Homomorphic Encryption Random Beacon

Alisa Cherniaeva Corestar [*]   Ilia Shirobokov Corestar [†]   Omer Shlomovits ZenGo [‡]

November 14, 2019

Abstract

A reliable source of randomness is a critical element in many cryptographic systems. A public randomness beacon is a randomness source generated in a distributed manner that satisfies the following requirements: Liveness, Unpredictability, Unbiasability and Public Verifiability.

In this work we introduce HERB: a new randomness beacon protocol based on additively homomorphic encryption. We show that this protocol meets the requirements listed above and additionaly provides Guaranteed Output Delivery.

HERB has a modular structure with two replaceable modules: an homomorphic cryptosystem and a consensus algorithm.

In our analysis we instantiate HERB using ElGamal encryption and a public blockchain. We implemented a prototype using Cosmos SDK to demonstrate the simplicity and efficiency of our approach. HERB allows splitting all protocol participants into two groups that can relate in any way. This property can be used for building more complex participation and reward systems based on the HERB solution.

1 Introduction

A lot of modern applications require a reliable source of randomness. Individual parties can rely on local random generators but there are many cases when participants want to obtain a publicity verifiable randomness source that cannot be biased by a third party. This approach is especially important for decentralized systems since even a single centralized node within a system eliminates the whole idea of a decentralized approach. Distributed random generation can benefit such systems as Tor Hidden Services [SDM04]; secure generation of cryptographic protocols parameters [Ber+15]; voting protocols like Helios [Adi08]; private messaging systems like Vuvuzela [VDH+15] or Herbivore [GRPS03]; gambling companies such as FunFair [LH17], TruePlay [TP], DAO Casino [DC]; Byzantine agreement algorithms [CKS05]; etc.

Rabin [Rab83] first proposed the concept of randomness beacon that publishes unpredictable and unbiased numbers at some regular intervals. The original concept relies on a trusted third party such as NIST random beacon [NIST], Random.org [Rand] or Oraclize.it [Ora]. This approach carries drawbacks associated with centralized services, in particular,

[*]alice.chernyaeva@gmail.com
[†]shirobokov.research@gmail.com
[‡]omer@ZenGo.com

inability to verify a random number generated, risk of tampering and risk of benefit from the prior knowledge of the randomness. In this paper we aim to develop a decentralized randomness beacon with the following properties, specified in other randomness beacon research works [AMM18], [Syt+17], [SJSW18]:

- **Availability (or liveness)** - any single participant (or colluding participants) should not be able to prevent a random number generation process.
- **Unpredictability** - any single participant (or colluding participants) should not be able to obtain any information about future random beacon values.
- **Unbiasability (or bias-resistance)** - any single participant (or colluding participants) should not be able to influence future random beacon values to their advantage.
- **Public-Verifiability** - third parties should be able to verify the correctness of generated values using public information only.

Definition 1.1. *We call a randomness generation process a Random Beacon if the properties of availability, unpredictability, bias-resistance and public verifiability holds.*

Below we introduce a publicly verifiable random beacon protocol based on partially homomorphic encryption scheme. Details about our implementation of the scheme and performance metrics can be found in Section 5.

1.1 Related Work

The decentralized random number generation problem was originally raised in [Blu83] by Blum. This article covers a protocol allowing to generate a random bit via a telephone line. Lack of a bias-resistance property appeared to be the core issue of the coin tossing protocol and of more complex protocols [LMR83], [MNS09], [BOO10] built on it. Cleve [Cle86] demonstrated that for any r-round coin-tossing protocol there exists an efficient adversary that can bias the output of the honest party. This bias gets reduced as the number of rounds increases causing such protocols to be complicated communication-wise.

With decentralized systems and blockchain technology gaining popularity over the recent years, research in randomness beacons has gained interest.

A closely related result to our work is a concurrent paper by Nguyen-Van *et. al* [NV+]. This protocol uses homomorphic encryption, verifiable random functions and a public distributed ledger. The protocol is driven by leader (requester) and does not provide the liveness and bias-resistance properties without trusted third party.

A large class of considered random beacon protocols is based on the Schoenmakers' Public Verifiable Secret Sharing scheme [Sch99] and its improved modifications. This class includes number generation protocols like RandHound and RandHerd [Syt+17], Scrape [CD17], HydRand [SJSW18], as well as Caucus [AMM18] and Ouroboros [KRDO17] solutions. The approach features a high level of communication complexity and all attempts to reduce it entail drawbacks. [Syt+17] has failure probability, [SJSW18] solution provides probabilistic guarantees for unpredictability and complete unpredictability is only achievable for random data generated after a specific number of rounds. For some use cases this approach represents a considerable issue.

Dfinity protocol [HMW18] is based on unique signatures [GO92]. It uses the threshold version of the BLS signature [BLS01]. Unique signatures enable computation and communication complexities that fully comply with requirements to randomness beacon practical use. Yet, a new challenge arises when it comes to selecting the best initialization vector for this scheme. At a first glance, it may seem that we are back to the problem of distributed random number generation, however in terms of both security and performance this procedure may be subject to less strict requirements. The BLS signature scheme assumes elliptic curve pairings, which is an additional security assumption not necessary in our solution. Finally, Dfinity protocol requires trusted setup or distributed key generation phase.

Slow-timed hash functions defined by Lenstra and Wesolowski [LW15] represent one of the Verifiable Delay Functions (VDF) described in [BBBF18] by Boneh *et al.* The VDF concept was elaborated in papers by Wesolowski [Wes18], Pietrzak [Pie18] and Boneh , Bünz and Fisch [BBF18]. The main idea is that the adversary cannot bias the output of the random beacon because it cannot calculate the result in the given time window. Applying VDF is closely interrelated with issues posed by implementing ASICs or protection from it, as hardware solutions may considerably accelerate the VDF computation. Another challenge is designing a secure way of getting input for the VDF.

Honey Badger BFT protocol [Mil+16] is a randomized BFT asynchronous protocol with unpredictable output that can also be used as a randomness beacon. As the BLS-based protocol, Honey Badger requires trusted setup or distributed key generation. The idea to use a PoW blockchain as a random number source is covered by Bonneau *et al.* in [BCG15] and Bentov *et al.* in [BGZ16]. The main issue is that PoW blockchains cannnot provide bias-resistance property. Finally, there are approaches other than randomness beacon; these are aimed at number generation between two parties only, see Game Channels [CSD18] and Fate Channels [LH17].

More information about related solutions can be found in Section 7.

1.2 Our contribution

This paper introduces Homomorphic Encryption Randomness Beacon (HERB), a protocol geared towards the continuous provision of publicly verifiable randomness at regular intervals. We demonstrate an analysis including proofs showing that the protocol indeed meets the above requirements. Additionally, HERB protocol provides Guaranteed Output Delivery [RBO89] which is considered an important additional property for random beacon security in [CD17]. [KRDO17], [SJSW18]. We suggest using the threshold version of ElGamal cryptosystem as homomorphic encryption for HERB, but other threshold homomorphic encryption scheme can be suitable. We use ElGamal based on elliptic curve group of prime order with a message encrypted to support linear homomrphism (ElGamal in the exponant [ST04]). ElGamal is fast, well studied, and assumes only EC-DDH.

In simplified terms the random number generation process can be represented in the following way: each participant encrypts their part of the secret and publishes the ciphertext. Then, all the encrypted shares are joined. Only after being joined, the fragments are cooperatively decrypted by participants. Thus each participant contributes to the random number generation, and the result can only be derived through their joint effort.

This approach allows for obtaining a communication complexity of $\mathcal{O}(n^3)$ without using Public Blockchain and $\mathcal{O}(n^2)$ with it. The optimizations presented in this article may reduce complexity to $\mathcal{O}(n^2)$ without using Public Blockchain, but they require more elaborate research.

Each HERB participant can select their own entropy source, in particular, they can use the True Random Number Generators based on quantum physics [Ste+00].

Our protocol requires performing a distributed key generation protocol before the core generation phase.

Note that we are far from suggesting HERB as some silver bullet fighting all problems in decentralized random number generation. It is just one more approach that can be more suitable than others for some specific cases. For example, HERB fits well with smart-contract based random beacon implementation such as it has a straightforward structure, doesn't require pairings and participants have to send only two transactions with relatively small amounts of data.

This paper is organized as follows. Section 2 contains a network model description. Section 3 gives basic notations and cryptographic primitives used in the HERB protocol. Section 4 covers the HERB protocol and Distributed Key Generation protocol used as a setup phase for HERB. Section 5 contains detailed coverage of HERB's security properties. Section 6 describes HERB implementation. Section 7 compares HERB to other related protocols and concludes the paper.

2 Model

We assume a synchronous system with a fully connected network point–to–point channels between n parties (*nodes*). It is assumed that there is a malicious adversary able of controlling t system participants at most, where $n = 3t + 1$. The adversary can diverge from the specified protocol in any way.

3 Preliminaries

In this section, we introduce definitions and notation used throughout the paper.

We denote the set of nodes as $\{id_i\}_{i=1,\dots,n}$, where id_i is Identifier of i-th participant. Let $\mathbb{G}$ be a cyclic group written in additive notation of prime order q with generator G. We use an additive notation for all groups in this article since it is assumed that further implementation of the suggested protocols is most likely to occur within elliptic curves. When writing $x \overset{\$}{\leftarrow} S$ we mean that x is chosen uniformly at random from the set S. By *hash* we denote some cryptographic hash function.

Let $\mathbb{G}$ be a large cyclic group of prime order q with generator G. The Decisional Diffie-Hellman (DDH) problem is, for the given quadruple $G, x \cdot G, y \cdot G$ and $z \cdot G$, where $x, y, z \in \mathbb{Z}_q$, to determine whether or not $z \cdot G = (xy) \cdot G$.

We assume that the DDH problem is hard in the chosen cyclic group.

3.1 Zero Knowledge Proofs for Discrete Logarithm Relations

ZK-proofs are protocols involving two parties: the Verifier and the Prover. When the protocol is executed, the Prover convinces the Verifier that they have a solution for a certain mathematical witness without disclosing any further data. The Verifier has to make sure that the Prover indeed has the necessary information. Generally, the following core properties for such proof systems are specified:

1. **Completeness**. The Prover can convince the Verifier of a true statement using a witness.
2. **Soundness**. The Prover cannot convince the Verifier of a false statement.
3. **Zero-knowledge**. If a statement is true then the proof does not reveal any useful information about the witness.

We use non-interactive zero-knowledge proofs (NIZK) for discrete logarithm relations in our protocol. We describe these proofs for groups in additive notation.

NIZK of Correct ElGamal Encryption. We use Σ protocol described in Appendix A and turn it into NIZK using Fiat-Shamir heuristic [FS86]. Proof of Correct Encryption (CE) allows to the Prover convince the Verifier that they knows the witness $\omega = (x, r)$ for the statement $\delta = (G, Q, B, A)$, where $Q = xG$, $(A, B) = (rG, xG + rQ)$.

Algorithm 1 CE(G, Q, A, B)

```
Input: $G, Q, A, B \in \mathbb{G}$; $x, r \in \mathbb{Z}_q$
Output: $\pi = (e, z_1, z_2)$ - proof that the prover knows valid $x, r$.
1. $T = s_1 \cdot G + s_2 \cdot Q$, $E = s_2 \cdot G$, $s_1, s_2 \xleftarrow{\$} \mathbb{Z}_q$
2. $e = hash(A, B, T, E)$
3. $z_1 = s_1 + x \cdot e \pmod{q}$, $z_2 = s_2 + r \cdot e \pmod{q}$
4. $\pi = (e, z_1, z_2)$
```

Everyone, who knows π, G, Q, A, B can verify the proof by reconstructing the commitments T' and E' then checking equation how it shown in **CE-Verify**. This is an algorithm that either accepts $(b = 1)$ or rejects $(b = 0)$ proof π to secret values x, r.

Algorithm 2 CE-Verify(π, Q, T, A, B)

```
Input: $\pi = (e, z_1, z_2); G, Q, A, B \in \mathbb{G}$
Output: $b \in \{1, 0\}$
1. $T' = z_1 \cdot G + z_2 \cdot Q - e \cdot B$
2. $E' = z_2 \cdot G - e \cdot A$
3. $e' = hash(A, B, T', E')$
4. If $e' \overset{?}{=} e$ then $b = 1$
   Else $b = 0$
```

We use CE proofs to enforce parties to publish only correct ciphertexts.

NIZK for Discrete Logarithm Equality. Proof for Discrete Logarithm Equality (DLEQ) allows to the Prover convince the Verifier that they knows the witness $x \in \mathbb{Z}_q$ such that $Y = xG,\ Z = xQ$ for the statement (Y, G, Z, Q). We use Σ protocol descrived by Chaum and Pedersen [CP92] and turn it into a NIZK using Fiat-Shamir heuristic [FS86].

Algorithm 3 DLEQ (Y, G, Z, Q)

```
Input: $Y, G, Z, Q \in \mathbb{G}$, $x \in \mathbb{Z}_q$
Output: $\pi = (e, z)$ - proof.

1. $A_1 = w \cdot G,\ A_2 = w \cdot Q$ for $w \xleftarrow{\$} \mathbb{Z}_q$
2. $e = hash(Y, Z, A_1, A_2)$
3. $z = w - x \cdot e \pmod q$
4. $\pi = (e, z)$
```

Everyone, who knows π, Y, G, Z, Q can verify the proof by reconstructing the commitments A'_1 and A'_2 then checking equation how it shown in **DLEQ-Verify**. This is algorithm that either accepts $(b = 1)$ or rejects $(b = 0)$ proof π to secret value x.

Algorithm 4 DLEQ-Verify(π, Y, G, Z, Q)

```
Input: $\pi = (e, z); Y, G, Z, Q \in \mathbb{G}$
Output: $b \in \{1, 0\}$

1. $A'_1 = z \cdot G + e \cdot Y$
2. $A'_2 = z \cdot Q + e \cdot Z$
3. $e' = hash(Y, Z, A'_1, A'_2)$
4. If $e' \overset{?}{=} e$ then $b = 1$
   Else $b = 0$
```

We use DLEQ proof for verification of partial decryption of an ElGamal ciphertext.

3.2 Homomorphic Properties of Cryptosystems

Homomorphic encryption allows us to operate with ciphertext directly without decryption. Let's assume that there is an encryption function $E(k, m)$, where k stands for the encryption key and m is a public text. E is then considered partially homomorphic with respect to some operation $*$ as long as there is an effective T algorithm that receives $E(k, m_1)$ and $E(k, m_2)$ as input and returns an encrypted ciphertext $C = T(E(k, m_1), E(k, m_2))$ that yields a public $m_1 * m_2$ message after decryption C.

The difference between partially homomorphic encryption (PHE) and fully homomorphic encryption (FHE) is that FHE schemes support arbitrary computation on ciphertexts.

In this article PHE is considered and, unless stated otherwise, any further reference to homomorphic encryption implies PHE.

3.3 ElGamal

ElGamal Encryption system is an asymmetric key encryption algorithm for public key cryptography described by ElGamal in 1985 [ElG85]. This scheme is secure if the decision Diffie-Hellman assumption holds [TY98]. In this paper, we are going to use elliptic curve ElGamal cryptosystem. Below we are going to cover the classic and threshold versions of ElGamal cryptosystem.

Let p - large prime number. The elliptic curve E defined in field $\mathbb{F}_p$, its cyclic subgroup $\mathbb{G} \in E(\mathbb{F}_p)$ has prime order q. Generator of group $\mathbb{G}$ is G.

Classic ElGamal Cryptosystem defined by the following algorithms:

- **Setup**(1^λ): Choose a private key $x \overset{\$}{\leftarrow} \mathbb{Z}_q$ and a public key $Q = xG$.
- **Encrypt**(Q, m): Encode message m to the point $M \in \mathbb{G}$ (e.g. using Koblitz's method [BCR10]). The ciphertext is $C = (rG, M + rQ)$, $r \overset{\$}{\leftarrow} \mathbb{Z}_q$.
- **Decrypt**(C, x): For the ciphertext $C = (A, B) \in E(\mathbb{F}_p) \times E(\mathbb{F}_p)$, decrypted point is $M = B - xA$. Decode point M to the message m.

For our purpose, we need a threshold ElGamal cryptosystem. Threshold cryptosystem allows any system participant to encrypt a message so that only a subset of size greater t parties could decrypt it. Note that for the **Setup** phase we need a DKG protocol. We discuss this question in 4.1 section.

(t, n)-Threshold ElGamal Cryptosystem comprises five algorithms [FP01]:

- **Setup**$(n,\ t,\ 1^\lambda)$: Key Generation Procedure, which generates the common public key PK, participants' secret keys $sk_i, i \in \{1, ..., n\}$ and verification keys $VK_i, i \in \{1, ..., n\}$. This procedure detailed described in section 4.1.
- **Encrypt**$(PK,\ m)$: Same as classic ElGamal encryption.
- **ShareDecrypt**$(PK,\ C,\ sk_i)$: For the ciphertext $C = (A, B)$ decryption share $\mu_i = (i, \mu_i', \pi_i)$, where $\mu_i' = sk_i \cdot A$, π_i - proof of correctness of decryption share.
- **ShareVerify**$(PK,\ C,\ \mu_i,\ VK_i)$: Verification of the proof π_i.
- **Decrypt**$(PK,\ C,\ \mu_1, ..., \mu_{t+1})$: $D =$ **Recover**$(\mu_1, ..., \mu_{t+1})$. Decrypted message $M = B - D$.

Function **Recover** can reconstruct the point D from a list of public shares $\mu_1, ..., \mu_{t+1}$ using Lagrange interpolation.

4 HERB Protocol

The HERB protocol is aimed at implementing a bias-resistant, publicly-verifiable and unpredictable random beacon. The protocol is based on the ElGamal encryption system covered in §3.3. If needed, it can be replaced with any other threshold homomorphic encryption scheme that has an efficient algorithm for decentralized key generation. The HERB protocol implies two participant roles: *key holders* cooperating at all protocol stages and *entropy providers* only contributing when ciphertext are published. These two sets can relate in any way.

Let's introduce the notion of a ciphertext share. *Ciphertext share* is a random number encrypted by a single entropy provider via a specific encryption system.

HERB protocol works in rounds, Each round includes the publication and the disclosure phases as described below:

1. Setup phase: All key holders together generate the common public key and shares of the private key. Setup is carried out once.
2. Publication phase: Each entropy provider publishes a ciphertext share. Upon completion, every participant can aggregate the published ciphertext shares into a common ciphertext.
3. Disclosure phase: Threshold of Key holders together decrypt the aggregated ciphertext.

Note that for convenience we use a cryptographic hash function to convert the protocol result into a fixed-length bit string.

The homomorphic property of suitable cryptosystems makes them malleable, i.e. for given $C = Enc(PK, m)$ it is feasible to compute $C' = Enc(PK, f(m))$, where f is nontrivial function and $C' \neq C$ [Wik02]. Therefore, the entropy provider that is the last to publish their ciphertext share is able to obtain the random number they need. We avoid this by using non-interactive zero-knowledge proofs. As far as the protocol herein is concerned, the homomorphic property enables creating aggregated ciphertext and decrypting it without decrypting the shares it consists of. To ensure non-malleability, the validity of the ciphertext shares and of decryption shares will be confirmed via published NIZKs.

4.1 Setup

Let p is large prime number. The elliptic curve E defined over field $\mathbb{F}_p$. Group $\mathbb{G} = E(\mathbb{F}_p)$ has prime order q. For instance, E can be represented by P-256, P-384 and P-521 curves from FIPS 186-4[KD13]. Let G be the generator of group $\mathbb{G}$. Let H be another generator of $\mathbb{G}$ with unknown discrete log. H can be derived in a deterministic way from $\mathbb{G}$ by hashing $\mathbb{G}$ and interpret as a point or in a similar way as described in [Tan05]. We assume a group of n key holders and m entropy providers. Any subset of size $t + 1$ key holders can obtain the result of the random number generation.

Key generation is an initial phase for the threshold ElGamal system. We cannot use centralized solutions, e.g. Shamir's Secret Sharing [Sha79], which will make HERB predictable and non bias-resistant by a centralized key generator. Instead we are going to use distributed key generation (DKG) protocol. It allows a set of n key holders to generate jointly a public key and shares of the secret key spread among the participants of the DKG protocol such that any subset of size greater then *threshold* value t can use distributed secret key without revealing this key.

We prefer to use the DKG protocol described by Gennaro *et al.* in [Gen+07]. We note that one can use any other DKG protocol Depending on the system security assumptions suitable of one system.

We slightly modify the DKG from [Gen+07] by adding verification key computation at the last phase of the protocol (step 4.3). Verification keys ensure that each key holder is able to prove fair acting in the course of the joint message decryption. See Appendix for our modified DKG.

4.2 Random Generation Process

In this part, we introduce our random generation algorithm. Given in protocol 5. We use the linear homomorphism property to sum ciphertext shares from $t + 1$ entropy providers. Each entropy provider is free to select their own local entropy source for their random number share. In that sense the randomness is additive.

Communication can be done using a peer-to-peer fully connected network and Byzantine Fault Tolerance consensus algorithm but we describe our solution using a bulletin board or a public ledger that can run arbitrary logic (smart contract) on public information. More information about using a consensus algorithm presented in the Future Work section.

A bulletin-board ensures tamper-proof history, integrity, availability, support of user authentication and publicly auditability. In practice, a public blockchain can be used as a bulletin board. In the case of HERB, this approach is implemented as follows: entropy providers send their ciphertext shares C_i and relevant proofs to the blockchain where the system (e.g. smart contract) records and checks them. Once the predefined conditions are met (time period or a sufficient number of ciphertexts, depending on the system settings), an aggregated ciphertext C is built from all the obtained shares. Therefore, published ciphertext shares with the relevant proofs publicly available and each participant can check them independently.

Remark. To generate a random point M, participant takes $y \overset{\$}{\leftarrow} \mathbb{Z}_q$ and calculates $y \cdot G = M$. This random point generation method was selected to enable proofs π_{CE} generation, as to do it a participant has to know the $\log_G M$ number.

5 Security

This section covers how HERB meets guaranteed output delivery property and core requirements to a decentralized randomness beacon: liveness, unpredictability, bias-resistance and public verifiability. Let's assume that at the moment of execution the random number generation goal is fixed and cannot be changed (for example, if results are to be used in a round of roulette, all bets have to be off).

Since key generation is somewhat out of the scope of the present let us assume that the key generation protocol was already executed and that the private keys of the honest protocol participants were kept private. Security of the selected DKG within the suggested model is proved in [Gen+07].

We consider adaptive adversary, that can choose which nodes to corrupt after the protocol execution begins. It is also assumed that each honest node complies with the protocol

Protocol 5 HERB

We start by running DKG between n key holders $\{id_i\}_{1\leq i\leq n}$, where it is assumed that at most t are malicious. These key holders use the DKG protocol to obtain keys for the ElGamal cryptosystem. There is a set of m entropy providers $\{e_j\}_{1\leq j\leq m}$ that are able to submit ciphertext shares.

1. Each entropy provider e_j, $1 \leq j \leq m$, generates random point $M_j \in \mathbb{G}$. Then encrypts it:

$$\mathbf{Encrypt}(Q, M_j) = (r_j G, M_j + r_j Q) = (A_j, B_j) = C_j, \text{ for } r_j \xleftarrow{\$} \mathbb{F}_q.$$

2. e_j publishes C_j along with NIZK of correct encryption (see algorithm 1):

$$C_j = (A_j, B_j),\ \pi_{CE_j} = \mathbf{CE}(G, Q, A_j, B_j)$$

3. When C_j is published, participants agree that the ciphertext share is correct, if **CE-Verify**$(\pi_{CE_j}, G, Q, A_j, B_j) = 1$ (see 2).

4. When all correct C_j are published, participants calculate $C = (A, B)$, where

$$A = \textstyle\sum_{j\in J} A_j, \quad B = \sum_{j\in J} B_j$$

   where J is subset of entropy providers which published correct C_j.
   Notice that C is ciphertext for plaintext $M = \sum_{j\in J} M_j$. Also notice that size of J depends on system parameters.

5. Key holder id_i, $1 \leq i \leq n$, with secret key x_i publishes decryption shares along with NIZK of discrete logarithm equality (see algorithm 3):

$$D_i = x_i \cdot A, \pi_{DLEQ_i} = \ \mathbf{DLEQ}(D_i, A, VK_i, G)$$

6. When D_i is published, participants verify that **DLEQ-Verify**$(\pi_{DLEQ_i}, D_i, A, VK_i, G) = 1$ (see algorithm 4).

7. When $t + 1$ decryption shares published, participants calculate M:

   7.1 $D = \sum\limits_{i\in I} \lambda_i \cdot D_i$, where $|I| = t + 1$ - set of indexes of parties, who published decryption shares and λ_i - Lagrange coefficients.

   7.2 Resulting random number is $M = B - D$

and timely publishes the required data. A malicious node is a node controlled by a polynomial time Byzantine adversary; it can diverge from the defined protocol in any way. Our model assumes that an adversary cannot take over more than t nodes within a single generation epoch (between two DKG protocol execution sessions). The adversary cannot predict the results of honest participants local PRNG.

It is assumed that all cryptographic primitives within the protocol provide their intended security properties. The ElGamal encryption scheme is secure if decision Diffie-Hellman assumption holds [TY98]. Security for multi-user setting was proved by Bellare, Boldyreva and Micali [BBM00]. Non-malleable property for the ElGamal system is achieved via zero-knowledge proofs. For simplicity, we use (t, n)-threshold ElGamal scheme.

We assume a synchronous communication model as in [Gen+07].

Our proof uses a bulletin board model with a set of entropy providers coinciding with the set of key holders. Each entropy provider can send only one ciphertext share per round.

It is also assumed that before embarking on the aggregation of the joint ciphertext C at least $t+1$ ciphertext shares have to be published. In general, an increase of the set of entropy providers compared to the set of key holders does not necessarily entail an increase in the security level or an improvement in the random number generation quality.

Proactive security. After a certain interval expressed as the number rounds executed, the protocol participants perform a new DKG procedure. Thanks to this regular key update, loss/theft of a participant secret key has a smaller impact on the overall protocol security level.

Unpredictability

Lemma 5.1. *An adversary cannot precompute future random beacon values unless with negligible probability.*

Proof Sketch. According to the security model, the number of malicious nodes cannot exceed t. Denote adversary ciphertext shares published at stage 2 as $C_1 = E(M_1), ..., C_{t+1} = E(M_{t+1})$. According to the protocol model, generation of the common ciphertext C requires at least $t+1$ ciphertext shares C_i, therefore, an adversary lacks at least one plaintext $M_{t+j}, j > 1$ to compute the algorithm outcome without decryption.

Therefore, an adversary cannot obtain M without assistance from the honest nodes. These nodes do not publish decryption shares until step 5 of the protocol. So, it can be concluded that the prediction of the algorithm outcome at the stage of common ciphertext generation is not possible.

At the stage of decryption, an adversary can compute the algorithm outcome after the first D_i is published by any of the honest nodes. After $t+1$ decryption shares are published, the outcome becomes known to all parties. This short delay at the point of decryption gives an adversary no advantage whatsoever since by this point they cannot change decisions related to the random number generation goal recorded before. □

Bias-resistance

Lemma 5.2. *Neither a separate node nor a smaller then threshold set of malicious nodes can bias (influence) the random value at the end of the protocol.*

Proof Sketch. Each new ciphertext share $C_i \neq 0$ changes the aggregated ciphertext $C = \sum C_i$, therefore the algorithm outcome M can change as well. At the same time, according to lemma 5.1, the adversary cannot learn the nature of the last changes. Thus, the adversary needs to generate a ciphertext that somehow depends on the rest of the published C_i. Given that non-malleable encryption is applied by condition, the only way the adversary can consciously impact the algorithm outcome to turn it to their advantage is by attempting to obtain plaintexts M_i of all parties. From unpredicatbility property, the adversary cannot do it at earlier protocol stages. By the time the adversary gets all the data needed to bias the output, the final ciphertext C (and, therefore, the final random number) is already generated and thus cannot be changed. Therefore, in the course of a round, neither a single node nor a set of malicious nodes can bias the algorithm outcome. □

Liveness

Lemma 5.3. To generate a common ciphertext C, we need $t+1$ nodes.

Proof Sketch. Immediate derived from the adversary model. □

Lemma 5.4. According to the security model, the algorithm completes successfully.

Proof Sketch. Assuming there are at least $t+1$ honest nodes timely publishing valid data, we get $t+1$ ciphertext shares C_i. The previous lemma demonstrates that this number is sufficient to ensure secure generation of a joint ciphertext C.

Given $t+1$ of honest nodes, at least $t+1$ decryption shares D_i are to be published; generally, these do not have to come from the same nodes that generated the common ciphertext C. The $t+1$ published decryption shares are sufficient to disclose the random M message within the given threshold scheme (t,n). Thus all the nodes involved in the generation, as well as network users having access to the published data get a random number and the algorithm completes successfully. □

Guaranteed Output Delivery

Lemma 5.5. Malicious nodes cannot prevent honest nodes from receiving the protocol's output.

Proof Sketch. Follow from proof of the lemma 5.4 and using bulletin boards. □

Public Verifiability

Lemma 5.6. A third-party verifier can verify the published random message M.

Proof Sketch. Given the use of a public verifiable bulletin board, all published C_i values are known to a third party. Also, the third party has access to the CE NIZK-proof that ensure non-malleability of the published C_i. A verifier can verify this proof using algorithm 2; therefore, they can verify the correctness of the published ciphertext shares. The common ciphertext $C = \sum C_i$ can be independently computed by a third party. Thus a verifier can make sure that the C is generated correctly.

Once $C = (A, B)$ is generated, each key holder publishes decryption shares $D_i = x_i * A$ with the ZK-proof of discrete logarithm equality into a bulletin board, where x_i represents the share of the common private key. To verify D_i a third party only needs to verify the ZK-proof by using algorithm 4. A DLEQ proof allows convincing a verifier that the D_i decryption share was obtained from A by the id_i private key.

Having $t+1$ of valid D_i, a verifier can compute $D = \sum \lambda_i \cdot D_i$, where λ_i are Lagrange coefficients. $M = B - D$ is the algorithm output. Thus, third party verifier can verify the the HERB output. ☐

Theorem 5.7. *HERB protocol is a random beacon per* ***definition 1***

Proof. Follow from lemmas 5.1, 5.2, 5.3, 5.4, 5.6, ☐

6 Implementation

We present a random beacon protocol with the following selectable parameters: key generation protocol, encryption scheme, consensus algorithm and ratio between entropy providers and key holders. A specific HERB protocol implementation is demonstrated as a proof of concept. It has the following parameters:

- **DKG**: Gennaro et al.'s protocol [Gen+07];
- **Encryption scheme**: Threshold ElGamal encryption [ElG85];
- **Consensus**: Using public blockchain as bulletin board;
- **Entropy Providers and Key Holders**: Both set are equal.

In the previous section, we demonstrated that such scheme meets all requirements to availability, unpredictability, bias-resistance and public-verifiability. We also implemented[1] this scheme [HERB] using Kyber library [Kyb], Cosmos-SDK [Cos] and Tendermint [Ten].

- For test purposes, the $(67, 100)$-threshold ElGamal scheme is used.
- Common ciphertext is being aggregated after $t = 67$ ciphertext shares were sent.
- Each entropy provider can send only one ciphertext share per round.
- For our tests, we use DigitalOcean's [DO] machines (4 CPU 2.6 GHz, 8GB RAM).
- We launch a test network with 34 blockchain full nodes and 100 entropy providers/key holders.
- We use Prometheus [Prom] for metrics recording.
- We implemented transactions for sending ciphertext shares and decryption shares. Calculations with participants secret keys are performed on the client side. Aggregation is taking place at server (blockchain) side.
- Average block time is 1 second.
- DigitalOcean's droplets were randomly located in the different regions: NYC1 - 5 droplets, TOR1 - 3, SGP1 - 7, LON1 - 2, BLR1 - 6, FRA1 - 4, SF2 - 5, AMS3 - 2.

Third party can verify the HERB results with parameters selected as specified above. They need:

[1] `https://github.com/corestario/HERB`

- $t + 1$ ciphertextes C_i and $t + 1$ CE-proofs;
- $t + 1$ decryption shares D_i and $t + 1$ DLEQ proofs.

Our application writes only the first $t+1$ transactions with ciphertext shares to the blockchain as well as only first $t + 1$ transactions with decryption shares.

Random number generation is taking around 4.1 seconds with these parameters (see figure 1). Each block contains from 35 to 65 transactions. One transaction with a ciphertext share weighs 640 bytes. One transaction with a decryption share weighs 744 bytes. Other measurements about transactions are described in the table below.

Transactions set	Weight
One tx (ciphertext share)	640 bytes
One tx (decryption share)	744 bytes
All sent txs per block (publication phase)	from 22 to 41 Kb
All sent txs per block (disclosure phase)	from 26 to 48 Kb
All sent txs per round	136Kb
All saved to blockchain txs per round	93Kb

Note that our protocol can be both used as a stand-alone solution for distributed random numbers generation and as an additional source of incoming entropy for other random number generation protocols. For example, Dfinity's approach [HMW18] suggests using the nothing-up-my-sleeve number as the starting input value for VRF. However, this method is subject to the possibility of manipulation [Ber+14], therefore, to obtain input values we have to apply an additional source of randomness, for example, the HERB protocol.

7 Analysis

Assume that Public Blockchain is used as a Bulletin Board. Most blockchains require to pay fees for computations on the blockchain side. So participants use public blockchain only as storage.

Such as the most of randomness beacons do not require public blockchain, we appended a consensus version of HERB to our comparison. Tendermint is used as a default consensus mechanism for HERB (see Future Work for more details about consensus algorithms). It has communication complexity $\mathcal{O}(n^2)$. For ease of comparison we take the case of the key holders set of n participants coinciding with the set of entropy providers.

Let's look at communication and computational complexities of the protocol.

At the first stage of the protocol (collecting C_i shares), each participant has to send its ciphertext share to other participants. In total, $(n - 1) \cdot n$ ciphertexts are sent. Then participants execute a consensus algorithm to obtain the joint ciphertext C. When the Tendermint [BKM18] protocol is used to send $n - 1$ ciphertext shares, the $\mathcal{O}(n^3)$ communication complexity is obtained. Then each key holder sends their decryption share to other $n - 1$ participants for decryption C. The overall protocol complexity is $\mathcal{O}(n^3)$. And communication complexity per node is $\mathcal{O}(n^2)$.

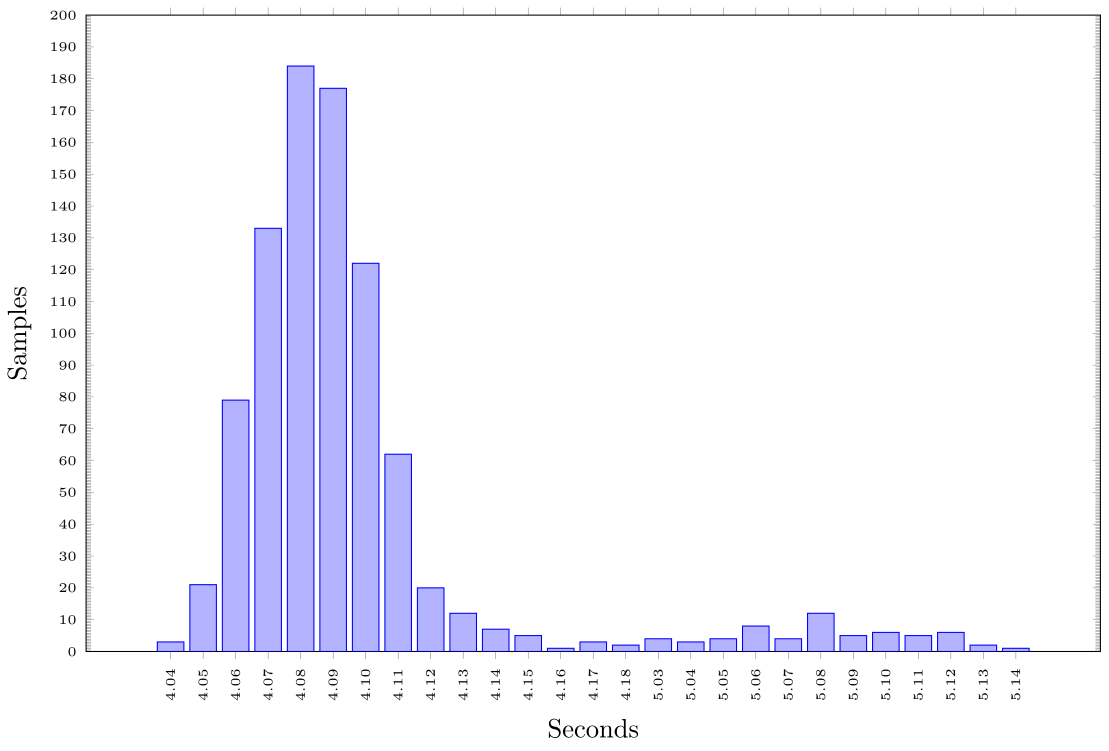

Figure 1: Test results

Clearly, with this approach applied, the HERB protocol yields a fairly high communication complexity level. One of optimization option is using the Avalanche [Roc18] algorithm as consensus. Avalanche communication complexity is $\mathcal{O}(kn)$ for some small (constant) security parameter $k << n$. Then the overall communication complexity is $\mathcal{O}(kn^2)$ and communication complexity per node is $\mathcal{O}(kn)$.

Third party verifier has to verify all proofs published during the protocol and check whether the aggregated ciphertext C is computed correctly. To do it, it is sufficient to perform $\mathcal{O}(n)$ arithmetic operations over elliptic curve points. Protocol participants have to perform additional computations. However, these will not increase computational costs in $\mathcal{O}$-notation. Thus, computational complexity is $\mathcal{O}(n)$.

Now we will consider existing random number generation algorithms covered in HydRand article [SJSW18]. For a detailed description of the solutions provided in the TABLE I, please, refer to the [SJSW18]. We appended [NV+] work to this table.

The table contains three protocol types:

1. Stand-alone protocols.
2. Protocols designed for the purpose of generating randomness, leveraging resources of existing systems.
3. Protocols, which produce randomness as a byproduct of their operation.

Given that HERB is a stand-alone solution, we will perform a more detailed comparison between it and similar protocols. Also, we will include the concept of using the unique threshold signatures covered in [HMW18] and [CKS05] into this comparison. Although both these articles don't cover stand-alone protocols, there are random beacon implementations (e.g. [Dra]) based on the same concept.

The protocol presented in [NV+] uses ElGamal encryption as PHE. As of November 2019, the protocol does not impose any restrictions on the sent ciphertexts (e.g. Proof of Correct Encryption presented in our work) so ElGamal encryption is malleable. Malleability means that an adversary is able to obtain the random number they need. Using a requester decreases overall complexity on the account of protocol security. The requester may refuse to decrypt the result or collude with a contributor to bias the output. Finaly, [NV+] requires VRF as an extra cryptographic primitive. The main disadvantage of the HERB protocol is the requirement to use the DKG protocol during the setup phase. HERB setup phase does not support frequent changes within the key holders set, as it consumes additional time and requires transferring more data than the main generation cycle.

The HERB protocol has the same communication complexity as Scrape [CD17] and Ouroboros [KRDO17]. Avalanche optimization can help improve bottleneck at the consensus step and leads to lower overall communication complexity of the HERB protocol. HERB's drawback is the requirement to use DKG.

HydRand [SJSW18] has a drawback: the complete unpredictability can only be achieved for random numbers generated in t rounds. It limits the use of the protocol in systems that require a frequent response from the PRNG tool, e.g. multiplayer gambling applications. This drawback reduces the benefit of smaller communication complexity obtained the Hydrand authors compared to Scrape [CD17] or Ouroboros [KRDO17] for some applications. However, HydRand achieve very low level of probability of successful prefiction after a few (constant) number of rounds. HydRand, just as Scrape and Ouroboros, does not need the DKG procedure.

Rand* protocol family [Syt+17] contains protocols with different properties. RandShare has all the Scrape's benefits and drawback. The HydRand [SJSW18] authors point out that the liveness property is lost within an asynchronous mode, so when a different communication model is selected, RandShare has all the properties required for a Random Beacon. RandHound and RandHerd have a relatively low communication complexity and offer quite good scalability options. The disadvantage of this is the failure probability, the reduction of which increases the overall complexity of communication

The HERB protocol falls behind the unique signatures schemes in terms of the communication complexity. At the same time, HERB does not require elliptic curve pairings, unlike BLS-based protocols and each new HERB's output doesn't depend on the previous one. Also, the HERB protocol allows participants to use any entropy sources for their random share including truly random number generators (e.g. [Guo+10]).

Regardless of the fact the HERB protocol is inferior to some of the mentioned protocols in terms of scalability, it has another interesting property to make up for it. In general, the entropy providers and key holders sets are independent of one another. It means that there can be more entropy providers than key holders. This property can be useful in some contexts, for example, in a lottery. Each ticket buyer becomes an entropy provider contributing to the generation of the joint lottery result. At the same time the key holders group is smaller and sampled to contain at most t malicious nodes.

Another advantage of the HERB protocol is an option to replace the partially homomorphic encryption scheme with one of the fully homomorphic lattice-based schemes. These schemes support the distributed key generation and threshold decryption (e.g. [BD10]). This modification ensures the post-quantum security for the HERB protocol.

TABLE I: Comparison of approaches for generating publicly-verifiable randomness

		Communication model	Liveness / failure probability	Comm. complexity (overall protocol)	Unpredicrability	Bias-Resistance	Comp. complexity (per hode)	Verification complexity (per passive verifier)	Characteristic cryptographic primitive(s)	Trusted dealer or DKG required
[CM16]	Algorand	semi-syn.	10^{-12}	$\mathcal{O}(cn)$	$\overset{t}{\nearrow}$	✗	$\mathcal{O}(c)$	$\mathcal{O}(1)$	VRF	no
[CKS05]	Cachin et al.	asyn.	✓	$\mathcal{O}(n^2)$	✓	✓	$\mathcal{O}(n)$	$\mathcal{O}(1)$	uniq. thr. sig.	yes
[AMM18]	Caucus	syn.	✓	$\mathcal{O}(n)$	$\overset{t}{\nearrow}$	✗	$\mathcal{O}(1)$	$\mathcal{O}(1)$	hash func.	no
[HMW18]	Dfinity	syn.	10^{-12}	$\mathcal{O}(cn)$	✓	✓	$\mathcal{O}(c)$	$\mathcal{O}(1)$	BLS sig.	yes
[SJSW18]	HydRand	syn.	✓	$\mathcal{O}(n^2)$	$\overset{t}{\nearrow}$✓	✓	$\mathcal{O}(n)$	$\mathcal{O}(n)$	PVSS	no
[KRDO17]	Ouroboros	syn.	✓	$\mathcal{O}(n^3)$	✓	✓	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	PVSS	no
[DGKR18]	Ourob. Praos	semi-syn.	✓	$\mathcal{O}(n)$	$\overset{t}{\nearrow}$	✗	$\mathcal{O}(1)$	$\mathcal{O}(1)$	VRF	no
[BO83]	Proof-of-Work	syn.	✓	$\mathcal{O}(n)$	$\overset{t}{\nearrow}$	✗	very high	$\mathcal{O}(1)$	hash func.	no
[BGB17]	Proof-of-Delay	syn.	✓	$\mathcal{O}(n)$	✓	✓	very high	$\mathcal{O}(log(\Delta))$	hash func.	no
[Syt+17]	RandShare	asyn.	✗	$\mathcal{O}(n^3)$	✓	✓	$\mathcal{O}(n^3)$	$\mathcal{O}(n^3)$	PVSS	no
[Syt+17]	RandHound	syn.	0.08%	$\mathcal{O}(c^2n)$	✓	✗	$\mathcal{O}(c^2n)$	$\mathcal{O}(c^2n)$	PVSS/CoSi	no
[Syt+17]	RandHerd	syn.	0.08%	$\mathcal{O}(c^2log(n))$	✓	✓	$\mathcal{O}(c^2log(n))$	$\mathcal{O}(1)$	PVSS/CoSi	yes
[CD17]	Scrape	syn.	✓	$\mathcal{O}(n^3)$	✓	✓	$\mathcal{O}(n^2)$	$\mathcal{O}(n^2)$	PVSS	no
[NV+]	Nguyen-Van et. al	unknown	✗	$\mathcal{O}(n)$	✓	✗	$\mathcal{O}(1)$	$\mathcal{O}(n)$	PHE, VRF, Public Blockchain	no
	HERB with Tendermint	syn.	✓	$\mathcal{O}(n^3)$	✓	✓	$\mathcal{O}(n)$	$\mathcal{O}(n)$	PHE	yes
	HERB with Avalanche	syn.	✓	$\mathcal{O}(kn^2)$	✓	✓	$\mathcal{O}(n)$	$\mathcal{O}(n)$	PHE	yes
	HERB with Public Blockchain	syn.	✓	$\mathcal{O}(n^2)$	✓	✓	$\mathcal{O}(n)$	$\mathcal{O}(n)$	PHE, Public Blockchain	yes

7.1 Future work

Consensus Algorithm. When the HERB protocol is executed, we can run the consensus algorithm between parties instead using bulletin boards. Existing consensus algorithms able to resolve the Byzantine agreement problem described in [LSP82]. The Practical Byzantine Fault Tolerance (PBFT) algorithm [CL+99] is a classical solution. More modern PBFT versions include the Ripple Protocol consensus algorithm (RPCA) [SYB+14], the Tendermint consensus algorithm [BKM18] or the Avalanche algorithm [Roc18].

HERB protocol looks similar to bulletin boards-based solution. Participants use a chosen consensus algorithm instead of storing data on a bulletin board to achieve consensus about published ciphertext shares (step 2). This approach allows to avoid using bulletin boards and related inconveniences. The drawback is higher requirements for the network connection between participants. For example, a participant has to be online to get random results.

HERB with Leader. Using a leader in each HERB round appears to be a promising approach to reduce the communication complexity to $\mathcal{O}(n^2)$. The core protocol modification then will imply node communication only with a selected leader. Participants choose a leader randomly (for example, the previously generated random number can be used) and only for one round. Here is the simplest case breakdown:

1. Setup phase: all key holders generate a common public key and shares of the private key. The Setup is executed once for a predefined number of random number generation rounds. Each of these rounds contains leader selection, publication and disclosure phases.

2. Leader election phase: the leader for the current round is selected randomly.

3. Publication phase: each entropy provider sends a computed ciphertext share with the relevant proof to the protocol leader. The leader then verifies the received messages, generates the aggregated C and sends C, valid ciphertext shares C_i and their proofs to key holders.

4. Disclosure phase: if the key holders accept the received C, they send the signed message, which contains ciphertext C and the decryption shares with the relevant proofs to the leader. Upon verifying the received decryption shares, the leader discloses the protocol result M, aggregates signatures from the key holders and sends the obtained random M number, aggregated signature and decryption shares to them together with the DLEQ proofs.

Note that this approach uses aggregated signatures, for example [Syt+16], [Max+18]. Communication complexity of this protocol is $\mathcal{O}(n^2)$. However, detailed specification of this protocol that complies with the randomness beacon requirements, as well as analysis of its security takes additional research and will be considered in future work.

Aggregation Protocols. Another feasible approach implies using aggregation protocols, in particular, Handel [BGKL19]. However, ciphertext aggregation has been studied less than signature aggregation. Therefore, implementing Handel [BGKL19] in the HERB protocol first takes selecting a ciphertext aggregation procedure that enables proving the validity of such aggregation.

8 Acknowledgements

We are grateful to Vasiliy Shapovalov for a lot of discussions, advice, and feedback.

We thank Philipp Schindler and Felix Crisan for the helpful feedback.

We thank Eugene Danilenko, Andrei Zavgorodnii, and Andrey Rybnov for help with the proof-of-concept implementation.

We also thank Maria Kondorskaya for her help with translation.

References

[Adi08] Ben Adida. "Helios: Web-based Open-Audit Voting." In: *USENIX security symposium*. Vol. 17. 2008, pp. 335–348.

[AMM18] Sarah Azouvi, Patric McCorry, and Sarah Meiklejohn. "Winning the Caucus Race: Continuous Leader Election via Public Randomness". In: *arXiv preprint arXiv:1801.07965* (2018).

[BBBF18] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. "Verifiable delay functions". In: *Annual International Cryptology Conference*. Springer. 2018, pp. 757–788.

[BBF18] Dan Boneh, Benedikt Bünz, and Ben Fisch. *A Survey of Two Verifiable Delay Functions*. Tech. rep. Cryptology ePrint Archive, Report 2018/712, 2018. https://eprint. iacr. org/2018/712, 2018.

[BBM00] Mihir Bellare, Alexandra Boldyreva, and Silvio Micali. "Public-key encryption in a multi-user setting: Security proofs and improvements". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2000, pp. 259–274.

[BCG15] Joseph Bonneau, Jeremy Clark, and Steven Goldfeder. "On Bitcoin as a public randomness source." In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 1015.

[BCR10] Padma Bh, D Chandravathi, and P Prapoorna Roja. "Encoding and decoding of a message in the implementation of Elliptic Curve cryptography using Koblitz's method". In: *International Journal on Computer Science and Engineering* 2.5 (2010), pp. 1904–1907.

[BD10] Rikke Bendlin and Ivan Damgård. "Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems". In: *Theory of Cryptography Conference*. Springer. 2010, pp. 201–218.

[Ber+14] Daniel J Bernstein, Tung Chou, Chitchanok Chuengsatiansup, Andreas Hülsing, Tanja Lange, Ruben Niederhagen, and Christine van Vredendaal. "How to manipulate curve standards: a white paper for the black hat." In: *IACR Cryptology ePrint Archive* 2014 (2014), p. 571.

[Ber+15] Daniel J Bernstein, Tung Chou, Chitchanok Chuengsatiansup, Andreas Hülsing, Eran Lambooij, Tanja Lange, Ruben Niederhagen, and Christine Van Vredendaal. "How to Manipulate Curve Standards: A White Paper for the Black Hat http://bada55. cr. yp. to". In: *International Conference on Research in Security Standardisation*. Springer. 2015, pp. 109–139.

[BGB17] Benedikt Bünz, Steven Goldfeder, and Joseph Bonneau. “Proofs-of-delay and randomness beacons in ethereum”. In: *IEEE Security and Privacy on the blockchain (IEEE S&B)* (2017).

[BGKL19] Olivie Begassat, Nicolas Gailly, Blazej Kolad, and Nicolas Liochon. “Handel Aggregation protocol for large scale Byzantine committee”. Stanford Blockchain Conference, Stanford university. 2019.

[BGZ16] Iddo Bentov, Ariel Gabizon, and David Zuckerman. “Bitcoin beacon”. In: *arXiv preprint arXiv:1605.04559* (2016).

[BKM18] Ethan Buchman, Jae Kwon, and Zarko Milosevic. “The latest gossip on BFT consensus”. In: *arXiv preprint arXiv:1807.04938* (2018).

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. “Short signatures from the Weil pairing”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2001, pp. 514–532.

[Blu83] Manuel Blum. “Coin flipping by telephone a protocol for solving impossible problems”. In: *ACM SIGACT News* 15.1 (1983), pp. 23–27.

[BO83] Michael Ben-Or. “Another advantage of free choice (extended abstract): Completely asynchronous agreement protocols”. In: *Proceedings of the second annual ACM symposium on Principles of distributed computing*. ACM. 1983, pp. 27–30.

[BOO10] Amos Beimel, Eran Omri, and Ilan Orlov. “Protocols for multiparty coin toss with dishonest majority”. In: *Annual Cryptology Conference*. Springer. 2010, pp. 538–557.

[BPW12] David Bernhard, Olivier Pereira, and Bogdan Warinschi. “How not to prove yourself: Pitfalls of the fiat-shamir heuristic and applications to helios”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2012, pp. 626–643.

[CD17] Ignacio Cascudo and Bernardo David. “SCRAPE: Scalable randomness attested by public entities”. In: *International Conference on Applied Cryptography and Network Security*. Springer. 2017, pp. 537–556.

[CDS94] Ronald Cramer, Ivan Damgård, and Berry Schoenmakers. “Proofs of partial knowledge and simplified design of witness hiding protocols”. In: *Annual International Cryptology Conference*. Springer. 1994, pp. 174–187.

[CKS05] Christian Cachin, Klaus Kursawe, and Victor Shoup. “Random oracles in Constantinople: Practical asynchronous Byzantine agreement using cryptography”. In: *Journal of Cryptology* 18.3 (2005), pp. 219–246.

[CL+99] Miguel Castro, Barbara Liskov, et al. “Practical Byzantine fault tolerance”. In: *OSDI*. Vol. 99. 1999, pp. 173–186.

[Cle86] Richard Cleve. “Limits on the security of coin flips when half the processors are faulty”. In: *Proceedings of the eighteenth annual ACM symposium on Theory of computing*. ACM. 1986, pp. 364–369.

[CM16] Jing Chen and Silvio Micali. “Algorand”. In: *arXiv preprint arXiv:1607.01341* (2016).

[Cos] *Cosmos SDK - A Framework for Building High Value Public Blockchains*. `https://github.com/cosmos/cosmos-sdk`.

[CP92] David Chaum and Torben Pryds Pedersen. “Wallet databases with observers”. In: *Annual International Cryptology Conference*. Springer. 1992, pp. 89–105.

[CSD18] Alisa Chernyaeva, Ilya Shirobokov, and Alexander Davydov. “Game Channels: state channels for the gambling industry with built-in PRNG”. In: (2018).

[DC] *DAO.Casino White Paper*. https://github.com/DaoCasino/Whitepaper/blob/master/DAO.Casino%20V

[DGKR18] Bernardo David, Peter Gaži, Aggelos Kiayias, and Alexander Russell. “Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2018, pp. 66–98.

[DO] *DigitalOcean - The developer cloud*. `https://www.digitalocean.com`.

[Dra] *Drand - A Distributed Randomness Beacon Daemon*. `https://github.com/dedis/drand`.

[ElG85] Taher ElGamal. “A public key cryptosystem and a signature scheme based on discrete logarithms”. In: *IEEE transactions on information theory* 31.4 (1985), pp. 469–472.

[FP01] Pierre-Alain Fouque and David Pointcheval. “Threshold cryptosystems secure against chosen-ciphertext attacks”. In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2001, pp. 351–368.

[FS86] Amos Fiat and Adi Shamir. “How to prove yourself: Practical solutions to identification and signature problems”. In: *Conference on the Theory and Application of Cryptographic Techniques*. Springer. 1986, pp. 186–194.

[Gen+07] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. “Secure distributed key generation for discrete-log based cryptosystems”. In: *Journal of Cryptology* 20.1 (2007), pp. 51–83.

[GO92] Shafi Goldwasser and Rafail Ostrovsky. “Invariant signatures and non-interactive zero-knowledge proofs are equivalent”. In: *Annual International Cryptology Conference*. Springer. 1992, pp. 228–245.

[GRPS03] Sharad Goel, Mark Robson, Milo Polte, and Emin Sirer. *Herbivore: A scalable and efficient protocol for anonymous communication*. Tech. rep. Cornell University, 2003.

[Guo+10] Hong Guo, Wenzhuo Tang, Yu Liu, and Wei Wei. “Truly random number generation based on measurement of phase noise of a laser”. In: *Physical Review E* 81.5 (2010), p. 051137.

[HERB] *Homomorphic Encryption Randomness Beacon*. `https://github.com/dgamingfoundation/HERB`.

[HMW18] Timo Hanke, Mahnush Movahedi, and Dominic Williams. “DFINITY Technology Overview Series, Consensus System”. In: *arXiv preprint arXiv:1805.04548* (2018).

[KD13] Cameron F Kerry and Charles Romine Director. “FIPS PUB 186-4 federal information processing standards publication digital signature standard (DSS)”. In: (2013).

[KRDO17] Aggelos Kiayias, Alexander Russell, Bernardo David, and Roman Oliynykov. "Ouroboros: A provably secure proof-of-stake blockchain protocol". In: *Annual International Cryptology Conference*. Springer. 2017, pp. 357–388.

[Kyb] *Kyber - Advanced crypto library for the Go language*. `https://github.com/dedis/kyber`.

[LH17] Jeremy Longley and Oliver Hopton. *Funfair technology roadmap and discussion*. `https://funfair.io/wp-content/uploads/FunFair-Technical-White-Paper.pdf`. 2017.

[LMR83] Michael Luby, Silvio Micali, and Charles Rackoff. "How to simultaneously exchange a secret bit by flipping a symmetrically-biased coin". In: *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*. IEEE. 1983, pp. 11–22.

[LSP82] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine generals problem". In: *ACM Transactions on Programming Languages and Systems (TOPLAS)* 4.3 (1982), pp. 382–401.

[LW15] Arjen K Lenstra and Benjamin Wesolowski. "A random zoo: sloth, unicorn, and trx." In: *IACR Cryptology ePrint Archive* 2015 (2015), p. 366.

[Max+18] Gregory Maxwell, Andrew Poelstra, Yannick Seurin, and Pieter Wuille. "Simple schnorr multi-signatures with applications to bitcoin". In: *Designs, Codes and Cryptography* (2018), pp. 1–26.

[Mil+16] Andrew Miller, Yu Xia, Kyle Croman, Elaine Shi, and Dawn Song. "The honey badger of BFT protocols". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM. 2016, pp. 31–42.

[MNS09] Tal Moran, Moni Naor, and Gil Segev. "An optimally fair coin toss". In: *Theory of Cryptography Conference*. Springer. 2009, pp. 1–18.

[NIST] *NIST Randomness Beacon*. `https://www.nist.gov/programs-projects/nist-randomness-beacon`.

[NV+] Thanh Nguyen-Van, Tuan Nguyen-Anh, Tien-Dat Le, Minh-Phuoc Nguyen Ho, Tuong Nguyen-Van, Quang Nhat Le, and Khuong Nguyen-An. "Scalable distributed random number generation based on homomorphic encryption". In: ().

[Ora] *Oraclize - Blockchain oracle service, enabling data-rich smart contracts*. `http://www.oraclize.it`.

[Pie18] Krzysztof Pietrzak. "Simple Verifiable Delay Functions." In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 627.

[Prom] *Prometheus - Monitoring system and time series database*. `https://prometheus.io`.

[Rab83] Michael O Rabin. "Transaction protection by beacons". In: *Journal of Computer and System Sciences* 27.2 (1983), pp. 256–267.

[Rand] *RANDOM.org - True Random Number Service*. `https://www.random.org`.

[RBO89] Tal Rabin and Michael Ben-Or. "Verifiable secret sharing and multiparty protocols with honest majority". In: *Proceedings of the twenty-first annual ACM symposium on Theory of computing*. ACM. 1989, pp. 73–85.

[Roc18] Team Rocket. *Snowflake to avalanche: A novel metastable consensus protocol family for cryptocurrencies.* 2018.

[Sch99] Berry Schoenmakers. “A simple publicly verifiable secret sharing scheme and its application to electronic voting”. In: *Annual International Cryptology Conference.* Springer. 1999, pp. 148–164.

[SDM04] Paul Syverson, R Dingledine, and N Mathewson. “Tor: The secondgeneration onion router”. In: *Usenix Security.* 2004.

[Sha79] Adi Shamir. “How to share a secret”. In: *Communications of the ACM* 22.11 (1979), pp. 612–613.

[SJSW18] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. “HydRand: Practical Continuous Distributed Randomness”. In: *2020 IEEE Symposium on Security and Privacy (SP).* Ieee. 2020, to appear.

[ST04] Berry Schoenmakers and Pim Tuyls. “Practical two-party computation based on the conditional gate”. In: *International conference on the theory and application of cryptology and information security.* Springer. 2004, pp. 119–136.

[Ste+00] André Stefanov, Nicolas Gisin, Olivier Guinnard, Laurent Guinnard, and Hugo Zbinden. “Optical quantum random number generator”. In: *Journal of Modern Optics* 47.4 (2000), pp. 595–598.

[SYB+14] David Schwartz, Noah Youngs, Arthur Britto, et al. “The Ripple protocol consensus algorithm”. In: *Ripple Labs Inc White Paper* 5 (2014).

[Syt+16] Ewa Syta, Iulia Tamas, Dylan Visher, David Isaac Wolinsky, Philipp Jovanovic, Linus Gasser, Nicolas Gailly, Ismail Khoffi, and Bryan Ford. “Keeping authorities" honest or bust" with decentralized witness cosigning”. In: *2016 IEEE Symposium on Security and Privacy (SP).* Ieee. 2016, pp. 526–545.

[Syt+17] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J Fischer, and Bryan Ford. “Scalable bias-resistant distributed randomness”. In: *Security and Privacy (SP), 2017 IEEE Symposium on.* Ieee. 2017, pp. 444–460.

[Tan05] Caimu Tang. “ECDKG: A Distributed Key Generation Protocol Based on Elliptic Curve Discrete Logarithm”. In: *sE· CURECOMM* (2005), pp. 353–364.

[Ten] *Tendermint Core (BFT Consensus) in Go.* `https://github.com/tendermint/tendermint`.

[TP] *TruePlay White Paper.* https://trueplay.io/docs/en/TruePlayWhitePaper.pdf.

[TY98] Yiannis Tsiounis and Moti Yung. “On the security of ElGamal based encryption”. In: *International Workshop on Public Key Cryptography.* Springer. 1998, pp. 117–134.

[VDH+15] Jelle Van Den Hooff, David Lazar, Matei Zaharia, and Nickolai Zeldovich. “Vuvuzela: Scalable private messaging resistant to traffic analysis”. In: *Proceedings of the 25th Symposium on Operating Systems Principles.* ACM. 2015, pp. 137–152.

[Wes18] Benjamin Wesolowski. “Efficient verifiable delay functions.” In: *IACR Cryptology ePrint Archive* 2018 (2018), p. 623.

[Wik02] Douglas Wikström. “A note on the malleability of the El Gamal cryptosystem”. In: *International Conference on Cryptology in India*. Springer. 2002, pp. 176–184.

Appendices

A ElGamal Encryption Correctness

We use the definition of Σ-protocols with properties completeness, special soundness and special honest verifier zero-knowledge as defined in [CDS94]

Proof of Correct Encryption.

This is a Σ protocol with relation $R = \{(\delta, \omega)|(B = xG + rQ, A = r * G\}$, where stament $\delta = (G, Q, A, B)$ and witness $\omega = (x, r)$.

Common input: The prover P and verifier V both have δ.

Private input: P has value ω

1. P chooses s_1, s_2 , computes $T = s_1G + s_2Q$ and $E = s_2G$ and sends V message (T, E).
2. V sends P challenge e.
3. P computes $z_1 = s_1 + ex$, $z_2 = s_2 + er$ and sends V reply (z_1, z_2, e)
4. V accepts, if $z_1G + z_2Q = T + eB$ and $z_2G = E + eA$ else rejects it.

Proof. In order to see that completeness holds observe that when P runs the protocol honestly we have:

$z_1G + z_2Q = s_1G + exG + s_2Q + erQ = T + e(xG + rQ) = T + eB$

$z_2G = s_2G + erG = E + eA.$

To show special soundness we consider two accepted transcripts $\{T, E, e, z_1, z_2\}$ and $\{T, E, e', z'_1.z'_2\}$. From $z_1 = s_1 + ex$ and $z'_1 = s_1 + e'x$ we can easy extract x. From $z_2 = s_2 + er$ and $z'_2 = s_2 + e'r$ we can extract r. We get that $x = (z_1 - z'_1)/(e - e')$, $r = (z_2 - z'_2)/(e - e')$.

This protocol is a special honest verifier zero-knowledge. We can simply choose random $z_1, z_2, e \in \mathbb{Z}_q$ and compute $E = z_2G - eA$, $T = z_1G + Z_2Q - eB$. The transcript $\{T, E, e, z_1, z_2\}$ will be the output of the simulation. Also we can use just z_1, z_2 with e as input value. □

Remark. Special soundness and special honest verifier zero-knowledge properties of the Σ protocol can be turned into soundness and zero-knowledge respectively using Fiat-Shamir heuristic. [BPW12].

B Modified DKG

Protocol 6 Distributed Key Generation

1. Sharing phase. Each key holder has identifier id_i, where $i = 1, ..., n$. Key holder id_i shares random value z_i as a dealer:

   1.1 id_i generates two polynomials:

   $$\begin{aligned} f_i(x) &= a_{i,0} + a_{i,1}x + ... + a_{i,t}x^t \\ f_i^{'}(x) &= b_{i,0} + b_{i,1}x + ... + b_{i,t}x^t \\ z_i &= f_i(0) = a_{i,0} \end{aligned}$$

   1.2 id_i commits these polynomials as set $\mathbf{V} = \{V_{i,k}\}_{k\in[0,t]}$, where $V_{i,k} = a_{i,k}G + b_{i,k}H$ and publishes $\mathbf{V}$.

   1.3 id_i calculates secret shares for $j = 1, ..., n$:

   $$s_{i,j} = f_i(id_j),\ s_{i,j}^{'} = f_i^{'}(id_j)$$

   1.4 id_i sends $s_{i,j}, s_{i,j}^{'}$ to the party id_j using the private channel between id_i and id_j.

   1.5 Each key holder $id_j, j = 1, ..., n$:

   - Verifies
   $$s_{i,j}G + s_{i,j}^{'}H \stackrel{?}{=} \sum_{k=0}^{t} id_j^k V_{i,k} \tag{1}$$
   - If shares doesn't satisfy the condition above then id_j broadcasts complaint against id_i.

   1.6 If id_i recieve complaint from id_j then id_i broadcasts $s_{i,j}$ and $s_{i,j}^{'}$ that satisfy the equation above. Correct $s_{i,j}$ and $s_{i,j}^{'}$ are accepted by id_j.

2. Secret shares calculation. Each participant id_i, $i = 1, ..., n$:

   2.1 id_i marks as disqualified any party that either

   - received more than t complaints in Step 1.5 or
   - answered a complaint in Step 1.6 with values that falsify eq. (1)

   2.2 id_i builds the set of non-disqualified parties $QUAL$. All honest parties build the same set $QUAL$ (proof of that presented in [Gen+07]).

   2.3 $x_i = \sum_{j\in QUAL} s_{j,i}$ is secret key share of id_i.

3. Public shares publication phase. Each key holder id_i, $i \in QUAL$:

   3.1 id_i broadcasts $A_{i,k} = a_{i,k}G, k = 0, ..., t$

   3.2 Each key holder id_j, $j \in QUAL$:

   - Verifies
   $$s_{i,j}G \overset{?}{=} \sum_{k=0}^{t} id_j^k A_{i,k} \tag{2}$$
   - If published values doesn't satisfy the conditions above then id_j broadcasts complaint against id_i.

   3.3 If id_i receives complaint from id_j then id_i broadcasts $s_{i,j}$ and $s_{i,j}^{'}$ that satisfy all equations.

4. Public key calculation phase. Each key holder id_i, $i \in QUAL$:

   4.1 For parties id_i who receives at least one *valid* complaint, i.e., values which satisfy eq. (1) and not eq. (2), the other parties run the reconstruction phase of Pedersen-VSS protocol described in [Gen+07] to compute z_i, $f_i(x)$ and $A_{i,k}, k = 1, ..., t$ in the clear.

   4.2 $PK_i = A_{i,0}, i \in QUAL$ are public key shares. Common public key is $PK = \sum_{i \in QUAL} PK_i$.

   4.3 $VK_i = \sum_{h \in QUAL} \sum_{k=0}^{t} id_j^k \cdot A_{h,k}$, $i \in QUAL$ are verification keys.