Near Protocol Randomness Beacon

Alex Skidanov
Near Protocol
/AlexSkidanov
alex@nearprotocol.com

August 2019

Abstract

We present a randomness beacon scheme that is unpredictable and unbiasable for as long as more than 1/3 of participants follow the protocol, is live for as long as 2/3 of participants follow the protocol, doesn't depend on verifiable delay functions and doesn't require distributed key generation. The disadvantage of the approach is $O(n^3)$ network overhead.

1 Introduction and Related Work

This short document presents the algorithm Near Protocol uses for the randomness beacon.

Distributed randomness beacon is a construction that allows n participants continuously create random numbers with the following properties:

1. **Unpredictable**. No participant should be able to predict the resulting number or reason about any properties of it before the number is created.
2. **Unbiasable**. No participant should be able to influence the resulting number in any way.

We also want the algorithm to tolerate some percentage of malicious actors that are either offline or intentionally deviate from the protocol. We differentiate between the number of malicious actors that need to cooperate to stall the algorithm and the number of malicious actors that is needed to make the algorithm predictable or biasable.

There are multiple approaches to distributed randomness beacon that are being implemented in blockchain projects.

RANDAO + VDFs. Ethereum Serenity plans to use RANDAO, the output of which is then fed to a verifiable delay function, output of which is the generated random number. This approach has a very desirable property that it remains live, unbiasable and unpredictable for as long as there are at least two honest actors. Recent advances in VDFs ([1], [2]) make this approach feasible in

practice. However it is extremely hard to configure the VDF and the algorithm in such a way that it finishes in meaningful time on conventional hardware, but the adversary with a specialized ASIC doesn't compromise the unpredictability of the algorithm.

Threshold Signatures. Dfinity [3], at its core, relies on a randomness beacon based on DKG-friendly Threshold Signatures. While can be configured differently, one possible configuration provides liveness for as long as more than 2/3 of the participants are online and follow the protocol, and remains unbiasable and unpredictable for as long as more than 1/3 participants are online and follow the protocol.

The disadvantage of this approach is the necessity to perform a very complex distributed key generation (DKG) step. DKG is an active area of research, and while there were multiple breakthroughs in the recent years, it remains an extremely complex task.

RandShare and RandHound. RandShare and RandHound are two randomness beacon schemes presented in [4]. RandShare requires more than 2/3 honest participants for both liveness and to remain unbiasable and unpredictable, which is worse than the 1/3 threshold for unpredictability that threshold signatures provide. It also has $O(n^3)$ network overhead.

RandHound is a more complex version that reduces the network overhead.

In this paper we present a very simple approach that has the same thresholds for liveness and to remain unbiasable and unpredictable as Threshold Signatures, but doesn't require DKG. It does impose $O(n^3)$ network overhead, which is undesirable, however only $O(n^2)$ messages are sent (each of which is $O(n)$ long) as opposed to $O(n^3)$ constant sized messages in RandShare.

2 Construction

Let n be the total number of participants, and k be $\lfloor 2n/3 \rfloor + 1$.

We assume the existence of a public key cryptography scheme such that

1. It's deterministic, i.e. if one participant encrypted a message m with a public key pk and obtained the result em, then any participant that encrypted m with pk obtains em.

2. It's verifiable, i.e. if a participant that has access to a private key sk was given a message that was claimed to be encoded with pk, they can either decrypt it and get some message m that is, when encrypted with pk again, results in em, or can prove that no such message m exists.

We discuss a suitable scheme in section 4.

Each random number is generated in five steps:

Commitment. Each participant j generates (using a local source of randomness) a random vector r_j of size k where each element is a 256 bit random numbers, computes an (n, k) erasure code of r_j resulting in a vector s_j of size n with n shares such that any k shares are sufficient to reconstruct r_j, and encodes

each share $s_{j,i}$ with the public key of participant i to get a vector e_j of size n. They then broadcast e_j. Since each share in e_j is encoded with a public key of different participant, it is impossible to recover r_j unless k participants collude and share their decoded shares.

Consensus. Participants collectively use any byzantine fault tolerant consensus, for example Tendermint [5], to reach a consensus on exactly k published vectors e. Say the vectors on which the consensus was reached were published by participants $\{z_1, z_2, z_3 ... z_k\}$

Reveal. Each participant i for each $j \in 1..k$ decrypts $e_{z_j,i}$ and either publishes the decrypted $s_{z_j,i}$ or a proof that $e_{z_j,i}$ cannot be decrypted. Assuming no more than $n - k$ participants are offline or otherwise not following the protocol, at some point for each vector e_{z_j} at least k such decrypted s_{z_i}, i elements (or proofs of impossibility to decrypt them) will be published.

Recovery. Once for each vector e_{z_j} at least k elements were decrypted or proven non-decryptable, each participant locally performs the following for each such vector:

1. If for at least one element a proof that such element cannot be decrypted is provided, the vector is discarded;
2. Otherwise, we have at least k elements of s_{z_j}, which is sufficient to reconstruct r_{z_j}. If the reconstruction fails, the vector is discarded;
3. Otherwise, once r_{z_j} is reconstructed, we recompute the erasure code s'_{z_j}, and then encrypt each element i of s'_{z_j} with the public key of participant i to compute e'_{z_j}. If $e_{z_j} \neq e'_{z_j}$, the vector is discarded;
4. Otherwise, r_{z_j} is kept.

Computing output. The final output is the XOR of all the kept r_{z_j}.

3 Analysis

Theorem 1 (Liveness)**.** *For as long as at least k participants are online and follow the protocol, the protocol will terminate assuming partially synchronous network.*

Proof. For this proof we assume that the consensus in the second step is Tendermint, though it naturally applies to many other consensus algorithms. Since at least k participants are online and follow the protocol, there will be a moment when at least k vectors e_j are published. Starting with the next round in Tendermint any honest leader will be able to propose a set of k of those vectors. Since Tendermint is live under partially synchronous network, a consensus on some subset of k vectors will be reached in finite time.

Once the consensus is reached, since at least k participants are online and follow the protocol, there will be a moment in time when each participant observes at least k revealed elements $s_{z_j,i}$ for each j, and thus would move to the

recovery step. Both the recovery and computing output steps are performed offline and do not require further communication. □

Lemma 1. *During the recovery phase for any $j \in 1..k$ either all the honest participants will discard vector e_{z_j}, or all the honest participants will successfully reconstruct and keep r_{z_j}*

Proof. It is sufficient to prove that if at least one honest participant recovered r_{z_j} and kept it, then all the honest participants will recover r_{z_j} and will keep it.

Indeed, if a honest participant successfully recovered and kept r_{z_j} that means that their locally recomputed e'_{z_j} matched the published and agreed upon e_{z_j}, but that means that each element $e_{z_j,i}$ can be decrypted, and corresponds to the correct share $s_{z_j,i}$, and thus all the shares were successfully decrypted by all honest participants, and each honest participant successfully reconstructed $r_{z_j,i}$ and (assuming the erasure code is deterministic) obtained matching $e'_{z_j,i}$. □

Theorem 2 (Safety)**.** *Once the protocol terminates, all the participants will agree on the produced output.*

Proof. Assuming the consensus protocol used in consensus phase is safe, all the participants agreed on the same set $z_1, z_2...z_k$. According to lemma 1, all the participants then kept the same subset of the published vectors. Since the final output is a deterministic function of such vectors, all the participants obtained the same output. □

Theorem 3. *For as long as less than k participants are malicious and cooperate, the protocol is unbiasable and unpredictable.*

Proof. The resulting number is determined once the consensus phase is over. After that the malicious actors can only stall the protocol, but cannot influence the number in any way, since once the vector z is agreed upon, all the participants are guaranteed to keep the same set of r_j and will observe the same output.

Since less than k participants are malicious and cooperate, at least one of the agreed upon z_j corresponds to an honest participant. Similarly, since less than k actors cooperate, they can't have access to k shares of s_{z_j}, thus they cannot recover r_{z_j} of any honest participant, and cannot reason about its value. Therefore, until the reveal phase starts, the malicious actors have no way to learn the value of r_{z_j} of at least one honest actor, and thus have no insight into the value of the resulting output.

The protocol is unpredictable because the moment when the number is determined (end of consensus phase) is before the first moment when any participant gets any insight into the resulting generated number (the beginning of the reveal phase).

The protocol is unbiasable because the last moment when any participant has any influence on the number (the commitment and consensus phases) is before the first moment when any participant gets any insight into the resulting generated number (the beginning of the reveal phase). □

4 Public Key Cryptography Scheme

The algorithm described in section 2 relies on public key cryptography scheme that is determenistic and veriable.

At the core of the scheme we use ElGamal [6] encryption scheme. However, ElGamal is not determenistic, specifically it uses an ephemeral randomly generated key as one of the encryption steps. It is not determenistic by design, because in most practical applications of ElGamal it must be impossible to brute-force the input. In the application to the randomness beacon described above it is not a concern, since the input to the encryption is a random 256 bit number, and thus cannot be brute-forced. We make ElGamal determenistic by changing the derivation of the ephemeral key be a determenistic function of the input.

Assuming cyclic group $\mathscr{G}$ with the generator G, secret key x and public key $P = xG$, and some M which is an element of $\mathscr{G}$, the encrypted M is computed as:

$$\begin{aligned} k &= hashToScalar(M) \\ C_1 &= kG \\ C_2 &= M + kP \\ out &= (C_1, C_2) \end{aligned}$$

where k is the ephemeral secret key and C_1 is the ephemeral public key.

Once (C_1, C_2) is received, the message can be decrypted as

$$M = C_2 - xC_1$$

Indeed,

$$C_2 - xC_1 = (M + kP) - xkG = M + xkG - xkG = M$$

This algorithm is determenistic, in a sense that the same message encrypted with the same public key will always result in the same encrypted message. However, it is not verifiable under the definition that we provided in section 2. Specifically, if a malicious actor used a different value of k when encoding a message M and then distributed resulting (C_1, C_2), then once the message M is decrypted and re-encrypted following the protocol, a different pair (C_1', C_2') will be obtained.

We want the participant that has access to the secret key x in this situation to be able to prove that the published pair (C_1, C_2) was not obtained by following the protocol. We will do that by publishing a proof that consists of M such that when encrypted following the protocol doesn't result in (C_1, C_2), the value $S = xC_1$, and a cryptographic proof that the participant knows value x such that when multiplied by C_1 results in published S without revealing x.

To prove that there's such x that $xC_1 = S$ without revealing x we can prove instead that

$$dlog(S, C_1) = dlog(P, G)$$

Proof systems for such statements about discrete logarithms is a well-researched area, for example see [7]. Specifically, we can proof the statement above in the following way:

$$\begin{aligned} r &= randomScalar() \\ R_1, R_2 &= rC_1, rG \\ e &= hashToScalar(R_1, R_2) \\ s &= r + xe \\ out &= (R_1, R_2, s) \end{aligned}$$

Now everyone can verify that $dlog(S, C_1) = dlog(P, G)$ by recomputing e from (R_1, R_2) and then verifying that:

$$\begin{aligned} sC_1 &= R_1 + eS \\ sG &= R_2 + eP \end{aligned}$$

With all this instrumentation a full proof that the message M was not encrypted properly consists of:

$$(M, xC_1, R_1, R_2, s)$$

5 Conclusion

We presented a simple randomness beacon design that can tolerate less than 1/3 of malicious actors for liveness, and less than 2/3 of malicious actors to remain ubiasable and unpredictable, with $O(n^3)$ network overhead. The protocol doesn't depend on any complex cryptographic primitives such as distributed key generation or verifiable delay functions.

This makes the protocol applicable in practical settings in which the cubic network overhead is acceptable, but the desired threshold to remain ubiasable is high.

6 Acknowledgements

Thanks to Daniel Robinson for major contributions to section 4.

References

[1] Benjamin Wesolowski. Efficient verifiable delay functions. 11478:379–407, 2019.

[2] Krzysztof Pietrzak. Simple verifiable delay functions. Cryptology ePrint Archive, Report 2018/627, 2018. https://eprint.iacr.org/2018/627.

[3] Timo Hanke, Mahnush Movahedi, and Dominic Williams. DFINITY technology overview series, consensus system. *CoRR*, abs/1805.04548, 2018.

[4] Ewa Syta, Philipp Jovanovic, Eleftherios Kokoris Kogias, Nicolas Gailly, Linus Gasser, Ismail Khoffi, Michael J. Fischer, and Bryan Ford. Scalable bias-resistant distributed randomness. Cryptology ePrint Archive, Report 2016/1067, 2016. https://eprint.iacr.org/2016/1067.

[5] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on BFT consensus. *CoRR*, abs/1807.04938, 2018.

[6] Taher El Gamal. A public key cryptosystem and a signature scheme based on discrete logarithms. In *Proceedings of CRYPTO 84 on Advances in Cryptology*, pages 10–18, New York, NY, USA, 1985. Springer-Verlag New York, Inc.

[7] Jan Camenisch and Markus Stadler. Proof systems for general statements about discrete logarithms. Technical report, 1997.