ASHOKA UNIVERSITY, DEPTARTMENT OF COMPUTER SCIENCE

CS2361 Blockchain and Cryptocurrencies - CS2362 Computer Security and Privacy *Final Report*: **Publicly Verifiable Randomness via Blockchain**

Nikhil Bhave, Vibodh Nautiyal
May 2, 2022

The project links can be found at:

- Website Link: https://vibodhnautiyal.github.io/randomness-beacon/ (Note: You need to have Metamask extension enabled on your browser and be on the Kovan Testnet)
- Github Link: https://github.com/vibodhnautiyal/randomness-beacon

1 INTRODUCTION

Several entities in the modern world require some form of randomness- elections, lotteries, conscription or sports drafts. However, in most of these cases, the participants are required to place their faith in some central authority who is in charge of these entities. There have been cases in history where the authenticity of these central authorities has been questioned. One such instance was the Vietnam Selective Service Lottery in which young men across the United States were being conscripted into the Vietnam War. The lottery system consisted of 366 numbers that were to be drawn at random. The number indicated the birthday of those who were to be conscripted. However, statisticians later found out that due to faults with the lottery system, those with birthdays later in the year were more likely to be chosen. The

consequences of this are quite somber- many of the men conscripted never returned home.

This leads to to believe that there is a necessity for a form of randomness that is:

1. public
2. tamper-proof
3. unpredictable
4. eliminates the need for trust in a central authority

In this report, we look at **Blockchain as a platform upon which to build a beacon of publicly-verifiable randomness**. We aim to give an intuitive understanding of how a randomness beacon is set up and how it is expected to perform, to analyse the methods with which the system could be tampered with and finally, to implement an interface that emits random numbers using the Ethereum blockchain.

2 VERIFIABLE RANDOMNESS

How will the consumers of a randomness beacon be convinced that the numbers emitted by the beacon are, in fact, truly random (or as close as possible to truly random numbers)? First, let us set up the conditions under which our beacon will operate.

There are two major properties of random numbers that our Beacon must satisfy:

1. **Uniformity**
2. **Independence**

Let there be an *n-bit* source of randomness that goes into our beacon
The beacon, in turn, emits an *m-bit* random number at time interval *t* such that m < n
Let X be a random variable that denotes our m-length random number

To satisfy the uniformity condition, our beacon must emit a random number "a" with the following probability:

$$P[X = a] \approx \frac{1}{2^m}$$

To satisfy the independence condition, first consider another random variable Y that denotes a different random number "b". Then, the following must be satisfied:

$$P[(X = a), (Y = b)] = P[X = a] \times P[Y = b]$$

In order to test these properties of Randomess, we have used the NIST Test Suite and ran tests on the random numbers generated by our Beacon. More information on the same is given later in the report.

3 Preliminaries and Major Concepts

3.1 Entropy

Entropy is a mathematical measure of information or uncertainty, and is computed as a function of a probability distribution. As an example, suppose a random variable X represents the toss of a fair coin. In such a case, the entropy of a coin toss is one bit, since we could encode *heads* by 1 and *tails* by 0. The entropy of n independent coin tosses would be n, since the n coin tosses could be encoded by a bitstring of length n. Entropy is defined as follows:

Suppose X is a discrete random variable that takes on values from a finite set X. Then the **entropy** of the random variable X is defined to be the quantity

$$H(X) = -\sum_{x \in X} P[x] \log_2 P[x]$$

The key idea behind entropy is to ask the question: how much information, on average, would we need to encode an outcome from the distribution? We have seen above that the number is 1 bit in case 1 coin flip, and n bits in case of n coin flips.

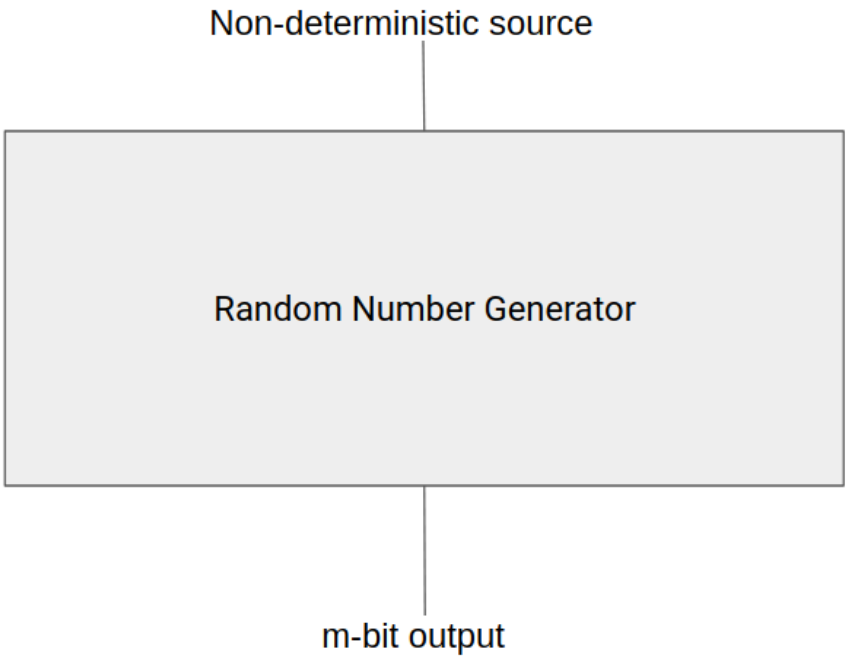
3.2 Min-entropy

Min-entropy is the information (in bits) needed to encode the **most likely outcome** from a distribution.

$$\text{min-entropy} = -\log_2 p_{max}$$

Note that for a uniform distribution, the min-entropy is equal to the Shannon entropy. This is because in a uniform distribution, each outcome is equally likely and thus the most likely outcome would also be the same as each outcome.

For a random number generator that produces an m-bit output, the min-entropy should be equal to m.



A random number generator should produce an m-bit output with probability $\frac{1}{2^m}$.

min-entropy $= -\log_2 p_{max}$ where $p_{max} = \frac{1}{2^m} = -(-m) = m$ bits
We have ran our random numbers through the NIST Test Suite which includes the Approximate Entropy Test. More details of the same are given later in the report.

3.2.1 How is the Entropy derived from a cryptocurrency?

In the original paper titled "On Bitcoin as a public randomness source" that we have extensively referred to for our project, as the name suggests, the authors have primarily developed their method using Bitcoin. For kicking off our discussion, we still start out with Bitcoin and eventually transition to Ethereum.

A Bitcoin block consists of a header and a group of transactions. We are primarily interested in the header as that is our main source of entropy. We cannot use the transactions to generate randomness because the pool of UTXOs is available prior to being included in blocks as part of transaction pools. The header consists of various fields such as:

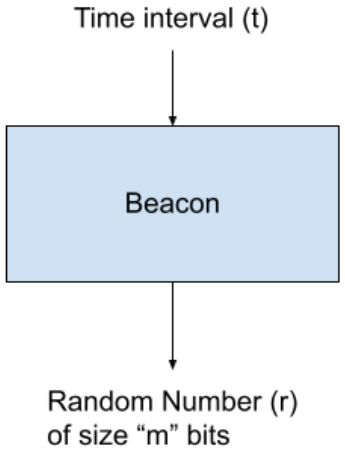
1. Version
2. Previous Block Hash
3. Merkle Tree (consists of Entropy)

4. Timestamp (consists of Entropy)
5. Current Difficulty
6. Nonce (consists of Entropy)

The currently difficulty of a block is usually decided by the number of 0s that are placed in front of the *Hash Puzzle*. This is usually referred to as the "d" value. This is periodically adjusted after around every 2016 blocks.

The Ethereum block header architecture is slightly different from that of Bitcoin's. It also significantly longer. It consists of header fields not found in Bitcoin such as: *ommer's Hash, State Root, Logs Bloom*, among others.

Figure 3.1: High-level overview of the Beacon



3.3 EXTRACTORS

An extractor is a function $y = Ext_k(x)$ that takes an n-bit input x of "sufficient" entropy and returns an m-bit input y of "high" entropy. While a Bitcoin header is 640 bits long, our own extractor function will take as input a 4096-bit Ethereum block header (thus, n = 4096). It will pass this header through SHA-256 hash function and then the output will be hashed once again. Finally, we will end up an m-length output where m = 256.

3.4 BEACON

A Beacon is a function $r = Beacon(t)$ which returns an m-bit near-uniform random value r at a time interval t. Given a sample distribution D_t, the value r is computed using an "extractor":

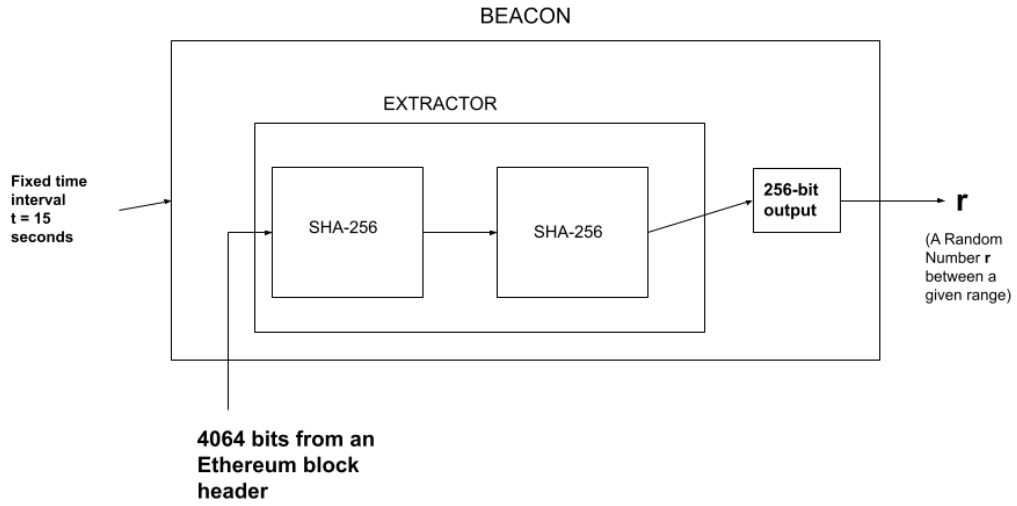
$$r \leftarrow Beacon(t) = Ext_k(D_t)$$

For the construction of the Beacon, we will use the Ethereum block header (E_t) as an input to our Extractor function:

$$Beacon(t) = Ext_k(E_t)$$

As we know, miners have to solve a hash puzzle and insert the answer into the nonce field. It is not possible to predict with high probability what the next nonce will be. If a miner is able to predict with probability greater than $\frac{1}{d}$ where d is the current Mining Difficulty, then there would be a faster miner process than trying random nonces. We use this fact to create our Beacon. We feed in the entire Block header into our extractor function, which will then output a random string.

Figure 3.2: Internal Implementation of the Beacon



3.5 WHERE DOES THE RANDOMNESS OF THE BEACON COME FROM?

The randomness of the Beacon is a result of the proof-of-work hash puzzle that miners have to solve to mine a Bitcoin or Ethereum block. Every block has a mining difficult d, which is the

number of consecutive 0s that the hash of the block header must start with.

Any miner may create a block which will have a probability of $\frac{1}{2^d}$ of being a valid block. This is because of the basic property of a hash function, namely that it behaves as a random oracle. Therefore, we have no way of knowing whether the first bit will be 0 or 1, and similarly for the next d bits. Since there are d bits that are specified to be 0 in the hash puzzle, and our chance of getting a 0 in each one of those d bits is $\frac{1}{2}$, the probability of us finding a number such that its hash contains d consecutive zeroes at the start is $\frac{1}{2^d}$.

We also know that there is no better way of choosing a block (including the nonce) such that it has a probability greater than $\frac{1}{2^d}$ of being a valid block prior to computing the hash. Therefore, the only feasible mining strategy is to try random values and check the hash.

This is where the randomness that powers the Beacon comes from: the d-bits of consecutive zeroes that the miners must find in the hash. Note that these d bits are actually spread out across two fields: the nonce and the Merkle hash, but we capture them both as we hash the block in our construction.

4 Implementation of the Randomness Beacon

4.1 The Backend

The backbone of our backend is a smart contract written in Solidity. The primary function of this smart contract is to obtain the (Current Block Number - 1) and use that to query the corresponding Block Hash. Finally, we pass this block hash along to the front-end.

This is the smart contract code logic:

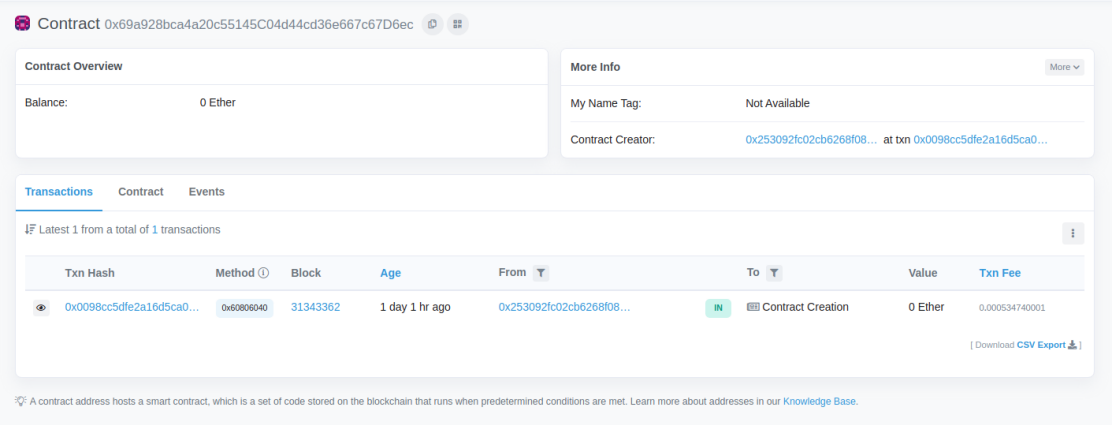
```
pragma solidity 0.8.13;

contract Beacon {
    function extractor() public view returns(bytes32) {
        uint _previousBlockNumber;
        bytes32 _previousBlockHash;
        bytes memory _hashinbytes;
        bytes32 _hashOfHash; // this is Hash of m
        _previousBlockNumber = uint(block.number - 1);
        _hashinbytes = abi.encodePacked(_previousBlockHash);
        _hashOfHash = sha256(_hashinbytes);
        return (_hashOfHash);
    }
}
```

The contract has been deployed to the Kovan testnet and the address is

0x69a928bca4a20c55145C04d44cd36e667c67D6ec

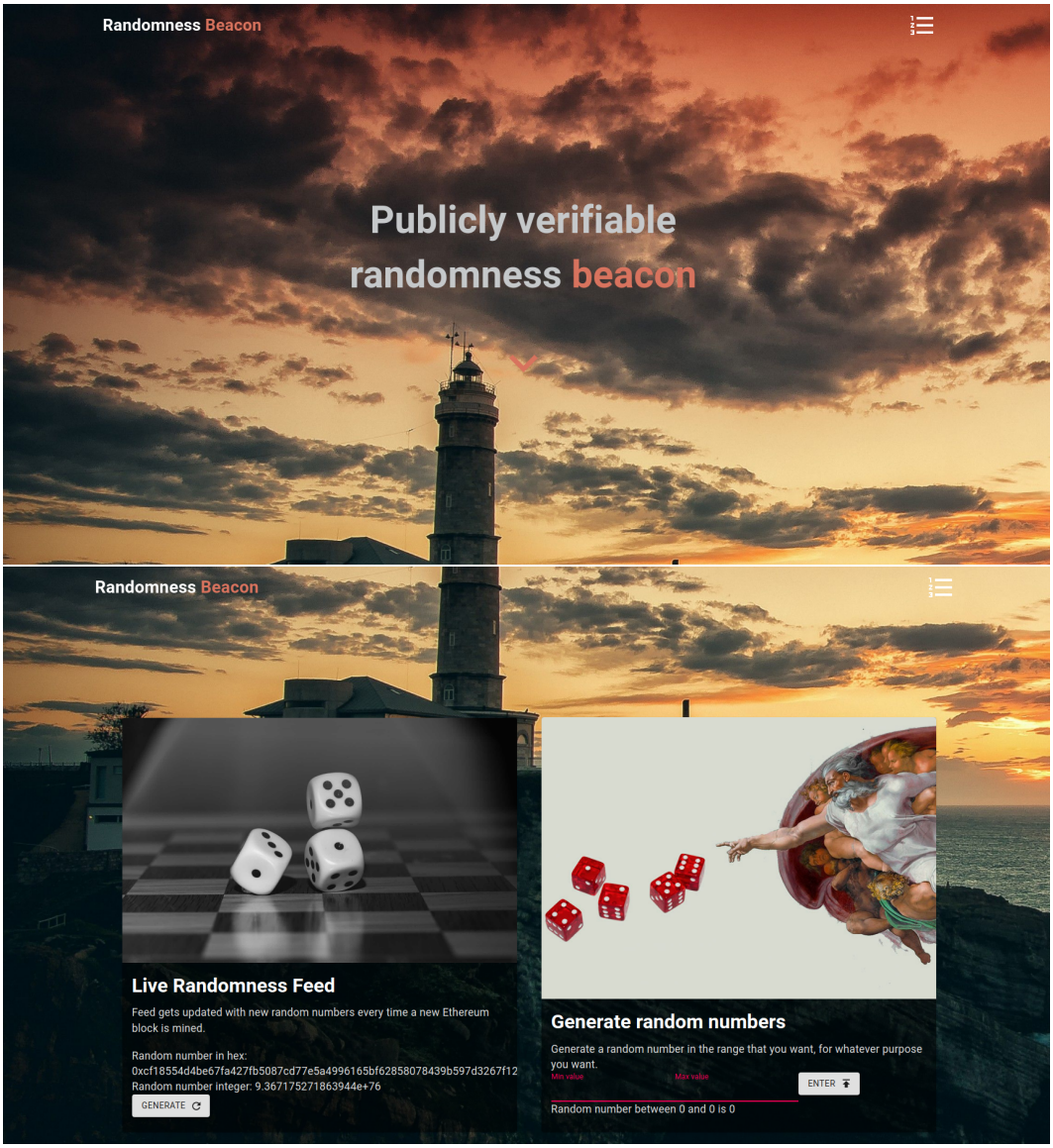
The contract can be called like an API call at any time from either Javascript using web3.js or from within another Solidity contract



4.2 The Frontend

The front-end is developed primarily in React.js. We make our Solidity smart contract interact with the React Front-end using web3.js. Web3.js uses a *fetch* call to interact with the smart contract and get the block hash. The smart contract gets the hash of the latest block that was mined and runs it through the extractor function. It then returns this hex value.

We display this number as a hex value and decimal in the live randomness feed. If the user wants a random number in a range, they can put the minimum and maximum value and we return a random value in that range.



5 Tests for Randomness

5.1 NIST's Statistical Suite

We used the NIST Test Suite to test the randomness of the numbers produced by our Beacon. The NIST Test Suite is a statistical package consisting of 15 tests that were developed to test the randomness of binary sequences. It is a well known testing suite for randomness, as well as one of the most trusted. The different tests and details on the same can be found here: https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf.

5.2 Test Results

We ran the tests with a sequence of over 6000 bits (more than the input value recommendation for most tests) and the Beacon's sequence of bits passed 14 out of the 15 tests. This gives confidence that the bit sequence being generated by the Beacon are indeed random.

```
Type of Test                                    P-Value                 Conclusion
01. Frequency Test (Monobit)                    0.5820834007409373      Random
02. Frequency Test within a Block               0.9951332420432067      Random
03. Run Test                                    0.44027255384499653     Random
04. Longest Run of Ones in a Block              0.796000516657981       Random
05. Binary Matrix Rank Test                     0.9264783048045885      Random
06. Discrete Fourier Transform (Spectral) Test  0.37063684749429837     Random
07. Non-Overlapping Template Matching Test      0.23568411221110266     Random
08. Overlapping Template Matching Test          0.35917484662565713     Random
09. Maurer's Universal Statistical test         -1.0                    Non-Random
10. Linear Complexity Test                      0.2635046460555814      Random
11. Serial test:
                                                0.2475402421846669      Random
                                                0.23890336235740753     Random
12. Approximate Entropy Test                    0.013794271755777977    Random
13. Cummulative Sums (Forward) Test             0.8363288399429446      Random
14. Cummulative Sums (Reverse) Test             0.8575749522835303      Random
15. Random Excursions Test:
                State       Chi Squared         P-Value                 Conclusion
                -4          2.726253907615122   0.7421001833497328      Random
                -3          1.9637517241379312  0.8541376093997681      Random
                -2          9.102171136653896   0.10505763687552792     Random
                -1          2.0114942528735633  0.8475523117064279      Random
                +1          5.942528735632184   0.3118513589375146      Random
                +2          11.910742159784308  0.036031500572255515    Random
                +3          6.066593103448275   0.29979295662536787     Random
                +4          8.038255133158119   0.15413987619055972     Random
16. Random Excursions Variant Test:
                State       COUNTS              P-Value                 Conclusion
                -9.0        146                 0.2780066489522469      Random
                -8.0        125                 0.4569900511193713      Random
                -7.0        93                  0.8996091206247475      Random
                -6.0        79                  0.8549077682975009      Random
                -5.0        84                  0.9395704158247962      Random
                -4.0        88                  0.9771410123331056      Random
                -3.0        76                  0.7091968762204706      Random
                -2.0        79                  0.7262257625654784      Random
                -1.0        91                  0.7617075639478075      Random
                +1.0        79                  0.5441970982678067      Random
                +2.0        90                  0.8955329031670437      Random
                +3.0        109                 0.45574603740414465     Random
                +4.0        104                 0.6261817855010814      Random
                +5.0        82                  0.8994551371792656      Random
                +6.0        63                  0.5832934499147818      Random
                +7.0        63                  0.6138253662393598      Random
                +8.0        65                  0.6667387615459992      Random
                +9.0        47                  0.46205748578579564     Random
```

6 Attacks and Adversaries

6.1 Getting a particular random number

We will do an analysis of an adversarial attack on this network and calculate the minimum cost of the reward required to incentives the adversary to manipulate the system.
The adversary is able to pay off any miners B, where B is the cost of a block reward for mining a block, whenever the adversary wishes to suppress a block from being pushed into the network.

If we assume the near-uniform randomness of our above-mentioned Beacon, then the proba-

bility of forcing the beacon to produce a certain outcome becomes a Bernoulli Trial.
Let the probability that a beacon outputs value "x" be: p
Thus, the probability that a beacon does not output value "x" is: $1 - p$

To make this more clear, let us take a concrete example. Let's say that the probability that a beacon outputs a certain value x is $\frac{1}{10} = 0.1$. Thus, p = 0.1 and p - 1 = 0.9

Therefore, on average, for every 10 blocks that a miner discards, he should be able to get a 1 block that, when inputted into the beacon, gives a particular value x. Thus, 10 - 1 = 9 blocks will have to be discarded.

Defined more formally, in order to obtain any particular value x of size m-bits, a miner will have to discard:

$$(\frac{1}{p} - 1) \text{ blocks}$$

Consider a bet on a single Beacon output: either a 0 or a 1. The adversary stands to win W with probability p for the outcome of a single bit decided by the Beacon. Without any manipulation, the adversary's expected winning is $W \cdot p$.

Note that if the adversary decides to attack, then the adversary must withhold blocks until they get a block that gives them the Beacon output that they desire. If the adversary has the power to withhold all blocks until success, then the expected earnings of the adversary are W, since the adversary is guaranteed to win the bet.

6.2 Manipulation resistant lottery

A manipulation resistant lottery is a lottery in which it is impossible for the adversary's expected reward after manipulation subtracted by the cost of manipulating the Beacon to be greater than the adversary's expected reward without any manipulation.

Therefore, the adversary's algorithm will be advantageous $\iff$ Earnings from manipulating beacon > Expected earnings without manipulation. This is same as (Lottery stake - cost of manipulating beacon) > expected earnings without manipulation.

$$W - \frac{1-p}{p} > p \cdot W$$

$$p - p \cdot W > \frac{1-p}{p}$$

$$W(1-p) > \frac{1-p}{p}$$

$$W > \frac{1}{p}$$

This bet is on a single Beacon output: either a 0 or a 1. Therefore, in this case $p = \frac{1}{2}$. So, $W > 2B$, where B is the block reward.

Currently, the Ethereum block reward is 3 Eth, while the Bitcoin block reward is 6.25 Bitcoins. So, the cost of manipulating a single bit outcome from the Beacon would be 6 Eth = $16,842 on Ethereum blockchain and $484,012 on Bitcoin.

This shows that the cost of manipulating even a single bit outcome from the lottery is quite expensive. The reward that the adversary would stand to earn must be greater than these amounts in order for the adversary to be incentivised to bribe all the miners.

7 Future Work and Related Concepts

One of the popular alternatives to the above mentioned method of generating randomness is using an off-chain, decentralized Oracle service called such as Chainlink. Oracles like Chainlink are useful in procuring real-world data like coin prices, weather stats, market rates and much more. The reason we use an oracle and not native API calls is because API calls might change every other second and different nodes replaying the transaction might get different results. This will end up breaking consensus.

Instead, oracles post the requested data as a transaction on the network itself in a publicly-available manner. Thus, all nodes refer to this common piece of information and will get the same results when they replay results. Chainlink VRF (Verifiable Random Function) provides a decentralized option of obtaining random numbers from an off-chain source (link: https://docs.chain.link/docs/chainlink-vrf/). This is the industry standard for getting random numbers into decentralized applications. However, in the absence of Oracles, our construction of the randomness beacon is still the best way to generate randomness on-chain.

8 REFERENCES

1. Bonneau, J., Clark, J., Goldfeder, S. (2015). *On Bitcoin as a public randomness source* - IACR. Retrieved April 2, 2022, from https://eprint.iacr.org/2015/1015.pdf

2. Rosenbaum, D. E. (1970, January 4). *Statisticians charge draft lotter'y was not random.* The New York Times. Retrieved April 2, 2022, from this link

3. Cryptography: Theory and Practice by Douglas Stinson and Maura Paterson

4. Properties of Random Numbers, Bucknell University

5. Oracles | ethereum.org

6. NIST Statistical Test Suite