# BCOLN 2019 – Challenge Task Report

## **Introduction**

In the course of the module Blockchain and Overlay Networks in 2019 the students were asked to implement a challenge task. This should be solved in teams and then presented. The theme of this year's task was to deploy a lottery as a decentralized app (dApp) on the Ethereum Blockchain. The lottery system should determine the random number with the help of an oracle, which like the lottery is also implemented as a Smart Contract.

Lottery systems have been in existence for a very long time and the integrity and correctness of a central authority (lottery organizer) must always be trusted. Since ever larger sums are handled thereby at least the temptation is given to try to take influence on the drawn numbers. Would it not be a good idea at this point to carry out the lottery on the blockchain and thus ensure a decentralized and in the best case fraud-proof system?

### **Requirements**

The system requirements essentially comprise the four points listed below:

1. The lottery must be fully functional, produce the output in graphical or textual form, and be implemented using Smart Contracts only. In addition, the random number must be provided by an Oracle Contract.
2. The oracle must be implemented as a separate Smart Contract. Thereby it does not matter whether the number is generated on the blockchain or externally.
3. The contract must automatically distribute the winnings to the winner, divide the winnings if several players have chosen the right number or cumulate the winnings for the next lottery if no winner is determined.
4. The contract and its functionality must be documented in a self-contained report.

The remainder of this report is structured as follows. In section 1.2 a certain background knowledge about the oracle is given and which possibilities exist to generate a random number.

Section 1.3 deals with the design decisions made, which are then explained in more detail in section 1.4, the implementation. Finally, a summary is presented in 1.5 and the conclusions drawn.

## **Background**

Implementing a lottery system in a "normal" environment may not be considered very demanding. A group of people report their numbers to a central office, the central office determines a random number and the winner receives the money. However, if we operate in the context of blockchains, there are some problems that were not considered important before. Just the basic question of what a random number is and how it is created is of great importance. The following subsections are dedicated to some approaches for generating random numbers, each of which will discuss the advantages and disadvantages.

The two main difficulties encountered are the following

1. How is a random number generated? The code in Solidity should be deterministic because it runs decentralized. A clock or something similar is not available to determine a random.
2. How does a smart contract know when the time has come to draw a winner? Again, there is no reliable time available.

Since the code should run deterministically, a generator is needed to generate the same number on multiple nodes if the code is called multiple times. This actually completely contradicts the term "random". The solution to this problem is to generate a random number once and use it on different nodes. So the code is run several times on several nodes, but it produces a unique number.

### **Why Using Random() is Not a Good Idea**

The Random function consists of the block timestamp and the block difficulty. Both variables cannot be influenced by the players, but by the miners to a certain extent. Accordingly, the use of the random() function for a lottery where unconditional independence should be guaranteed is not optimal.

In general, according to [1] it is **not advisable** to build your random on a **block variable**. This includes for example the coinbase (address of the miner of the block), the previously mentioned difficulty and timestamp as well as the block number or the gas limit. This is for the simple reason that they can be influenced by the miner.

### **Is it Better to Use The Blockhash?**

Some implementations suggest using a past blockhash, to generate a random. This is bad because an attack can create an exploit contract with the same PRNG code and then call the target contract via an internal message. The "random" numbers for the two contracts will be the same.

Some contracts suggest to use a future blockhash according to [1]. This approach is slightly better than the others and works as follows. One player makes a bet and the house stores the block number of the transaction. In a second call, the player asks the house if it can draw the winner. The house fetches the previously saved block number and generates a random one from the hash. The problem is that the blockhashes are only available from the last 256 blocks for scaling reasons. All older blockhashes will be zero. If the second call is not made within 256 blocks, the blockhash and random will be zero.

Another approach is to calculate a private seed with the random. It is true that private variables cannot be called by other contracts, but off-chain the variable can be accessed via web3.eth.getStorageAt() using web3. The blockchain should therefore not be used to store secrets in plaintext.

### **Maybe Using an External Service?**

Another possibility is to use an external service via a URL connector. This can be used, for example, to request a number from Random.Org. In the eyes of the writer, however, this is somewhat in keeping with the meaning of the blockchain, since it again relies on a central location, which, however, can be compromised. However, if this risk is considered to be negligible, it is still a good possibility.

### **All good things come in four**

A promising approach, the commit-reveal approach, as the name implies consists of two different phases. In a first phase, the participants commit a cryptographically secured secret in the form of a hash. In the second phase, all parties reveal their secrets in plain text. These will then be compared with the hash and if they are correct with the other secrets merged with an XOR operation and then used as a seed for a random value.

This approach should logically not be based on a single instance and should hash the user's secrets together with their address to ensure that only the actual owner of a secret can reveal it. Additionally, the possibility can be added that each commit will cost an amount and will only be refunded if revealed. If a secret is not revealed, no random will be generated and a new attempt will be made.

## **Design**

The design of the contract and the interactions were recorded in a class diagram, which is shown in fig. [1].

In the course of implementing the Challenge Task, we decided to implement the Commit Reveal approach, as we believe it is the most promising method.

The general procedure of the lottery is as follows. The lottery is opened, which at the same time starts a new campaign in the oracle. After that all participants of the lottery will have a number of commits and afterwards they will have to register as soon as there are enough players. If all players have revealed in a certain time interval, the random number will be calculated and the lottery will be dissolved according to the rules of the game. If more than one person wins, the prize is split. If only one person wins, they receive the entire pot. If no one wins, the pot is taken to the next round.

!!INSERT DIAGRAM HERE!!

## **Implementation**

## **Summary & Conclusion**

The evaluation was carried out using Web3 tests. The main focus was on the functionality of the contract and the different outputs. The test cases each include the following options:

* No one wins pot in the next round
* Several people win, pot is split
* One person wins, pot is paid to that person.

Since it is relatively difficult to calculate the effective amount of ether in an account (gas must be included when an action is executed) we had to make an approximation to determine the amount of money paid out and paid in.

## **References**

[1] Predicting Random Numbers in Ethereum Smart Contracts, <https://blog.positive.com/predicting-random-numbers-in-ethereum-smart-contracts-e5358c6b8620>, Online, Accessed last: 13.05.2019