Mt. Random : Multi-Tiered Randomness Beacons

Ignacio Cascudo*, Bernardo David†, Omer Shlomovits‡, Denis Varlakov§
*IMDEA Software Institute, Madrid, Spain, ignacio.cascudo@imdea.org
†IT University of Copenhagen, Copenhagen, Denmark, bernardo@bmdavid.com
‡ZenGo X, Tel Aviv, Israel, omer@zengo.com
§ZenGo X, Tel Aviv, Israel, denis@zengo.com

***Abstract*—Many decentralized applications require a common source of randomness that cannot be biased by any single party. Randomness beacons provide such a functionality, allowing any (third) party to periodically obtain random values and verify their validity (i.e. check that they are indeed produced by the beacon and consequently random). Protocols implementing randomness beacons have been constructed via a number of different techniques. In particular, several beacons based on time-based cryptography, Publicly Verifiable Secret Sharing (PVSS), Verifiable Random Functions (VRF) and their threshold variant (TVRF) have been proposed. These protocols provide a range of efficiency/randomness quality trade-offs but guarantee security under different setups, assumptions and adversarial models.**

In this work, we propose Mt. Random, a multi-tiered randomness beacon that combines PVSS and (T)VRF techniques in order to provide an optimal efficiency/quality trade-off without sacrificing security guarantees. Each tier is based on a different technique and provides a constant stream of random outputs offering progressing 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.

Mt. Random (and all of its building blocks) can be proven secure under the standard DDH assumption (in the random oracle model) using only a bulletin board as setup, which is a requirement for the vast majority of beacons. We showcase the efficiency of our novel building blocks and of the Mt. Random beacon via benchmarks made with a prototype implementation. Our experimental results confirm the benefits of our multi-tiered approach, showing that even though higher tiers provide fresh random outputs more often, lower tiers can be executed fast enough to keep higher tiers freshly seeded.

I. 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 random values to all parties. This is particularly important in Proof-of-stake protocols [31], [18], [14], where such random beacons are needed to carry out the leader elections to decide the next party to publish a block. In addition, random beacons are important for other blockchain-related applications where committees must be elected, such as sharding [43], [19], [45], as well as for smart contracts that require a source of randomness. In such settings it is desirable to implement a random beacon as a protocol among the mutually distrustful participants of the corresponding system, i.e., without assistance of a trusted third party; moreover, we want to have a protocol with guaranteed output delivery, and whose output correctness can be publicly verified. The output of the protocol should not be predictable beforehand and/or biasable by an adversary that corrupts up to a certain threshold of the parties.

To illustrate the non-immediate nature of the problem, notice that a simple commit-and-open strategy where parties commit to local randomness and then output the sum of the opened values not quite enough, as parties can bias the output with a selective abort strategy, where they open or not their commitments depending on their view so far.

Given that challenge, several alternatives for constructing randomness beacons have been proposed based on cryptographic primitives, such as publicly verifiable secret sharing (PVSS) [31], [11], [12], [41], [39], verifiable random functions (VRF) [14], [18], [17], [29], [24], [42], verifiable delay functions (VDF) [8], [44], [6], [5], [38] and homomorphic encryption [15]. Moreover, achieving fairness against rational adversaries has also been considered in works that rely on financial incentives or punishments to encourage parties to behave honestly [2], [1], [7], [32], [4]. In particular, this rational approach has been proposed in the specific context of randomness beacons by the RANDAO project [36].

Constructions of beacons from these different primitives present a trade-off between the complexity of the construction (in terms of computation and communication) and how unbiasable or unpredictable they really are. In this work, we will focus on the two first types of random beacons, namely based on PVSS and VRFs, because their security is based on standard assumptions. In fact, we consider two different types of VRF-based constructions, one using plain VRFs and another using so-called threshold VRFs [42], [24], [29] (or TVRF, also called distributed VRF or DVRF). Before describing our approach, we give a brief overview of the complexity vs. randomness quality trade-offs given by each of these types of beacons.

Constructions using plain VRFs require very little computation and communication, but are open to the type of selective abort bias that we mentioned above. Since they rely on the computation of a VRF that can only be carried out by a party having its secret key, an adversary can always bias the final output by choosing whether to reveal or not its own VRF output, a fact that is captured in previous security analysis of this type of beacon [18].

Distributed VRFs get rid of this bias by always allowing a set of parties larger than a threshold (e.g. a majority of parties) to compute the verifiable random function, after a setup that consists on a distributed key generation protocol. Nevertheless, TVRF-based random beacons that have been proposed consist on a round-by-round protocol where at each round the TVRF is applied to the output of the previous round (and the random beacon output is defined to be some fixed function of that output). This has the inconvenience of requiring 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 this exactly plays out.

Finally, PVSS-based beacons such as SCRAPE [11] and ALBATROSS [12] enhance the commit-and-open strategy mentioned above by having parties commit to their inputs via publicly verifiable secret sharing. This approach renders the selective abort strategy useless, since unopened secrets can always be reconstructed by honest parties (provided there is an honest majority). On the downside, such protocols require more communication and computation from the parties. The recent proposal ALBATROSS [12] amends this to some extent by allowing parties to generate a much larger output than SCRAPE at the cost of little additional communication and computation. Nevertheless, in ALBATROSS there is still the issue that, while the parties generate a large batch of elements in a group as output, these elements are all known at once, so it may not be usable in scenarios where one should generate randomness gradually, as it happens with TVRF based protocols.

Recently, there is a growing interest in constructing beacons from time based primitives, such as Time Lock Puzzles (TLP) [37], [9], [30], [22] and the Related notion of Verifiable Delay Functions (VDF) [8], [35], [44], [20]. Such randomness beacons [8], [6], [5] achieve communication complexity linear in the number of parties while requiring only a common reference string as setup. However, these constructions are based on sequential computation assumptions that are not well understood, such as the hardness of problems over supersingular isogenies [20] and of iterated squarings over groups of unknown order [37]. Since little is known about concrete security parameters for such constructions, we focus our approach on PVSS and (T)VRF based beacons. However, since these approaches provide uniform pseudorandom values, they can potentially be used as Tier 2 of our beacon (which will be discussed in details).

A 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:

- Tier 1 - Uniform Randomness via PVSS: This tier provides batches of uniformly random outputs while only requiring a Public Ledger and a Random Oracle as setup. However, communication and computational complexities are quadratic in the number of parties executing the tier.
- Tier 2 - Uniform Pseudorandomness via TVRFs: Besides the setup required for Tier 1, this tier requires a setup phase for distributed key generation, after which it provides uniformly pseudorandom outputs (one per execution). Communication and computational complexities are linear in the number of parties executing the tier. Since the seed must be periodically refreshed, this tier uses outputs from Tier 1 as seeds every time a refresh is needed.
- Tier 3 - Bounded-Biased Pseudorandomness via VRFs: Regarding setup, besides a Public Ledger and a Random Oracle, this tier requires a random nonce, which is obtained from the outputs of Tier 2. Communication and computational complexities can be adjusted at the expense of output bias, i.e. the lower the complexity the higher the upper bound for the bias an adversary can introduce.

Publicly Verifiable Distributed Key Generation: We show that the SCRAPE and ALBATROSS protocols can be adapted to create a publicly verifiable distributed key generation (DKG) protocol that can provide both the keys needed for the TVRF and for the threshold encryption that we use in GULL. This protocol gives each party a threshold public key/private key pair $(tpk_i; tsk_i)$ where tsk_i is a Shamir sharing of a global secret key sk in a prime-order field $\mathbb{Z}_q$ and $tpk_i = g^{tsk_i}$ in a DDH-hard cyclic group of order q generated by g;

the global public key $pk = g^{sk}$ is also publicly known. The security of our DKG scheme is entirely based on DDH (in the random oracle model) and, as a consequence of the unbiasability of SCRAPE and ALBATROSS, it does not suffer from the problem that the public key may be biased by a rushing adversary (which happens in some other alternatives). In terms of communication and computational complexities, our protocol is more efficient than previous unbiasable DKG schemes and essentially as efficient as the best biasable scheme (as discussed in Appendix E). We are not aware of this protocol being described anywhere else.

GULL (Gradually UnLeashed aLbatross): Finally we introduce GULL, a PVSS-based random beacon that generates large batches of outputs that remain secret until a opening phase where smaller "sub-batches" can be gradually released. GULL is constructed by modifying and augmenting the ALBATROSS beacon using threshold encryption. Basically, instead of revealing their shares as in ALBATROSS, parties in GULL threshold encrypt (functions of) their shares and prove in zero knowledge that the resulting ciphertexts are correctly generated. In order to do that, we present an efficient zero knowledge proof for the required language.

Due to the added threshold encryption and zero knowledge proof machinery, GULL is understandably slower than ALBATROSS in case a full batch of random outputs is required. However, in case many fresh unpredictable uniformly random outputs are required, the ability to gradually release sub-batches of outputs makes GULL significantly more efficient than ALBATROSS: instead of re-executing the full protocol in order to obtain a full batch that is completely revealed, GULL allows for simply opening an encrypted sub-batch, which is much cheaper than the full protocol execution. In other words, GULL allows for preprocessing a large amount of sub-batches of uniformly random outputs that can later be revealed at a low cost (instead of generating new outputs on-the-fly).

B. Other Related Works

Since one of the contributions of this paper is a distributed key generation protocol for discrete logarithm based schemes, in Appendix E we give an overview of some relevant works in the extensive literature on this topic, namely [34], [26], [23]. Here we note briefly that these protocols have diverse pros and cons: [34], [26] only assume DDH hardness as our protocol, while [23] uses Paillier encryption and therefore needs the decisional composite residue assumption but it only requires one round of communication (in contrast, [34] may require 3 rounds in case of complaints, our protocol may require 4, and [26] may require up to 5). Another issue is that the output global key in [34] and [23] may be biased by a rushing adversary, even though this may not be a big problem for many applications as shown in [28], and seems quite inherent to low round complexity. We also note that [28] also constructed a distributed key generation protocol with improved communication complexity based on a gossip strategy; however, this construction does not generate finite field as secret keys, like the other alternatives we mention, but rather group elements, so they may not be used for example in our application.

II. Preliminaries

A. General notation

For integers $m \leq n$ we denote by $[m, n]$ the set $\{m, m+1, \ldots, n\}$. We let $[n] = [1, n]$, i.e. $\{1, \ldots, n\}$. Our protocols will take place in a cyclic group G of prime order q. Observe that, in such a group, any element distinct from the identity is a generator. We denote by $\mathbb{Z}_q$ the finite field of q elements, consisting of the integers modulo q, and note that we can speak of g^a for $g \in G, a \in \mathbb{Z}_q$ and this respects the rule $g^a \cdot g^b = g^{a+b}$ where the sum is in $\mathbb{Z}_q$. We will assume the DDH problem is hard in our group, i.e. given (g, g^a, g^b, g^c) where g is in G, a, b are uniformly random and independent in $\mathbb{Z}_q$ and c may be (with same probability) either uniformly random in $\mathbb{Z}_q$ and independent of (a, b) or defined by $c = a \cdot b$, then it is hard to decide in which of the two cases we are with probability non-negligibly larger than $1/2$.

B. Adversarial and Communication Models

The protocols analysed in this work are proven secure against a malicious static adversary, i.e. the adversary may arbitrarily deviate from the protocol but it must choose what parties to corrupt before the execution starts. For the sake of simplicity, we assume access to an authenticated bulletin board. Once a party posts a message 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). Notice that 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. Moreover, we assume synchronous communication, i.e. all messages sent (or posted to the bulletin board) within a round are guaranteed to be received by all parties before the next round.

C. 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.

We recall the secret sharing scheme we refer to as (t, ℓ)-packed Shamir secret sharing, a well-known generalization of Shamir's secret sharing scheme that allows to share a vector of ℓ secrets $(s_0, s_1, \ldots, s_{\ell-1})$ in $\mathbb{Z}_q^\ell$ as long as $n + \ell \leq q$. Standard Shamir's scheme is the case $\ell = 1$.

To share the secret, the dealer selects a polynomial of degree at most $t + \ell - 1$ such that $f(-j) = s_j$ for $j \in [0, \ell - 1]$ and sends the evaluation $\sigma_i = f(i)$ to P_i for $i \in [n]$.

Polynomial interpolation uniqueness properties guarantee that the secret is distributed independently from any set of t or fewer shares (privacy); while on the other hand it can be fully reconstructed from any set of $t + \ell$ shares or more

(($t+\ell$)-reconstruction). Indeed given a set $\mathcal{A}$ of exactly $t+\ell$ shares, we apply Lagrange interpolation in each coordinate of the secret, namely

$$s_j = \sum_{i\in\mathcal{A}} \sigma_i L_{i;\mathcal{A}}(\beta_j)$$

for $j = 0,\dots,\ell-1$, where

$$L_{i;I}(X) := \prod_{k\in I;k\neq i} \frac{X-\alpha_i}{\alpha_k-\alpha_i}.$$

A larger subset can reconstruct the secret by applying this process to the shares of some subset $\mathcal{A}$ of $t+\ell$ parties.

D. Non-interactive zero knowledge proofs

In a zero knowledge proof of knowledge a prover wants to convince a verifier of the veracity of a statement and of the fact that she knows a piece of information (witness) that makes the statement true, without revealing anything about this witness. Non-interactive proofs carry out this with a single message from the prover. Proofs considered here will be 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 [13] and low-degree exponent interpolation (LDEI) [12]. In fact, DLEQ proofs can be seen as a special case of LDEI proofs, and both can realized from standard Sigma-protocol techniques.

In a LDEI proof, we consider the cyclic group $\mathbb{G}$ of prime order q, and let $\alpha_1,\dots,\alpha_m$ be fixed public pairwise-different elements in the field $\mathbb{Z}_q$. The statement is given by a vector of elements $g_1,\dots,g_m,x_1,\dots,x_m$ of the cyclic group, and some integer $0\le d<m$. The prover needs to show that there exists a polynomial $w(X)$ in $\mathbb{Z}_q[X]$ of degree at most d that interpolates the discrete logarithms of the x_i's with respective bases g_i on evaluation points α_i, i.e., $x_i = g_i^{w(\alpha_i)}$ for all $i\in[m]$.

A non-interactive proof of knowledge of the polynomial $w(X)$ was presented in [12] and is given in Figure 1. The proof works in the random oracle model, and we denote it by

$$\pi_{\mathrm{LDEI}}\ ((g_i)_{i=1}^m;(x_i)_{i=1}^m;d).$$

A well known special case is $d=0$, where we obtain a discrete logarithm equality, or DLEQ, statement: what the prover is showing in that case is that the discrete logarithms of the x_i with respective bases g_i are all equal, i.e. $x_i = g_i^w$ for all $i\in[m]$ where now $w\in\mathbb{Z}_q$. We subsequently define

$$\pi_{\mathrm{DLEQ}}\ ((g_i)_{i=1}^m;(x_i)_{i=1}^m) := \pi_{\mathrm{LDEI}}\ ((g_i)_{i=1}^m;(x_i)_{i=1}^m;0)$$

E. 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 [11] and the subsequent modifications in ALBATROSS [12]. The PVSSs in these papers follow in turn the blueprint of Schoenmakers' PVSS [40].

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

Setup: Group $\mathbb{G}$, fixed pairwise distinct elements $\alpha_1,\dots,\alpha_m$ in $\mathbb{Z}_q$, a random oracle $H(\cdot)$

Statement: $\{(g_1,\dots,g_m,x_1,\dots,x_m,d)\in\mathbb{G}^{2m}\times\mathbb{Z} : \exists w(X)\in\mathbb{Z}_q[X];\ \deg w\le d;\ x_i = g_i^{w(\alpha_i)}\ \forall i\in[m]\}$ (and the prover knows $w(X)$).

Protocol:

- The prover samples $u(X)\leftarrow\mathbb{Z}_q[X]$ with $\deg u\le d$ and computes $a_i = g_i^{u(\alpha_i)}$ for all $i\in[m]$, in addition to $e = H(g_1,\dots,g_m,x_1,\dots,x_m,a_1,\dots,a_m)$; and $z(X) = u(X) - e\cdot w(X)$. The proof is (e,z).
- The verifier computes $a_i = g_i^{z(\alpha_i)}x_i^e$ for all i and checks that $e = H(g_1,\dots,g_m,x_1,\dots,x_m,a_1,\dots,a_m)$ and that $\deg z\le d$, accepts if these two conditions are true, and otherwise rejects.

Fig. 1: LDEI zero knowledge proof of knowledge π_{LDEI} from [12].

We describe the PVSS in ALBATROSS, which 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 $t\le(n-\ell)/2$ 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 $\ell=1$. One important point in favor of this generalization is that the amortized computation and communication per secret shared becomes much better as ℓ grows. The construction of the PVSS in ALBATROSS can be seen in Figure 2.

PVSSs can be used to construct random beacons as follows: parties commit to a secret random choice in a group (in the case of ALBATROSS the group would be $\mathbb{G}^\ell$) by PVSSing it among the remaining participants. At that point all parties and any external verifier can check the validity of each sharing and determine the set $\mathcal{Q}$ of parties which have dealt correctly. Once the set $\mathcal{Q}$ of parties that have correctly shared a secret is pinpointed, each of these secrets will always be opened, even if the dealer refuses to open it; indeed, they can be reconstructed by the remaining parties, and also this process is publicly verifiable. In fact at the point where $\mathcal{Q}$ is determined, the output is also fully fixed. This output is constructed by applying a randomness extractor to the opened secrets, so that the result is independent from the input choice of any t parties.

This randomness extractor could simply consist on the group operation applied to the opened secrets. The result would be independent of any set of all but one of these secrets. However, ALBATROSS exploits the fact that by assumption there is more than one honest party in $\mathcal{Q}$, and extracts a larger output. This requires the notion of t-resilient matrix.

Definition 1. A matrix $M\in\mathbb{Z}_q^{r\times m}$ is t-resilient if for any $A=\{i_1,\dots,i_t\}\subset[m]$ of size t, Mv is independent from the coordinates of v indexed by A, i.e. for any

Packed PVSS in ALBATROSS [12].

Parameters: Let n be the number of parties that receive shares, and $1 \le t \le (n-\ell)/2$ be the corruption threshold, where $\ell \ge 1$ is an integer.
Setup: A public bulletin board, field $\mathbb{Z}_q$, and DDH-hard group G with generator g. Every party has a private key $sk_i \in \mathbb{Z}_q$, and public key $pk_i = g^{sk_i} \in G$.
Sharing:
The secret is a tuple $(g^{s_0}, \dots, g^{s_{\ell-1}}) \in G^\ell$, for $(s_0, \dots, s_{\ell-1}) \in \mathbb{Z}_q^\ell$ chosen by the dealer.

1) The dealer constructs Shamir's shares for $(s_0, \dots, s_{\ell-1}) \in \mathbb{Z}_q^\ell$ by selecting a polynomial $f \in \mathbb{Z}_q[X]$ of degree at most $t+\ell-1$, with $f(-j) = s_j$, $j = 0, \dots, \ell-1$, defines $\sigma_i = f(i)$, $i = 1, \dots, n$. We refer to s_j; σ_i as "Shamir secret and shares".
2) The dealer posts the "encrypted group shares" $\hat{S}_i = pk_i^{\sigma_i}$ on the bulletin board, together with the NIZK proof $\pi_{LDEI}((pk_i)_{i=1}^n, (\hat{S}_i)_{i=1}^n, t+\ell-1)$, asserting that indeed $(\sigma_1, \dots, \sigma_n) = (f(1), \dots, f(n))$ for a polynomial f of degree at most $t+\ell-1$.

Sharing verification:

1) Check whether π_{LDEI} is correct.

Reconstruction: A set A containing at least $t+\ell$ honest parties (whose existence is guaranteed if there are t corruptions) can reconstruct $(g^{s_0}, \dots, g^{s_{\ell-1}})$ as follows:

1) Using sk_i, the i-th party computes $S_i = (\hat{S}_i)^{sk_i^{-1}}$. Note this is supposed to be $S_i = g^{\sigma_i}$, the "group share".
2) The i-th party posts S_i on the bulletin board together with a NIZK proof of correct decryption $\pi_i = \pi_{DLEQ}((g, S_i), (pk_i, \hat{S}_i))$.[a]
3) Given any subset $I \subseteq A$ of exactly $t+\ell$ decrypted shares $(S_i)_{i \in I}$ for which π_i is correct (e.g. the first $t+\ell$ with that condition), any party or external verifier can reconstruct each g^{s_j} via Lagrange interpolation in the exponent:
$$g^{s_j} = \prod_{i \in I} S_i^{L_{i,I}(-j)}.$$

[a] Indeed note that $g^{sk_i} = pk_i$, $S_i^{sk_i} = \hat{S}_i$, and P_i knows sk_i. We also remark that swapping the roles of pk_i and S_i would not work, as P_i does not know the common exponent σ_i that would be needed for the proof in that case.

Fig. 2: Packed PVSS in ALBATROSS.

ALBATROSS Random beacon from PVSS

Setup and parameters: Parameters are exactly as in Figure 2, in particular $1 \le t \le (n-\ell)/2$ for some integer $\ell \ge 1$. Define $\ell' = n - 2t$ and note that $\ell \le \ell'$. Let $M \in \mathbb{Z}_q^{\ell' \times (n-t)}$ be a t-resilient matrix.
Protocol:

1) (Sharing) Each party P_a shares a random secret $(g^{s_0^{(a)}}, \dots, g^{s_{\ell-1}^{(a)}}) \in G^\ell$ with the sharing phase of the PVSS (Figure 2).
2) (Verification) After the sharing round is finished, Every party executes the sharing verification phase on every shared secret. Since verification is public, this fixes a set Q of the first $n-t$ parties P_a, $a \in Q$ who have correctly shared.
3) (Reconstruction) Every party P_a in Q opens the Shamir secret $(s_0^{(a)}, \dots, s_{\ell-1}^{(a)})$ and the randomness used and parties verify it is consistent with the sharing posted before and if so, set P_a's group secret as $(g^{s_0^{(a)}}, \dots, g^{s_{\ell-1}^{(a)}})$. If P_a refuses to open, or opens an invalid secret, the group secret $(g^{s_0^{(a)}}, \dots, g^{s_{\ell-1}^{(a)}})$ is reconstructed using the reconstruction phase in the PVSS.
4) (Aggregation) At this point we have a matrix of opened secrets, with rows corresponding to $a \in Q$, and columns $j \in [0, \ell-1]$. Now to every column j, the randomness extractor given by the t-resilient matrix M is applied (this can just be done by each party locally, as everything is public now). Index the columns of M with $a \in Q$ and rows with $k \in [\ell']$. Then for every $k \in [\ell']$, and every $j \in [0, \ell-1]$, the (k, j)-th output is
$$o_{k,j} = g^{\sum_{a \in Q} M_{k,a} s_j^{(a)}},$$
where o_{kj} can be computed from public information as
$$o_{k,j} = \prod_{a \in Q} (g^{s_j^{(a)}})^{M_{k,a}}.$$
This is a total of $\ell \cdot \ell'$ values (which is ℓ^2 if $\ell = \ell'$).

Fig. 3: ALBATROSS Random beacon using PVSS [12].

$(y_1, \dots, y_t) \in \mathbb{Z}_q^t$, the distribution of Mv when conditioned to $v_{i_1} = y_1, \dots, v_{i_t} = y_t$ 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 $r \le m - t$. An optimal choice (i.e. $r = m - t$) results of taking M to be a transpose of a Vandermonde matrix (we are assuming $q \ge m$). For computation efficiency reasons, [12] choose M to also be itself Vandermonde, i.e. $M_{ij} = \omega^{ij}$ for some $\omega \in \mathbb{Z}_q$ of large enough order. In summary, the random beacon protocol is as in Figure 3.

The parameter $\ell' = n - 2t$ is the size of the output of the t-resilient function. In ALBATROSS, parameters were set such that $\ell' = \ell$, and in SCRAPE, $\ell' = \ell = 1$. In this latter case we obtain that (for the optimal corruption $2t = n - 1$), $M \in \mathbb{Z}_q^{1 \times (n-t)}$ is in fact the vector $(1, 1, \dots, 1)$. The output consists of 1 element of the group in that case, namely the element $g^{\sum_{a \in Q} s^{(a)}}$ where $s^{(a)}$ is the (in SCRAPE's case, single) Shamir secret shared by P_a. We remark that, in this paper, we keep ℓ and $\ell' = n - 2t$ as two separate parameters.

F. Verifiable Random Functions (VRFs)

A verifiable random function (VRF) [33] is a pseudorandom function that can be evaluated by the owner of a secret key, who at the same time produces a proof or correct evaluation, which can be verified by using the corresponding public key. A VRF scheme consists on three algorithms (λ denotes a security parameter):

KeyGen(1^λ): outputs a pair (pk, sk) of a public and a secret key.
Eval(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.
Verify(pk, x, y, π) is a probabilistic algorithm that outputs 0 or 1 (respectively meaning ”reject” or ”accept” the proof).

It has been observed in [18] that the standard VRF definition is not sufficient in the randomness beacon setting. Notice that pseudorandomness only holds in case the key pair has been honestly generated (*i.e.* by KeyGen) but not when it is generated maliciously, allowing the adversary to bias VRF outputs computed under maliciously generated keys. Indeed, in VRF based beacons (*e.g.* Figure 4), the adversary can generate its own key pairs maliciously. Hence, in this setting, we require the VRF to be unpredictable under maliciously key generation as defined in [18]. In Appendix A we present the definition and a construction of a VRF with unpredictability under malicious key generation.

We show in Figure 4 a construction of a VRF based random beacon from [18]. The beacon uses an initial seed 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: Each party has a key-pair for a VRF and evaluates the VRF on the seed. The parties define the output of that round to be the hash of the XOR of the correctly computed evaluations (which the can check using the verification procedure and the public keys), and use that output to define the seed for the next round. Note this process opens the door for biasing strategies: malicious parties may simply wait until honest parties publish their evaluations of the VRF and then decide whether they publish theirs, thereby deciding the final result.

VRF-based beacon

Setup: The setup contains some initial seed ρ_0, and a random oracle $H : \{0,1\}^{\ell_{VRF}} \to \{0,1\}^m$.

Beacon:

1) Each party executes $\mathsf{KeyGen}(1^\lambda)$ of the VRF obtaining a key-pair (pk_i, sk_i), and publishes pk_i.
2) At round $r = 1, 2, \ldots$: Let $m_r = r \| \rho_{r-1}$.
   a) Every party P_i computes and publishes $(\sigma_r^i, \pi^i) = \mathsf{Eval}(sk_i, m_r)$.
   b) Each party verifies proofs of the remaining parties by applying $\mathsf{Verify}(pk_i, m_r, \sigma_r^i, \pi^i)$, defines I to be the set of parties that have posted a correct (σ_r^i, π^i), and computes $\rho_r = \bigoplus_{i \in I} \sigma_r^i$. The output of this round is $w_r = H(\rho_r)$

Fig. 4: VRF-based beacon from [18].

G. Threshold Verifiable Random Functions (TVRFs)

Analogously to the case of signatures, one can also 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 [25] we define a DVRF as the tuple of algorithms below, where as usual t denotes the corruption threshold:

$\mathsf{DistKeyGen}(1^\lambda)$: outputs secret keys $tsk_i, i \in [n]$, corresponding public partial keys tpk_i and a global public key tpk.

$\mathsf{PartialEval}(x, tsk_i, tpk_i)$ is a deterministic algorithm which outputs a pair $m_i = (y_i, \pi_i)$ where y_i is the evaluation of the (implicit) random function F at x and π_i is a proof.

$\mathsf{Combine}(tpk, \{tpk_i\}, x, A, (m_i)_{i \in A})$ is a probabilistic algorithm that takes a set of at least $t+1$ evaluations (indexed by A) and outputs either a pair (y, π) consisting of a global evaluation y and a global proof π, or $\perp$.

$\mathsf{Verify}(tpk, x, y, \pi)$ is a probabilistic algorithm that outputs 0 or 1 (respectively meaning “reject” or “accept” the proof).

Security definitions and a construction of a TVRF can be found in Appendix B.

Notice that, in the threshold scenario, the pseudorandomness property of the standard definition is sufficient to guarantee that VRF outputs are unbiased because the distributed key generation procedure guarantees that keys are correctly generated.

We present in Figure 5 a TVRF-based random beacon proposed by the DRAND [42] and Dfinity [29] projects and proven secure in [25]. The idea is to apply the verifiable random function iteratively starting with some seed as initial TVRF input and, in every subsequent round, applying the TVRF to the output of the previous round. The random beacon output at a certain round is the hash of that round's TVRF output.

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 ρ_0 and $H' : G \to \{0,1\}^\ell$ a hash function.

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

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

Fig. 5: The DRAND/Dfinity beacon.

H. 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. Similar to threshold signatures and threshold verifiable random functions, threshold encryption schemes require a distributed key generation protocols providing every decrypting party with a partial secret key, and publishing corresponding partial public keys and a global public key, the latter of which is used by any sender to encrypt a message, while the partial public keys guarantee that each decrypting party carries out the decryption correctly. In this work consider here El Gamal threshold encryption [21], which requires exactly the same ensemble of keys as the TVRF we have seen above. We present further security definitions threshold encryption and a construction of threshold El Gamal in Appendix C.

III. Distributed Key Generation via PVSS

In the following section we will need to run El Gamal threshold encryption protocol, and we therefore need a distributed key generation protocol to provide keys to the parties involved. We could use some of the existing protocols discussed in Appendix E but here we present an alternative based on the ideas from SCRAPE and ALBATROSS that is fully based on the DDH-assumption and compares rather positively to these alternatives.

Recall that our goal is to establish a common public key $tpk = g^{tsk}$, partial public keys $tpk_i = g^{tsk_i}$ such that tsk_i are Shamir shares for tsk, and in addition party P_i receives tsk_i. Thinking of the case $\ell = 1; \ell^0 = 1$ in ALBATROSS one realizes that the two first requirements are given by that protocol: the parties will have established a random value g^{tsk} (the output of ALBATROSS in that case), and can easily obtain the partial public keys g^{tsk_i} from the information known at the end of the protocol: while we did not need to compute these values explicitly in Figure 3, the i-th partial key can be computed by aggregating the decrypted shares of the i-th party for each of the secrets, in the same way as g^{tsk} is computed from the reconstructed group secrets.

However we still have the problem of how party P_i can compute tsk_i. This requires to modify the secret sharing phase so that when P_a deals a secret $s^{(a)}$ this party sends information that allows P_i not only to reconstruct $g^{\ _i^{(a)}}$ but also $_i^{(a)}$ (recall $_i^{(a)}$ is the Shamir share of $s^{(a)}$). We solve this by also sending a ciphertext $E_i^{(a)} = {}_i^{(a)} \quad H(g^{\ _i^{(a)}})$ containing $_i^{(a)}$ that can only be decrypted by learning $g^{\ _i^{(a)}}$, which in turn can only be obtained by party P_i with its secret key. We need then to discuss what happens if the encrypted message $E_i^{(a)}$ in does not correspond to the value in the exponent of $pk_i^{\ _i^{(a)}}$ which the dealer has also posted. In comparison to Fouque-Stern DKG, where the use of Paillier encryption allows the dealer to construct an elegant non-interactive proof of the fact that the two values are indeed the same, here we do not have this possibility. What we do is to simply have P_i complain if it sees that the value in $E_i^{(a)}$ does not match the exponent in $pk_i^{\ _i^{(a)}}$, in which case the dealer needs to reveal $_i^{(a)}$. This is not a problem since at this point we know that one of P_a or P_i is cheating. If party P_a is cheating, all values $_i^{(a)}$ for all $i \; 2 \; [n]$ will be ignored. On the other hand, if P_a is honest, the cheating complainer P_i reveals an additive share of its own tsk_i.

Finally, we also point out the following modification with respect to the order of operations in ALBATROSS, which we will also exploit later in GULL: in ALBATROSS parties would first decrypt their shares for each of the shared secrets (and prove decryption correctness) and reconstruct the secrets of each dealer (step 3 of Figure 3), and then these opened secrets would be aggregated (step 4); here, we note that instead parties can first aggregate their shares and then decrypt them and reconstruct the final result directly. Indeed, note that from the posted encrypted shares $pk_i^{\ _i^{(a)}}$ to P_i the aggregated value $pk_i^{\text{P}_{a2Q} \ _i^{(a)}}$ can be computed publicly, P_i can decrypt each value to $g^{\ _i^{(a)}}$ secretly, aggregate all to $g^{\text{P}_{a2Q} \ _i^{(a)}}$ and then post this value and a DLEQ proof that it is correct with respect to $pk_i^{\text{P}_{a2Q} \ _i^{(a)}}$. The complete protocol is in Figure 6.

The distributed key generation protocol has the properties that [26] called correctness and that are called robustness in [28], namely that all honest parties agree on a global public key, whose corresponding global secret key can be reconstructed from any set of partial secret keys containing at least $t+1$ honest ones, and the public transcript. In addition the public key is unbiasable. In order to capture these properties, we define an ideal functionality $F_{DDH \quad DKG}$ in Figure 7, which is tailored to the DDH setting we are working on. $F_{DDH \quad DKG}$ essentially outputs random partial public keys and secret key shares to honest parties while allowing for the adversary to arbitrary secret key share (and consequently arbitrary partial public keys) for corrupted parties. We remark that $F_{DDH \quad DKG}$ can be used as the DKG building block for a number of protocols, e.g. threshold El Gamal and the TVRFs in [25] (including the Dfinity TVRF).

We formally analyse the security of $_{DDH \quad DKG}$ from Figure 6 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 instance of the protocol can be executed in sequence while preserving their security. More details about this model can be found in [10].

Theorem 1. Under the DDH assumption and assuming an authenticated bulletin board, $_{DDH \quad DKG}$ securely realizes $F_{DDH \quad DKG}$ in the random oracle model against a malicious static PPT adversary A corrupting at most $t \quad \frac{n \ 1}{2}$ parties.

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

1) In round 1, S proceeds as follows:

   Upon receiving $(\text{GEN}; sid; P_a)$ from $F_{DDH \quad DKG}$ for an honest party P_a, S acts exactly as an honest party would, sampling a random $s^{(a)} \; 2 \; Z_q$, dealing it with the SCRAPE PVSS and, for all $2 \; [n]$, posting $\hat{S}_i^{(a)}; \ ^{(a)}; E_i^{(a)}$ on the bulletin board. Finally, add P_a to Q, i.e. the set of parties who provide valid shares.

   When A posts $\hat{S}_i^{(a)}; \ ^{(a)}; E_i^{(a)}$ for $i = 1; :::; n$ on the bulletin board on behalf of a corrupted party $P_a \; 2 \; P^A$, S checks whether to add P_a to Q or not:

   a) Verify the proof $^{(a)}$ is valid.
   b) Use the extractor from the zero knowledge proof $_{LDEI}$ to obtain $_i^{(a)}$ from $^{(a)}$ for all $i \; 2 \; [n]$.

Distributed key generation via SCRAPE - $_{DDH \quad DKG}$

Parameters: Let n be the number of parties that receive shares, and let $1 \quad t \quad (n \quad 1)=2$ be an integer, the corruption threshold.
Setup: A public bulletin board, field Z_q, and DDH-hard group G with generator g. Every party in the system has a private key sk_i 2 Z_q, and public key $pk_i = g^{sk_i}$. A random oracle $H : G \ ! \ f \ 0; 1g^{d\log qe}$. We also assume some injective encoding $Z_q \ ! \ f \ 0; 1g^{d\log qe}$ which is easy to invert.
Protocol

1) In round 1, each party P_a proceeds as follows:
   P_a chooses $s^{(a)}$ 2 Z_q and deals it with the SCRAPE PVSS: P_a selects a polynomial $f^{(a)}$ 2 $Z_q[X]$ of degree at most t, with $f^{(a)}(0) = s^{(a)}$ and, for all i 2 $[n]$, defines $^{(a)}_i = f^{(a)}(i)$, computes $\hat{S}^{(a)}_i = pk_i^{\ ^{(a)}_i}$ and computes $^{(a)} = \ _{LDEI}((pk_i)^n_{i=1}; (\hat{S}^{(a)}_i)^n_{i=1}; t)$.
   For all i 2 $[n]$, P_a computes $E^{(a)}_i = \ ^{(a)}_i \quad H(g^{\ ^{(a)}_i})$ and posts $\hat{S}^{(a)}_i; \ ^{(a)}; E^{(a)}_i$ on the bulletin board.
2) In round 2, for all i, P_i verifies the proof $^{(a)}$ for all a; for those a for which the proof rejects P_i posts a complaint against P_a on the bulletin board. Moreover P_i computes $^{(a)}_i$ from $E^{(a)}_i$ as $^{(a)}_i = H((\hat{S}^{(a)}_i)^{\frac{1}{sk_i}}) \quad E^{(a)}_i$ and checks whether $\hat{S}^{(a)}_i = pk_i^{\ ^{(a)}_i}$. If this does not hold then P_i posts a complaint against P_a to the bulletin board. Otherwise, P_i sets $S^{(a)}_i = g^{\ ^{(a)}_i}$.
3) If no complaints were posted, ignore this round and execute the instructions of round 4. Otherwise, in round 3, for all i, P_i proceeds as follows:
   If a proof $^{(a)}$ receives more than t complaints, P_a is disqualified.
   If a party P_a receives a complaint from P_i about its encrypted share, then P_a reveals $^{(a)}_i$. If $\hat{S}^{(a)}_i$ 6= $pk_i^{\ ^{(a)}_i}$ or $E^{(a)}_i$ 6= $^{(a)}_i \quad H(g^{\ ^{(a)}_i})$, P_a is disqualified.
   Let Q be the set of parties who have posted encrypted shares and proofs without being disqualified.
4) In round 4, for all i, party P_i proceeds as follows:
   a) P_i computes $\hat{S}_i = \prod_{a2Q} \hat{S}^{(a)}_i$ and $_i = \sum_{a2Q} \ ^{(a)}_i$. Also P_i sets $S_i = \prod_{a2Q} S^{(a)}_i$.
   b) P_i publishes $\hat{S}_i$, S_i and $_{DLEQ}(g; S_i; pk_i; \hat{S}_i)$ in the bulletin board.
5) Finally, after round 4, all parties proceed as follows:
   a) For all $\hat{S}_i; S_i; \ _{DLEQ}((g; S_i); (pk_i; \hat{S}_i))$ posted to the bulletin board, verify $\hat{S}_i = \prod_{a2Q} \hat{S}^{(a)}_i$ and the proof $_{DLEQ}((g; S_i); (pk_i; \hat{S}_i))$. Let I be the set of all indices for which these checks pass.
   b) Let $J \quad I$ be a set of cardinality $t+1$ (e.g. the first $t+1$). The output global public key is $tpk = S = \prod_{i2J} S_i^{L_{i;J}(0)}$. The i-th partial public key (for i 2 I) is $tpk_i = S_i$. The i-th partial secret key (for i 2 I) is $tsk_i = \ _i$. Finally, note the global secret key is implicitly defined as $tsk = s = \sum_{a2Q} s^{(a)}$.

Fig. 6: Protocol $_{DDH \quad DKG}$ for distributed key generation via SCRAPE.

Functionality $F_{DDH \quad DKG}$

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

Upon receiving $($GEN$; sid; P_i)$ from a party P_i:

1) If P_i is honest, forward $($GEN$; sid; P_i)$ to S.
2) If P_i is corrupted, wait for S to send $($SETSHARE$; sid; P_i; \ _i)$ where $_i$ 2 Z_q and set $tpk_i = g^{\ _i}$.
3) Let J be the set of all parties P_j who sent $($GEN$; sid; P_j)$. If all honest parties are in J, proceed as follows:
   a) Sample a random polynomial f of degree at most t with $f(i) = \ _i$ for all $_i$ sent by S in step 2): [a] For every honest party P_h, set $tpk_h = g^{\ _h}$ with $_h = f(h)$.
   b) Set $tpk = g^{f(0)}$.
   c) For all corrupted parties P_c 2 J, send $($KEYS$; sid; \ _c; f\, tpk_j\, g_{j2J}; tpk)$ to S.
   d) Wait for S to answer with $($ABORT$; sid; C)$ where C is a set of corrupted parties.
   e) For all j 2 J n C, send $($KEYS$; sid; \ _j; f\, tpk_k\, g_{k2JnC}; tpk)$ to P_j. [b]

[a] This is possible since the adversary can only set at most t values $_i$.
[b] Notice that $f\, tpk_k\, g_{k2JnC}$ can always be used to obtain $tpk = g^{f(0)}$ by Lagrange interpolation because jJ n $Cj \quad n \quad t > t$.

Fig. 7: Distributed Key Generation Functionality $F_{DDH \quad DKG}$

c) Verify that $E^{(a)}_i = \ ^{(a)}_i \quad H(g^{\ ^{(a)}_i})$ for all i 2 $[n]$.
d) If and only if all these checks pass, add P_a to Q.

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

2) For every corrupted party P_i 2 $P^A \setminus Q$, S computes the secret key shares $_i = \sum_{a2Q} \ ^{(a)}_i$ and sends $($GEN$; sid; P_i)$ and $($SETSHARE$; sid; P_i; \ _i)$ to $F_{DDH \quad DKG}$. S waits for message $($KEYS$; sid; \ _i; f\, tpk_j\, g_{j2Q}; tpk)$ for P_i 2 P^A from $F_{DDH \quad DKG}$. Notice that S can do that since it knows $^{(a)}_i$ provided by simulated honest parties and it has extracted the corresponding values from corrupted parties.
3) In rounds 2 and 3, S executes exactly the same instructions as an honest party. Notice that this will yield the same set Q computed in step 1.
4) In round 4, for every i such that P_i 2 Q is honest, computes $\hat{S}_i = \prod_{a2Q} \hat{S}^{(a)}_i$, uses the simulator from the ZK proof $_{DLEQ}$ to generate an accepting proof $_{DLEQ}(g; tpk_i; pk_i; \hat{S}_i)$ and posts $\hat{S}_i$, tpk_i and $_{DLEQ}(g; tpk_i; pk_i; \hat{S}_i)$ on the bulletin board.
5) After round 4, let C be the set of corrupted parties who post $\hat{S}_i$, S_i and $_{DLEQ}(g; S_i; pk_i; \hat{S}_i)$ with an invalid proof $_{DLEQ}(g; S_i; pk_i; \hat{S}_i)$. S sends $($ABORT$; sid; C)$ to $F_{DDH \quad DKG}$.
6) S executes the remainder of the protocol as an honest party would and, when A terminates, outputs whatever A outputs.

We now show that the execution with S and $F_{DDH \quad DKG}$ is indistinguishable from an execution of $_{DDH \quad DKG}$ with A. First of all, notice that in rounds 1, 2 and 3 all messages sent from S to A (through the bulletin board) are distributed exactly as in $_{DDH \quad DKG}$. Moreover, notice that after round

1 is finished $\mathcal{S}$ computes the same set $\mathcal{Q}$ as parties would compute after round 3 of ${}_{DDH-DKG}$. 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 ${}_i^{(a)}$ from ${}^{(a)}$ for all corrupted parties P_a. Having determined $\mathcal{Q}$, $\mathcal{S}$ is able to determine the choices of secret key shares ${}_a$ 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 ${}_a$ to $F_{DDH-DKG}$.

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 ${}_{DDH-DKG}$, which intuitively means that $\mathcal{A}$ cannot bias the global public key even though it can choose secret key shares ${}_a$ 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 ${}_i^{(a)}$ to $\mathcal{A}$. First, notice that it is proven in [11] 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 ${}_i^{(a)}$ such that $E_i^{(a)} = {}_i^{(a)} \quad H({}_i^{(a)})$ and thus learn ${}_i^{(a)}$ via $E_i^{(a)}$ with negligible probability, since it can only make $poly(k)$ queries to the random oracle and ${}_i^{(a)}$ is chosen uniformly at random from an $exp(k)$ 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 ${}_i^{(a)}$ 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 ${}_{LDEI}$, $\mathcal{S}$ 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 $S_i^{(a)}$. □

As an aside, we remark two interesting extensions of our distributed key generation, which we only explain informally.

Remark 1 (Refreshing partial keys): The protocol can be modified to one that, given a distributed key ensemble $(pk;\{pk_i\};\{sk_i\})$ in the form above (not necessarily created by our protocol) outputs fresh random partial secret and public keys tsk_i, 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 6. 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 $(pk';\{pk_i'\};\{sk_i'\})$ with $pk' = 1_G$. Now parties can define $\tilde{pk}_i = pk_i \; pk_i'$ and (privately by party P_i) $\tilde{sk}_i = sk_i + sk_i'$, and output the ensemble $(pk;\{\tilde{pk}_i\};\{\tilde{sk}_i\})$.

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 $\ell = 1$, $\ell' \; 1$, where assuming now $t \; (n \; \ell')=2$, we would obtain as output ℓ' independent instances $(tpk^k;\{tpk_i^k\};\{tsk_i^k\})$, $k \in [\ell']$.

The protocol works in the same way until step 4. In step 5 parties P_i compute $\hat{S}_{i;k} = \prod_{a\in\mathcal{Q}}(\hat{S}_i^{(a)})^{M_{k;a}}$, ${}_{i;k} = \sum_{a\in\mathcal{Q}} M_{k;a}\,{}_i^{(a)}$ and $S_{i;k} = \prod_{a\in\mathcal{Q}}(S_i^{(a)})^{M_{k;a}}$ for $k = 1;\dots;\ell'$. 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 clearly be extended to deal with refreshing ℓ' ensembles.

IV. GULL: Gradual Release of PVSS Outputs via Threshold Encryption

While the ALBATROSS construction provides a large uniformly random output, one problem is that the whole output is reconstructed by the participants at once. For applications, it is instead desirable that parts of this output are released gradually, while the rest of the output is still hidden. In this section, we depart from ALBATROSS to construct GULL, a random beacon that can accomplish this. Recall that in ALBATROSS as described in Figure 3, the output consisted of a total of $\ell \cdot \ell'$ group elements, that we can think of as consisting of ℓ' blocks of ℓ elements each; in our modification, parties carry out the beginning of the protocol as in ALBATROSS (until the whole output is fixed), but then are able to release every block independently. Every block can be released with little communication and computation and, furthermore, the blocks that have not yet been released are unpredictable given the ones that are known already.

In order to do this, we reutilize a trick from the previous section: note that after step 2 of the protocol in Figure 3, a set $\mathcal{Q}$ of well-behaved dealers (dealers who have shared their secret correctly) has been set. What we do now is to swap the order of steps 3 and 4, i.e., we have every party aggregate the shares before reconstructing the secrets. More precisely, we can do this in the following way: every party can compute from public information $R_{ik} = \prod_{a\in\mathcal{Q}}(\hat{S}_i^{(a)})^{M_{k;a}}$ for every i and every $k \in [1;\ell']$. Additionally, each P_i can compute the value $S_{ik} = R_{ik}^{sk_i^{-1}}$. Note that $S_{ik} = \prod_{a\in\mathcal{Q}}(S_i^{(a)})^{M_{k;a}}$:

Note that for every k, P_i could prove the correctness of the value S_{ik} if P_i were to open it, since R_{ik} is known by everyone, and P_i could then use ${}_{DLEQ}((g;S_{ik});(pk_i;R_{ik}))$. However, in our case P_i will not directly open S_{ik}, but rather encrypt it with threshold El Gamal. Namely, P_i publishes $E_{ik} = Enc(tpk;S_{ik}) = (g^{r_{ik}};tpk^{r_{ik}} \; S_{ik}) := (c_{ik};d_{ik})$ (where the randomness r_{ik} must be independent of each other for $k \in [1;\ell']$) and provides a zero-knowledge proof ${}_{EG}$ that the value S_{ik} encrypted as E_{ik} satisfies $S_{ik}^{sk_i} = R_{ik}$ where sk_i is the same as in the equation $g^{sk_i} = pk_i$. This proof is slightly more complicated than the DLEQ proof mentioned above, and we detail it in Appendix D.

Parties can now agree on a set $\mathcal{T}$ of $t + \ell + 1$ parties that have published correct proofs for every $k \in [1;\ell']$. For every $k \in [1;\ell']$ and every $j \in [0;\ell \; 1]$, and from the encrypted values everyone can compute $O_{kj} = \mathsf{Enc}(\mathsf{tpk}; \prod_{i \in I} S_{ik}^{L_{i,I}(-j)})$ using the linearity of El Gamal.

Then, at the opening stage parties could decrypt O_{kj} individually by using the threshold decryption protocol to obtain the outputs o_{kj} one by one. Nevertheless, one needs to take into account that opening one o_{kj} reveals information about the values $o_{kj'}$ for other $j' \in [0, \ell - 1]$. Therefore we consider that the batch $(o_{k0}, o_{k1}, \dots, o_{k(\ell-1)})$ is opened at once. However, the independence of the output "holds in the other coordinate", i.e., having opened batches $(o_{k0}, o_{k1}, \dots, o_{k(\ell-1)})$ for $k \in [1, \ell'']$, for some $\ell'' < \ell'$, the remaining unopened batches $(o_{k0}, o_{k1}, \dots, o_{k(\ell-1)})$, $k \in [\ell'' + 1, \ell']$ remain uniformly random in the view of the adversary.

Indeed, fix any j. We recall that o_{kj} is defined as $g^{\sum_{a \in Q} M_{k,a} s_j^{(a)}}$ with $s_j^{(a)}$ having been chosen by participant P_a, $a \in Q$. The properties of the t-resilient matrix imply that if v is the vector with containing all $s_j^{(a)}$, the output $y = Mv$ is uniformly random in $\mathbb{Z}_q^{\ell'}$ and independent from any set of t coordinates of v (which are the ones known by the adversary). Therefore, conditioned to some of the coordinates of this output y being revealed, the rest of the coordinates of y are still uniformly random in the view of the adversary. This translates of course to the independence of the unopened o_{kj}.[1]

As for unbiasability and uniformity of the random output, notice that GULL differs from ALBATROSS at a point where the output is already determined, and hence it inherits those properties from ALBATROSS.

V. Constructing Mt. Random

In this section, 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 in 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.

In this work, we use the DDH assumption (in the random oracle model) to prove the security of all of Mt. Random's building blocks, i.e. PVSS, DKG, TVRF and VRF. The goal is to obtain a final construction whose security can be analysed 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.

GULL: PVSS beacon with gradual release

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 $sk_i \in \mathbb{Z}_q$, and public key $pk_i = g^{sk_i}$. A t-resilient matrix $M \in \mathbb{Z}_q^{\ell' \times (n-t)}$ which we can take by setting its elements to $M_{ij} = \omega^{ij}$ for some $\omega \in \mathbb{Z}_q$ of order at least $\max\{n - t, \ell'\}$.

Setup from DKG: We assume that parties have established a global threshold public key tpk, partial threshold keys tpk_i and partial threshold secret keys tsk_i for threshold El Gamal.

Protocol:

1) Round 1 - (Sharing) Each party P_a shares a random secret $(g^{s_0^{(a)}}, \dots, g^{s_{\ell-1}^{(a)}}) \in \mathbb{G}^{\ell}$ with the sharing phase of the PVSS.
2) Round 2:
   a) (Verification) Every party executes the sharing verification phase on every shared secret. Since verification is public, this fixes a set Q of the first $n - t$ parties $P_a, a \in Q$ who have correctly shared.
   b) (Aggregation) Every party can compute
   $$R_{ik} = \prod_{a \in Q} (\hat{S}_i^{(a)})^{M_{k,a}}$$
   for every $i \in [n]$ and every $k \in [1, \ell']$. Additionally each P_i computes $S_{ik} = R_{ik}^{sk_i^{-1}}$ for every $k \in [1, \ell']$.
   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 non-interactive proof π_{EG} for the language
   $$\{((g, pk_i, R_{ik}, \mathsf{tpk}, c_{ik}, d_{ik}), (sk_i, r_{ik}, S_{ik})) :$$
   $$g^{sk_i} = pk_i, g^{r_{ik}} = c_{ik}, d_{ik} = \mathsf{tpk}^{r_{ik}} \cdot S_{ik}, S_{ik}^{sk_i} = R_{ik}\}$$
   which we detail in Appendix D.
3) (Lagrange computation) After round 2 is finished, let I be the set of the first $t + \ell$ parties who have posted correct proofs for every k. For every $k \in [\ell']$ and every $j \in [0, \ell - 1]$, parties compute:
   $$O_{k,j} = \left(\prod_{i \in I} (c'_{ik})^{L_{i,I}(-j)}, \prod_{i \in I} (c_{ik})^{L_{i,I}(-j)}\right).$$
4) (Opening) At any point after round 2 is finished, to open batch k' where $k' \in [\ell']$, parties threshold-decrypt $O_{k'j}$ for every $j \in [0, \ell - 1]$ to obtain output $(o_{k'0}, \dots, o_{k'(\ell-1)})$.

Fig. 8: GULL: PVSS beacon with gradual release.

We present the general structure of Mt. Random in Figure 9. In the remainder of this section, we discuss the building blocks used for each of Mt. Random's tiers and provide a security analysis of the full multi-tiered beacon.

A. Tier 1: Uniform Randomness via PVSS

The first tier of Mt. Random outputs true uniform randomness. It is important that this tier outputs uniformly random values because these outputs will be used as high min-entropy seeds for the next tier. In our construction we will instantiate this tier with GULL (Figure 8) using threshold encryption keys generated by our new DKG protocol (Figure 6). Being based on this protocol, this tier will arguably have the highest execution time and communication, outputting uniformly random

[1] We remark that the randomness r_{ik} chosen by party P_i in the El Gamal encryption of her shares must be independent for different values of k, as otherwise the adversary could obtain information about o_{kj} from their encryptions O_{kj} and the opened $o_{k'j}$.

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 [12] protocol, the full batch of uniformly random 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 (Figure 8), 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. Nevertheless, previously revealed but unused outputs can still be used as seeds for Tier 2.

B. Tier 2: Pseudorandomness via Threshold VRFs

The second tier of Mt. Random 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). In our construction, Tier 2 is instantiated with a DDH based version of the DRAND/Dfinity TVRF proposed in [25] coupled with our new DKG protocol (Figure 6). As discussed before, 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 [25]) 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 6) 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, as pointed out in Remark 1, our DKG protocol can be used to refresh secret key shares if parties are compromised.

C. 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 [18]). In fact, 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, which seeds each of its execution with the output of its last execution, we seed this protocol with an output from Tier 2. This crucial difference has the advantage of reducing the potential adversarial bias in Tier 3 outputs.

1) Combining Bounded Biased Randomness and Uniform Randomness: Apart from outputting bounded biased randomness, Tier 3 can also be used in conjunction with Tier 1 outputs and an extractor in order to obtain correlated but uniform randomness. Basically, an uniformly random output from Tier 1 can be used as a seed for an extractor that takes as input a sequence of outputs from Tier 3, outputting correlated (due to the use of the same seed) but uniform randomness.

D. Seeding Upper Tiers vs. Unpredictable Randomness

An important aspect of Mt. Random is that each lower tier is used to seed the next upper tier, i.e. Tier 1 seeds Tier 2, which in turn seeds Tier 3. When randomness from Tier 1 or 2 is requested to be used as a seed in the next tier, it is not necessary wait for a fresh random value to be produced. For this reason, Tiers 1 and 2 respectively keep lists AlbUn and TVRFUn of random outputs that have been obtained in the past but that have not yet been used as a seed by the next layer. However, many applications (e.g. a lottery and committee selection) require unpredictable random values that are not known in advance. In this case, a fresh unpredictable output can be obtained from Tier 1 or 2 as follows:

- Tier 1: A fresh unpredictable uniformly random output can be obtained from Tier 1 by executing Step 2 of the output request procedure, which decrypts an unused block of threshold encrypted outputs from AlbUnEnc and returns the first output from the freshly decrypted block.
- Tier 2: A fresh unpredictable pseudorandom output can be obtained by waiting for the output of the next round of the beacon executed by Tier 2.

E. Security Analysis

In order to analyse the security of Mt. Random, we first argue about the initialization phase and then focus on the security guarantees offered by each layer. Notice that in the initialization phase we execute our DKG protocol (Figure 6) 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 P_i is guaranteed to have obtained its secret share $\mathsf{tsk}_i; \mathsf{tsk}'_i$ as well as the same view of the public keys. This fact will be important when arguing about the security of Tiers 1 and 2, where these keys will be used for threshold encryption and TVRFs, respectively.

In Tier 1, we only execute GULL from Figure 8 using keys $\mathsf{tpk}; \mathsf{tpk}_i; \mathsf{tsk}_i$, which gives us two main guarantees as discussed in Section IV: 1) Executing up to Step 3 results in ℓ' output blocks that are guaranteed not only to be recoverable by a majority of the parties but also to remain secret until decryption is executed in Step 4; 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

Mt. Random: Multi-tiered Randomness Beacon

Parameters:

- n participants P_i, $i \in [n]$.
- Integer $\ell \geq 1$ (number of secrets in GULL output block).
- Integer corruption threshold $1 \leq t < (n - \ell)/2$.
- Integer $\ell' = n - 2t$ (number of blocks outputted by one round of GULL).
- Integers ℓ_{TVRF} and ℓ_{VRF} denoting the bitlength of outputs from Tier 2 and Tier 3 respectively.
- Integer $TVRF_{max} \geq 0$ (number of times the TVRF-based beacon at Tier 2 is applied iteratively starting from a given seed). If it is 0 then we are not using this tier
- Integer $VRF_{max} \geq 0$ (number of times the TVRF-based beacon at Tier 3 is applied iteratively starting from a given seed). If it is 0 then we are not using this tier.

Setup: An authenticated public bulletin board (BB), $e \in \mathbb{Z}_q$, and DDH-hard group $\mathbb{G}$ with generator g. Every party in the system has a private key $sk_i \in \mathbb{Z}_q$ and a public key $pk_i = g^{sk_i}$ (registered in BB) for Tier 1. A t-resilient matrix $M \in \mathbb{Z}_q^{\ell' \times (n-t)}$ given by $M_{ij} = \omega^{ij}$ for some $\omega \in \mathbb{Z}_q$ of order at least $\max\{n - t, \ell'\}$.

Initialization: All parties P_i keep initally empty Tables AlbUn, AlbUnEnc and TVRFUn. the first two tables will store unused GULL outputs from Tier 1: AlbUn stores plain outputs and AlbUnEnc stores outputs encrypted under threshold-El Gamal. Table TVRFUn stores outputs from Tier 2. 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 $\pi_{DDH-DKG}$ (Figure 6) to obtain keys for Tiers 1 and 2 (see Remark 2). The public outputs are global threshold public keys tpk, tpk' and partial threshold public keys tpk_i, 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 Distributed Key Generation phase, all parties execute GULL from Figure 8 until Step 3. 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 AlbUnEnc. When an output is requested:

1) If AlbUn is not empty, return the next output $o_{kj} \in$ AlbUn and remove o_{kj} from AlbUn.
2) If AlbUn is empty and AlbUnEnc is not empty, all parties decrypt the next $B_k \in$ AlbUnEnc, store the resulting values $o_{k1}, o_{k2}, \ldots, o_{k\ell}$ in AlbUn and remove B_k from AlbUnEnc. Return the next $o_{kj} \in$ AlbUn and remove o_{kj} from AlbUn.
3) If AlbUn and AlbUnEnc are empty, return $\perp$ and execute GULL until Step 3 to refill AlbUnEnc.

Tier 2: Parties request an output o_{kj} from Tier 1 (repeating the request until $o_{kj} \neq \perp$) and execute the protocol in Figure 5 using tpk', tpk'_i, tsk'_i with initial seed $\sigma_0 = o_{kj}$. In each round $r \in \{1, \ldots, TVRF_{max}\}$, a value $z_r \in \{0,1\}^{\ell_{TVRF}}$ is outputted by the protocol and stored in table TVRFUn. When an output is requested, if TVRFUn is not empty, return the next $z_r \in$ TVRFUn and remove z_r from TVRFUn, else, return $\perp$. When $r = TVRF_{max}$, reset r to 0 and re-start Tier 2.

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 4 using z_r as initial seed. In each round $r' \in \{1, \ldots, VRF_{max}\}$, the output $w'_r \in \{0,1\}^{\ell_{VRF}}$ is the output of the beacon. When $r' = VRF_{max}$, r is reset to 0 and Tier 3 is started again.

Fig. 9: Mt. Random: Multi-tiered Randomness Beacon.

Fig. 10: Comparison of time for carrying out each Tier with fixed $t = \lfloor \frac{n}{3} \rfloor$, $\ell = 1$

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 case more encrypted output blocks must be generated). In case fresh unpredictable randomness is required, we remark that it can be obtained by executing step 2 of Tier 1's output request procedure, which decrypts the next unused encrypted output block and returns the first freshly decrypted output value.

In Tier 2, we execute the TVRF-based beacon protocol from Figure 5, which is proven to output pseudorandom values in [25]. Since we periodically re-seed this protocol with uniformly random values from Tier 1, its outputs are guaranteed to be pseudorandom even after long execution times. Notice that we can re-seed Tier 2 with outputs from Tier 1 that are already revealed but still not used as a Tier 2 seed. By the security of the TVRF scheme used in Tier 2 (proven in [25]), an adversary who controls less than the required threshold of 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 again the TVRF security properties hold since we use unbiased threshold keys tpk', tpk'_i, tsk'_i.

In Tier 3, we execute the protocol from Figure 4, which is proven to output bounded biased values in [18] 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. Once again, using outputs from Tier 2 that are already known but still not used as a seed in Tier 3 preserves the security of the scheme, since even by knowing the seed in advance the adversary can only bias the output of this tier by a bounded amount (as proven in [18]).

Fig. 11: Comparison of communication size for carrying out each Tier with fixed $t = \lfloor \frac{n}{3} \rfloor$, c, $\ell = 1$

Fig. 12: Amortized cost of a single random element generated at Tier 1 with fixed $n = 25$, $t = 8$. For given ℓ, number of output random elements is 9ℓ

VI. Efficiency Analysis

We provide a reference implementation for each one of the tiers.[2] Our main goal is to demonstrate the trade-off in efficiency between the three tiers. We also highlight the sensitivity of the different random beacons 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). Our experiments do not include network latency or delay. The reason is simple: Network latency is larger than our computation times and therefore will mask them. Since the number of rounds of Tier 1 is larger than the number of rounds 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 because for all tiers the communication bandwidth is small enough for network to not be a bottleneck. All our measurements were done using a benchmark tool and are averaged over many runs.

Computation time and communication size: In Figure 10 we compare the computation time for a single run of each tier as a function of the number of parties n. As can be seen from the figure, Tier 1 is the slowest, Tier 3 is the fastest and Tier 2 is in the middle. This is coherent with how we suggest to hierarchically compose the different tiers in the paper. Figure 11 shows the communication size of the three tiers, for various number of parties 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 appendix E 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 over and over again throughout the life time of the system.

Fig. 13: Average total running time of Tier 1 for various threshold t with fixed $n = 25$, $\ell = 1$

Tier 1 and Tier 2 sensitivity: We measured Tier 1 without gradual release (Albatross), that is, all random values are released at once. In Figure 12 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 Figures 13 and 14 we fix the number of parties and change the threshold t and number of culprits c, respectively. As can be viewed from

[2] All our code is open sourced and provided here: https://github.com/ZenGo-X/random-beacon

Fig. 14: Average total running time of Tier 1 for various number of culprits c with fixed $n = 25$, $t = 8$

Fig. 15: Average total running time of Tier 2 for various threshold t with fixed $n = 25$

the figures both parameters impact the total running time in a meaningful way. Increasing threshold t decreases running time as it decreases number of output random elements and decreases number of messages every party needs to process. Finally, for Tier 2, we conducted an experiment, Figure 15, for fixed number of parties n and various threshold t. Observe that as expected, the computation time is linear in the number of parties.

Acknowledgements

Research leading to these results has been partially funded by a research grant from Nomadic Labs and the Tezos Foundation, by the Spanish Government under grant number PID2019-110873RJ-I00 (SecuRing), by the Concordium Foundation and by the Independent Research Fund Denmark with grants number 9040-00399B (TrA^{2}C) and number 9131-00075B (PUMA).

The authors thank Diego Aranha for providing experimental data on the performance of Paillier cryptosystem operations and group operations over a DDH-hard group.

References

[1] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Fair two-party computations via bitcoin deposits. In R. Böhme, M. Brenner, T. Moore, and M. Smith, editors, FC 2014 Workshops, volume 8438 of LNCS, pages 105–121. Springer, Heidelberg, Mar. 2014.

[2] M. Andrychowicz, S. Dziembowski, D. Malinowski, and L. Mazurek. Secure multiparty computations on bitcoin. In 2014 IEEE Symposium on Security and Privacy, pages 443–458. IEEE Computer Society Press, May 2014.

[3] 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.

[4] C. Baum, B. David, and R. Dowsley. Insured MPC: Efficient secure computation with financial penalties. In J. Bonneau and N. Heninger, editors, FC 2020, volume 12059 of LNCS, pages 404–420. Springer, Heidelberg, Feb. 2020.

[5] C. Baum, B. David, R. Dowsley, J. B. Nielsen, and S. 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.

[6] C. Baum, B. David, R. Dowsley, J. B. Nielsen, and S. Oechsner. Tardis: A foundation of time-lock puzzles in uc. to appear at EUROCRYPT 2021, 2020. https://eprint.iacr.org/2020/537.

[7] I. Bentov and R. Kumaresan. How to use bitcoin to design fair protocols. In J. A. Garay and R. Gennaro, editors, CRYPTO 2014, Part II, volume 8617 of LNCS, pages 421–439. Springer, Heidelberg, Aug. 2014.

[8] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In H. Shacham and A. Boldyreva, editors, CRYPTO 2018, Part I, volume 10991 of LNCS, pages 757–788. Springer, Heidelberg, Aug. 2018.

[9] D. Boneh and M. Naor. Timed commitments. In M. Bellare, editor, CRYPTO 2000, volume 1880 of LNCS, pages 236–254. Springer, Heidelberg, Aug. 2000.

[10] R. Canetti. Security and composition of multiparty cryptographic protocols. Journal of Cryptology, 13(1):143–202, Jan. 2000.

[11] I. Cascudo and B. David. SCRAPE: Scalable randomness attested by public entities. In D. Gollmann, A. Miyaji, and H. Kikuchi, editors, ACNS 17, volume 10355 of LNCS, pages 537–556. Springer, Heidelberg, July 2017.

[12] I. Cascudo and B. David. ALBATROSS: Publicly AttestabLe BATched Randomness based On Secret Sharing. In S. Moriai and H. Wang, editors, ASIACRYPT 2020, Part III, volume 12493 of LNCS, pages 311–341. Springer, Heidelberg, Dec. 2020.

[13] D. Chaum and T. P. Pedersen. Wallet databases with observers. In E. F. Brickell, editor, CRYPTO'92, volume 740 of LNCS, pages 89–105. Springer, Heidelberg, Aug. 1993.

[14] J. Chen and S. Micali. Algorand: A secure and efficient distributed ledger. Theor. Comput. Sci., 777:155–183, 2019.

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

[16] V. Cortier, D. Galindo, S. Glondu, and M. 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.

[17] P. Daian, R. Pass, and E. Shi. Snow white: Provably secure proofs of stake. Cryptology ePrint Archive, Report 2016/919, 2016. https://eprint.iacr.org/2016/919, To Appear in the Proceedings of Financial Crypto 2019.

[18] B. David, P. Gazi, A. Kiayias, and A. Russell. Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain. In J. B. Nielsen and V. Rijmen, editors, EUROCRYPT 2018, Part II, volume 10821 of LNCS, pages 66–98. Springer, Heidelberg, Apr. / May 2018.

[19] B. David, B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi. Gearbox: An efficient uc sharded ledger leveraging the safety-liveness dichotomy. Cryptology ePrint Archive, Report 2021/211, 2021. https://eprint.iacr.org/2021/211.

[20] L. De Feo, S. Masson, C. Petit, and A. Sanso. Verifiable delay functions from supersingular isogenies and pairings. In S. D. Galbraith and S. Moriai, editors, *ASIACRYPT 2019, Part I*, volume 11921 of *LNCS*, pages 248–277. Springer, Heidelberg, Dec. 2019.

[21] Y. Desmedt and Y. Frankel. Threshold cryptosystems. In G. Brassard, editor, *CRYPTO'89*, volume 435 of *LNCS*, pages 307–315. Springer, Heidelberg, Aug. 1990.

[22] N. Ephraim, C. Freitag, I. Komargodski, and R. Pass. Non-malleable time-lock puzzles and applications. Cryptology ePrint Archive, Report 2020/779, 2020. https://eprint.iacr.org/2020/779.

[23] P.-A. Fouque and J. Stern. One round threshold discrete-log key generation without private channels. In K. Kim, editor, *PKC 2001*, volume 1992 of *LNCS*, pages 300–316. Springer, Heidelberg, Feb. 2001.

[24] D. Galindo, J. Liu, M. Ordean, and J.-M. Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. Cryptology ePrint Archive, Report 2020/096, 2020. https://eprint.iacr.org/2020/096.

[25] D. Galindo, J. Liu, M. Ordean, and J.-M. Wong. Fully distributed verifiable random functions and their application to decentralised random beacons. Cryptology ePrint Archive, Report 2020/096, 2020. https://eprint.iacr.org/2020/096.

[26] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin. Secure distributed key generation for discrete-log based cryptosystems. In J. Stern, editor, *EUROCRYPT'99*, volume 1592 of *LNCS*, pages 295–310. Springer, Heidelberg, May 1999.

[27] J. Groth. Non-interactive distributed key generation and key resharing. 2021. https://eprint.iacr.org/2021/339.

[28] K. Gurkan, P. Jovanovic, M. Maller, S. Meiklejohn, G. Stern, and A. Tomescu. Aggregatable distributed key generation. 2021. https://eprint.iacr.org/2021/005.

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

[30] J. Katz, J. Loss, and J. Xu. On the security of time-lock puzzles and timed commitments. In R. Pass and K. Pietrzak, editors, *TCC 2020, Part III*, volume 12552 of *LNCS*, pages 390–413. Springer, Heidelberg, Nov. 2020.

[31] A. Kiayias, A. Russell, B. David, and R. Oliynykov. Ouroboros: A provably secure proof-of-stake blockchain protocol. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part I*, volume 10401 of *LNCS*, pages 357–388. Springer, Heidelberg, Aug. 2017.

[32] R. Kumaresan and I. Bentov. How to use bitcoin to incentivize correct computations. In G.-J. Ahn, M. Yung, and N. Li, editors, *ACM CCS 2014*, pages 30–41. ACM Press, Nov. 2014.

[33] S. Micali, M. O. Rabin, and S. P. Vadhan. Verifiable random functions. In *40th FOCS*, pages 120–130. IEEE Computer Society Press, Oct. 1999.

[34] T. P. Pedersen. A threshold cryptosystem without a trusted party (extended abstract) (rump session). In D. W. Davies, editor, *EUROCRYPT'91*, volume 547 of *LNCS*, pages 522–526. Springer, Heidelberg, Apr. 1991.

[35] K. Pietrzak. Simple verifiable delay functions. In A. Blum, editor, *ITCS 2019*, volume 124, pages 60:1–60:15. LIPIcs, Jan. 2019.

[36] randao.org. RANDAO: Verifiable random number generation, 2017. https://www.randao.org/whitepaper/Randao_v0.85_en.pdf accessed on 20/02/2020.

[37] R. L. Rivest, A. Shamir, and D. A. Wagner. Time-lock puzzles and timed-release crypto, 1996.

[38] P. Schindler, A. Judmayer, M. Hittmeir, N. Stifter, and E. R. Weippl. Randrunner: Distributed randomness from trapdoor vdfs with strong uniqueness. In *28th Annual Network and Distributed System Security Symposium, NDSS 2021, virtually, February 21-25, 2021*. The Internet Society, 2021.

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

[40] B. Schoenmakers. A simple publicly verifiable secret sharing scheme and its application to electronic. In M. J. Wiener, editor, *CRYPTO'99*, volume 1666 of *LNCS*, pages 148–164. Springer, Heidelberg, Aug. 1999.

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

[42] D. team. DRAND project website, 2020. https://drand.love accessed on 21/03/2021.

[43] G. Wang, Z. J. Shi, M. Nixon, and S. 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.

[44] B. Wesolowski. Efficient verifiable delay functions. In Y. Ishai and V. Rijmen, editors, *EUROCRYPT 2019, Part III*, volume 11478 of *LNCS*, pages 379–407. Springer, Heidelberg, May 2019.

[45] M. Zamani, M. Movahedi, and M. Raykova. RapidChain: Scaling blockchain via full sharding. In D. Lie, M. Mannan, M. Backes, and X. Wang, editors, *ACM CCS 2018*, pages 931–948. ACM Press, Oct. 2018.

Appendix

A. Verifiable Random Functions: Definition and Construction

A VRF scheme $(\mathsf{KeyGen}(1\ ), \mathsf{Eval}(sk, x), \mathsf{Verify}(pk, x, y,\ ))$ with unpredictability under malicious key generation is secure if it holds that:

- (complete provability): for every (pk, sk) generated by KeyGen, and every x, then if $(y,\ ) = \mathsf{Eval}(sk, x)$, we have that $\mathsf{Verify}(pk, x, y,\ ) = 1$ with overwhelming probability;
- (unique provability): for every x, for any $y_1 \neq y_2$, and any proofs $\ _1,\ _2$, then at least one of $\mathsf{Verify}(pk, x, y_1,\ _1)$ or $\mathsf{Verify}(pk, x, y_2,\ _2)$ output 0 with overwhelming probability.
- (pseudorandomness): no PPT adversary can distinguish between $\mathsf{Eval}(sk, x)$ and a uniformly random string, even when having chosen x, after seeing pk.
- (unpredictability under malicious key generation) no PPT adversary who generated (pk, sk) arbitrarily can distinguish between $\mathsf{Eval}(sk, x)$ and a uniformly random string for an unknown uniformly random x.

We describe in Figure 16 the VRF with unpredictability under malicious key generation from [18].

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:

- $\mathsf{KeyGen}(1\ )$ chooses a uniformly random $sk \in \mathbb{Z}_q$, sets $pk = g^{sk}$ and outputs (pk, sk)
- $\mathsf{Eval}(sk, x)$ sets $y = H\ (x, u)$ where $u = H'(x)^{sk}$. It moreover defines $\ = (u,\ _{DLEQ}((g, H'(x)), (pk, u)))$, the latter being the proof that $g^k = pk$ and $H'(x)^k = u$ for a common k, in this case $k = sk$. It outputs $(y,\ )$.
- $\mathsf{Verify}(pk, x, y,\ )$ parses $\ = (u,\ ')$, checks that $\ '$ is a correct DLEQ proof for $(g, H'(x)), (pk, u))$ and checks $y = H\ (x, u)$. It accepts if all these checks pass.

Fig. 16: VRF with unpredictability under malicious key generation [18].

B. Threshold Verifiable Random Functions: Definition and Construction

A Threshold Verifiable Random Function (TVRF) has the following properties:

- Consistency: Given any x, when we apply Combine to any $t+1$ correct partial evaluations $(m_i)_{i \in A}$, we obtain the same y.
- Robustness: If Combine outputs a pair $(y; \ )$, then Verify$(tpk; x; y; \ ) = 1$
- Uniqueness: for every x, for any $y_1 \neq y_2$, and any proofs $_1; \ _2$, then at least one of Verify$(tpk; x; y_1; \ _1)$ or Verify$(tpk; x; y_2; \ _2)$ output 0 with overwhelming probability.
- Pseudorandomness: roughly, the adversary corrupting t parties cannot distinguish the output of the function from a uniformly random value, even when chosing the input.

We describe in Figure 17 a DDH-based threshold VRF inspired by a threshold Boneh-Lynn-Shacham (BLS) signatures from in [25]. 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 [25] 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 DistKeyGen explicit, as we both introduced our own scheme in Section III and discuss a number of alternatives in Appendix E.

DDH-based threshold VRF (DDH-DVRF in [25])

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 $H' : \{0,1\} \to Z_q$ for the DLEQ proof.

Commands:

- DistKeyGen$(1 \ )$ The distributed key generation creates $tsk_i \in Z_q$ such that $(tsk_i)_{i=1}^n$ is a valid Shamir sharing of some secret $tsk \in Z_q$. It outputs public $tpk_i = g^{tsk_i}$ and $tpk = g^{tsk}$, and privately tsk_i only to party P_i, for $i \in [n]$.
- PartialEval$(x; tsk_i; tpk_i)$: y_i is computed by P_i as $y_i = H(x)^{tsk_i}$. In addition compute $_i = \ _{DLEQ} \ ((g; H(x)); (tpk_i; y_i))$.
- Combine$(pk; \{tpk_i\}; x; A; (y_i; \ _i)_{i \in A})$: A subset $A^0 \ A$ is selected such that A^0 has cardinality $t+1$ and $_i$ is accepted for $i \in A^0$. Then $y = \prod_{i \in A^0} y_i^{L_{i;A^0}(0)}$ and $= (y_i; \ _i)_{i \in A^0}$
- Verify$(tpk; x; y; \ )$: Parse $= (y_i; \ _i)_{i \in A^0}$, verify all $_i$ for $i \in A^0$, and check whether $y = \prod_{i \in A^0} y_i^{L_{i;A^0}(0)}$. Output 1 if all checks pass, otherwise output 0.

Fig. 17: DDH-based threshold VRF (DDH-DVRF in [25]).

C. Threshold Encryption: Definition and Construction

A threshold encryption scheme is composed by the following algorithms:

- DistKeyGen$(1 \ )$: outputs secret keys $tsk_i; i \in [n]$, corresponding public partial keys tpk_i and a global public key tpk.
- Enc$(tpk; m)$ takes as input the global public key and a message m, and outputs a cyphertext E
- LocalDec$(tpk_i; tsk_i; E)$ takes a cyphertext E and a partial key pair (tpk_i, tsk_i), and outputs a partial decrypted message x_i.
- GlobalDec$(tpk; I; \{tpk_i\}_{i \in I}; \{x_i\}_{i \in I}; E)$ takes as input a set $I \ [n]$ with $|I| \ t+1$, the global public key, the partial public keys of I, the cyphertext E and the partial decrypted messages x_i and outputs a decrypted message m^0 or an error $\perp$.

We describe informally the properties we want from a threshold encryption scheme, following the work of [16], which we refer to for formal definitions.

- 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.

The threshold version of El Gamal is then as in Figure 18

Threshold El Gamal encryption scheme.

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

Commands:

- DistKeyGen$(1 \ )$: The distributed key generation creates $tsk_i \in Z_q$ such that $(tsk_i)_{i=1}^n$ is a valid Shamir sharing of some secret $tsk \in Z_q$. It outputs public $tpk_i = g^{tsk_i}$ and $tpk = g^{tsk}$, and privately tsk_i only to party P_i, for $i \in [n]$.
- Enc$(tpk; m)$: To encrypt a message $m \in G$, sample r uniformly at random in Z_q, and output $E = (g^r; tpk^r \ m) := (c; d) \in G^2$
- LocalDec$(tpk_i; tsk_i; E)$ outputs $x_i = (y_i; \ _i)$ where $y_i = c^{tsk_i}$ and $_i = \ _{DLEQ} \ ((g; c); (tpk_i; y_i))$.
- GlobalDec$(tpk; I; \{tpk_i\}_{i \in I}; \{x_i\}_{i \in I}; c)$ outputs $\perp$ if no more than t DLEQ proofs $_i; i \in I$ pass. Otherwise, it takes a subset $I^0 \ I$ of cardinality exactly $t+1$ such that $_{i \in I^0}$ are all correct, and computes

$$m^0 = d \ (\prod_{i \in I^0} y_i^{L_{i;I^0}(0)})$$

Fig. 18: Threshold El Gamal encryption scheme

D. Zero-knowledge proof $_{EG}$

In this section we provide a zero-knowledge proof for the EG relation that we need in the GULL construction in Section IV, which is a discrete logarithm equality type of relation, except that one of the elements that would be public in the DLEQ relation now is encrypted by El Gamal (threshold) encryption. In order to alleviate the notation, the relation and its elements will be denoted as follows for the rest of the section:

$$f((g_1; x_1; x_2; t; c; d); (s; r; g_2)) \; 2 \; G^6 \quad (Z_q^2 \quad G) :$$
$$g_1^s = x_1; \; g_1^r = c; \; d = t^r \quad g_2; \; g_2^s = x_2 g$$

The problem here is that g_2 is part of the witness, and should not be revealed. The third and fourth equalities can be combined by raising the third to s and substituting $g_2^s = x_2$ in, but this results in an equation $d^s \quad t \quad ^{rs} = x_2$ with a product rs in the exponent. This is now solved by linearization, namely consider $w = \quad rs$ as a new variable and, using one of the first two equations, for example the second, introduce a new one that guarantees that w is of the right form.

More concretely, the prover will show knowledge of exponents $r; s; w$ with:

$$g_1^r = c$$
$$g_1^s = x_1$$
$$d^s \quad t^w = x_2$$
$$c^s \quad g_1^w = 1$$

This can be proved by a standard -protocol, as we will see. If the prover is honest then $w = \quad rs$ will satisfy the equations. On the other hand, knowledge of $(r; s; w)$ satisfying these equations implies knowledge of $(r; s; g_2)$ satisfying the relation, so the only way of a cheater prove to succeed convincing the verifier of a false statement is by breaking the soundness of the protocol for this system of equations, which will happen with negligible probability. Zero-knowledge is quite trivial. We formally state and prove security of the protocol now.

Protocol $_{EG}$

Setup: A random oracle H

1) The prover chooses $u_r; u_s; u_w \quad 2 \quad Z_q$ unifomly at random, and constructs $a_1 = g_1^{u_r}; a_2 = g_1^{u_s}; a_3 = d^{u_3} \quad t^{u_w}; a_4 = c^{u_s} \quad g_1^{u_w}$. She creates $e = H(g_1; x_1; x_2; t; c; d; a_1; a_2; a_3; a_4)$. She computes $z_r = u_r + e \quad r$, $z_s = u_s + e \quad s$, $z_w = u_w \quad e \quad r \quad s$. The proof is $(e; z_r; z_s; z_w)$
2) The verifier computes $a_1 \quad c^e = g_1^{z_r}; a_2 \quad x_1^e = g_1^{z_s}; a_3 \quad x_2^e = d^{z_s} \quad t^{z_w}; a_4 = c^{z_s} \quad g_1^{z_w}$ and accepts if $e = H(g_1; x_1; x_2; t; c; d; a_1; a_2; a_3; a_4)$, otherwise rejects.

Fig. 19: Protocol $_{EG}$

Proposition 1. Protocol $_{EG}$ is a correct proof of knowledge of $(s; r; g_2)$ 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 $w = \quad rs$ implies $d^s \quad t^w = x_2$, $c^s \quad g_1^w = 1$ 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 $e \; 6 \; e^0$ with $z_r; z_s; z_w$ and respectively $z_r^0; z_s^0; z_w^0$. This means that the 4 checks by the verifier pass in both cases. From here it is easy to see that $c^{e \quad e^0} = g_1^{z_r \quad z_r^0}$ and $x_1^{e \quad e^0} = g_1^{z_s \quad z_s^0}$ so one can extract

$$r = (z_r \quad z_r^0)=(e \quad e^0) \; , \; s = (z_s \quad z_s^0)=(e \quad e^0) \text{ and } g_2 = d \; t \quad ^r$$

Note that these values satisfy that $g_1^s = x_1; \; g_1^r = c; \; d = t^r \quad g_2$, 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 $1 = c^{z_s \quad z_s^0} \quad g_1^{z_w \quad z_w^0}$, which implies $1 = c^{s(e \quad e^0)} g_1^{z_w \quad z_w^0}$. Since we already knew $c = g_1^r$ for the extracted r, this means $g_1^{rs(e \quad e^0) + z_w \quad z_w^0} = 1$. Since we are in a group of prime order, so g_1 is a generator, it must hold that

$$rs(e \quad e^0) + z_w \quad z_w^0 = 0:$$

Finally from the fact that the third check passes in both instances we have $x_2^{e \quad e^0} = d^{z_s \quad z_s^0} t^{z_w \quad z_w^0}$, which, using the information deduced in the previous line and the expression for the extracted s, means

$$x_2^{e \quad e^0} = (d^s t \quad ^{rs})^{e \quad e^0}:$$

Now since $e \quad e^0 \; 6 \; 0$ and we are in a group of prime order, this means $x_2 = d^s t \quad ^{rs}$. 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 Z_q, and defines $a_1 \quad c^e = g_1^{z_r}; a_2 \quad x_1^e = g_1^{z_s}; a_3 \quad x_2^e = d^{z_s} \quad t^{z_w}; a_4 = c^{v_s} \quad g_1^{v_w}$. This generates a transcript which is indistinguishable from one of an actual protocol, as it is easy to see.

□

E. Distributed Key Generation

There are many known instantiations of the distributed key generation protocol $DistKeyGen(1 \quad)$ from Figure 17. The structure of most of these protocols is similar to the one we have presented, namely parties each secret share a random field element with Shamir's secret sharing and post some

Scheme	Comp. (Exp/Enc/Dec)	Comm. (bits)	Rounds	Bias	Assump.
Pedersen [34]	$nt + 5n + t + 1$	$(2n^2 + tn + n)k_q$	$1 + 2$	Yes	DDH
Gennaro et al. [26]	$2nt + 11n + 3t + 3$	$(4n^2 + 2tn + 2n)k_q$	$2 + 3$	No	DDH
Fouque-Stern [23]	$(nt + 5n + t + 1)$ Exp. $+4n$ Enc$+n$ Dec	$(2n^2 + tn + n)k_q$ $+2n^2k_h + 3n^2k_N$	1	Yes	DDH +DCR
Fouque-Stern [23] in terms of Exp. and k_q	$(nt + 18005n + t + 1)$ Exp.	$(28n^2 + tn + n)k_q$	1	Yes	DDH +DCR
Our Result	$9n + t + 2$	$(2n^2 + tn + 5n)k_q$	$2 + 2$	No	DDH

TABLE I: Comparison of DKG schemes when 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 in $\mathbb{G}$ (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.

related information. The global implicit secret key is the sum of the secrets dealt by a set Q of parties who have shared correctly (the partial secret keys are similarly computed by the corresponding party by summing the received shares from parties in Q), and the public information is used to derive the public partial and global keys. The differences lie on how parties can prove the correct sharing of their initial secrets, and their consistency with the public information they post.

Possibly the best known is Pedersen's protocol [34], where parties use a verifiable secret sharing scheme (VSS), namely Feldman's VSS to do this, while they post a commitment to the coefficients of the polynomial. Parties reach an agreement, via the VSS properties, on a set Q of parties that have correctly shared their value. The protocol has 1 round of interaction, and 2 additional rounds if there are disputes.

As discussed in Gennaro et al. [26], one caveat of Pedersen distributed key generation protocol is the fact that malicious parties can bias the public global key. [26] also showed a modification of the protocol that fixes this problem, using a different commitment to the coefficients of the sharing polynomial. However this introduces a new round of interaction and a new round of dispute resolution.

[23] proposed a one-round distributed key generation protocol based on Paillier cryptosystem, where parties only speak once, by posting their message in a public bulletin board. This protocol is publicly verifiable but again the public key can be biased by a rushing adversary.

Nevertheless, a recent work by Gurkan et al. [28] shows that the public key biasability from [26] should not be a problem for applications to threshold encryption, signatures and verifiable random functions, due to a property named rekeyability, introduced in that work. We also remark that in the same work [28], the authors construct a publicly verifiable distributed key generation protocol with a much improved asymptotical communication complexity $O(n)$, based on the notion of aggregation via gossip. However, this protocol is not only based on pairing assumptions (stronger than our DDH assumption), but also outputs group elements as secret keys (rather than elements in $\mathbb{Z}_q$), i.e., the output is to be used with pairing-based threshold schemes, so it cannot be used for example for its use with threshold El Gamal encryption scheme, at least directly. It would be very interesting to achieve the type of output keys we need with their gossip techniques.

Another recent work [27] introduces a non-interactive (but biasable) DKG protocol that generates keys with the same structure as ours. However, the preliminary version of [27] does not present any efficiency analysis of the proposed protocol, making it hard to present a comparison. Moreover that construction requires pairing hardness assumptions.

In Table I, we compare the amount of computation, communication, number of rounds (separated in number of fixed rounds plus number of rounds that may be required to resolve disputes), assumptions and biasability of the globel public key pk by a rushing adversary. We denote by k_q the number of bits to describe a field element in $\mathbb{Z}_q$, which we assume to also be roughly equal to the number of bits to describe an element in $\mathbb{G}$; in the case of Fouque-Stern, we denote by k_N the number of bits to describe an element in $\mathbb{Z}_N$ for the use of Paillier scheme (hence $2k_N$ describes an element in $\mathbb{Z}_{N^2}$). Since Pedersen's and Gennaro et al.'s protocols involve private communication between parties, in order to properly compare the communication complexity, we have assumed that this communication is done through the public ledger using El Gamal encryption, which requires posting $2k_q$ bits and computing 2 exponentiations per encryption, while decryption costs 1 exponentiation. For the sake of comparison to Fouque-Stern we measured the time for Paillier encryption and decryption with 128-bit security, obtaining 180 milliseconds and 244 milliseconds, respectively, on a Intel(R) Core(TM) i7-10510U CPU @ 1.80GHz using the RELIC library [3]. On the same platform and security level, a group operation over a DDH-hard group takes 50 microseconds.

As one can see, 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 typically 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.

Our novel DKG protocol's performance is further showcased in our benchmarks. Figures 20 and 21 show the DKG computation time and communication size for changing number of parties n for tiers 1 and 2.