DDH-DKG}$ is indistinguishable from its view in an ideal execution with $\mathcal{S}$ and $\mathcal{F}_{DDH-DKG}$. Let $P^{\mathcal{A}}$ be the set of corrupted parties. $\mathcal{S}$ simulates the bulletin board and the random oracle towards $\mathcal{A}$ and proceeds as follows:

1. In round 1, $\mathcal{S}$ proceeds as follows:
   - Upon receiving (Gen, sid, P_a) from $\mathcal{F}_{DDH-DKG}$ for an honest party P_a, $\mathcal{S}$ acts exactly as an honest party would, sampling a random $s^{(a)} \in \mathbb{Z}_q$, dealing it with the SCRAPE PVSS and, for all $i \in [n]$, posting $\hat{S}_i^{(a)}, \pi^{(a)}, E_i^{(a)}$ on the bulletin board. Finally, add $\mathcal{P}_a$ to $\mathcal{Q}$, *i.e.* the set of parties who provide valid shares.
   - When $\mathcal{A}$ posts $\hat{S}_i^{(a)}, \pi^{(a)}, E_i^{(a)}$ for $i = 1, \ldots, n$ on the bulletin board on behalf of a corrupted party $P_a \in P^{\mathcal{A}}$, $\mathcal{S}$ checks whether to add $\mathcal{P}_a$ to $\mathcal{Q}$ or not:
     (a) Verify the proof $\pi^{(a)}$ is valid.
     (b) Use the extractor from the zero knowledge proof π_{LDEI} to obtain $\sigma_i^{(a)}$ from $\pi^{(a)}$ for all $i \in [n]$.
     (c) Verify that $E_i^{(a)} = \sigma_i^{(a)} \oplus H(g^{\sigma_i^{(a)}})$ for all $i \in [n]$.
     (d) If and only if all these checks pass, add $\mathcal{P}_a$ to $\mathcal{Q}$.

   When Round 1 is finished, $\mathcal{S}$ has computed $\mathcal{Q}$ exactly as in $\pi_{DDH-DKG}$, since it checked that all messages $\hat{S}_i^{(a)}, \pi^{(a)}, E_i^{(a)}$ from corrupted partiers pass the checks in Rounds 2 and 3 before adding these parties to $\mathcal{Q}$.

Distributed key generation - $\pi_{DDH-DKG}$

Parameters: Let n be the number of parties that receive shares, and let $1 \leq t \leq (n-1)/2$ be an integer, the corruption threshold.
Setup: A public bulletin board, field $\mathbb{Z}_q$, and DDH-hard group $\mathbb{G}$ with generator g. Every party in the system has a private key $\mathsf{sk}_i \in \mathbb{Z}_q$, and public key $\mathsf{pk}_i = g^{\mathsf{sk}_i}$. A random oracle $H : \mathbb{G} \to \{0,1\}^{\lceil \log q \rceil}$. We also assume some injective encoding $\mathbb{Z}_q \to \{0,1\}^{\lceil \log q \rceil}$ which is easy to invert.
Protocol

1. Each party P_a proceeds as follows:
   - P_a chooses $s^{(a)} \in \mathbb{Z}_q$ and deals it with the SCRAPE PVSS: P_a selects a polynomial $f^{(a)} \in \mathbb{Z}_q[X]_{\leq t}$ with $f^{(a)}(0) = s^{(a)}$ and, for all $i \in [n]$, defines $\sigma_i^{(a)} = f^{(a)}(i)$, $\hat{S}_i^{(a)} = \mathsf{pk}_i^{\sigma_i^{(a)}}$ and $\pi^{(a)} = \pi_{LDEI}((\mathsf{pk}_i)_{i=1}^n, (\hat{S}_i^{(a)})_{i=1}^n, t)$.
   - For $i \in [n]$, P_a computes $E_i^{(a)} = \sigma_i^{(a)} \oplus H(g^{\sigma_i^{(a)}})$ and posts $\hat{S}_i^{(a)}, \pi^{(a)}, E_i^{(a)}$.
2. Each P_i verifies the proof $\pi^{(a)}$ and posts a complaint on the bulletin board if this proof is invalid. Moreover P_i computes $\sigma_i^{(a)}$ from $E_i^{(a)}$ as $\sigma_i^{(a)} = H((\hat{S}_i^{(a)})^{\frac{1}{\mathsf{sk}_i}}) \oplus E_i^{(a)}$ and checks whether $\hat{S}_i^{(a)} = \mathsf{pk}_i^{\sigma_i^{(a)}}$. If this does not hold then P_i posts a complaint against P_a to the bulletin board. Otherwise, P_i sets $S_i^{(a)} = g^{\sigma_i^{(a)}}$.
3. If no complaints were posted, ignore this round and go to round 4. Otherwise:
   - If a proof $\pi^{(a)}$ receives more than t complaints, P_a is disqualified.
   - If P_i complains against P_a about its received encrypted share, then P_a reveals $\sigma_i^{(a)}$. If $\hat{S}_i^{(a)} \neq \mathsf{pk}_i^{\sigma_i^{(a)}}$ or $E_i^{(a)} \neq \sigma_i^{(a)} \oplus H(g^{\sigma_i^{(a)}})$, P_a is disqualified.
4. Let $\mathcal{Q}$ be the set of parties who posted encrypted shares and proofs without being disqualified in step 3. For all i, party P_i proceeds as follows:
   (a) P_i computes $\hat{S}_i = \prod_{a \in \mathcal{Q}} \hat{S}_i^{(a)}$, $S_i = \prod_{a \in \mathcal{Q}} S_i^{(a)}$, and $\sigma_i = \sum_{a \in \mathcal{Q}} \sigma_i^{(a)}$.
   (b) P_i publishes $\hat{S}_i$, S_i and $\pi_{DLEQ}(g, S_i, \mathsf{pk}_i, \hat{S}_i)$ in the bulletin board.
5. Finally, after round 4, all parties proceed as follows:
   (a) For all $\hat{S}_i, S_i, \pi_{DLEQ}((g, S_i), (\mathsf{pk}_i, \hat{S}_i))$ posted to the bulletin board, verify $\hat{S}_i = \prod_{a \in \mathcal{Q}} \hat{S}_i^{(a)}$ and the proof $\pi_{DLEQ}((g, S_i), (\mathsf{pk}_i, \hat{S}_i))$. Let I be the set of all indices for which these checks pass.
   (b) Let $J \subseteq I$ be a set of cardinality $t+1$ (e.g. the first $t+1$ parties in I). The output global public key is $\mathsf{tpk} = S = \prod_{i \in J} S_i^{L_{i,J}(0)}$ (where $L_{i,J} = \prod_{k \in J} \frac{-k}{i-k}$). The i-th partial public key (for $i \in I$) is $\mathsf{tpk}_i = S_i$. The i-th partial secret key (for $i \in I$) is $\mathsf{tsk}_i = \sigma_i$. Finally, note the global secret key is implicitly defined as $\mathsf{tsk} = s = \sum_{a \in \mathcal{Q}} s^{(a)}$.

Fig. 7: Protocol $\pi_{DDH-DKG}$ for distributed key generation via SCRAPE.

2. For every corrupted party $P_i \in P^{\mathcal{A}} \cap \mathcal{Q}$, $\mathcal{S}$ computes the secret key shares $\sigma_i = \sum_{a \in \mathcal{Q}} \sigma_i^{(a)}$ and sends $(\textsc{Gen}, sid, P_i)$ and $(\textsc{SetShare}, sid, P_i, \sigma_i)$ to $\mathcal{F}_{DDH-DKG}$. $\mathcal{S}$ waits for message $(\textsc{Keys}, sid, \sigma_i, \{\mathsf{tpk}_j\}_{j \in \mathcal{Q}}, \mathsf{tpk})$ for $P_i \in P^{\mathcal{A}}$ from $\mathcal{F}_{DDH-DKG}$. Notice that $\mathcal{S}$ can do that since it knows $\sigma_i^{(a)}$ provided by simulated honest parties and it has extracted the corresponding values from corrupted parties.

Functionality $\mathcal{F}_{DDH-DKG}$

$\mathcal{F}_{DDH-DKG}$ is parameterized by a DDH-hard cyclic group $\mathbb{G}$ of prime order q, with generator g. Let n and $1 \leq t \leq (n-1)/2$ be integers. $\mathcal{F}_{DDH-DKG}$ interacts with parties $P_1, \ldots, P_n$ and an adversary $\mathcal{S}$ that corrupts at most t parties. $\mathcal{F}_{DDH-DKG}$ works as follows:

– Upon receiving $(\textsc{Gen}, sid, P_i)$ from a party P_i:
  1. If P_i is honest, forward $(\textsc{Gen}, sid, P_i)$ to $\mathcal{S}$.
  2. If P_i is corrupted, wait for $\mathcal{S}$ to send $(\textsc{SetShare}, sid, P_i, \sigma_i)$ where $\sigma_i \in \mathbb{Z}_q$ and set $\mathsf{tpk}_i = g^{\sigma_i}$.
  3. Let J be the set of all parties P_j who sent $(\textsc{Gen}, sid, P_j)$. If all honest parties are in J, proceed as follows:
     (a) Sample a random polynomial $f \in \mathbb{Z}_q[X]_{\leq t}$ with $f(i) = \sigma_i$ for all σ_i sent by $\mathcal{S}$ in step 2). [a] For every honest party P_h, set $\mathsf{tpk}_h = g^{\sigma_h}$ with $\sigma_h = f(h)$.
     (b) Set $\mathsf{tpk} = g^{f(0)}$.
     (c) For corrupted parties $P_c \in J$, send $(\textsc{Keys}, sid, \sigma_c, \{\mathsf{tpk}_j\}_{j\in J}, \mathsf{tpk})$ to $\mathcal{S}$.
     (d) Wait for $\mathcal{S}$ to answer with $(\textsc{Abort}, sid, C)$ where C is a set of corrupted parties.
     (e) For all $j \in J \setminus C$, send $(\textsc{Keys}, sid, \sigma_j, \{\mathsf{tpk}_k\}_{k\in J\setminus C}, \mathsf{tpk})$ to P_j. [b]

[a] This is possible since the adversary can only set at most t values σ_i.

[b] Notice that $\{\mathsf{tpk}_k\}_{k\in J\setminus C}$ can always be used to obtain $\mathsf{tpk} = g^{f(0)}$ by Lagrange interpolation because $|J \setminus C| \geq n - t > t$.

Fig. 8: Distributed Key Generation Functionality $\mathcal{F}_{DDH-DKG}$

3. In rounds 2 and 3, $\mathcal{S}$ executes exactly the same instructions as an honest party. Notice that this will yield the same set $\mathcal{Q}$ computed in step 1.
4. In round 4, for every i such that $P_i \in \mathcal{Q}$ is honest, computes $\hat{S}_i = \prod_{a\in\mathcal{Q}} \hat{S}_i^{(a)}$, uses the simulator from the ZK proof π_{DLEQ} to generate an accepting proof $\pi_{DLEQ}(g, \mathsf{tpk}_i, \mathsf{pk}_i, \hat{S}_i)$ and posts $\hat{S}_i$, tpk_i and $\pi_{DLEQ}(g, \mathsf{tpk}_i, \mathsf{pk}_i, \hat{S}_i)$ on the bulletin board.
5. After round 4, let C be the set of corrupted parties who post $\hat{S}_i$, S_i and $\pi_{DLEQ}(g, S_i, \mathsf{pk}_i, \hat{S}_i)$ with an invalid proof $\pi_{DLEQ}(g, S_i, \mathsf{pk}_i, \hat{S}_i)$. $\mathcal{S}$ sends $(\textsc{Abort}, sid, C)$ to $\mathcal{F}_{DDH-DKG}$.
6. $\mathcal{S}$ executes the remainder of the protocol as an honest party would and, when $\mathcal{A}$ terminates, outputs whatever $\mathcal{A}$ outputs.

We now show that the execution with $\mathcal{S}$ and $\mathcal{F}_{DDH-DKG}$ is indistinguishable from an execution of $\pi_{DDH-DKG}$ with $\mathcal{A}$. First of all, notice that in rounds 1, 2 and 3 all messages sent from $\mathcal{S}$ to $\mathcal{A}$ (through the bulletin board) are distributed exactly as in $\pi_{DDH-DKG}$. Moreover, notice that after round 1 is finished $\mathcal{S}$ computes the same set $\mathcal{Q}$ as parties would compute after round 3 of $\pi_{DDH-DKG}$. This is so because $\mathcal{S}$ is able to perform all the verification done by individual parties in rounds 2 and 3 all at once after extracting $\sigma_i^{(a)}$ from

$\pi^{(a)}$ for all corrupted parties P_a. Having determined $\mathcal{Q}$, $\mathcal{S}$ is able to determine the choices of secret key shares σ_a from all corrupted parties, which might be made after the adversary has seen all honest party messages in round 1. Hence, $\mathcal{S}$ provides consistent values σ_a to $\mathcal{F}_{DDH-DKG}$.

It remains to be shown that the messages exchanged by $\mathcal{S}$ and $\mathcal{A}$ in round 4 are indistinguishable from those exchanged by honest parties and $\mathcal{A}$ in an execution of $\pi_{DDH-DKG}$, which intuitively means that $\mathcal{A}$ cannot bias the global public key even though it can choose secret key shares σ_a for corrupted parties. In round 4, we take advantage of the fact that, for i and a such that parties $P_i \in \mathcal{Q}$ and $P_a \in \mathcal{Q}$ are honest, $\hat{S}_i^{(a)}$ and $E_i^{(a)}$ reveal no information about $\sigma_i^{(a)}$ to $\mathcal{A}$. First, notice that it is proven in [8] that $\hat{S}_i^{(a)}$ is indistinguishable from a random group element for $\mathcal{A}$ under the DDH assumption. Moreover, since $\mathcal{A}$ is PPT, it can only guess $\sigma_i^{(a)}$ such that $E_i^{(a)} = \sigma_i^{(a)} \oplus H(\sigma_i^{(a)})$ and thus learn $\sigma_i^{(a)}$ via $E_i^{(a)}$ with negligible probability, since it can only make $poly(k)$ queries to the random oracle and $\sigma_i^{(a)}$ is chosen uniformly at random from a $exp(k)$ large space where k is the security parameter. Hence, for all a where $P_a \in \mathcal{Q}$ is an honest party, $\mathcal{A}$ learns only t values $\sigma_i^{(a)}$ and $S_i^{(a)}$, which are not sufficient to recover the degree t polynomials that defines honest parties' $S_i^{(a)}$ values and consequently tpk_a. Since $\mathcal{A}$ learns nothing about tpk_i values of honest parties before round 4, leveraging the zero knowledge property of π_{LDEI}, $\mathcal{S}$ can generate an accepting proof that honest parties have obtained tpk_i from $\hat{S}_i^{(a)}$ instead of the value they should have obtained from $S_i^{(a)}$.

Remark 1 (Refreshing partial keys). The protocol can be modified to one that, given a distributed key ensemble $(\mathsf{tpk}, \{\mathsf{tpk}_i\}, \{\mathsf{tsk}_i\})$ in the form above (not necessarily created by our protocol) outputs fresh random partial secret and public keys $\widetilde{\mathsf{tsk}}_i$, $\widetilde{\mathsf{tpk}}_i$ corresponding to the same global keys tsk, tpk. This is done by having each party P_a share the value $s^{(a)} = 0$ in step 1) of Figure 7. It is easy to modify the LDEI proof to additionally prove in zero knowledge that the PVSS is indeed a sharing to 0 (in Figure 1, the prover just chooses $u(X)$ with the additional condition $u(0) = 0$ and the verifier checks that $z(0) = 0$). Modifying the DKG protocol in this way will output the ensemble $(\mathsf{tpk}', \{\mathsf{tpk}'_i\}, \{\mathsf{tsk}'_i\})$ with $\mathsf{tpk}' = 1_{\mathbb{G}}$. Now parties can define $\widetilde{\mathsf{tpk}}_i = \mathsf{tpk}_i \cdot \mathsf{tpk}'_i$ and each party P_i can privately compute $\widetilde{\mathsf{tsk}}_i = \mathsf{tsk}_i + \mathsf{tsk}'_i$.

Remark 2 (Outputting ℓ' key ensembles). Our DKG protocol would correspond to the case $\ell = \ell' = 1$ in the analogy with ALBATROSS, but of course we can also easily adapt the protocol for $\ell = 1$, $\ell' \geq 1$, where assuming now $t \leq (n-\ell')/2$, we would obtain as output ℓ' independent instances $(\mathsf{tpk}^{(k)}, \{\mathsf{tpk}_i^{(k)}\}, \{\mathsf{tsk}_i^{(k)}\})$, $k \in [\ell']$. The protocol works in the same way until step 4. In step 5 parties P_i compute $\hat{S}_{i,k} = \prod_{a \in \mathcal{Q}} (\hat{S}_i^{(a)})^{M_{k,a}}$, $\sigma_{i,k} = \sum_{a \in \mathcal{Q}} M_{k,a} \sigma_i^{(a)}$ and $S_{i,k} = \prod_{a \in \mathcal{Q}} (S_i^{(a)})^{M_{k,a}}$ for $k = 1, \ldots, \ell'$. Then steps 6, 7, 8 are executed independently for each k, where in step 7 parties verify $\hat{S}_{i,k} = \prod_{a \in \mathcal{Q}} (\hat{S}_i^{(a)})^{M_{k,a}}$. Moreover, the refreshing technique (Remark 1) can be easily extended to deal with refreshing ℓ' ensembles.

4 GULL: Gradual Release of PVSS Outputs via Threshold Encryption

Before presenting GULL, we describe a zero-knowledge proof for the EG relation that we will need, which is similar to discrete logarithm equality, except that one of the elements that would be public in the DLEQ relation now is encrypted by El Gamal (threshold) encryption. The relation is as follows:

$$\mathcal{R}_{EG} = \{((g_1, x_1, x_2, t, c, d), (s, r, g_2)) \in \mathbb{G}^6 \times (\mathbb{Z}_q^2 \times \mathbb{G}) : \\ g_1^s = x_1,\ g_1^r = c,\ d = t^r \cdot g_2,\ g_2^s = x_2\}$$

The problem here is that g_2 is part of the witness, and should not be revealed. Our solution consists on reducing this to proving knowledge of exponents r, s, w with $g_1^r = c$, $g_1^s = x_1$, $d^s \cdot t^w = x_2$, $c^s \cdot g_1^w = 1$, which can be done with a standard Σ-protocol. The point is that if (s, r, g_2) is a witness for $\mathcal{R}_{EG}$, then setting $w = -rs$ will satisfy the equations, while on the other hand, knowledge of (r, s, w) satisfying these equations implies knowledge of (r, s, g_2) satisfying $\mathcal{R}_{EG}$. We present protocol π_{EG} in Figure 9 and formally state and prove its security in Proposition 1

NIZK proof of knowledge π_{EG} for $\mathcal{R}_{EG}$

Setup: Random oracle H. **Common Input:** (g_1, x_1, x_2, t, c, d). **Witness:** (s, r, g_2)

1. The prover chooses $u_r, u_s, u_w \leftarrow \mathbb{Z}_q$, constructs $a_1 = g_1^{u_r}$, $a_2 = g_1^{u_s}$, $a_3 = d^{u_s} \cdot t^{u_w}$, $a_4 = c^{u_s} \cdot g_1^{u_w}$, creates $e = H(g_1, x_1, x_2, t, c, d, a_1, a_2, a_3, a_4)$ and computes $z_r = u_r + e \cdot r$, $z_s = u_s + e \cdot s$, $z_w = u_w - e \cdot r \cdot s$. The proof is (e, z_r, z_s, z_w).
2. The verifier computes $a_1 = g_1^{z_r} \cdot c^{-e}$, $a_2 = g_1^{z_s} \cdot x_1^{-e}$, $a_3 = d^{z_s} \cdot t^{z_w} \cdot x_2^{-e}$, $a_4 = c^{z_s} \cdot g_1^{z_w}$ and accepts iff $e = H(g_1, x_1, x_2, t, c, d, a_1, a_2, a_3, a_4)$.

Fig. 9: NIZK π_{EG} for $\mathcal{R}_{EG}$

Proposition 1. *Protocol π_{EG} is a correct proof of knowledge of (s, r, g_2) such that $((g_1, x_1, x_2, t, c, d), (s, r, g_2)) \in \mathcal{R}_{EG}$, with special soundness (with soundness error $1/q$), and zero knowledge in the random oracle model, assuming the Fiat-Shamir heuristic holds.*

Proof. We prove that the interactive public-coin version of this protocol where e is chosen uniformly at random by the verifier is correct, special-sound and zero knowledge and the Fiat-Shamir heuristic implies the properties above for the non-interactive version.

Correctness: The protocol is easily seen to be correct, as setting $w = -rs$ implies $d^s \cdot t^w = x_2$, $c^s \cdot g_1^w = 1$ if the relation is correct, as argued above, and hence all of the checks will pass.

Special-soundness: Now suppose that a prover can answer two different challenges $e \neq e'$ with z_r, z_s, z_w and respectively z'_r, z'_s, z'_w. This means that the 4 checks by the verifier pass in both cases. From here it is easy to see that $c^{e-e'} = g_1^{z_r - z'_r}$ and $x_1^{e-e'} = g_1^{z_s - z'_s}$ so one can extract $r = (z_r - z'_r)/(e - e')$, $s = (z_s - z'_s)/(e - e')$ and $g_2 = d \cdot t^{-r}$. Note that these values satisfy that $g_1^s = x_1,\ g_1^r = c,\ d = t^r \cdot g_2$, so in order to show that the extracted (s, r, g_2) is a witness, we only need to additionally show that $g_2^s = x_2$ From the fact that the fourth check passes in both cases, we get that $1 = c^{z_s - z'_s} \cdot g_1^{z_w - z'_w}$, which implies $1 = c^{s(e-e')} g_1^{z_w - z'_w}$. Since we already knew $c = g_1^r$ for the extracted r, this means $g_1^{rs(e-e') + z_w - z'_w} = 1$. Since we are in a group of prime order, so g_1 is a generator, it must hold that $rs(e - e') + z_w - z'_w = 0$. Finally from the fact that the third check passes in both instances we have $x_2^{e-e'} = d^{z_s - z'_s} t^{z_w - z'_w}$, which, using the information deduced in the previous line and the expression for the extracted s, means $x_2^{e-e'} = (d^s t^{-rs})^{e-e'}$. Now since $e - e' \neq 0$ and we are in a group of prime order, this means $x_2 = d^s t^{-rs}$. But the right hand side is exactly g_2^s so $x_2 = g_2^s$ as we wanted to show.

Zero knowledge: The simulator samples z_r, z_s, z_w, e independently and uniformly at random in $\mathbb{Z}_q$, and defines a_1, a_2, a_3, a_4 as the verifier would do in the proof verification. This generates a transcript which is indistinguishable from one of an actual protocol, as it is easy to see.

We now construct GULL, a random beacon that allows for generating $O(n^2)$ outputs that can be opened in individual batches of $O(n)$ outputs. We present GULL in Figure 10 and formally state its security in Theorem 2.

Theorem 2. *Assuming DDH holds in $\mathbb{G}$, for any static adversary $\mathcal{A}$ corrupting t parties, with probability at least $1 - t/q$ the following holds (in the random oracle model) for protocol GULL in Figure 10:*

- *All honest parties obtain the same output $(o_{k,j})_{k \in [\ell'], j \in [0, \ell-1]} \in \mathbb{G}^{\ell \cdot \ell'}$.*
- *(Unbiasability) Regardless of the actions of $\mathcal{A}$, the distribution of the output is computationally indistinguishable from uniform in $\mathbb{G}^{\ell \cdot \ell'}$.*
- *(Unpredictability after opening k' batches [6]). For every $k' \in \ell'$, consider the following experiment played after the values $o_{k,j}, k \in [k'], j \in [0, \ell - 1]$ have been opened in step 4. A challenger chooses $b \in \{0, 1\}$ at random; if b=0, it sets $w_{k,j} = o_{k,j}$ (the true unopened outputs) for $k \in [k'+1, \ell']$, $j \in [0, \ell-1]$; if $b = 1$, it chooses all these $w_{k,j}$ independently and uniformly at random in $\mathbb{G}$. The challenger sends all $w_{j,k}$ to $\mathcal{A}$ who makes a guess b'. Then the probability that $b' = b$ is at most $1/2 + \mathsf{negl}(\lambda)$ where λ is the security parameter.*

Proof. We first note that if a corrupted party cheats in one of the zero knowledge proofs, that party will be caught with probability at least $1 - 1/q$. Therefore

[6] We remark that the definition of unpredictability is based on the definition of IND1-security of a PVSS, where the same unpredictability guarantees are required for an adversary that sees the sharing of a given secret (see e.g. [23, 9]).

GULL: PVSS beacon with gradual release.

Setup: A public bulletin board, a DDH-hard group $\mathbb{G} = \langle g \rangle$ of prime order q, a t-resilient matrix $M \in \mathbb{Z}_q^{\ell' \times (n-t)}$, key-pairs $(\mathsf{sk}_i, \mathsf{pk}_i = g^{\mathsf{sk}_i}) \in \mathbb{Z}_q \times \mathbb{G}$ for each P_i.
Setup from DKG: Parties have a global threshold public key tpk, partial threshold keys tpk_i and partial threshold secret keys tsk_i, where $\mathsf{tpk}_i = g^{\mathsf{tsk}_i}$ and $\mathsf{tsk}_i = h(i)$ for some secret random $h \in \mathbb{Z}_q[X]_{\leq t+1}$, and $\mathsf{tpk} = g^{h(0)}$.
Preparation:

1. (ALBATROSS Sharing) Each P_a PVSSs a secret $(g^{s_0^{(a)}}, \ldots, g^{s_{\ell-1}^{(a)}}) \in_R \mathbb{G}^\ell$ by choosing $f^{(a)} \in_R \mathbb{Z}_q[X]_{\leq t+\ell-1}$ setting $s_j^{(a)} := f^{(a)}(-j)$, $\sigma_i^{(a)} := f^{(a)}(i)$ and publishing $\hat{S}_i^{(a)} = \mathsf{pk}_i^{\sigma_i^{(a)}}$ for $i \in [n]$, together with a proof of correct sharing $\pi^{(a)} = \pi_{LDEI}((\mathsf{pk}_i)_{i=1}^n, (\hat{S}_i^{(a)})_{i=1}^n, t+\ell-1)$.
2. (a) (Verification) Parties check the validity of $\pi^{(a)}$ for all $a \in [n]$. This fixes a set $\mathcal{Q}$ of the first $n-t$ parties $P_a, a \in \mathcal{Q}$ whose $\pi^{(a)}$ are correct.
   (b) (Aggregation) Every party can compute, for every $i \in [n]$ and $k \in [1, \ell']$,
$$R_{ik} = \prod_{a \in \mathcal{Q}} (\hat{S}_i^{(a)})^{M_{k,a}}$$
   Additionally each P_i computes $S_{ik} = R_{ik}^{\mathsf{sk}_i^{-1}}$ for every $k \in [1, \ell']$. Note $S_{ik} = g^{f_k(i)}$ where $f_k(X) = \sum_{a \in \mathcal{Q}} M_{k,a} \cdot f^{(a)}(X)$.
   (c) (Encryption) For every $k \in [\ell']$, P_i posts $E_{ik} = \mathsf{Enc}(\mathsf{tpk}, S_{ik}) = (g^{r_{ik}}, \mathsf{tpk}^{r_{ik}} \cdot S_{ik}) := (c_{ik}, d_{ik})$ and a NIZK proof $\pi_{EG,i,k}$ (detailed in Figure 9) for the fact that $((g, \mathsf{pk}_i, R_{ik}, \mathsf{tpk}, c_{ik}, d_{ik}), (\mathsf{sk}_i, r_{ik}, S_{ik}))$ is in $\mathcal{R}_{EG}$, i.e. $g^{\mathsf{sk}_i} = \mathsf{pk}_i,\ g^{r_{ik}} = c_{ik},\ d_{ik} = \mathsf{tpk}^{r_{ik}} \cdot S_{ik},\ S_{ik}^{\mathsf{sk}_i} = R_{ik}$.

Opening: Let I be the set of the first $t+\ell$ parties P_i who have posted correct proofs $\pi_{EG,i,k}$ for every $k \in [\ell']$. At any point after round 2 is finished, the k'-th batch (for an arbitrary $k' \in [\ell']$) of ℓ outputs can be opened as follows:

3) (Lagrange computation) For every $j \in [\ell]$, parties compute: $O_{k',j} = (\prod_{i \in I} (c_{ik'})^{L_{i,I}(-j)}, \prod_{i \in I} (d_{ik'})^{L_{i,I}(-j)})$
4) (Decryption) Parties threshold-decrypt $O_{k'j}$ for every $j \in [0, \ell-1]$ to obtain output $(o_{k'0}, \ldots, o_{k'(\ell-1)})$. More concretely, call $O_{k',j} = (C_{k',j}, D_{k',j})$. Then:
   – Each party P_i posts the values $u_{k',j,i} = C_{k',j}^{\mathsf{tsk}_i}$, for all $j \in [0, \ell-1]$ and a proof $\pi_{k',j,i} = \pi_{DLEQ}((g, C_{k',0}, \ldots, C_{k',\ell-1}), (\mathsf{tpk}_i, u_{k',0,i}, \ldots, u_{k',\ell-1,i}))$.
   – Let J be a set of the first $t+1$ parties that have posted correct $\pi_{k',i}$. Then the outputs $o_{k',j}$, $j \in [0, \ell-1]$ are reconstructed by every party as
$$o_{k',j} = D_{k',j} \cdot \prod_{i \in J} u_{k',j,i}^{-L_{i,J}(0)}$$

Fig. 10: GULL: PVSS beacon with gradual release. Setup and Preparation

the probability that a corrupted party deviates from the protocol and yet is included in $\mathcal{Q}, I$ or J is at most t/q. Hence, except with this probability, all parties included in $\mathcal{Q}, I, J$ have behaved honestly in the respective steps (sharing, threshold encryption, threshold decryption). We assume this from now on. Since there are at least $t+\ell$ honest parties, $|I| \geq t+\ell$ and $|J| \geq t+1$, so the protocol always finalizes. In fact, the outputs are already determined at the end of step 1: they are $o_{k,j} = g^{f_k(-j)}$, where $f_k = \sum_{a \in \mathcal{Q}} M_{k,a} f^{(a)}$ where $\mathcal{Q}$ is fixed at the end of step 1. Since up to this point the protocol is exactly as in Albatross the unbiasability under DDH follows from the results there. In a nutshell that argument relies on the fact that the honest inputs are indistinguishable from uniform for the adversary at that point (we will in fact prove a stronger claim when we show unpredictability in our case), together with the t-resiliency of the matrix guaranteeing that the result will be uniform in that case.

We now argue unpredictability. For the sake of notation, assume that the adversary $\mathcal{A}$ corrupts $P_{n-t+1}, \ldots, P_{n-t}$. Suppose that after the first k' batches $(o_{k,j})_{k \in [k'], j \in [0,\ell-1]}$ have been opened, $\mathcal{A}$ is given a vector v_b for uniformly random $b \in \{0,1\}$ where $v_0 = (o_{k,j})_{k \in [k'+1,\ell'], j \in [0,\ell-1]}$ and v_1 is uniformly random in $\mathbb{G}^{\ell \cdot (\ell' - k')}$. We will prove that if $\mathcal{A}$ can guess b with probability non-negligibly larger than $1/2$, then we can construct $\mathcal{D}$ that solves the $((n-t)\cdot\ell)$-DDH problem: namely $(h, h^{\alpha}, h^{\beta_{11}}, \cdots, h^{\beta_{(n-t)\ell}}, h^{\gamma_{11}}, \cdots, h^{\gamma_{(n-t)\ell}})$ is sampled as a challenge, where h is uniformly random in the group $\mathbb{G}$ and α, and all β_{ij} are chosen independently and uniformly at random in $\mathbb{Z}_q$ while the value of γ_{ij} is dictated by a bit b' chosen uniformly at random: if $b' = 0$, then $\gamma_{ij} = \alpha\beta_{ij}$ for all i, j; while if $b' = 1$, γ_{ij} are uniformly random in $\mathbb{Z}_q$ if $b' = 1$. This challenge is sent to $\mathcal{D}$, who has to guess b'. The $((n-t)\cdot\ell)$-DDH problem is equivalent to DDH as long as the number $(n-t)\cdot\ell$ is polynomial in the security parameter.

We construct a distinguisher $\mathcal{D}$ which runs an internal copy of $\mathcal{A}$. $\mathcal{D}$ will simulate an execution of GULL where $g = h^{\alpha}$ and the vector of secrets chosen by each honest party P_a is $g^{\beta_{aj}}$ (we will show how this can be done). These equal $h^{\gamma_{aj}}$ if $\gamma_{aj} = \alpha\beta_{aj}$. At the end of the simulation, $\mathcal{D}$ sets as a challenge for $\mathcal{A}$ the outputs that one would obtain if the honest parties had shared the values $h^{\gamma_{aj}}$. $\mathcal{A}$ now makes a guess of whether this are correct outputs or random values. In the former case $\mathcal{D}$ guesses that $\gamma_{aj} = \alpha\beta_{aj}$ and in the latter that γ_{aj} are random.

The simulation is as follows: $\mathcal{D}$ sets $g = h^{\alpha}$, samples u_i in $\mathbb{Z}_q$ for $i \in [n-t]$ and sets $\mathsf{pk}_i = h^{u_i}$, implicitely defining $\mathsf{sk}_i = u_i/\alpha$ (which $\mathcal{D}$ does not know). It waits for $\mathcal{A}$ to choose pk_i and sk_i for malicious parties. Moreover, the distributed key generation algorithm is run to establish the threshold keys $\mathsf{tsk}_i, \mathsf{tpk}, \mathsf{tpk}_i$.

$\mathcal{D}$ chooses τ_{ai}, at random in $\mathbb{Z}_q$ for $a \in [n-t]$, $i \in [n-t+1, n]$. Let $f^{(a)}$ be the polynomial in $\mathbb{Z}_q[X]_{\leq t+\ell}$ with $f^{(a)}(i) = \tau_{ai}$ for $i \in [n-t+1, n]$ and $f^{(a)}(j) = \beta_{aj}$ for $j \in [0, \ell-1]$. $\mathcal{D}$ does not know $f^{(a)}$ but it can compute $h^{f^{(a)}(i)}$, for $i \in [n-t]$, from the values $h^{\beta_{aj}}$ for $j \in [0, \ell-1]$ and $h^{\tau_{ai}}$ for $i \in [n-t+1, n]$ by Lagrange interpolation. Note that $h^{\beta_{aj}}$ are part of the challenge. $\mathcal{D}$ sets $\hat{S}_i = (h^{f^{(a)}(i)})^{u_i}$ for $i \in [n-t]$ and $\hat{S}_i = \mathsf{pk}_i^{\tau_{ai}}$ for $i \in [n-t+1, n]$. Now $\hat{S}_i = \mathsf{pk}_i^{f^{(a)}(i)}$ for all i, hence it is a sharing for the vector $(g^{f^{(a)}(-j)})_{j \in [0,\ell-1]} = (g^{\beta_{aj}})_{j \in [0,\ell-1]}$. $\mathcal{D}$ simulates the proof $\pi^{(a)}$ using the zero knowledge simulator. It waits for the adversary to send the corresponding information from Round 1, which determines $\mathcal{Q}$. Let $\mathcal{Q}_0 = \mathcal{Q} \cap [1, n-t]$, $\mathcal{Q}_1 = \mathcal{Q} \cap [n-t+1, n]$.

Now $\mathcal{D}$ cannot reconstruct the "real" S_{ik}, as it does not know the secret keys of the honest participants. Instead it simply samples these at random (we call them S'_{ik} so that we can make the distinction later). It then simulates the proof $\pi_{EG,i,k}$ using the simulator for the zero knowledge protocol. The rest of the protocol is carried out from these values as in Figure 10 with $\mathcal{D}$ playing for the honest parties and $\mathcal{A}$ for the corrupted ones. After opening k' batches, $\mathcal{A}$ is given as a challenge the values x_{kj} defined by $x_j = \prod_{a\in\mathcal{Q}_0}(h^{\gamma_{aj}})^{M_{k,a}} \cdot \prod_{a\in\mathcal{Q}_1}(g^{f^{(a)}(-j)})^{M_{k,a}}$, where $f^{(a)}$ was extracted from $\pi^{(a)}$. These would be the outputs obtained in the real protocol if $\gamma_{aj} = \alpha\beta_{aj}$, since the secrets of honest parties P_a are implicitely defined as $g^{\beta_{aj}} = h^{\alpha\beta_{aj}}$. While, if γ_{aj} are uniformly random, x_{kj} are also uniformly random due to the properties of the t-resilient matrix.

$\mathcal{A}$ makes a guess about whether the vector of x_{kj} are the secrets or random values, and $\mathcal{D}$ outputs the same guess about whether $\gamma_j = \alpha\beta_j$. Note x_{kj} are the actual secrets if and only if $\gamma_{aj} = \alpha\beta_{aj}$. So if $\mathcal{A}$'s view $View_{\mathcal{A},k'}$ at this point is as in a real protocol where honest parties have input $g^{\beta_{aj}}$, we would be done.

We argue this by induction. Consider first $k' = 0$, i.e. no outputs have been opened yet. Then apart from $\pi^{(a)}$, $\pi_{EG,i}$, which have been simulated with the zero knowledge simulator (so they are indistinguishable from the real view), the other point where the simulation differs from the real protocol is in the fact that S'_{ik} are chosen at random. But by IND-CPA property of El Gamal (in turn based on DDH) their encryptions are indistinguishable from encryptions of the "real" S_{ik}. For $k' > 0$, the situation is more delicate because now also the opened outputs in the simulated run depend on the simulated S'_{ik}. By induction, the view $View_{\mathcal{A},k'-1}$ of the adversary before the opening of the k'-th output batch is indistinguishable from that in the real protocol. We have argued that, under those conditions and the DDH assumption, our unpredictability claim is true and hence the distribution of the remaining outputs, in particular of the next batch $(o_{k',j})_{j\in[0,\ell-1]}$, conditioned to that view is indistinguishable from uniform in both the real and simulated run. We conclude that the views $View_{\mathcal{A},k'}$ in the real and simulated protocol are also indistinguishable which finalizes the proof.

5 Constructing Mt. Random

In Figure 11, we present Mt. Random, our multi-tiered beacon composed by the building blocks presented so far. As discussed earlier, we have three tiers: Tier 1 - Uniform Randomness, Tier 2 - Pseudorandomness and Tier 3 - Bounded Biased Randomness. Starting from Tier 1, going up each tier represents a trade-off between efficiency and randomness quality, where more efficiency is gained at the cost of quality. In other words, higher tiers generate random outputs faster than lower tiers albeit with losses in randomness quality, *i.e.* going from uniformly random values to values with a bounded adversarial bias. Moreover, each higher tier uses outputs from the previous tier as seeds, ensuring that all tiers operate within a desired level of bias while maintaining efficiency. We use the DDH assumption (in the random oracle model) to prove security of all of Mt.

Mt. Random: Multi-tiered Randomness Beacon

Parameters:
- n participants P_i, $i \in [n]$, corruption threshold $1 \leq t \leq (n-\ell)/2$.
- Integers $\ell \geq 1$ (number of secrets in GULL output block) and $\ell' = n-2t$ (number of blocks outputted by one round of GULL).
- Integers ℓ_{TVRF} and ℓ_{VRF} denoting the bitlength of outputs from Tiers 2 and 3.
- Integer $\mathsf{TVRF}_{max} \geq 0$ and $\mathsf{VRF}_{max} \geq 0$ denoting, respectively, the number of times the TVRF-based beacon at Tier 2 and the VRF-based beacon at Tier 3 are applied iteratively starting from a given seed.

Setup: An authenticated public bulletin board (BB), field $\mathbb{Z}_q$, and DDH-hard group $\mathbb{G}$ with generator g. A key pair $\mathsf{sk}_i \in \mathbb{Z}_q$, $pk_i = g^{\mathsf{sk}_i}$ registered in BB for each P_i. A t-resilient matrix $M \in \mathbb{Z}_q^{\ell' \times (n-t)}$. Random oracle $H : \{0,1\}^\star \to \{0,1\}^k$.

Initialization: All parties P_i keep an initially empty table Batch that stores unopened GULL sub-batches encrypted under threshold-El Gamal. All parties first execute the **Distributed Key Generation** phase and then execute **Tier 1**, **Tier 2** and **Tier 3** as soon as seed randomness from the previous tier is available. Tiers are re-executed as more outputs are needed.

Distributed Key Generation: All parties execute the DKG protocol in Figure 7 twice to obtain keys for Tiers 1 and 2. The public outputs are global threshold public keys $\mathsf{tpk}, \mathsf{tpk}'$ and partial threshold public keys $\mathsf{tpk}_i, \mathsf{tpk}'_i$ for $i \in [n]$, while each party $P_i, i \in [n]$ obtains partial threshold secret keys tsk_i and tsk'_i.

Tier 1: Using keys tpk and tsk_i obtained in the DKG phase, all parties execute the Preparation phase of GULL in Figure 10. At this point all parties obtain ℓ' blocks $B_k = (O_{k1}, O_{k2}, \ldots, O_{k\ell})$, $k \in [\ell']$ consisting of threshold El-Gamal encryptions of o_{kj} under tpk, which are stored in Batch. When an output is requested:

1. If Batch is not empty, parties execute the Opening phase of GULL to decrypt the next $B_k \in \mathsf{Batch}$, obtaining $o_{k1}, o_{k2}, \ldots, o_{k\ell}$. Output $H(o_{k1}), H(o_{k2}), \ldots, H(o_{k\ell})$ and remove B_k from Batch.
2. If Batch is empty, return $\perp$ and run the Preparation phase of GULL to refill it

Tier 2: Initially, parties request an output o_{kj} from Tier 1 (repeating the request until $o_{kj} \neq \perp$) and set $\sigma_0 = o_{kj}, r = 1$. When an output is requested, all parties execute the DRAND/Dfinity beacon described in section 2.6 using $\mathsf{tpk}', \mathsf{tpk}'_i, \mathsf{tsk}'_i$ with seed $\sigma_0 = o_{kj}$ to obtain output z_r, outputting $H(z_r)$. When $r = \mathsf{TVRF}_{max}$, get a new output o_{kj} from Tier 1 and set $\sigma_0 = o_{kj}, r = 1$.

Tier 3: All parties request an output z_r from Tier 2 (repeating the request until $z_r \neq \perp$) and run the VRF-based beacon in Figure 3 using z_r as initial seed. In each round $r' \in \{1, \ldots, \mathsf{VRF}_{max}\}$, output $H(w'_r)$ where $w'_r \in \{0,1\}^{\ell_{VRF}}$ is the output of the beacon. When $r' = \mathsf{VRF}_{max}$, r is reset to 0 and Tier 3 is started again.

Fig. 11: Mt. Random: Multi-tiered Randomness Beacon.

Random's building blocks, *i.e.* PVSS, DKG, TVRF and VRF, thus obtaining a final construction whose security is based on a single standard assumption while achieving competitive concrete efficiency. However, we remark that other constructions of these building blocks can be used within our framework in order to achieve better efficiency at the cost of having security underpinned by multiple and possibly less standard assumptions. Moreover, each Tier could be constructed from other primitives that yield random outputs with similar guarantees, *e.g.* Tier 2 could be instantiated using VDF based beacons [5]. We will now discuss the building blocks used for tier and provide a security analysis.

Tier 1: Uniform Randomness via PVSS: The first tier of Mt. Random outputs true uniform randomness. It is important to output uniformly random values at this tier because these values will be used as seeds for Tier 2. We instantiate Tier 1 with GULL using threshold encryption keys generated by our DKG protocol (Figure 7). Tier 1 has the highest execution time and communication, outputting uniformly random values less frequently than higher tiers. On the other hand, instead of outputting a single value, Tier 1 will output a *batch* of uniformly random values that can be used to seed Tier 2 multiple times (instead of requiring a full execution of Tier 1 every time Tier 2 needs a new seed).

In the original ALBATROSS [9] protocol, the full batch of outputs is revealed as soon as the protocol terminates. This is not an issue when seeding Tier 2, since Tier 2 outputs cannot be predicted without a threshold key. However, it might be a problem in the case where fresh uniformly random outputs from Tier 1 are required for applications other than seeding Tier 2. Hence, we instantiate Tier 1 with GULL, which allows for gradually revealing smaller "sub-batches" of outputs. Under this regime, whenever a fresh uniformly random output is required for other applications, a fresh sub-batch can be revealed, which is significantly more efficient than re-executing the full ALBATROSS protocol.

Tier 2: Pseudorandomness via Threshold VRFs: Tier 2 outputs pseudorandom values instead of truly uniformly random values. While these values are not suitable for some applications (*e.g.* seeding PRGs), they are sufficient for a number of popular applications (*e.g.* selecting random committees). We instantiate Tier 2 with a DDH based version of the DRAND/Dfinity TVRF proposed in [19] coupled with our new DKG protocol (Figure 7).We choose to use a DDH-based TVRF in order to instantiate all of our building blocks from a single standard assumption. However, a more efficient TVRF (*e.g.* GLOW [19]) can be used for better performance at the cost of a stronger assumption.

There are two main hurdles in using TVRF-based beacons: 1. keys must be generated in a distributed manner; 2. being essentially a distributed PRG, the beacon must be re-seeded periodically. Mt. Random respectively solves these issues by employing our new DDH-based DKG (Figure 7) and by periodically re-seeding Tier 2 with uniformly random outputs from Tier 1. Using our DKG, we maintain public verifiability of threshold key validity and consequently of Tier 2's output without requiring extra assumptions. Moreover our DKG protocol can be used to refresh secret key shares if parties are compromised (see Remark 1).

Tier 3: Bounded Biased Randomness via VRFs: The third tier of Mt. Random outputs pseudorandom values that may be biased by the adversary up to a certain upper bound. While this sort of biased randomness finds less applications than unbiased pseudorandomness or uniform randomness, it is still sufficient for important applications such as selecting block creators in Proof-of-

Stake based blockchains (*e.g.* Ouroboros Praos [15]). We instantiate Tier 3 with the VRF and VRF-based beacon protocols from Ouroboros Praos, which are secure under the CDH assumption (implied by DDH). However, differently from the original Ouroboros Praos beacon, where each execution is seeded with the output of the previous one, we seed this protocol with an output from Tier 2. This crucial difference reduces the potential adversarial bias in Tier 3 outputs.

5.1 Security Analysis

Theorem 3. *Under the DDH assumption in $\mathbb{G}$, for any static adversary corrupting $t < n/2$ parties, with probability at least $1 - t/q$, the following holds (in the random oracle model) for Mt. Random in Figure 11:*

- *All Tiers have guaranteed output delivery, all honest parties obtain the same outputs $H(o_{k1}), H(o_{k2}), \ldots, H(o_{k\ell})$, $H(z_r)$ and $H(w'_r)$ from each output request from Tiers 1, 2 and 3, respectively.*
- *(Unbiasability for Tiers 1 and 2) Regardless of the actions of the adversary, the distribution of $H(o_{k1}), H(o_{k2}), \ldots,\ H(o_{k\ell})$ (resp. $H(z_r)$) from Tier 1 (resp. Tier 2) is computationally indistinguishable from uniform.*
- *(Unpredictability for Tiers 1 and 2). Given all previous outputs from Tiers 1 and 2, the adversary cannot predict future outputs from Tiers 1 and 2.*
- *(Bounded Bias and Predictability for Tier 3) The adversary can predict and bias at most t bits of the output of Tier 3.* [7]

Proof. (Sketch) Notice that in the initialization phase we execute our DKG protocol (Figure 7) before initiating the execution of the tiers. Due to the security of the DKG protocol (Theorem 1), the resulting global and partial public keys $\mathsf{tpk}, \mathsf{tpk}_i$ and $\mathsf{tpk}, \mathsf{tpk}'_i$ for $i \in [n]$ are guaranteed to be unbiased and each party $\mathcal{P}_i$ is guaranteed to have obtained its secret share $\mathsf{tsk}_i, \mathsf{tsk}'_i$ as well as the same public keys. Moreover, using the simulator constructed in the proof of Theorem 1, we can extract adversarial secret keys.

In Tier 1, we execute GULL from Figure 10 using keys $\mathsf{tpk}, \mathsf{tpk}_i, \mathsf{tsk}_i$, which gives us two main guarantees as shown in Theorem 2: 1. The Preparation phase results in ℓ' output blocks that are guaranteed to be recoverable by a majority of the parties but remain secret until the Opening phase is executed; 2. All ℓ values of each output block are guaranteed to be uniformly random. Hence, when Tier 1 is initiated, ℓ' output blocks with ℓ uniformly random values become available. When an output is requested, executing the procedures of Tier 1 clearly returns either an uniformly random output (or $\perp$, in which case more output blocks are obtained executing step 2 of Tier 1's output request procedure).

In Tier 2, we execute the TVRF-based beacon protocol from Figure 5, which is proven to output pseudorandom values in [19]. Since we periodically re-seed this protocol with uniformly random values from Tier 1, its outputs are guaranteed to be pseudorandom. Notice that we can re-seed Tier 2 with outputs from

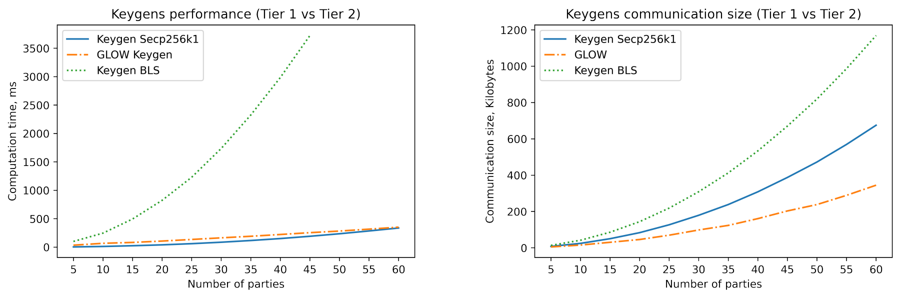
[7] A formal definition and proof is shown in [15], where Tier 3's protocol was originally described

Tier 1 that are already revealed but still not used as a Tier 2 seed. By the security of the TVRF scheme (proven in [19]), an adversary controlling a minority of the parties cannot predict the output of the TVRF on any given input. Hence, the outputs of Tier 2 cannot be predicted by the adversary (who only corrupts a minority of the parties) upon learning the seed. Notice that the TVRF security properties hold since we use unbiased threshold keys $\mathsf{tpk}', \mathsf{tpk}'_i, \mathsf{tsk}'_i$.

In Tier 3, we execute the protocol in Figure 3, proven to output bounded biased values in [15] even when it is seeded with outputs of a previous execution of itself. Hence, seeding this protocol with the unbiased pseudorandom outputs from Tier 2, not only preserves but improves on the proven bias bounds for its outputs. Using outputs from Tier 2 that are already known but still not used as a seed in Tier 3 preserves the security, since even by knowing the seed in advance the adversary can introduce a bounded bias to the output as proven in [15].

6 Efficiency Analysis

Distributed Key Generation Our novel DKG protocol's performance is further showcased in Figures 12a and 12b, which show the DKG computation time and communication size for changing number of parties n for Tiers 1 and 2.



(a) (b)

Fig. 12: a) DKG computation time for Tiers 1 and 2 for changing number of parties n with fixed $t = \lfloor \frac{n}{3} \rfloor$. b) DKG communication size for Tiers 1 and 2 for changing number of parties n with fixed $t = \lfloor \frac{n}{3} \rfloor$

Mt. Random We provide a reference implementation for each one of the tiers here: `https://github.com/ZenGo-X/random-beacon`. Our main goal is to demonstrate the trade-off in efficiency between the three tiers. We also highlight the sensitivity to changing number of parties n, the threshold t and culprits c when relevant. All our measurements were done on a t3.medium AWS instance (2 vCPU of Intel(R) Xeon(R) Platinum 8259CL CPU @ 2.50GHz, 4GB RAM).

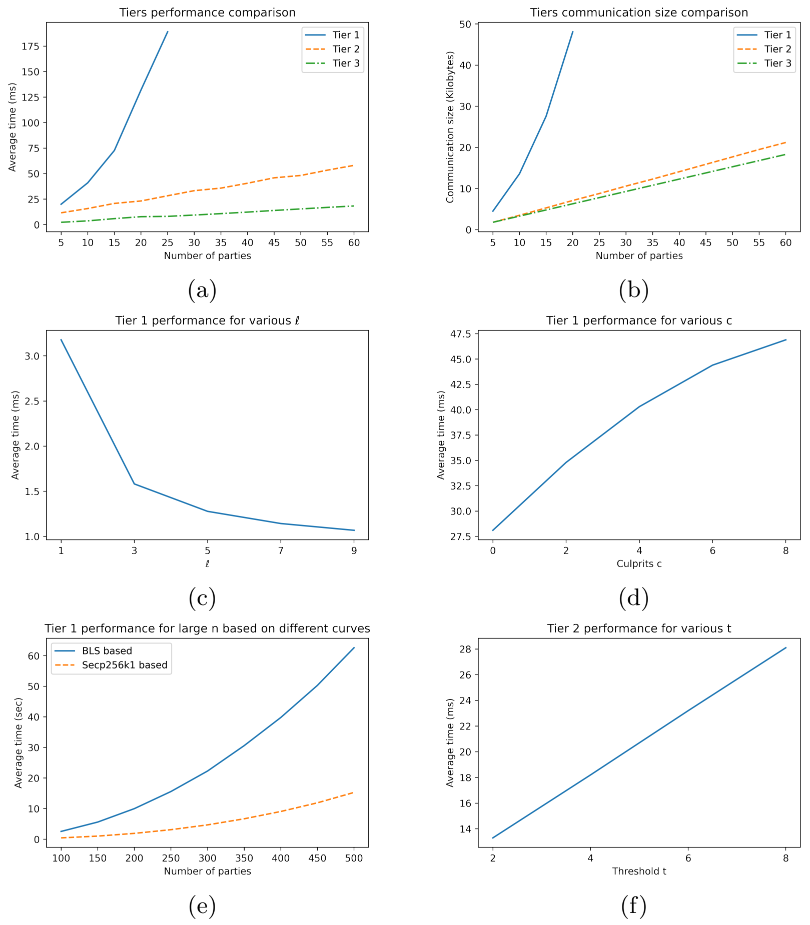


Fig. 13: Execution time (a) and communication (b) of each Tier with fixed $t = \lfloor \frac{n}{3} \rfloor$, $\ell = 1$. (c) Amortized cost of generating a single element at Tier 1 with fixed $n = 25$, $t = 8$, $\ell = 9$. Average execution time of Tier 1 for a number c of corruptions with fixed $n = 25$, $t = 8$ (d) and for large n with fixed $t = \lfloor \frac{n}{3} \rfloor$, $\ell = 1$ (e). (f) Average execution time of Tier 2 for threshold t with $n = 25$.

Our experiments do not include network latency or delay, as network latency is larger than our computation times and would mask them. Since the number of rounds of Tier 1 is larger than that in Tier 2 and Tier 3, and communication size of Tier 2 is larger than communication size of Tier 3, if we include latency, we trivially get our expected hierarchy. Network delay is of no interest: the communication bandwidth is small enough for the network to not be a bottleneck. Measurements were done using a benchmark tool and averaged over many runs.

Computation time and communication size: In Figure 13a we compare the computation time for a single run of each tier as a function of the number of parties n. Tier 1 is the slowest, Tier 3 is the fastest and Tier 2 is in the middle, which is coherent with how we suggest to hierarchically use the different tiers in the paper. Figure 13b shows the communication size of the three tiers, for various n. Here again we see a clear hierarchy where Tier 1 requires the most communication, Tier 3 the last and Tier 2 is in the middle. For completeness, we provide in Section 6 the same measurements, but for running distributed key generation for tiers 1 and 2. Key generation and setup is not our focus as we consider it a one-time operation running at the beginning of the execution. On the other hand, producing random values is done periodically throughout the life time of the system.

Tier 1 and Tier 2 sensitivity: We measured Tier 1 without gradual release (ALBATROSS), i.e., all random values are released at once. In Figure 13c we show how changing ℓ , a parameter proportional to the number of random elements output by Tier 1, impacts the amortized cost of a single random element. As expected, the more random elements we pack in a single run the more efficient the amortized computation per a single random element is. This result hints to the effectiveness of running GULL in settings were fresh unpredictable output is needed by an application other than Tier 2.

In Figure 13d, we fix n and change the number of culprits c, which impacts the total running time in a meaningful way, since less computation is done in an optimistic case with less misbehaving parties. Figure 13e shows Tier 1 performance for large n. The choice of the curve dramatically affects efficiency. In this case, the Secp256k1 curve implementation outperforms BLS12-381 (we used libraries `https://github.com/rust-bitcoin/rust-secp256k1` for Secp256k1 and `https://github.com/algorand/pairing-plus` for BLS12-381). All other Tier 1 benchmarks are based on BLS12-381 curve in order to make results comparable to Tier 2. Finally, Figure 13f, shows computation time of Tier 2 for fixed n and various threshold t. As expected, the computation time is linear in n.

References

1. D. F. Aranha, C. P. L. Gouvêa, T. Markmann, R. S. Wahby, and K. Liao. RELIC is an Efficient LIbrary for Cryptography. `https://github.com/relic-toolkit/relic`.
2. Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. Craft: Composable randomness beacons and output-independent abort mpc from time. Cryptology ePrint Archive, Report 2020/784, 2020. `https://eprint.iacr.org/2020/784`.
3. Carsten Baum, Bernardo David, Rafael Dowsley, Jesper Buus Nielsen, and Sabine Oechsner. TARDIS: A foundation of time-lock puzzles in UC. LNCS, pages 429–459. Springer, Heidelberg, 2021.
4. Adithya Bhat, Nibesh Shrestha, Zhongtang Luo, Aniket Kate, and Kartik Nayak. RandPiper - reconfiguration-friendly random beacons with quadratic communication. pages 3502–3524. ACM Press, 2021.
5. Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In Hovav Shacham and Alexandra Boldyreva, editors, *CRYPTO 2018, Part I*, volume 10991 of *LNCS*, pages 757–788. Springer, Heidelberg, August 2018.

6. Dan Boneh and Moni Naor. Timed commitments. In Mihir Bellare, editor, *CRYPTO 2000*, volume 1880 of *LNCS*, pages 236–254. Springer, Heidelberg, August 2000.
7. Ran Canetti. Security and composition of multiparty cryptographic protocols. *Journal of Cryptology*, 13(1):143–202, January 2000.
8. Ignacio Cascudo and Bernardo David. SCRAPE: Scalable randomness attested by public entities. In Dieter Gollmann, Atsuko Miyaji, and Hiroaki Kikuchi, editors, *ACNS 17*, volume 10355 of *LNCS*, pages 537–556. Springer, Heidelberg, July 2017.
9. Ignacio Cascudo and Bernardo David. ALBATROSS: Publicly AttestabLe BATched Randomness based On Secret Sharing. In Shiho Moriai and Huaxiong Wang, editors, *ASIACRYPT 2020, Part III*, volume 12493 of *LNCS*, pages 311–341. Springer, Heidelberg, December 2020.
10. David Chaum and Torben P. Pedersen. Wallet databases with observers. In Ernest F. Brickell, editor, *CRYPTO'92*, volume 740 of *LNCS*, pages 89–105. Springer, Heidelberg, August 1993.
11. Jing Chen and Silvio Micali. Algorand: A secure and efficient distributed ledger. *Theor. Comput. Sci.*, 777:155–183, 2019.
12. Alisa Cherniaeva, Ilia Shirobokov, and Omer Shlomovits. Homomorphic encryption random beacon. Cryptology ePrint Archive, Report 2019/1320, 2019. `https://eprint.iacr.org/2019/1320`.
13. Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. Distributed elgamal à la pedersen: Application to helios. In *Proceedings of the 12th annual ACM Workshop on Privacy in the Electronic Society, WPES 2013*, pages 131–142. ACM, 2013.
14. Phil Daian, Rafael Pass, and Elaine Shi. Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake. In Ian Goldberg and Tyler Moore, editors, *FC 2019*, volume 11598 of *LNCS*, pages 23–41. Springer, Heidelberg, February 2019.
15. Bernardo David, Peter Gazi, Aggelos Kiayias, and Alexander Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In Jesper Buus Nielsen and Vincent Rijmen, editors, *EUROCRYPT 2018, Part II*, volume 10821 of *LNCS*, pages 66–98. Springer, Heidelberg, April / May 2018.
16. Luca De Feo, Simon Masson, Christophe Petit, and Antonio Sanso. Verifiable delay functions from supersingular isogenies and pairings. In Steven D. Galbraith and Shiho Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 248–277. Springer, Heidelberg, December 2019.
17. Pierre-Alain Fouque and Jacques Stern. One round threshold discrete-log key generation without private channels. In Kwangjo Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 300–316. Springer, Heidelberg, February 2001.
18. Cody Freitag, Ilan Komargodski, Rafael Pass, and Naomi Sirkin. Non-malleable time-lock puzzles and applications. LNCS, pages 447–479. Springer, Heidelberg, 2021.
19. David Galindo, Jia Liu, Mihai Ordean, and Jin-Mann Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. In *IEEE European Symposium on Security and Privacy, EuroS&P 2021, Vienna, Austria, September 6-10, 2021*, pages 88–102. IEEE, 2021.
20. Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. In Jacques Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 295–310. Springer, Heidelberg, May 1999.

21. Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. LNCS, pages 147–176. Springer, Heidelberg, 2021.
22. Timo Hanke, Mahnush Movahedi, and Dominic Williams. Dfinity technology overview series, consensus system, 2018.
23. Somayeh Heidarvand and Jorge L. Villar. Public verifiability from pairings in secret sharing schemes. In *Selected Areas in Cryptography, 15th International Workshop, SAC 2008*, volume 5381 of *Lecture Notes in Computer Science*, pages 294–308. Springer, 2008.
24. Jonathan Katz, Julian Loss, and Jiayu Xu. On the security of time-lock puzzles and timed commitments. In Rafael Pass and Krzysztof Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 390–413. Springer, Heidelberg, November 2020.
25. Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, August 2017.
26. Silvio Micali, Michael O. Rabin, and Salil P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, October 1999.
27. Torben P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In Donald W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 522–526. Springer, Heidelberg, April 1991.
28. Krzysztof Pietrzak. Simple verifiable delay functions. In Avrim Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, January 2019.
29. randao.org. RANDAO: Verifiable random number generation, 2017. `https://www.randao.org/whitepaper/Randao_v0.85_en.pdf` accessed on 20/02/2020.
30. Ronald L Rivest, Adi Shamir, and David A Wagner. Time-lock puzzles and timed-release crypto, 1996.
31. Philipp Schindler, Aljosha Judmayer, Markus Hittmeir, Nicholas Stifter, and Edgar R. Weippl. RandRunner: Distributed randomness from trapdoor VDFs with strong uniqueness. The Internet Society, 2021.
32. Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar R. Weippl. HydRand: Efficient continuous distributed randomness. In *2020 IEEE Symposium on Security and Privacy*, pages 73–89. IEEE Computer Society Press, May 2020.
33. Berry Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In Michael J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 148–164. Springer, Heidelberg, August 1999.
34. Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris-Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. In *2017 IEEE Symposium on Security and Privacy*, pages 444–460. IEEE Computer Society Press, May 2017.
35. DRAND team. DRAND project website, 2020. `https://drand.love` accessed on 21/03/2021.
36. Gang Wang, Zhijie Jerry Shi, Mark Nixon, and Song Han. Sok: Sharding on blockchain. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies, AFT 2019, Zurich, Switzerland, October 21-23, 2019*, pages 41–61. ACM, 2019.
37. Benjamin Wesolowski. Efficient verifiable delay functions. In Yuval Ishai and Vincent Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.

A The ALBATROSS random beacon

In this section, we describe the ALBATROSS [9] random beacon in the notation of this paper. To avoid too much complications we omit certain details of [9] addressed to distribute the computational complexity of the protocol among the parties. The precise steps are collected in Figure 14. ALBATROSS works as follows:

Each party P_a shares a secret vector of group elements by applying packed Shamir in the exponent: the secret of P_a is $(g^{f^{(a)}(0)}, \dots, g^{f^{(a)}(-(\ell-1))}) \in \mathbb{G}^\ell$ where $s_j^{(a)} = f^{(a)}(-j)$ for a random polynomial $f^{(a)}$ of degree at most $t+\ell-1$. Then P_a sends posts encrypted shares $\mathsf{pk}_i^{f^{(a)}(i)}$, $i \in [n]$. In addition P_a posts a proof that the encrypted shares indeed have as exponents $\sigma_i^{(a)} = f^{(a)}(i)$ for a polynomial of the right degree.

At this point, because proofs are publicly verifiable, parties can pinpoint a set of size $\mathcal{Q}$ of $n-t$ parties that have correctly shared secret vectors.

These secrets can be opened [8] as follows: when reconstructing the secret from dealer P_a, each party P_i decrypts the i-th share to $g^{f^{(a)}(i)}$ using its secret key and proves the correctness of the decryption using a discrete logarithm equality proof. In fact, P_i can prove decryption correctness for all $a \in \mathcal{Q}$ with a single DLEQ proof. Once at least $t+\ell$ parties have done that, we can reconstruct the secret $(g^{f^{(a)}(0)}, \dots, g^{f^{(a)}(-(\ell-1))})$ by Lagrange interpolation in the exponent.

Finally the actual output is obtained by applying a t-resilient matrix to the vectors $(S_0^{(a)}, \dots, S_{\ell-1}^{(a)}) = (g^{f^{(a)}(0)}, \dots, g^{f^{(a)}(-(\ell-1))})$, $a \in \mathcal{Q}$. More precisely, the matrix is applied, for each $j \in [0, \ell-1]$ to the vector of $(S_j^{(a)})_{a \in \mathcal{Q}}$. This makes the result uniformly random independently from the choices of any subset of at most t colluding parties in $\mathcal{Q}$. Each application of the matrix gives as output a vector of ℓ' group elements $(o_{kj})_{k \in [\ell']}$. Since we have applied this for every $j \in [0, \ell-1]$, the overall output is a matrix of $\ell \cdot \ell'$ independent random group elements.

[8] An alternative is that the dealer opens the secret by simply announcing $f^{(a)}$, upon which everyone can check this opening is correct. We omit this from Figure 14 to focus on how to open the secrets even if dealers do not cooperate at this point

ALBATROSS PVSS-based randomness beacon

Setup: A public bulletin board, a DDH-hard group $\mathbb{G} = \langle g \rangle$ of prime order q, a t-resilient matrix $M \in \mathbb{Z}_q^{\ell' \times (n-t)}$, e.g., setting $M_{ij} = \alpha^{i \cdot j}$ for some $\alpha \in \mathbb{Z}_q^*$ of order at least $\max\{n-t, \ell'\}$. Every party has a key-pair $(\mathsf{sk}_i, \mathsf{pk}_i) \in \mathbb{Z}_q \times \mathbb{G}$ where $pk_i = g^{\mathsf{sk}_i}$.
Preparation:

1. (Sharing) Each party P_a PVSSs a random secret $(g^{s_0^{(a)}}, \dots, g^{s_{\ell-1}^{(a)}}) \in \mathbb{G}^\ell$ by choosing $f^{(a)} \in \mathbb{Z}_q[X]_{\leq t+\ell-1}$ with $f^{(a)}(-j) = s_j^{(a)}$, setting $\sigma_i^{(a)} = f^{(a)}(i)$ and publishing $\hat{S}_i^{(a)} = \mathsf{pk}_i^{\sigma_i^{(a)}}$, $i \in [n]$, and a proof
$$\pi^{(a)} = \pi_{LDEI}((\mathsf{pk}_i)_{i=1}^n, (\hat{S}_i^{(a)})_{i=1}^n, t+\ell-1)$$
of correct sharing.
2. (Verification) Parties check the validity of $\pi^{(a)}$ for all $a \in [n]$. This fixes a set $\mathcal{Q}$ of the first $n-t$ parties $P_a, a \in \mathcal{Q}$ whose $\pi^{(a)}$ are correct.
3. (Opening)
   – Each party P_i, $i \in [n]$ computes, for all $a \in \mathcal{Q}$, the decrypted group share $S_i^{(a)} := g^{\sigma_i^{(a)}} = (\hat{S}_i^{(a)})^{1/\mathsf{sk}_i}$, and a correctness decryption proof
   $$\pi_i = \pi_{DLEQ}((g, (S_i^{(a)})_{a\in\mathcal{Q}}), (\mathsf{pk}_i, ((\hat{S}_i^{(a)})_{a\in\mathcal{Q}})).$$
   Note this proof asserts knowledge of $w = \mathsf{sk}_i$ that simultaneously satisfies $g^w = \mathsf{pk}_i$ and $(S_i^{(a))})^w = \hat{S}_i^{(a)}$ for all $a \in \mathcal{Q}$.
   P_i posts $S_i^{(a)}$ for all $a \in \mathcal{Q}$ and π_i
   – Every party verifies proofs π_i for each $i \in [n]$. Let I be the set of the first $t+\ell$ parties P_i who have posted correct proofs.
   – For $a \in \mathcal{Q}$, $j \in [0, \ell-1]$, parties reconstruct $S_j^{(a)} = \prod_{i\in I}(S_i^{(a)})^{L_{i,I}(-j)}$
4. (Output) The output is a $(\ell' \times \ell)$-matrix O of group elements where for $k \in [\ell'], j \in [0, \ell-1]$ the (k,j)-th entry is defined as
$$o_{k,j} = (\prod_{a\in\mathcal{Q}} S_j^{(a)})^{M_{k,a}}$$

Fig. 14: ALBATROSS