SCHOOL OF BUSINESS

Nicosia, Cyprus

Implementing electronic lotteries by using hash values of Bitcoin blocks as source of entropy

BY

GEORGIOS TOPALIDIS

"This Project was submitted in partial fulfillment of the requirements for the Masters of Digital Currency degree at the University of Nicosia, School of Business, Nicosia, Cyprus, **(May/2018)**

University of Nicosia
46 Makedonitissas Avenue
P.O.Box 24005
1700 Nicosia
Cyprus

Date: **(May/2018)**

A Master's Project

In

Digital Currency

By

GEORGIOS TOPALIDIS

"Submitted in partial fulfillment of the requirements for the Masters of Digital Currency degree at the University of Nicosia"

Approved by:	**Date of Approval:**
……………………………… **(name)** (position)	………………………
……………………………… **(name)** (position)	………………………
……………………………… **(name)** (position)	………………………

Acknowledgements

I would first like to thank my thesis-project advisor Professor Mr George Giaglis of the School of Business at University of Nicosia. I am extremely thankful and indebted to him for sharing expertise, and sincere and valuable guidance and encouragement extended to me.

Finally, I must express my very profound gratitude to my parents and to my girlfriend Maria for providing me with unfailing support and continuous encouragement throughout my years of study and through the process of researching and writing this thesis -project. This accomplishment would not have been possible without them.

Abstract

This project aims to implement a decentralized application for conducting electronic draws and lotteries. Our approach is to build a service which can ensure decentralization, security, transparency and verifiability. For this purpose our application is using the hash values of Bitcoin blocks as source of randomness in order to capture the high needed entropy. Moreover, the application is taking advantage of the OP_RETURN transaction for storing successfully the results of the draw-lottery to the blockchain. The steps towards building the application as well as the experimental tests implemented for securing the functionality of the algorithm are demonstrated in this project.

Table of Contents

List of Tables

List of Figures

List of Appendices

Chapter 1 Introduction

1.1 Introduction

In this chapter is going to be examined the objective of the current project, the rationale, the theoretical framework, the methodology and the data that have been used as input for the electronic draw algorithm. Moreover, there is a brief reference of the most notorious scandals in the history of lottery games in order the reader to understand the insufficiencies and the inaccuracies in which a lot of lotteries depend on. Finally there will be a brief description of the design of the algorithm scheme.

1.2 Objective

The objective of my project is the development of a service that uses decentralized digital currency technology to solve an existing problem. More in particular, my service is an algorithm, implementing electronic draws and raffles with the aid of blockhash, which is a unique number identifying every block in the Bitcoin blockchain, which can be used as source of entropy with increased randomness. Moreover, the algorithm is taking advantage of the Bitcoin blockchain which is used as a public database for ensuring the transparency and the security of the whole process.

The present project will not introduce new things to the related literature. On the other hand, the present project is a significant effort introducing how it is possible to take advantage of seeds with high entropy in order to implement verifiable draws. The project will focus more on technical steps rather than making a research about the alternative options.

The aim is to deliver a trusted, transparent and fair service which takes advantage of big and open data like the blockhashes of Bitcoin blocks and can guarantee at a scale of maximum 100000 participants a secure and verifiable outcome. This service might be used by companies, organizations, governments, institutions and individuals who would like to proceed to a transparent and secure raffle.

1.3 Rationale

The reasoning for implementing a service like this is because there are not many algorithms or services that guarantee at the same time security, verifiability and transparency in the field of electronic draws and raffles. Moreover, the project it is structured in a way that is easy to communicate to the public the importance of the blockchain technology for solving everyday life problems which related with proof of existence and safe storage of information.

In addition, this project will add more results to the topic area and more importantly will add some more evidence about the use of verifiable seeds and outcomes. Finally, there is no doubt that the present project is very important for the prediction market, addressing a problem and introducing a construction method which is relatively new for the prediction market reality.

In chapter 2, on the one hand, we are going to examine more in detail a successful approach of using the blockchain technology for ensuring secure, verifiable and transparent draws by a company called Saferandom [1] (Saferandom.com, 2018) and on the other hand we are going to examine an unsuccessful approach by a company with the name Firelotto [2] (Firelotto.io, 2018) and reason why.

But, before going on examining these test cases of companies using the blockchain technology with the one way or the other, it will be wise to present at this point some scandals in the history of lottery games in order the reader to understand the insufficiencies of a system which not rely on blockchain technology. The absence of vital elements that blockchain technology provides in lottery games could not guarantee a system with high security, transparency and verifiability.

1.4 Reference on most notorious scandals in the history of lottery games

There are several big scandals in the history of lottery games. We can mention here in brief the most notorious in order to see the insufficiencies of procedures which not rely on open and big data and as a result suffer from opacity and unreliability. Moreover, apart from common lotteries there were cases where participants deceived by people who created fake websites or fake online services which promise fair and random draws for giveaways. A characteristic example of a case like this is the imitation of random.org list randomizer [3] (RANDOM.ORG, 2018).

The first lottery scandal was took place in Pennsylvania in 1980 and is known with the name Triple Six Fix. In this lottery the organizers weight all the balls apart from those with the numbers 4 and 6, hoping that one out of eight combinations with these numbers will be feasible. Indeed, the combination of 6, 6, 6 was finally the fixed combination that won the lottery. This unusual pattern alerted the authorities which at the end arrest the fixers of this scam [4] (En.wikipedia.org, 2018). We can point out that in this case the lottery lacks confidence, security and transparency.

Table 1.1: Possible fixed combinations in Triple Six Fix scandal

Fixed Possible Combinations with numbers 4 and 6
444
446
464
644
666
664
646
466

A second less infamous scam than the Triple Six Fix mentioned above, but costly enough was the scandal of Milan Lotto. In this case blindfolding children recruited in order to predict the numbers of the balls. Children were bribed to learn how to squint through their blindfolds. Moreover, there was a significant help from the organizers who marked some balls in order to be easier for the children to choose [5] (Hastley et al., 2018). Also, in this case we observe the interference of third parties which can have influence on the final outcome without securing transparent and fair procedures.

Another lottery scandal was occurred in China. A person called Zhao Liqun discovered a loophole in China's lottery drawing. By taking advantage of this loophole, by choosing the winning numbers after the announcement of the outcome, this guy bought a qualified and playable ticket several times after the ending of the draw. By doing this and taking advantage of this loophole he won the lottery with this unacceptable way [5] (Hastley et al., 2018). Furthermore, in this case we saw that a lottery may be prone to human errors which have to do with the right time of the announcement of the winning numbers, the conditions of implementing a draw and finally with measures securing the whole procedure.

The case of Ontario in June of 2004 with a store clerk hoodwinked a customer out of his winning ticket and the case of the group of construction workers participating in a group lottery draw in 2003 which won the ticket but earlier a store clerk had swapped out this ticket when he realized how much it had won, showed the necessity of building secure systems [5] (Hastley et al., 2018). With the usage of blockchain technology in prediction market industry, the participants are not only sure that their numbers are stored successfully in the blockchain but also that the draw outcome is securely stored also and could not be changed by someone with malicious intents. Moreover, once a participant acquires his/her unique ticket, this ticket accompanies him/her until the fulfillment of the draw procedure.

Finally, a fixed lottery draw happened in Serbia when the first three out of five numbers were called out as normal, but there was confusion when the fourth ball was called, the number 27 and the screen displayed it as the number 21. When the next ball was pulled it was number 21, leading to viewers to question whether the game known as Lotto in Serbia, had been fixed [6] (Mail Online, 2018). This false example also showed to us how important is to have in a draw a random number generator or in other words a source of randomness with high entropy. The Serbian Lotto number generator proved to be not so random.

1.5 Theoretical Framework, Methodology and Data

The present project will examine the whole theoretical framework about capturing data and use them as a seed to an algorithm. More precisely, the theoretical framework about the algorithm of implementing the draws will be examined with an assessment of its general strengths and weaknesses. Having a clear idea about the related theory and the problem of implementing verifiable, secure and transparent draws we will undertake an algorithm of making a draw. This algorithm will give the results of the draw. This algorithm is very important for the project because is the core part which takes the data and afterwards appears the results and moreover because it is unusual to use entropy from the blocks of a digital coin in order to capture the randomness of data.

So, the process has to do with capturing the data from the current last block of Bitcoin by using its hash after finding the solution to the Proof of Work algorithm. The hash of the blocks will be captured with the aid of the library BitcoinJ [7] (Bitcoinj.github.io, 2018) of Java. In our project it is explained why these data are verifiable and why they are produced by sources with high entropy.

It is very important the fact that our captured data will come from an unbiased source which is able to capture data with high randomness. We need high randomness on our data in order to be difficult for someone to foresee or assume the values of our captured data. The more entropy our source produces the more random the data come from this source would be. In order to keep the captured data verifiable, we have to store immediately the captured data from the local draw application to Bitcoin blockchain, in order everyone to see that the captured data are verifiable.

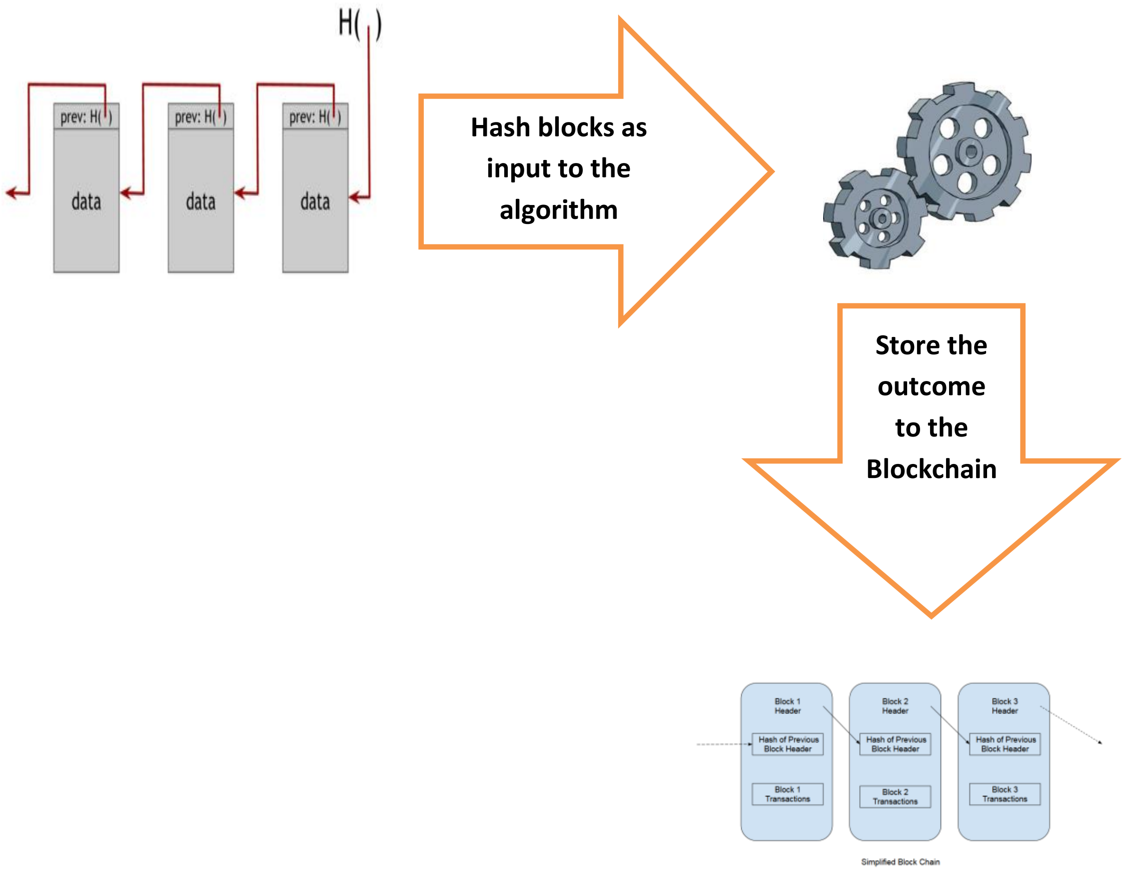
1.6 Design of the Algorithm Scheme

My design scheme in order to implement some core parts of the functionality of a prediction market, will consist of the below elements:

- Bitcoin Block Hash Results
- Draw Algorithm
- Blockchain Ledger

The captured hash results from Bitcoin blocks will be used as the random seed to my algorithm. Afterwards, the algorithm will define the winner of the draw. The results will be stored in the Bitcoin blockchain in order to be verifiable by all participants and at the same time secure.

Figure 1.1: Design of algorithm scheme



1.7 Summary

In this chapter there was an introduction about the objective and the reason for writing this project. Furthermore, there was a reference about the theoretical framework under which it is going to be examined our project and reference also about the methodology and the data which will feed the draw algorithm. Finally there was a brief description of the scheme in order the reader to have a picture of what it is going to be examined further.

In the following chapter, there will be a more detailed mention to the theory in which the draw algorithm counts in order to support its features, there will be literature review on electronic draws and finally a description on two test cases of companies trying to implement electronic draws.

Chapter 2 Introduction

2.1 Introduction

In this chapter is going to be discussed what is entropy, the role of entropy in a successful and secure draw and what is considered as a good entropy source. Moreover, in this chapter there are examples of estimating entropy based on stock market values (closing prices), based on Proof of Work algorithm of Bitcoin and based on Scrypt algorithm of Litecoin. These examples are helping to understand the use of financial data as a random beacon and the importance of counting on a good source of entropy for implementing procedures which require high levels of randomness. For this reason, among the other parts, there will be a subchapter analyzing and comparing the different sources used for acquiring entropy. Finally, there is an analysis of two test cases where two companies are taking advantage of the entropy provided by the mined blocks of Bitcoin network. On the one hand, there is a successful implementation of a secure and transparent electronic lottery but on the other hand we will see that the core elements for implementing a secure system are missing.

2.2 What is entropy and what is considered a good entropy source

Generally, entropy refers to disorder or uncertainty. This means that the less the possibility of an event to happen creates more information about its fulfillment. In other words, if one of the events is more probable than others, observation of that event is less informative. Shannon entropy quantifies all these considerations and it takes into account not the meaning of an event, for example the outcome of tossing a coin, but it depends on the information encapsulated in the underlying probability distribution of that event.

The formula of entropy is: **H(x) = E[I(x)] = E[-ln$(\boldsymbol{P}(\boldsymbol{x}))$]** [8] (En.wikipedia.org, 2018) where:

H: Entropy

E: Expected Value Operator

P: Probability of x event to happen

I: Information Content of x

Shannon entropy is measured in bits and we can calculate this entropy from the formula: S=$\log_2 2^N$, where 2^N is the number of possible outcomes. The value of entropy is strictly related to the source which creates the entropy. To make it clear, let assume that we have a coin, as a source of entropy, which has two heads and no tails. This coin will always have a zero entropy, since the coin will always come up heads and as a result will lead to a perfectly predicted outcome,S1=$\log_2 2^0$=0. We can see the difference now taking into account that the same source (coin) has higher entropy when it has one head and one tail, because now we have S2=$\log_2 2^1$= 1. The result of each toss of this coin delivers one full bit of information. So, we can come up with the idea that the more random a source could be, the higher the entropy that creates. To extend the previous simple example of tossing a coin and to make clear the notion of entropy we will analyze now the case of two fair coin tosses. The entropy in bits is the result of raising the number of possible outcomes to the base 2 logarithm. So, with two cases we end up having four possible outcomes and two bits of entropy because S3=$\log_2 2^N$=$\log_2 2^2$=2, where N=2.

To sum up, information entropy is the average amount of information conveyed by an event, when considering all possible outcomes. So, by capturing and comparing the outcome of three different sources in the upcoming parts of this chapter is vital in order to understand why randomness plays a vital role in electronic lottery systems and why some sources are more trustworthy than other, producing more entropy bits. Consider the fact that when there is a reference on a source of entropy is actually a reference on a random number generator, so the more random the numbers produced by the generator-source, the more secure and reliable the lottery procedure depend on that generator-source could be.

2.3 Capturing entropy from financial data (Matlab example)

After giving the definition of entropy, a small theoretical background and a brief example on how to estimate the entropy of tossing one coin and two coins accordingly, at this point we will see why financial data could be potentially a good source of randomness and also estimate the entropy by using financial data like stock market values. So, based on the scientific paper "On the Use of Financial Data as a Random Beacon" is proved that closing prices for common stocks contain sufficient min-entropy for generating random challenges. These challenges can be used in elections or other cryptographic applications [9] (Clark and Hengartner, 2010).

"In standard voting procedures, random audits are one method for increasing election integrity. In the case of cryptographic (or end-to-end) election verification, random challenges are often used to establish that the tally was computed correctly. In both cases, a source of randomness is required. In two recent binding cryptographic elections, this randomness was drawn from stock market data. This approach allows anyone with access to financial data to verify the challenges were generated correctly and, assuming market fluctuations are unpredictable to some degree, the challenges were generated at the correct time. It may be acceptable to the reader that financial data, like stock market prices, exhibits some unpredictable behaviour, but it is not likely intuitive how much randomness there is." [9] (Clark and Hengartner, 2010). Based on the above abstract part taken from the scientific paper mentioned above, is proved that financial data can be used as source of randomness. As a result, financial data such as stock market values can behave as source of entropy, since randomness can be drawn from these kind of data. Financial data such as the closing price of a stock of Dow Jones, is not only verifiable, since it's value is published every day from regulated official authorities but also is not able to be manipulated because it is very difficult to determine the actual price of a stock by certain actions. We can see now a series of simple steps on how we can derive entropy when we hold the closing price of Coca-Cola company (KO).

1st Step: We define the time period in which we want to gather information about closing price of Coca-Cola. In our case we gather the closing prices of Coca-Cola in the period 21/8/2009-19/8/2010 (one year). From the website finance.yahoo.com we can search for our data [10] (Finance.yahoo.com, 2018).

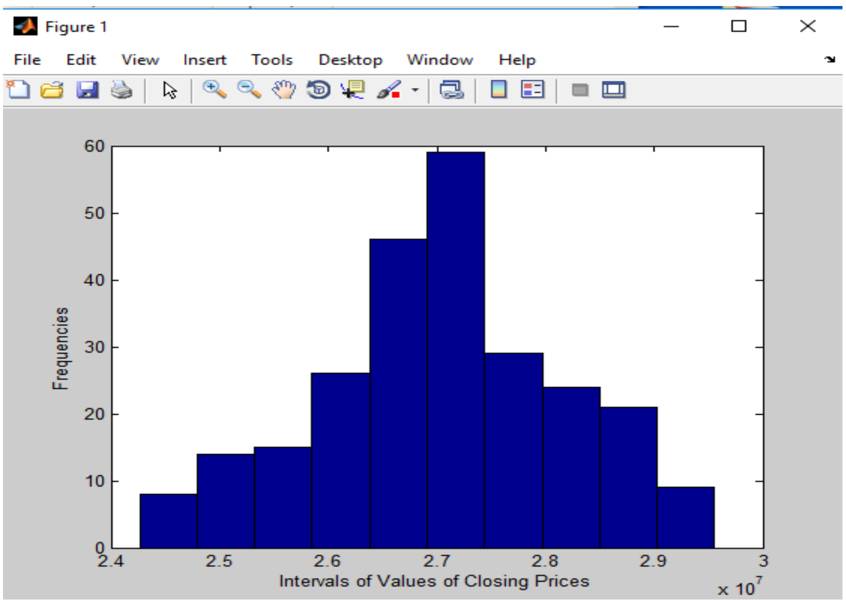
2nd Step: In a computational tool like Matlab, we can insert the data (closing prices) and the period T= days, concerning the number of trading days in the provided interval.

```
//inserting closing prices
>> Close

//estimating size, which is 251 days
>> SizeClose=size(Close)
```

3rd Step: In order to calculate the entropy, we need the frequencies. It is easy to capture the frequencies by plotting the closing prices in a histogram which has in the x axis the intervals of values of closing prices and in the y axis the frequencies.

```
//histogram of closing prices
>> hist(Close)
>> xlabel('Intervals of Values of
Closing Prices')
>> ylabel('Frequencies')
>> FreqCocaCola=hist(Close)
```

Figure 2.1: Histogram of closing prices of Coca Cola



<u>4th Step:</u> Estimating the relative frequencies. This is the step where we divide the values of frequencies with the sum of all frequencies.

```
>> RelFreqCocaCola=FreqCocaCola / sum (FreqCocaCola)

RelFreqCocaCola =

  Columns 1 through 7

    0.0319    0.0558    0.0598    0.1036    0.1833
0.2351    0.1155

  Columns 8 through 10

    0.0956    0.0837    0.0359
```

<u>5th Step:</u> Estimating the entropy. This is the last step of this procedure. We measure the outcome of entropy in bits. More bits are indicating more randomness. So, a source of randomness which gives a lot of bits of entropy could be a secure source for a draw.

```
>> EntropyCocaCola=-
RelFreqCocaCola*log2(RelFreqCocaCola')

EntropyCocaCola =

    3.0673
```

So, the entropy of the closing price of Coca-Cola in the period 21/8/2009-19/8/2010 (one year) is 3.0673 bits.

2.4 Capturing Entropy from Bitcoin and Proof of Work algorithm

Apart from using financial data, cryptocurrencies and more in particular the algorithms of cryptocurrencies requiring computational power for their solution. In these algorithms cryptocurrencies rely on and these algorithms are responsible for building systems with high entropy. And this can happen because randomness is needed for finding the solution of the algorithm. The solution based on this random procedure is vital in order the system to produce new coins and ensure its existence. So, at this point of the current chapter we will examine how it is possible to catch entropy from Bitcoin network taking advantage of the structure of Proof of Work algorithm.

A great source of entropy relies on the process of finding a solution for the Proof of Work algorithm. As we know, miners with their computational power trying every 10 minutes approximately to secure the network of Bitcoin. Bitcoin system and specifically the process of finding a solution for the Proof of Work algorithm is related with randomness and as a result with entropy. Randomness, actually can be found (is enclosed) in a number which miners trying to find in order to secure the system and to keep adding valid blocks of transactions to the blockchain transparent ledger. Miners, as trying to find this random number, they spent computational power. More in particular, the solution for this algorithm must be lower from a number target which changes from time to time in order the difficulty of finding a new block to consort with the increasingly computational power which miners spent. This adaption leads the Bitcoin system to produce by the procedure of mining, new bitcoins every 10 minutes approximately.

For simplicity, we can define the Proof of Work algorithm as the inequation below:
H(nonce || Prev_hash || tx || tx….|| tx) < target

- H(): The SHA256 hash function in which the Proof of Work algorithm is based on.
- nonce: a random number which has to be found in order to lead to a result. The value of the result must be lower than the target number.
- target(256 bits number): A number which changes based on the progress of computational power of miners. The bigger this number, the more difficult for the miners to find the appropriate hash (result of a hash function) which will satisfy the inequation.
- Prev_hash: The hash result of the previous valid block
- tx: The hash pointers of the transactions which have incorporated in this block.

To sum up, for a block to be valid it must hash to a value less than the current target. This means that for each block some work has been done in order to generate the block.

Bitcoin network uses the hash function SHA256 in order to come up with a desired solution. For this reason the hash is effectively a random number between 0 and 2** 256-1. However, this desired number is strictly related with the difficulty which changes every 2016 blocks based on the assumption that miners solve the algorithm every 10 minutes as we mentioned before. The difficulty parameter d, at the time (2/2/2018) writing this article is d= 2,603,077,300,219 [11] (Bitinfocharts.com, 2018) and based on this difficulty we can calculate the hash rate, hr, needed for finding the successful nonce in 10 minutes. But the hash rate is also given and it is hr=19,580,820,931,540,541,440 H/s (or 19.581 Ehash/s) [11] (Bitinfocharts.com, 2018) Based on d and hr, we can find now the probability (p=1 / tries) and afterwards the entropy (e). The current hash rate is hr=19,580,820,931,540,541,440 H/s. So we will estimate now the hr in Hashes/Second instead of Exa Hashes/Second. All the calculations done with the use of scientific calculator [12] (Web2.0calc.com, 2018).

X=19,580,820,931,540,541,440 hashes / second

So, in 10 minutes=10 * 60=600 seconds we have:

X* 600 tries➔19,580,820,931,540,541,440 * 600 tries➔11748492558924324864000 tries

So, the probability of a successful solution to the Proof of Work algorithm will be:

P= 1/ 11748492558924324864000 ➔ P=0.00000000000000000000008512

So, afterwards the entropy will be:

e= - log2(P)➔ e= -log2(0.00000000000000000000008512)➔ e=-(-73.31484803157024)➔ e= 73.31484803157024 of entropy. After the computations is clear that finding the solution to the Proof of Work algorithm is extremely hard and it needs tremendous computational power.

2.5 Capturing Entropy from Litecoin

After estimating the entropy that derives from Bitcoin network, it is time to compute also the entropy coming from the Litecoin. We chose this cryptocurrency mainly for two reasons. The first is that this cryptocurrency is well known in the cryptocurrency ecosystem and the second is because it depends its functionality and existence on a totally different algorithm called Scrypt algorithm. Litecoin is an open source, global payment network that is fully decentralized without any central authorities. Just like Bitcoin, Litecoin as a cryptocurrency is generated also by the procedure of mining. Litecoin however has different characteristics concerning the Proof of Work algorithm. Major differences with Bitcoin are among others the coin limit, the mean block time and the block reward details.

More in particular, Litecoin uses the Scrypt algorithm, originally named as s-crypt but pronounces as ‘script’. This algorithm incorporates the SHA256 algorithm, but its calculations are much more serialized than those of SHA256 in Bitcoin. Scrypt favours large amount of high speed RAM, rather than raw processing power alone. As a result, Scrypt is known as a memory hard problem. Miners compete for solving this particular Scrypt algorithm having a certain probability based on the current hash rate of the network of miners and also based on the estimated adjusted difficulty of the system. The difficulty parameter d of the time writing this article (2/2/2018) is d= 4,155,819 and the hash rate is hr= 125,528,158,035,059H/s (or 125.528 Thash/s) [13] (Bitinfocharts.com, 2018). Based on the above elements, we can find now the probability (p) of mining a block of Litecoin and afterwards the entropy (e) that resides in this probability.

The probability will be 1 / tries are occurred every 2.5 minutes, which is the duration for solving the Scrypt algorithm. Every 2.5 minutes (150 seconds) we will have:
X=125,528,158,035,059 hashes / second
So, in 2.5 minutes=2.5 * 60=150 seconds we have:
X* 150 tries➔125,528,158,035,059* 150 tries➔18829223705258850 tries

So, the probability of a successful solution to the Scrypt algorithm will be:
P= 1/ 18829223705258850
P=0.00000000000000005311

So, afterwards the entropy will be:
e= - log2(P)➔e= -log2(0.00000000000000005311)➔ e=-(54.06379408365442)➔e= 54.06379408365442 bits of entropy. We can see also here, that despite the fact that Litecoin network has lower bits of entropy than Bitcoin network, still the probability for finding the solution is extremely small and the solution needs tremendous hashing power.

2.6 Compare entropies and comment on big data for capturing entropy

In our previous analysis on capturing entropy from three different sources we came up with three different outcomes. The source with the higher entropy encapsulates more randomness and as a result it is more difficult for someone to manipulate a procedure or predict an outcome based on this entropy. In our case, we prove with mathematical calculations that electronic draws or lotteries based on Bitcoin protocol are more secure and random than potential draws which may use financial data like closing prices of a stock or Litecoin protocol as a source of randomness.

As we calculate previously, the probability for someone to solve the Proof of Work algorithm [14] (En.bitcoin.it, 2018) is P=0.00000000000000000000008512 based on the difficulty that we count our calculations. Imagine that the probability of winning the Greek Tzoker Lotto game is one in 24.435.180 or 0.0000000409246013 which is a number bigger than the probability of solving the Proof of Work. The lower the probability, the more difficult is for someone to make the right prediction.

In this project and more in particular in the project's algorithm that we are going to build, we will prove that the winner is the participant whose randomly given ticket (based on a unique id) can give the smallest outcome if it is concatenated with the seed, which in our case is the SHA256(blockHash) and afterwards this outcome hashed again with SHA256 hash function. In other words, winner is the one who owns the smallest SHA256(SHA256(blockHash) || ticket) number. The algorithm is secure because is depending not only on the blockHash which is a number that can be predicted with probability P=0.00000000000000000000008512 but also on the random procedure of sharing the tickets to the participants.

In this project, we are privileged to count the implementation of our lottery algorithm on open data. Open data as the name implies, is the data like the block hash number of the Bitcoin blocks, which should be freely available to everyone for use and republish. Moreover, this kind of data are public and can be reached easily. Open and big data for the above reasons and not only, are a great source of entropy under circumstances. These circumstances have to do with the verifiability, the transparency and the frequency of the data used for a purpose like implementing an electronic draw. Big data can guarantee fair procedures and eliminate cheaters of manipulating procedures because everything is public. So, the source of randomness can define the success of a draw procedure. Good source of randomness could be big data like the values of shares at stock market, the Bitcoin blocks, as we mentioned before and other cryptocurrencies, the flight landings, the weather conditions and GDP or unemployment rates just to mention a few.

2.7 Test cases where Bitcoin hash values used as source of entropy for implementing lotteries

Bitcoin block hashes can be used as a source of entropy in order to create transparent and verifiable lotteries. At this point we are going to present two different test cases where on the one hand the bytes of block hashes are used in a way in which there is a bias on the outcome of the lottery but on the other hand there is a correct usage of hashes of the blocks. So, on the one hand there is a false approach on how to take advantage and handle the bits of Bitcoin blocks and on the other hand we can see a correct application and usage of these open data.

In the case of Firelotto which is a company implementing online lotteries based on Bitcoin blockchain hash values and smart contracts [15] (Firelotto.io, 2018), transparency is not imply fairness. The company from its first steps has to solve a lot of issues concerning privacy of the transactions and randomness of its random generation algorithm. Firelotto has been criticized by users for its approach to selling tickets. The platform demands a wallet's private keys to participate in the lottery, which has led to accusations that Firelotto is attempting a scam [16] (Cryptovest, 2018). Moreover, there was a serious bug in the code which could let the owner (in our case Firelotto) to arbitrarily choose the winner. It was reported that the owner (drawer) can end the game choosing whatever Bitcoin hash they want [17] (GitHub, 2018). Finally, a research scientific post proved that their random generation algorithm is biased (modulo biased) [18] (Quora.com, 2018) and the distribution is skewed, favouring the smaller numbers [19] (Anon, 2018).

At this point, we will create a small example in order to understand what is modulo bias and how this bias affects Firelotto's algorithm fairness. Let's assume that we have a list of numbers A. The numbers in our list are: A = {30, 40, 50, 60, 70} and we want to pick randomly one of them. In order to choose one of our five numbers of the list A, a common action is to generate a random number R between a predefined range of 0…Z and then normalize the outcome by doing a modulo operation (%) on the length of our array A, which in our case is 5. For our example, we make the assumption that the random number R is between the range 0-12. In order to find the winning index, based on the modulo normalization, we have to perform the operation: Index=R%5 because the A list consists of 5 numbers possible to win based on their index. Giving any number to R between 0-12, we are sure that the outcome of the above operation will be between 0 and 4. So, for instance if our random number is R=9 then Index=9%5=4 which means that the number with index 4 is winning and this number is 70 in our case.

Despite the fact that this procedure of computing the winning number by taking advantage of the modulo operation seems to be fair and uniform, in reality it is not. And this is obvious for someone computing all the possible modulo operations for the random number R in the range of 0-12 for our array A as we did below.

Table 2.1: Modulo Operation Results

R value	Modulo 5	Result Index
0	0%5	0
1	1%5	1
2	2%5	2
3	3%5	3
4	4%5	4
5	5%5	0
6	6%5	1
7	7%5	2
8	8%5	3
9	9%5	4
10	10%5	0
11	11%5	1
12	12%5	2

By estimating the occurrences per index we end up having 3 times appearance of indices 0-2 and 2 times only for the index 3 and 4. This conclusion based on the distribution, gives higher probability for the numbers of array A with indices 0-2 to be picked as the winning numbers of a lottery. So, there is not a fair distribution and the procedure is obvious that is modulo bias.

Firelotto on its own algorithm replaces the random number R with values in the range of 0-255(usage only of 4 bytes of the block hash) and accordingly replaces the list A with numbers in the range of 1-20. This tactic suffers from modulo bias as we saw previously. To sum up, Firelotto's approaching on draws based on Bitcoin block hashes requires refactoring and redesign. At this point is important to mention that the analysis of modulo bias part in this chapter was done based on the scientific article of Konstantinos Chalkias with the name "Why Firelotto's blockchain-based random engine is not fair?" [19] (Anon, 2018).

On the other hand, there is a startup company called Saferandom which also implements electronic lotteries based on the Bitcoin block hashes as a source of entropy. In this case Saferandom uses all the bytes of the last Bitcoin block hash produced as the solution of Proof of Work algorithm and not only the last n bytes as Firelotto does. The algorithm is based on the function MIN(Hash(Source || token)) where: Hash, the hash function used for hashing the input. Source, bytes from the source of entropy, in this case the bytes of last bitcoin block hash. Token, participant's unique ticket.

In this approach all the hashed results are randomly stored in a list in which the first element finally wins the contest. The Saferandom algorithm is very close to the algorithm that it is going to be presented in this project and gives a fair and transparent outcome when it is used for electronic lotteries.

2.8 Summary

In this chapter there was a reference on what is entropy and also on the important role that entropy plays concerning randomness in electronic draws. Furthermore, there was a brief description on how entropy is calculated for sources like closing prices of stocks, Bitcoin network and Litecoin network. We end up, in the scope of this project that Bitcoin network is more secure and gives the highest level of randomness comparing to the other two sources of entropy. Finally, there was a description on an unsuccessful use of Bitcoin block hash for implementing electronic lotteries and on a successful use on the other hand. In the following chapter, there will be the presentation of the documentation of the code of the draw application.

Chapter 3 Introduction

3.1 Introduction

This chapter is referring to the documentation of the project and more in particular to the documentation of the code which is responsible for the construction of the draw algorithm. So, there will be a high level overview of the functionality of the project, a detailed analysis of all modules participating in the architecture of the system and a brief reference on the technologies used for the implementation of the project. To summarize, this chapter focuses on the technical implementation and not to the theoretical background of implementing electronic lotteries.

3.2 High level overview of functionality

The project simulates an electronic lottery. In this lottery each participant participates in the contest with his unique ticket of identification. This unique ticket given to each participant is a random generated number based on the length of the list of the participants. In our case, we choose the unique identifier to be a number, but it is acceptable to be a unique identifier in general, like the identity card number or the passport number for example. Every participant in the list can take part in the contest. Every contest is characterized by its unique id, timestamp, list of participants and of course by the seed which holds the appropriate randomness or better the bits of entropy in order the contest to hold security and randomness. In our case the seed is the SHA256(blockHash) value. Normally the seed is the hash of the last block (blockHash) of Bitcoin’s blockchain, but in our application we define the seed as the SHA256(blockHash). Based on the seed and the unique ticket, our algorithm defines the winner as the participant whose the outcome of SHA256(SHA256(blockHash) || ticket) is the smallest among all the other participant’s outcome. The winner’s ticket and unique identifier, the list of participants with their unique identifiers and moreover the information about the block used as source of entropy are stored in a file which is going to be stored in the Bitcoin’s blockchain. The information about the block contains data such as the block hash of the last mined block, the difficulty of the Bitcoin’s system, the nonce, which is the number that miners have to find in order to solve the Proof of Work algorithm, the timestamp of the current block, the hash of the previous block, the height of the current last block, the number of transactions and finally the seed which is the core element of the algorithm. The uniqueness of these data helping our algorithm in order to be transparent and secure and in order to be used as a proof of existence after the completion of a draw. After running successfully a contest, the above information is stored in a file. This file is hashed with the hash function SHA256 and afterwards with the aid of OP_RETURN transaction is stored as evidence in the blockchain of Bitcoin. This action is the vital key since everyone after the end of the lottery can validate the results of the contest on his own. Finally, after the brief description of the system we can say that our algorithm holds the features of security, randomness and transparency and for this reason could be used potentially for electronic lotteries. In the next subchapter there is a reference on the classes participating as modules to this Java based project and also reference on the architecture and the technology used for the implementation of the project.

3.3 Architecture and technology

In this part of the chapter we are going to mention the Java classes of the project in order the reader to have a general overview about the modules composing the project. These modules are responsible for the functionality of the system.

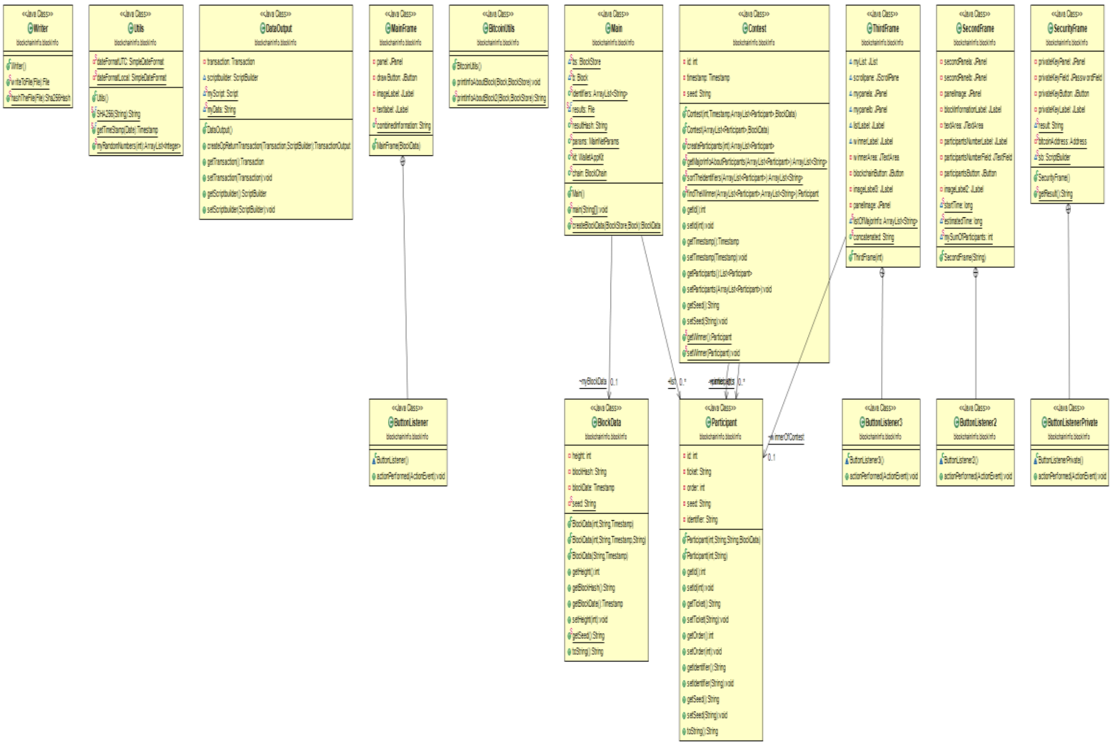
Table 3.1: Java classes of the project

A/A	**Class Name**
1	BitcoinUtils.java
2	BlockData.java
3	Contest.java
4	DataOutput.java
5	Main.java

6	MainFrame.java
7	Participant.java
8	SecondFrame.java
9	SecurityFrame.java
10	ThirdFrame.java
11	Utils.java
12	Writer.java

After pointing out the classes of our project we are going to visualize the architecture of our project by using a UML diagram. “The Unified Modeling Language (UML) is a general purpose, developmental, modeling language in the field of software engineering, that is intended to provide a standard way to visualize the design of a system.” [20] (En.wikipedia.org, 2018). So, with the aid of UML diagram we will have the opportunity to have a general overview of the connection of the modules of our system.

Figure 3.1: UML diagram of the system



At this point is important to mention the technology used for the completion of the project. The whole process of programming was completed using the Java programming language. More in particular, we took advantage of the BitcoinJ library [7] (Bitcoinj.github.io, 2018) of Java language in order to simulate the key entities of a Bitcoin environment through the API (Application Programming Interface) of the library [21] (Bitcoinj.github.io, 2018) and moreover because it was easy to fetch the

Bitcoin's blockchain information such as the block hash of the last block in order to achieve constructing our algorithm based on that information. Furthermore, for the graphical representation of some parts of the lottery process and also for making the application user friendly we used graphical components from the Swing library [22] (Docs.oracle.com, 2018) of Java. So, the aim of this chapter is to understand the reader, the functionality of the classes of the Java code and the functionality of their methods.

3.4 Analysis of modules

The analysis of the classes of the project is going to be done with alphabetical order. For each class the analysis consist of three parts, The first part is the description of the class, the second part is the presentation of its functions with the aid of a table and finally the third part has to do with the visualization of the class by using screenshots in order the reader to have a more solid overview on how the project looks like. Speaking about the third part it is important to mention that some of the screenshots refer to outcomes in the console of the debugging environment after running the program. The results in the console are indicators of successful running of the program. Moreover, for the three classes implementing graphical content, the MainFrame.java class, the SecondFrame.java class and the ThirdFrame.java class there is one descriptive analysis because all of them are using more or less same components and same or similar functions for their implementation of functionality. The graphical components of these classes are parts of the Java library Swing [22] (Docs.oracle.com, 2018)

3.4.1 BitcoinUtils class analysis

3.4.1.1 Description

This class after the procedure of fetching the last block of blockchain as object and the Blockstore object, is giving to the user some important information which can be used as proof of existence for transparency after the completion of running a draw. These information are very valuable also, because with the captured of the BlockHash of the last block, the user can create the seed (SHA256(blockhash)) which is going to be used for implementing the algorithm for the draw. This class has two functions, the first one is the printInfoAboutBlock() which prints in the console the captured information about the last captured block, and the printInfoAboutBlock2() function which returns all the captured information in the format of a string.

3.4.1.2 Table of functions

Table 3.2: BitcoinUtils class functions

A/A	Name	Description
1	printInfoAboutBlock(Block b, BlockStore bs)	This function takes as input parameters a BlockStore object and a Block object and prints out messages about the hash of the last block, the difficulty of the proof of work that this block should meet, the nonce which is an arbitrary value that exists only to make the hash of the block header fall below the difficulty target, the time at which the block was solved and broadcast, according to the clock of the solving node , the previous block hash, the last block height and the number of transactions of the last block
2	printInfoAboutBlock2(Block b, BlockStore bs)	This function takes as input parameters a BlockStore object and a Block object and prints out one message about the hash of the last block, the difficulty of the proof of work that this block should meet, the nonce which is an arbitrary value that exists only to make the hash of the block header fall below the difficulty target, the time at which the block was solved and broadcast, according to the clock of the solving node , the previous block hash, the last block height and the number of transactions of the last block

3.4.1.3 Screenshots

Figure 3.2: Functions of BitcoinUtils class

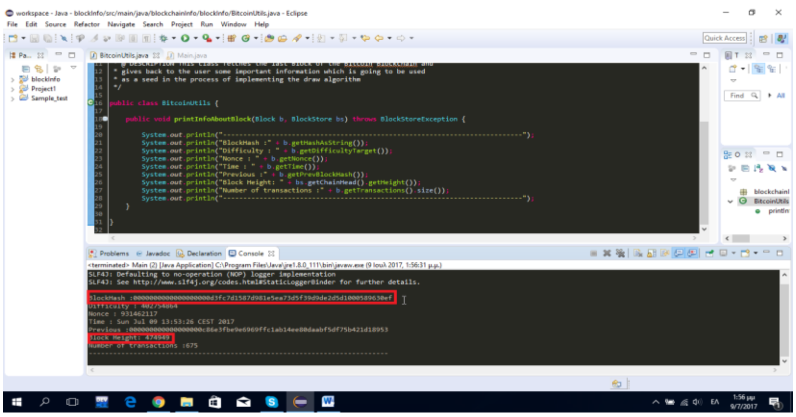
```
public class BitcoinUtils {

    public void printInfoAboutBlock(Block b, BlockStore bs) throws BlockStoreException {

        System.out.println("BlockHash : " + b.getHashAsString());
        System.out.println("Difficulty : " + b.getDifficultyTarget());
        System.out.println("Nonce : " + b.getNonce());
        System.out.println("Time : " + b.getTime());
        System.out.println("Previous : " + b.getPrevBlockHash());
        System.out.println("Block Height : " + bs.getChainHead().getHeight());
        System.out.println("Number of transactions : " + b.getTransactions().size());
    }

    public static String  printInfoAboutBlock2(Block b, BlockStore bs) throws BlockStoreException {
        String myInfoBlockOutcome="BlockHash : " + b.getHashAsString() + "\n" + "Difficulty : " + b.getDifficultyTarget()
        + "\n" + "Nonce : " + b.getNonce() + "\n" + "Time : " + b.getTime() + "\n" + "Previous : " + b.getPrevBlockHash()
        + "\n" + "Block Height : " + bs.getChainHead().getHeight() + "\n" + "Number of transactions : " + b.getTransactions().size();

        return myInfoBlockOutcome;
    }

}
```

3.4.1.4 Validation of Captured Block

During this phase of capturing data about the seed coming from the hash of the last block of Bitcoin blockchain, we have to be sure that the information of our last block captured in Java code is the same with the information published in the Bitcoin blockchain. For this reason we have to provide evidence of our outcome.

After running the Main.java class, the printInfoAboutBlock() function was called inside the BitcoinUtils.java class and the console of eclipse returned the below results.

Figure 3.3: Validation of captured block, step 1



The second step requires checking the validity of returned data with the data existing in a website which captures the blockchain activity. In our case we use the website http://blockr.io/ to check if the data is the same. As we can see from the below screenshots the data concerning the last block in the Bitcoin blockchain at the time we run our program are the same with the data from the website.

Figure 3.4: Validation of captured block, step 2

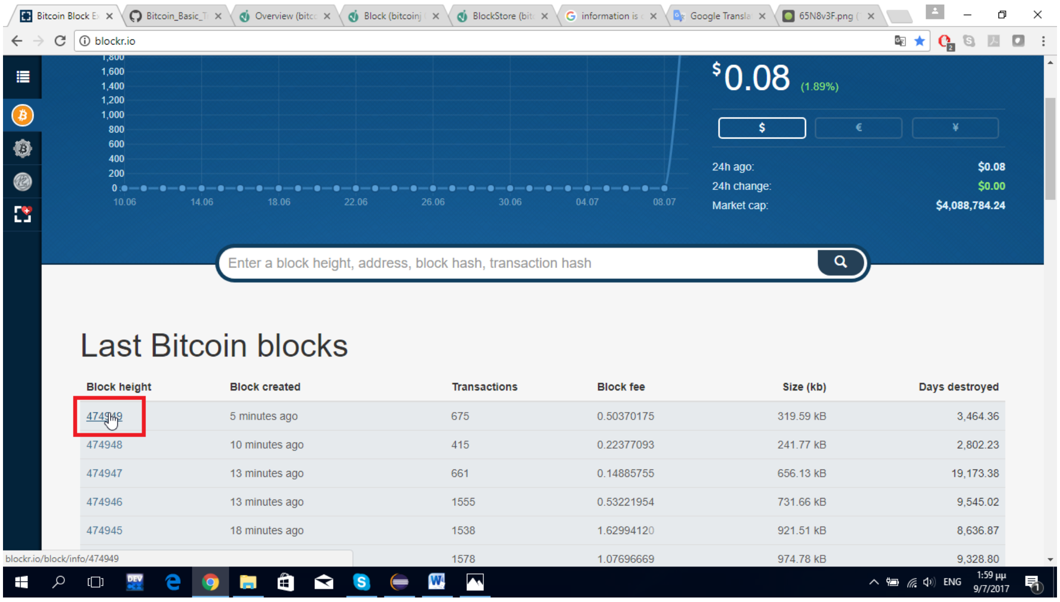
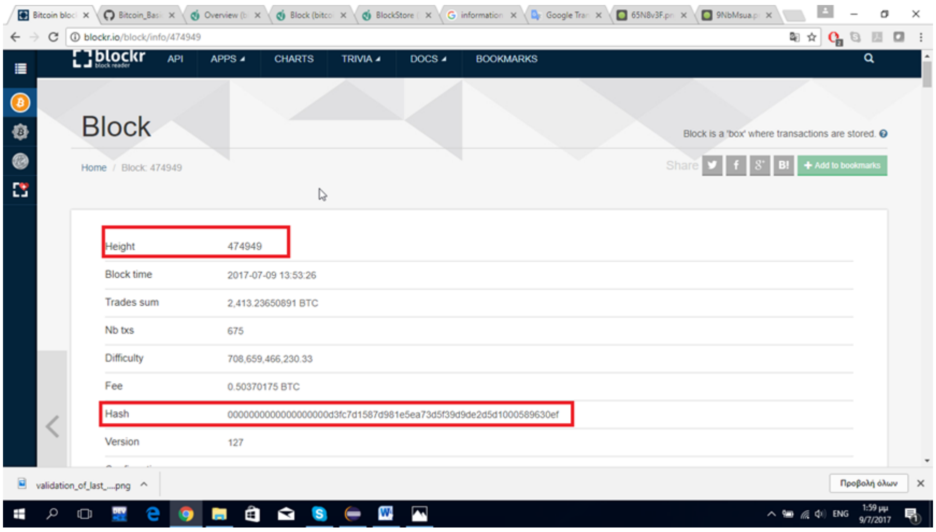


Figure 3.5: Validation of captured block, step 3



So, we can guarantee that we captured the valid data needed concerning our seed.

3.4.2 BlockData class analysis

3.4.2.1 Description

This java class simulates a block (the latest one) which was captured by the Bitcoin network. The class contains information about the height of the block concerning the sequence in the Bitcoin blockchain, the blockHash which is a unique number of leading zeros which adapts based on the difficulty to find a solution to the Proof of Work algorithm, the blockDate which is the time at which the block was solved and broadcast, according to the clock of the solving node and finally the seed which is the outcome of SHA256(blockHash), a number which holds entropy and will be used as a seed to our draw algorithm.

3.4.2.2 Table of functions

Table 3.3: BlockData class functions

A/A	Name	Description
1	getHeight()	Returns the height of the current capture block
2	getBlockHash()	Returns the hash number of the current capture block which is a number of leading zeros and is calculating by doing SHA256(SHA256(Block_Header))
3	getBlockDate()	Returns the time at which the block was solved and broadcast, according to the clock of the solving node
4	setHeight(int height)	Changes the height of the block according to the input parameter. The height however can not change because along with the blockHash are unique attributes of a block
5	getSeed()	Returns the seed of the current block, which is the SHA256(blockHash)
6	toString()	Returns all the information concerning a block in a custom way

3.4.2.3 Screenshots

Figure 3.6: Attributes of the BlockData class

```java
package blockchainInfo.blockInfo;

import java.sql.Timestamp;

 * @author Giorgos Topalidis

/*
 * @ DESCRIPTION This class contains all the appropriate information
 * concerning the data of a block
 */

public class BlockData {

    private int height; // the block height
    private String blockHash; // the hash of the block
    private Timestamp blockDate; // the time at which the block was solved and broadcast, according to the clock of the solving node
    private String seed; // the sha256(blockHash)

    public BlockData(int height, String blockHash, Timestamp blockDate) {
        this(blockHash, blockDate);
        this.height = height;

    }
```

Figure 3.7: Console output with the information of the last captured block

```
BlockHash :0000000000000000009f308febd1a198e85a56a26ac70ace49d3b84bdf4d0f5c
Difficulty : 402736949
Nonce : 2354805029
Time : Sun Jul 30 14:29:25 CEST 2017
Previous :0000000000000000001056816ba73e6fbb75f88c2aae220cd33f4a75689679d9
Block Height: 478259
Number of transactions :2060

BlockData [height=478259
blockHash=0000000000000000009f308febd1a198e85a56a26ac70ace49d3b84bdf4d0f5c
blockDate=2017-07-30 12:29:25.0
seed=e330fa1574b3ed64b491f4df1fab92fdd2cce60fddd4af98695eb06d739ac71e]
```

3.4.3 Contest class analysis

3.4.3.1 Description

This class simulates the procedure of a draw or lottery based on the ids of the participants which produced randomly from the createParticipants(int numberOfParticipants) function. According to these ids, every participant will produce its unique identifier which constitutes its unique ticket. In the function sortTheIdentifiers(ArrayList<Participant>participantList) the class sorts the unique identifiers for every participant in ascending order. The identifiers are type of String in our program, so it is easy to sort these strings alphabetically. We define as a rule of our contest that the winner will be this participant whose identifier will be the first of our sorted identifiers in the list of identifiers. Just to remind that the identifier String is the concatenation between the SHA256(blockHash) and the ticket of the participant, which is a unique string for every participant based on the random produced id. In this place we have to mention that we do not care about the id and as a result the ticket itself, but we care about the seed which in our case is the SHA256(blockHash). This value defines the winner every time (per 10 minutes) we run the algorithm, because this value holds randomness and the bits of entropy it produced are sufficient to make us use this value as a source of randomness (seed). So finally, the findTheWinner(ArrayList<Participant> participantList,ArrayList<String> identifiers***)*** function help us find the winner, which is the first value located in index with number zero in our sorted list. In order the function to define our winner it takes as input parameters the list of participants which have been created randomly (the ids) and the sorted list of the identifiers from all the participants of the current contest.

The attributes of this class are the id which is the id of the contest and is useful for identification of a contest if we want to store the contest details to a database, the timestamp which is the attribute which gives information about the time that the contest took place, the participants which is the list of all participants in the contest, the seed which is the SHA256(blockHash) value. Normally the seed is the blockHash, but in our application we define the seed as the SHA256(blockHash) and finally the winner which is the number of the id participant, which ticket is based on this unique id. Winner is the participant whose the outcome of SHA256(SHA256(blockHash) || ticket) is the smallest among all the other.

For my own convenience and in order to have only the appropriate and major information to display in my graphical user interface, I create the getMajorInfoAboutParticipants(ArrayList<Participant> pList***)*** function which captures and returns in an arraylist of strings only the id and the unique identifier of every participant.

3.4.3.2 Table of functions

Table 3.4: Contest class functions

A/A	Name	Description
1	createParticipants(int numberOfParticipants)	This function takes as input a number which simulates how many participants are going to participate and enroll in our contest. Afterwards creates a list of random numbers based on this input number and finally returns a list of participants with random ids. The size of this list is equal with the number of the input of the function
2	sortTheIdentifiers(ArrayList<Participant> participantList)	This function takes as input the list of the participants which has created in the previous function. Afterwards, accesses the value of identifier through the function getIdentifier() for every participant in the list and adds this value to a list of strings. Then the function sorts this list of string alphabetically and returns it
3	findTheWinner(ArrayList<Participant> participantList,ArrayList<String> identifiers)	This class takes as input the list of the participants from the

		createParticipants() function and the list of the sorted identifiers from the sortTheIdentifiers() function and defines the winner based on the identifier which included in the position with index zero of the sorted list of identifiers. This selection is done because we define the rule of alphabetically order in our list of identifiers
4	getId()	Returns the id of the contest
5	setId(int id)	Changes the id of the contest based on the input parameter
6	getTimestamp()	Returns the Timestamp of the contest
7	setTimestamp(Timestamp timestamp)	Changes the Timestamp of the contest based on the input parameter
8	getParticipants()	Returns the list of the participants of the contest
9	setParticipants(List<Participant> participants)	Changes the list of the participants based on the input parameter list
10	getSeed()	Returns the seed of the contest
11	setSeed(String seed)	Changes the seed of the contest based on the input new seed. This function maybe can have use when

		we want to choose a different seed from SHA256(blockHash) to be our random source of entropy
12	getWinner()	Returns the number of the id of the participant whose ticket is based on this unique id
13	setWinner(int winner)	Changes the winning number based on the input integer
14	getMajorInfoAboutParticipants(ArrayList<Participant> pList)	This class takes as input the list of the participants and returns a list of strings with information only about the id and the unique identifier of every participant

3.4.3.3 Screenshots

Figure 3.8: Attributes of the Contest class

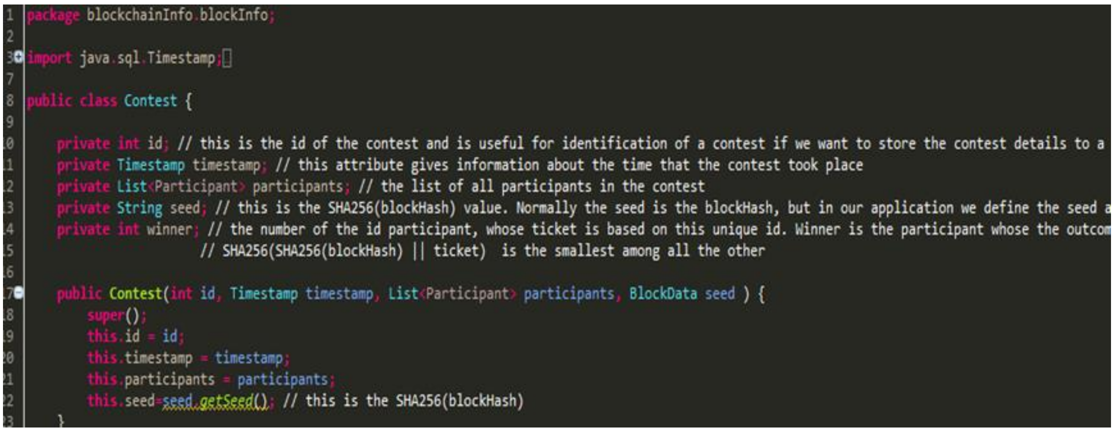
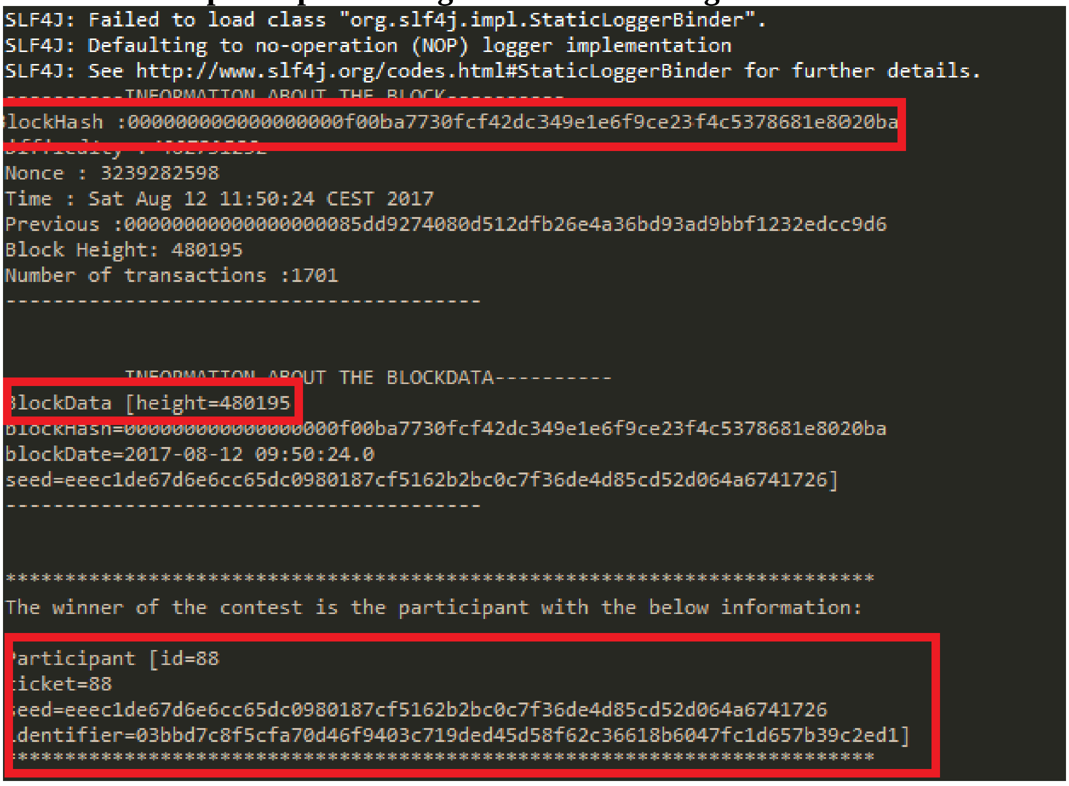


Figure 3.9: Console output with the information after running a contest with 100 participants using the block with height=480195



```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
----------INFORMATION ABOUT THE BLOCK----------
BlockHash :000000000000000000f00ba7730fcf42dc349e1e6f9ce23f4c5378681e8020ba
Difficulty : 402751232
Nonce : 3239282598
Time : Sat Aug 12 11:50:24 CEST 2017
Previous :00000000000000000085dd9274080d512dfb26e4a36bd93ad9bbf1232edcc9d6
Block Height: 480195
Number of transactions :1701
----------------------------------------


----------INFORMATION ABOUT THE BLOCKDATA----------
BlockData [height=480195
blockHash=000000000000000000f00ba7730fcf42dc349e1e6f9ce23f4c5378681e8020ba
blockDate=2017-08-12 09:50:24.0
seed=eeec1de67d6e6cc65dc0980187cf5162b2bc0c7f36de4d85cd52d064a6741726]
----------------------------------------


**************************************************************************
The winner of the contest is the participant with the below information:

Participant [id=88
ticket=88
seed=eeec1de67d6e6cc65dc0980187cf5162b2bc0c7f36de4d85cd52d064a6741726
identifier=03bbd7c8f5cfa70d46f9403c719ded45d58f62c36618b6047fc1d657b39c2ed1]
**************************************************************************
```

Figure 3.10: Console output with the information after running a contest with 100 participants using the block with height=480196

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (12 Auy 2017, 12:15:17 μ.μ.)
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
----------INFORMATION ABOUT THE BLOCK----------
BlockHash :0000000000000000001887df063d50d1cd6c0c8ce1c6180f2b72be157cd0ebaf
Difficulty : 402751232
Nonce : 1641741838
Time : Sat Aug 12 12:14:46 CEST 2017
Previous :000000000000000000f00ba7730fcf42dc349e1e6f9ce23f4c5378681e8020ba
Block Height: 480196
Number of transactions :2082
----------------------------------------


----------INFORMATION ABOUT THE BLOCKDATA----------
BlockData [height=480196
blockHash=0000000000000000001887df063d50d1cd6c0c8ce1c6180f2b72be157cd0ebaf
blockDate=2017-08-12 10:14:46.0
seed=a198b3925e51a9e4d848c4d2032c0586ee5c28f8b4d1d9b8ee96d1e07d1bb81f]
----------------------------------------


**************************************************************************
The winner of the contest is the participant with the below information:

Participant [id=26
ticket=26
seed=a198b3925e51a9e4d848c4d2032c0586ee5c28f8b4d1d9b8ee96d1e07d1bb81f
identifier=00f8162da1dfd53df8cfa6fb4c479842c2b054ef25fde348a2b6a20cd9e7bc77]
**************************************************************************
```

Figure 3.11: Console output with the information after running a contest with 1000 participants using the block with height=480198

```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
          INFORMATION ABOUT THE BLOCK
BlockHash :00000000000000000025b2bf6251ff2b67bbf5a017b26d2cc8db39f4588b8367
Difficulty : 402731232
Nonce : 47983225
Time : Sat Aug 12 12:28:51 CEST 2017
Previous :0000000000000000010b25e437d7e2cd61d8eb39cf83d3f1eae19617291e6a5f
Block Height: 480198
Number of transactions :1527
----------------------------------------


----------INFORMATION ABOUT THE BLOCKDATA----------
BlockData [height=480198
blockHash=00000000000000000025b2bf6251ff2b67bbf5a017b26d2cc8db39f4588b8367
blockDate=2017-08-12 10:28:51.0
seed=056d4453ee2b4bd353312a89a149f815ab0c12d85bf55882f887d04471815f508]
----------------------------------------


*************************************************************************
The winner of the contest is the participant with the below information:

Participant [id=98
ticket=98
seed=056d4453ee2b4bd353312a89a149f815ab0c12d85bf55882f887d04471815f508
identifier=001c4005194c2d63b54ea031c364f61372ff9e402fa602a425899ae52e50e94b]
*************************************************************************
```

Figure 3.12: Console output with the information after running a contest with 1000 participants using the block with height=480199

```
<terminated> Main (2) [Java Application] C:\Program Files\Java\jre1.8.0_111\bin\javaw.exe (12 Αυγ 2017, 12:36:12 μ.μ.)
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
          INFORMATION ABOUT THE BLOCK
BlockHash :000000000000000000ce41d36f6523624239b5b913273f9c3713cdc4d2fb0c94
Difficulty : 402731232
Nonce : 1933285770
Time : Sat Aug 12 12:35:32 CEST 2017
Previous :00000000000000000025b2bf6251ff2b67bbf5a017b26d2cc8db39f4588b8367
Block Height: 480199
Number of transactions :2093
----------------------------------------


          INFORMATION ABOUT THE BLOCKDATA----------
BlockData [height=480199
blockHash=000000000000000000ce41d36f6523624239b5b913273f9c3713cdc4d2fb0c94
blockDate=2017-08-12 10:35:32.0
seed=249c9a60d23cd48218138555b03b2319784b48b896e9100f0297035c36d9c644]
----------------------------------------


*************************************************************************
The winner of the contest is the participant with the below information:

Participant [id=764
ticket=764
seed=249c9a60d23cd48218138555b03b2319784b48b896e9100f0297035c36d9c644
identifier=00124a797bf7b57d228ceec090a2722e8338e7b3b726cf4042c45ef5b842524f]
*************************************************************************
```

As we can see from the information captured from the above screenshots, every time we run a contest and this can happen every 10 minutes, because every 10 minutes we have a new solution to the Proof of Work algorithm, in other words every 10 minutes we have a new different seed to feed our algorithm, we have finally a different winner. This is the ideal approach. However, based on our construction of the algorithm, in our contests we can have more than one draws with the same seed because the identifier is based on a random given id, apart from the seed. If we choose to have the identity number as id for example, then we could make only one draw every 10 minutes.

In our examples, we experiment by running the algorithm for 100 participants for the blocks 480195 and 480196 at the first experiment and running the algorithm for 1000 participants for the blocks 480198 and 480199 at the second experiment. Every time we run the algorithm we end up having a different winner. And this is the expected outcome since every time we rely on different seed, the SHA256(blockHash), to be our source of randomness and our algorithm and the procedure of draw has nothing to do with the ids and the identifiers of every participant which produced with a random way. Even if the ids and the identifiers were fixed and known, the outcome will be again different but it will not rely on the fixed values but only from the source of randomness.

To summarize our findings, we provide the outcomes of our experiments in the next table. The reader can easily check the results of my table by calculating the SHA256 results in an online SHA-256 hash calculator [23] (Xorbin.com, 2018).

Table 3.5: Outcomes of running the algorithm for 100 participants for the blocks with height 480195 and 480196 and running the algorithm for 1000 participants for the blocks with height 480198 and 480199

A/A	Number of experiment	Block Height	Block Hash	Number of Participants	Seed	Id of winner	Identifier of winner
1	1	480195	0000 0000 0000 0000 00f0 0ba7 730f	100	eeec 1de6 7d6e 6cc6 5dc0 9801 87cf	88	03bb d7c8 f5cf a70d 46f9 403c 719d

			cf42 dc34 9e1e 6f9c e23f 4c53 7868 1e80 20ba		5162 b2bc 0c7f 36de 4d85 cd52 d064 a674 1726		ed45 d58f 62c3 6618 b604 7fc1 d657 b39c 2ed1
2	1	480196	0000 0000 0000 0000 0018 87df 063d 50d1 cd6c 0c8c e1c6 180f 2b72 be15 7cd0 ebaf	100	a198 b392 5e51 a9e4 d848 c4d2 032c 0586 ee5c 28f8 b4d1 d9b8 ee96 d1e0 7d1b b81f	26	00f8 162d a1df d53d f8cf a6fb 4c47 9842 c2b0 54ef 25fd e348 a2b6 a20c d9e7 bc77
3	2	480198	0000 0000 0000 0000 0025 b2bf 6251 ff2b 67bb f5a0 17b2 6d2c c8db 39f4 588b 8367	1000	056d 4453 ee2b 4bd3 5331 2a89 a149 f815 ab0c 12d8 5bf5 5882 f887 d047 1815 f508	98	001c 4005 194c 2d63 b54e a031 c364 f613 72ff 9e40 2fa6 02a4 2589 9ae5 2e50 e94b
4	2	480199	0000 0000	1000	249c 9a60	764	0012 4a79

			0000 0000 00ce 41d3 6f65 2362 4239 b5b9 1327 3f9c 3713 cdc4 d2fb 0c94		d23c d482 1813 8555 b03b 2319 784b 48b8 96e9 100f 0297 035c 36d9 c644		7bf7 b57d 228c eec0 90a2 722e 8338 e7b3 b726 cf40 42c4 5ef5 b842 524f

3.4.4 DataOutput class analysis

3.4.4.1 Description

This class simulates the creation of an output of OP_RETURN transaction. The main functionality has to do with taking the SHA256 of the information included in the file and hashing with SHA256 for the second time the outcome. As we have mentioned before in the file the user stored information about the current block which is used, information about the winner and information about the participants. After the second hashing the class creates a script which imports the information of the file as output. This output is the evidence for a secure and transparent draw which is going to be stored in the blockchain as a proof of evidence.

3.4.4.2 Table of functions

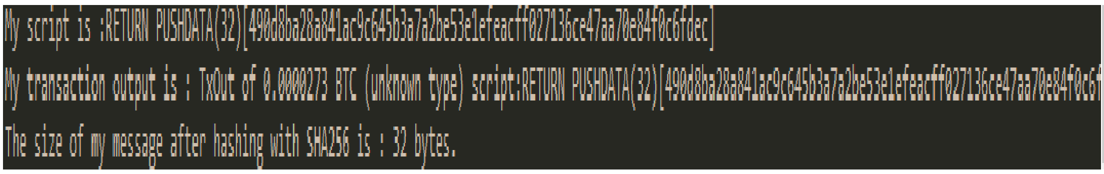
Table 3.6: DataOutput class functions

A/A	Name	Description
1	createOpReturnTransaction(Transaction tx, ScriptBuilder sb)	This function has access to the hash of the file which contains the important information about the contest. Moreover, hashes for the second time the hashing result of the file and afterwards creates a

		script and includes to this the SHA256(SHA256(data)) as an output of the OP_RETURN transaction. Finally this function prints to the console information about the structure and the size of the script
2	getTransaction()	This function returns the transaction of OP_RETURN type
3	setTransaction(Transaction transaction)	This function sets the transaction of OP_RETURN type
4	getScriptbuilder()	This function returns the Scriptbuilder which is responsible for building the transaction
5	setScriptbuilder(ScriptBuilder scriptbuilder)	This function sets the Scriptbuilder which is responsible for building the transaction

3.4.4.3 Screenshots

Figure 3.13: Information in the console concerning information about the size and the structure of OP_RETURN transaction



3.4.5 Main class analysis

3.4.5.1 Description

The main class is the starting point for running the application. This class calls the appropriate functions and creates objects in order the organizer of the lottery to be able to run a contest. More in particular, in this class there is a declaration and construction of objects which help for initializing a connection with the blockchain database. In other words, the class is setting up a Simplified Payment Verification (SPV) node [24] (Bitcoin.org, 2018) which is used in order not to download the full size of the blockchain but instead only the headers of the blocks during the initial synchronization process with the blockchain. And the project initializes a SPV node because it only needs the information which are encapsulated in the last block of the blockchain. These information contain the randomness that our algorithm needs.

Furthermore, in the comment area of the class there are several snippets of code which created in order to test certain functions at the stage of implementation. For experimental reasons the code of these snippets is visible. This code has to do with printing information about the last block of blockchain, printing information about the BlockData class, printing information about the participant of the contest, creation of 100 random numbers between 1-100, acting like participant's id and finally sorting the 100 numbers in order to check the order and the uniqueness.

3.4.5.2 Table of functions

Table 3.7: Main class function

A/A	Name	Description
1	createBlockData(BlockStore bs,Block b)	This function takes as input parameters a BlockStore object and a Block object and returns a BlockData object. In order to return this BlockData object the function first has to initialize it by capturing the height, the blockHash, the blockDate and the seed. The capturing of these information is feasible with the aid of BlockStore which is a map of hashes and with the aid of Block object which is a group of transactions.

3.4.5.3 Screenshots

Figure 3.14: Attributes of the Main class

```
public class Main {

    static BlockStore bs;
    static Block b;
    public static ArrayList<Participant> list=new ArrayList<Participant>();
    public static ArrayList<String> identifiers=new ArrayList<String>();
    static BlockData myBlockData;
    final static File results = new File("C:\\Users\\Maria\\workspace\\blockInfo\\Contest_Results.txt");
    public static String resultHash;
    public static MainNetParams params = MainNetParams.get();
    public static WalletAppKit kit;
    public static BlockChain chain;

    public static void main(String[] args) throws BlockStoreException, InterruptedException, ExecutionException {

         kit = new WalletAppKit(params,new java.io.File("."), "test");
        kit.startAndWait();
        chain = kit.chain();
         bs = chain.getBlockStore();
        Peer peer = kit.peerGroup().getDownloadPeer();
         b = peer.getBlock(bs.getChainHead().getHeader().getHash()).get();
```

3.4.6 GUI classes analysis

3.4.6.1 Description

We are not going to analyze in depth the three classes which taking advantage of graphical components, because they are using a plenty of graphical components and the analysis of these elements is not in our scope. However, we are going to explain the usage of these classes and how they connect with each other in order to provide to the user a graphical user interface which is more user friendly than running the application and see its results in the console.

The three classes, MainFrame.class, SecondFrame.class and ThirdFrame.class are extending the utility of the JFrame class and taking advantage of the several graphical components provided by this class. The reader can find documentation of the JFrame class (Docs.oracle.com, 2018). In our case, in order to build our graphical user interface we use the most usual components for building a graphical interaction which are explained in the below table.

3.4.6.2 Table of functions

Table 3.8: Major JFrame components

A/A	Name	Description
1	JPanel	Creation of a panel, in which the user adds all the components in order to be visible
2	JButton	A button which is clickable and triggers certain actions when it is clicked
3	JLabel	A component which provides a label ,which usually explains something, or is indication of something to our graphical environment
4	JTextArea	An area, in which the user can display information
5	JTextField	A field, in which the user can write a text and use it afterwards
6	JList	A graphical list, usually scrollable in which the user shows up information about a list of instances
7	ImageIcon	An implementation of the Icon interface that paints Icons from Images

3.4.6.3 Screenshots

The MainFrame class creates the main window which shows the main screen of the draw implementation. It contains a button for running a contest.

Figure 3.15: Graphical user interface of MainFrame class

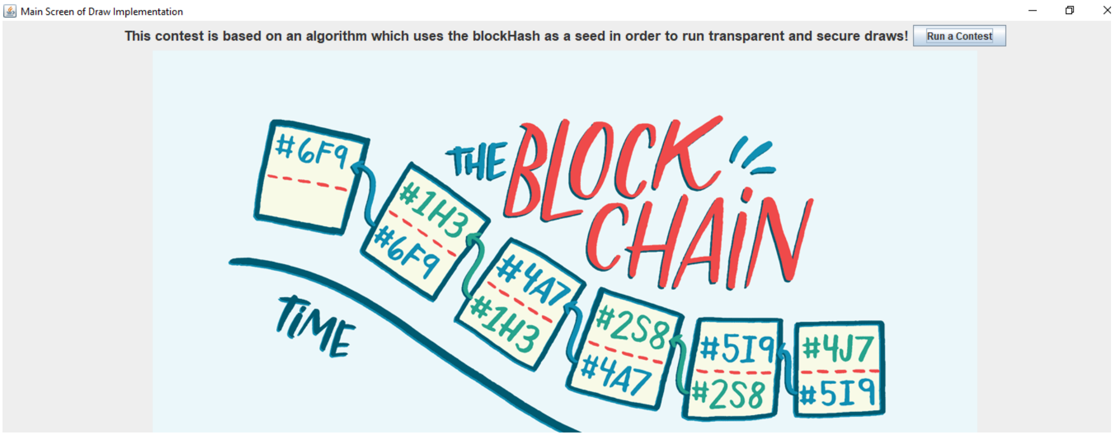


Figure 3.16: Attributes and functions of MainFrame class

```java
package blockchainInfo.blockInfo;

import java.awt.Font;


public class MainFrame extends JFrame {


    private JPanel panel;
    private JButton drawButton;
    private JLabel imageLabel;
    private JLabel textlabel;

    public MainFrame(BlockData myBlockData) {
        panel = new JPanel();
        ImageIcon image = new ImageIcon("C:\\Users\\Maria\\workspace\\blockInfo\\src\\main\\java\\blockchainInfo\\blockInfo\\BlockChain-An
        Font font = new Font("Blue", Font.BOLD, 17);

        textlabel = new JLabel(
                "This contest is based on an algorithm which uses the blockHash as a seed in order to run transparent and secure draws!");
        textlabel.setFont(font);
        panel.add(textlabel);

        drawButton = new JButton("Run a Contest");
        panel.add(drawButton);

        ButtonListener listener = new ButtonListener();

        drawButton.addActionListener(listener);
```

The SecondFrame class presents to the user all the appropriate information concerning the last block fetched by the Bitcoin blockchain. The user can see in the text area all the vital information about the block, the block hash of which is going to be used as a seed for our draw algorithm.

Moreover, in this frame the user who is the responsible for running a draw, can define the number of the participants of the draw by writing down the number in the provided text field area. Afterwards, by pressing the button, the draw algorithm is running, giving to us information about the participants and of course about the winner of the contest.

Figure 3.17: Graphical user interface of SecondFrame class

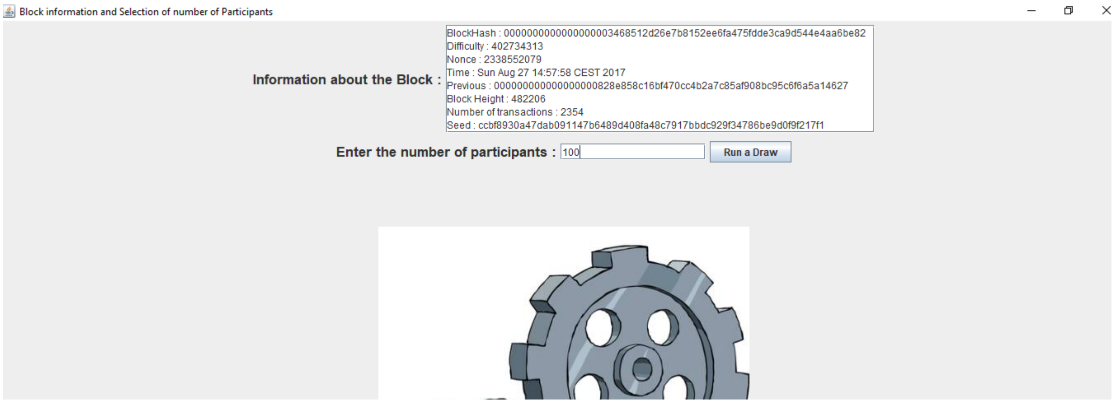


Figure 3.18: Attributes and functions of SecondFrame class

```java
package blockchainInfo.blockInfo;

import java.awt.BorderLayout;


public class SecondFrame extends JFrame {

    private JPanel secondPanela;
    private JPanel secondPanelb;
    private JPanel panelImage;

    private JLabel blockInformationLabel;// these components will be added to
                                          // the panela
    private JTextArea textArea;

    private JLabel participantsNumberLabel;// these components will be added to
                                          // the panelb
    private JTextField participantsNumberField;
    private JButton participantsButton;

    private JLabel imageLabel2;

    public SecondFrame(String myInput) throws IOException {

        secondPanela = new JPanel();
        secondPanelb = new JPanel();
        panelImage = new JPanel();
```

The ThirdFrame class finally displays to the user with the aid of a JList the list of the participants. More in particular the list encapsulates information about the id and the unique identifier of every single participant of this draw. In addition, the text area of this frame shows the information about the winner. Both the list and the information about the winner are going to be stored as a next step in the Bitcoin blockchain in order to stay there as a proof of existence. This action will happen if the user presses the “Store draw result in Blockchain!” button. This storing activity to the blockchain is a unique action which could be implemented with the aid of OP_RETURN transaction, which is supported by the Bitcoin network and is responsible for storing information bytes as an evidence in the blockchain.

Figure 3.19: Graphical user interface of ThirdFrame class

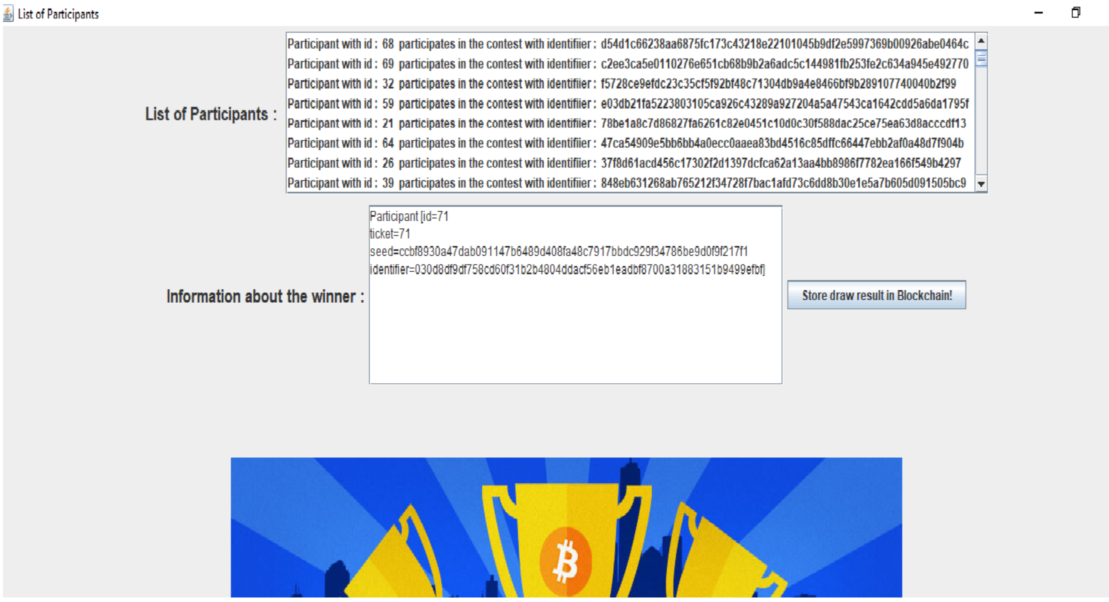


Figure 3.20: Attributes and functions of ThirdFrame class

```java
public class ThirdFrame extends JFrame {

    JList myList;
    JScrollPane scrollpane;

    JPanel mypanela;
    JPanel mypanelb;

    JLabel listLabel;
    JLabel winnerLabel;

    private JTextArea winnerArea;
    private JButton blockchainButton;

    private JLabel imageLabel3;
    private JPanel panelImage;



    public ThirdFrame(int numberOfParticipants){
```

3.4.7 Participant class analysis

3.4.7.1 Description

This java class simulates a person who participates in the procedure of draw. Every participant must be identified by a unique ticket such as his identity number or his passport number, a unique id, a unique e-mail address or whatever gives to a participant a uniqueness. In our case the id in a string format will be the unique ticket. The id of the participant denotes a number which is generated by a random function which sets the limits of the produced numbers, according to the sum of the participants. In other words a random generator gives to every participant a random generated number which is going to be his unique ticket.

Despite the fact that in our case the ticket is produced by a function based on the random unique id for every participant, this procedure is not mandatory. And this conclusion is easily understood because the draw algorithm is affected only from the randomness that has the block hash and not from any other external factor although this external factor concatenates its value with the SHA256(blockhash) and the result is hashing again with SHA256 in order to produce the unique final identifier which is the unique representative identifier for every participant.

This class contains information about the id, the ticket which is a unique identifier for every participant, the order which is a number which simulates how near or how far the identifier string is from the hash block solution. The outcome of ordering will define the winner of the draw, since the shorter distance that has the identifier from the hash block is the rule that defines the winner. Moreover, in this class we have information about the seed which is the SHA256(blockHash) and finally the identifier which is the outcome of SHA256(SHA256(blockHash) || ticket) and is the unique evidence of every participant to the contest.

3.4.7.2 Table of functions

Table 3.9: Participant class functions

A/A	Name	Description
1	getId()	Returns the id number of the participant
2	setId(int id)	Changes the id of the participant (this is not going to happen in our application, because the id is acting as a unique feature to produce the ticket)
3	getTicket()	Returns the ticket of the participant
4	setTicket(String ticket)	Changes the ticket of the participant (forbidden action in our application)
5	getOrder()	Returns the order of the participant based on the outcome of the draw result
6	setOrder(int order)	Changes the order of the participant (forbidden action in our application)
7	getIdentifier()	Returns the identifier of the participant
8	setIdentifier(String identifier)	Changes the identifier of the participant (forbidden action in our application)
9	getSeed()	Returns the seed of the participant

10	setSeed(String seed)	Changes the seed of the participant (forbidden action in our application)
11	toString()	Returns all the information concerning a participant in a custom way

3.4.7.3 Screenshots

Figure 3.21: Attributes of the Participant class

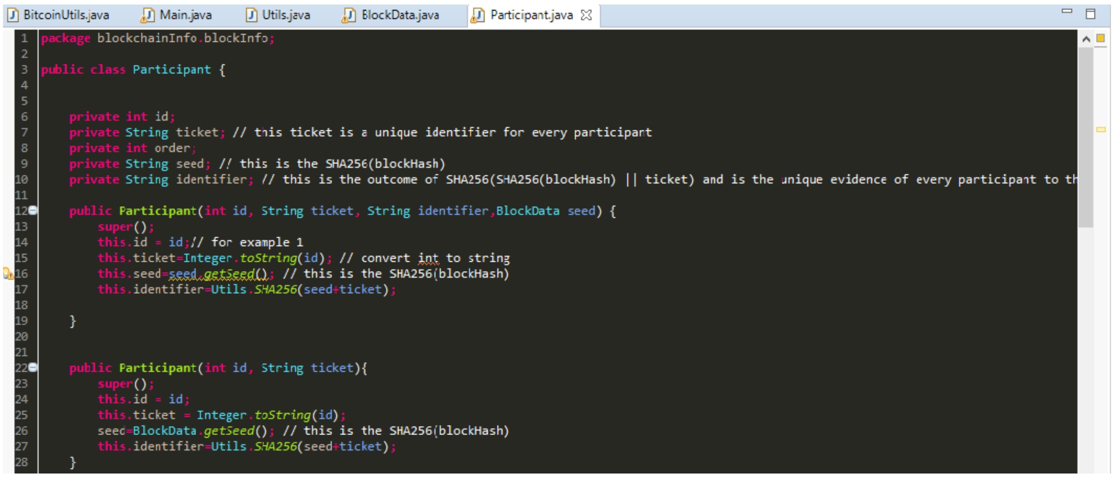
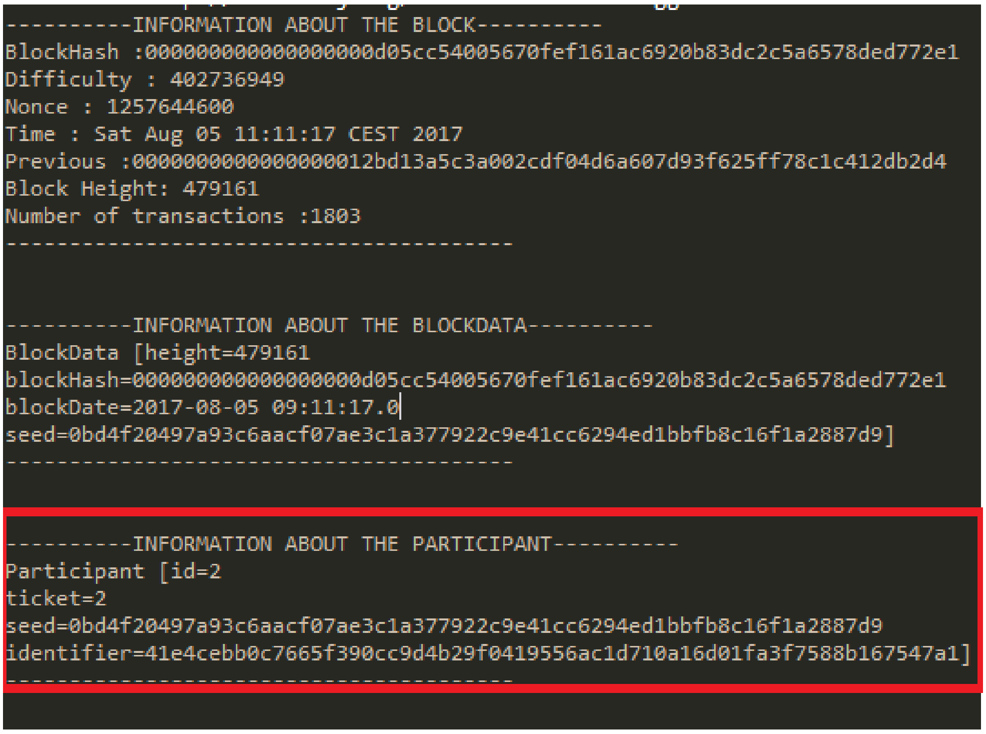


Figure 3.22: Console output with the information about a participant with id=2



3.4.8 SecurityFrame class analysis

3.4.8.1 Description

This Java class simulates the procedure of creating a wallet and importing an elliptic curve key to this wallet based on a specific private key. This private key will be used to sign the OP_RETURN transaction which will gather the appropriate information about the result of the draw and about the participants of the contest. After the configuration of the wallet of the user, this class with the aid of the DataOutput class creates a transaction output which holds the SHA256(SHA256(data)) which are going to be stored in the Bitcoin blockchain. Finally this class creates a request for sending and propagates the OP_RETURN transaction to the active nodes of the network in order to do the validation based on certain criteria and import the transaction in the next valid block.

3.4.8.2 Table of functions

This class contains only the getResult() function which returns the private key of the user. In other words this command simulates the dumpprivkey command which shows the private key in a Base58 check called the Wallet Import Format(WIF) [25] (En.bitcoin.it, 2018). Moreover, the class contains the constructor which instantiates graphical user interface attributes in order to create the Java window for typing the private key in WIF format and press the button for storing our double SHA256 data to the blockchain.

Finally the ButtonListenerPrivate.class contains the actionPerformed() function which is responsible for creating the wallet, creating the ECKey, printing information about the Bitcoin address and about the balance of the wallet, creating a transaction output for OP_RETURN transaction and setting up a connection with the Bitcoin nodes when the user of the application clicks on the "Sign my OP_RETURN transaction" button.

3.4.8.3 Screenshots

Figure 3.23: Private Key Insertion window

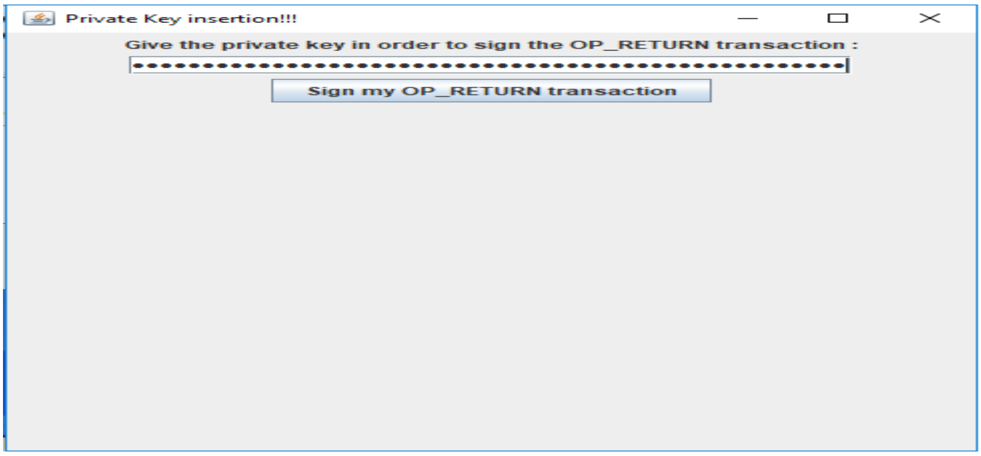


Figure 3.24: Information about the contest and the output of the OP_RETURN transaction

3.4.9 Utils class analysis

3.4.9.1 Description

This class provides to the application some functions which help the user to present the data in a certain format. For example, this class provides the function SHA256(), which takes the blockHash and returns the 256 bits hashing result of the blockHash. Moreover, it provides the getTimestamp() function which takes the date of the captured block as input and converts this date in the format yyyy-MMM-dd HH:mm:ss, which is an easy representation of a date.

Another useful function is the myRandomNumbers(int number) function which takes as input argument the number of the participants that we want to have for our draw and returns an array list of random integer numbers between a range that we can define, simulating the random id numbers that we want to give to our participants. Afterwards, these random id numbers will define the tickets of the participants because the tickets are the interpretation in string format of the id integer numbers.

3.4.9.2 Table of functions

Table 3.10: Utils class functions

A/A	Name	Description
1	SHA256(String input)	Returns a String, which is the outcome of SHA256(input), in our case we use this function to calculate the outcome of SHA256(blockHash)
2	getTimeStamp(Date d)	Returns a timestamp which has a defined format. This timestamp represents the time at which the block was solved and broadcast, according to the clock of the solving node
3	myRandomNumbers(int number)	Returns an array list of random integers between a range. This list of integers contains the id numbers which are going to be given to every participant in a draw

3.4.9.3 Screenshots

Figure 3.25: Attributes and functions of Utils class

BitcoinUtils.java   Main.java   Utils.java   BlockData.java

```java
package blockchainInfo.blockInfo;

import java.sql.Timestamp;

public class Utils {


    private static SimpleDateFormat dateFormatUTC = new SimpleDateFormat("yyyy-MMM-dd HH:mm:ss");
    private static SimpleDateFormat dateFormatLocal = new SimpleDateFormat("yyyy-MMM-dd HH:mm:ss");

    static {
        dateFormatUTC.setTimeZone(TimeZone.getTimeZone("UTC"));
    }

    public final static String SHA256(String input) {
        return org.apache.commons.codec.digest.DigestUtils.sha256Hex(input);
    }

    public final static Timestamp getTimeStamp(Date d) {
        try {
            return new Timestamp(dateFormatLocal.parse( dateFormatUTC.format(d) ).getTime());
        } catch (ParseException e) {
            e.printStackTrace();
        }
        return null;
    }

}
```

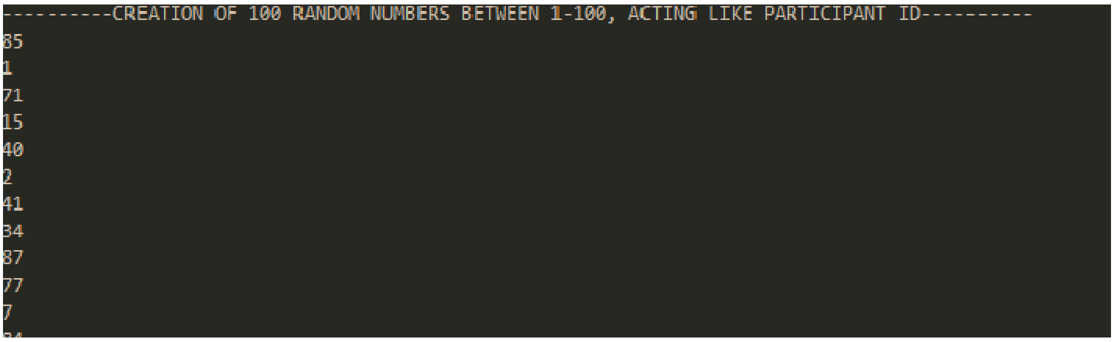
Figure 3.26: Capturing yyyy-MMM-dd HH:mm:ss format

```
BlockData [height=478277
blockHash=0000000000000000121dc9992e3537174ce9e84d30da13a680415adc5cf1be2
blockDate=2017-07-30 16:06:26.0
seed=8d1d25f1919101612d63fdd8dd20a45b353e5205e0ee0a217f6b15c8ba41d6c3]
```

Figure 3.27: Generation of random numbers

```java
    public static ArrayList<Integer> myRandomNumbers(int number) {
        ArrayList<Integer> mynumbers = new ArrayList<Integer>();
        Random rand = new Random();
        int i = 0;

        for (i = 0; i < number; i++) {
            int n = rand.nextInt(number) + 1;
            while (mynumbers.contains(n)) {
                // generate a different integer as id
                n = rand.nextInt(number) + 1;

            }
            mynumbers.add(n);

        }
        return mynumbers;
    }
```

Figure 3.28: Capturing random numbers between 1-100, simulating participant id generation



3.4.10 Writer class analysis

3.4.10.1 Description

This Java class simulates the procedure of storing data to a file. We need this class because we want to create a file which will contain all the appropriate information concerning our draw and our results. The information with the aid of this class will be stored in a file in the local system of the user. In order to be possible to store these information to the blockchain, we need to hash the data, in our case to hash the file which contains the data. The Writer class for our requirements provides the user with two functions. One for writing data to a file and the other for hashing the file which contains our data. In order to store the data as a proof of existence in the blockchain we obliged to store the data hashed until the size of 80 bytes.

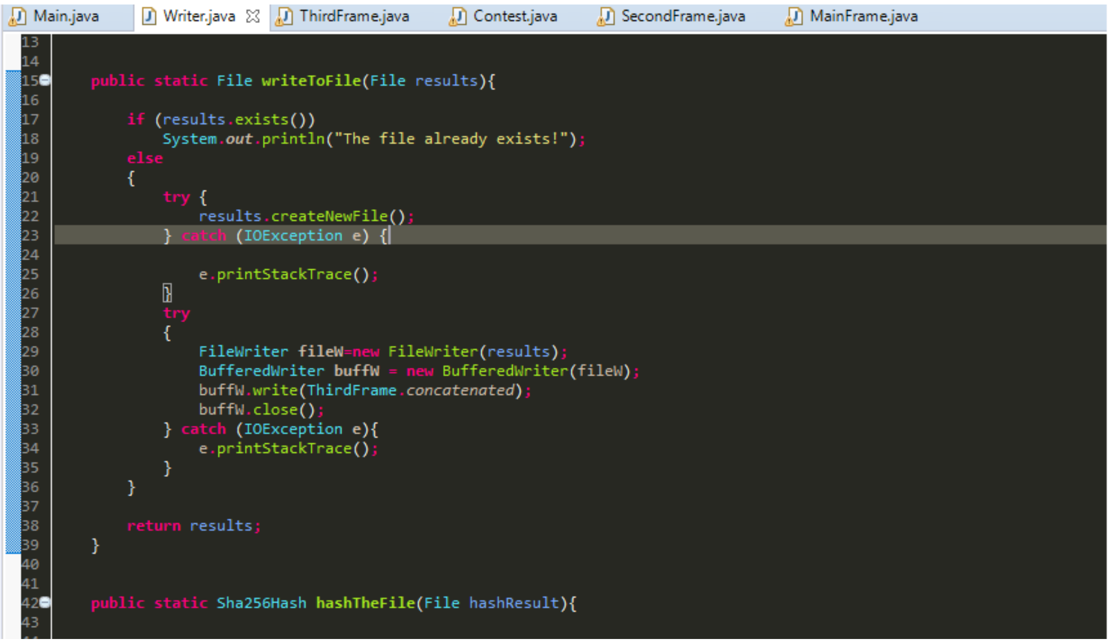
3.4.10.2 Table of functions

Table 3.11: Writer class functions

A/A	Name	Description
1	writeToFile()	This function creates a new file if it is not exists in the system of the user and writes into this file the information provided by the user concerning data about the block, the participants and the winner of the draw. It returns the file with the appended data
2	hashTheFile()	This function returns the SHA256 outcome of the imported file

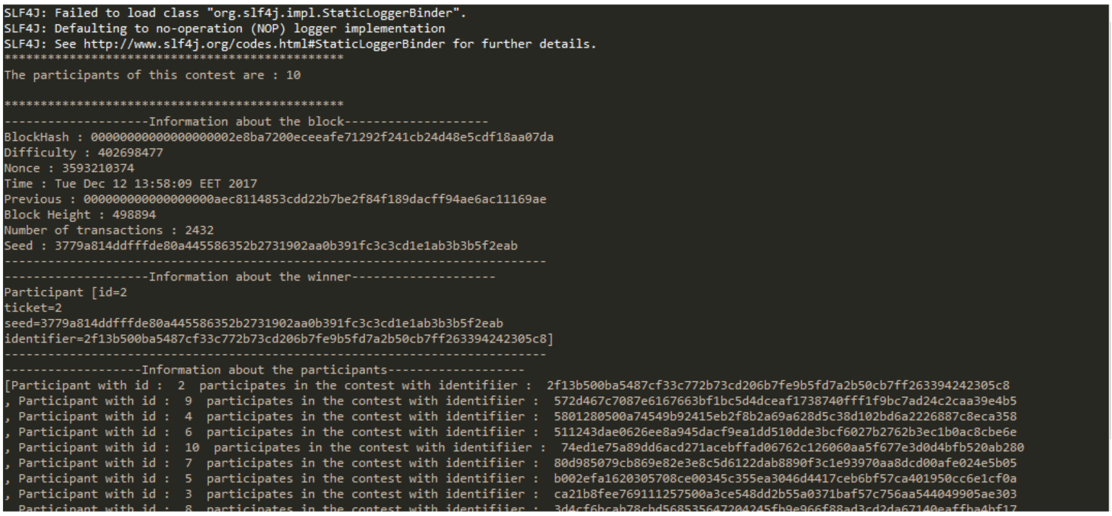
3.4.10.3 Screenshots

Figure 3.29: Functionality of the Writer class




```java
    public static File writeToFile(File results){

        if (results.exists())
            System.out.println("The file already exists!");
        else
        {
            try {
                results.createNewFile();
            } catch (IOException e) {

                e.printStackTrace();
            }
            try
            {
                FileWriter fileW=new FileWriter(results);
                BufferedWriter buffW = new BufferedWriter(fileW);
                buffW.write(ThirdFrame.concatenated);
                buffW.close();
            } catch (IOException e){
                e.printStackTrace();
            }
        }

        return results;
    }


    public static Sha256Hash hashTheFile(File hashResult){
```

Figure 3.30: Information which have been stored in the file and are going to be stored in the blockchain



```
SLF4J: Failed to load class "org.slf4j.impl.StaticLoggerBinder".
SLF4J: Defaulting to no-operation (NOP) logger implementation
SLF4J: See http://www.slf4j.org/codes.html#StaticLoggerBinder for further details.
***********************************************
The participants of this contest are : 10

***********************************************
--------------------Information about the block--------------------
BlockHash : 0000000000000000002e8ba7200eceeafe71292f241cb24d48e5cdf18aa07da
Difficulty : 402698477
Nonce : 3593210374
Time : Tue Dec 12 13:58:09 EET 2017
Previous : 000000000000000000aec8114853cdd22b7be2f84f189dacff94ae6ac11169ae
Block Height : 498894
Number of transactions : 2432
Seed : 3779a814ddfffde80a445586352b2731902aa0b391fc3c3cd1e1ab3b3b5f2eab
---------------------------------------------------------------------------
--------------------Information about the winner--------------------
Participant [id=2
ticket=2
seed=3779a814ddfffde80a445586352b2731902aa0b391fc3c3cd1e1ab3b3b5f2eab
identifier=2f13b500ba5487cf33c772b73cd206b7fe9b5fd7a2b50cb7ff263394242305c8]
---------------------------------------------------------------------------
-------------------Information about the participants-------------------
[Participant with id :  2  participates in the contest with identifiier :  2f13b500ba5487cf33c772b73cd206b7fe9b5fd7a2b50cb7ff263394242305c8
, Participant with id :  9  participates in the contest with identifiier :  572d467c7087e6167663bf1bc5d4dceaf1738740fff1f9bc7ad24c2caa39e4b5
, Participant with id :  4  participates in the contest with identifiier :  5801280500a74549b92415eb2f8b2a69a628d5c38d102bd6a2226887c8eca358
, Participant with id :  6  participates in the contest with identifiier :  511243dae0626ee8a945dacf9ea1dd510dde3bcf6027b2762b3ec1b0ac8cbe6e
, Participant with id :  10  participates in the contest with identifiier :  74ed1e75a89dd6acd271acebffad06762c126060aa5f677e3d0d4bfb520ab280
, Participant with id :  7  participates in the contest with identifiier :  80d985079cb869e82e3e8c5d6122dab8890f3c1e93970aa8dcd00afe024e5b05
, Participant with id :  5  participates in the contest with identifiier :  b002efa1620305708ce00345c355ea3046d4417ceb6bf57ca401950cc6e1cf0a
, Participant with id :  3  participates in the contest with identifiier :  ca21b8fee769111257500a3ce548dd2b55a0371baf57c756aa544049905ae303
```

3.4.11 Summary

In this chapter there was a detailed reference on the role that every class of our project plays in order the organizer to execute a successful draw. Moreover, there was an analysis of the functions of every class in order the reader to understand in a more technical level how the algorithm works. Finally, for validating some outcomes and in order to show the project in a more user friendly way there were some notable screenshots for every class. In the next chapter of the project we are going to point out some elements concerning the testing and the scalability of the lottery algorithm. Moreover, we are going to talk about how it is possible to validate the output of an OP_RETURN transaction.

Chapter 4 Introduction

4.1 Introduction

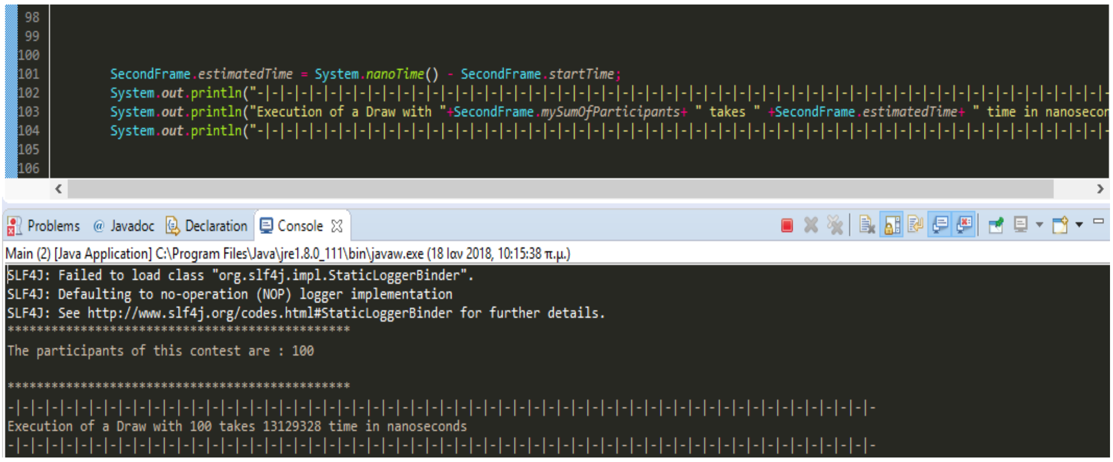
This chapter is about testing the scalability of the lottery algorithm. In order to be as realistic as we can, we run several test cases with different number of participants as input. Firstly we present the results in summary and secondly we present a more detailed presentation. Afterwards, we gather the results in order to comment on the executional time and make important conclusions about the potentiality of using the lottery algorithm as a service. A second part of this chapter is the validation of the output of the OP_RETURN transaction. In order to move on the final step of storing the draw outcomes to the Bitcoin's blockchain, first we have to be sure that the outcome of two times hashing the file holding the information about the draw would be the same data putted as an output to the OP_RETURN transaction.

4.2 Testing Results of Algorithm Execution in summary

An important part of the project is the testing procedure. For this reason we created several test cases in order to simulate how our algorithm performs with different incoming data. We captured the time and tested actually the part of the algorithm which is responsible for feeding the application with a desired number of participants, the creation of the contest based on the number of participants given from the previous step, the sorting procedure of the unique identifier number that every participant possess and finally the finding of the winner of the contest which is the participant whose the outcome of SHA256(SHA256(blockHash) || ticket) is the smallest among all the other participants.

In our Java code it is possible to capture the execution of a part of my algorithm by using the function nanotime() [26] (Docs.oracle.com, 2018) which returns the current value of the most precise available system timer, in nanoseconds. The reader can see a snapshot of my Java code concerning the nanotime() function below.

Figure 4.1: Execution of nanotime() function for 100 participants



As a result the user can capture how long time it takes for a part of algorithm to execute. In our case, I created four different test cases for running my part of the algorithm which implements the functions which I have described above. In the below table the reader can see the summarized outcomes of the tests after running the algorithm for every test case, ten times. The detailed test cases can be found in the next subchapter which present for different number of participants the running times of the algorithm.

Table 4.1: Average executional time of algorithm

A/A	Number of Participants	Average Time in Nanoseconds after running the test cases 10 times	Average Time in seconds after running the test cases 10 times
1	100	18040702.9	0.0180407029
2	1000	38368960.7	0.0383689607
3	10000	1829125543	1.829125543
4	100000	69455143486	69.455143486

The attempt for running the algorithm with 1000000 participants was not taken into consideration because the executional time of the algorithm in this case was high and as a result could not simulate a realistic executional time of an electronic draw or contest.

4.3 Testing Results of Algorithm Execution in detail

The previous summarized results concerning the running time of our algorithm are based on the below detailed tables holding the results of running test cases with different input. For every test case we execute the algorithm ten times. In our case we executed the algorithm from a Dell Precision 3510 laptop with i5-6300HQ CPU running with 2.30GHz frequency.

Table 4.2: Test case with 100 participants

A/A	Number of Participants	Time in Nanoseconds	Time in Seconds
1	100	61822195	0.061822195
2	100	12807994	0.012807994
3	100	12209772	0.012209772
4	100	12710660	0.01271066
5	100	13304438	0.013304438
6	100	13725327	0.013725327
7	100	14007550	0.01400755
8	100	14431549	0.014431549
9	100	11568439	0.011568439
10	100	13819105	0.013819105
	Average	18040702,9	0.0180407029

Table 4.3: Test case with 1000 participants

1	1000	31836431	0.031836431
2	1000	42556425	0.042556425
3	1000	40298648	0.040298648
4	1000	36614206	0.036614206
5	1000	44742646	0.044742646
6	1000	41242649	0.041242649
7	1000	41291092	0.041291092
8	1000	31963986	0.031963986
9	1000	38457761	0.038457761
10	1000	34685763	0.034685763
	Average	38368960,7	0.0383689607

Table 4.4: Test case with 10000 participants

1	10000	1909543595	1.909.543.595
2	10000	2062632861	2.062.632.861
3	10000	1751966333	1.751.966.333
4	10000	2002568888	2002568888,00
5	10000	1923398701	1.923.398.701
6	10000	1644767269	1.644.767.269
7	10000	1968284459	1.968.284.459
8	10000	1645257936	1.645.257.936
9	10000	1820341413	1.820.341.413
10	10000	1562493972	1.562.493.972
	Average	1829125543	1.829.125.543

Table 4.5: Test case with 100000 participants

1	100000	64648860600	646.488.606
2	100000	65532979763	65.532.979.763
3	100000	63809298306	63.809.298.306
4	100000	64120516391	64.120.516.391
5	100000	73448384245	73.448.384.245
6	100000	69337656738	69.337.656.738
7	100000	69270024769	69.270.024.769
8	100000	69298457645	69.298.457.645
9	100000	83613977504	83.613.977.504
10	100000	71471278901	71.471.278.901
	Average	69455143486	69.455.143.486

4.4 Testing Blockchain Applications in different networks

At this point, speaking about testing, it will be helpful to make a brief reference on the three different Bitcoin networks existing for testing purposes and implementation. In other words we can say that the Bitcoin Core [27] (Bitcoin.org, 2018), which is the standard Bitcoin client has three networks it can run which is the mainnet, the testnet and the regtest. **Despite the fact that we have a problem of storing the output of the drawing lottery to the blockchain,** we experiment mostly on regtest network trying to simulate some basic actions. We will mention these basic actions then by making a reference on the remote procedure calls (RPCs) [28] (Bitcoin.org, 2018) that we used in order to have certain outcomes.

Speaking about mainet we can say that this is the real Bitcoin network. In other words, this is the production network with the blockchain that everyone participating in this network uses. Some important issues to mention is that synchronizing the blockchain is mandatory in order to use this type of network, block mining happens every 10 minutes and writing software which handles transaction can cost real money in terms of Bitcoin value.

Figure 4.2: Overview of Bitcoin Core-Wallet in mainet

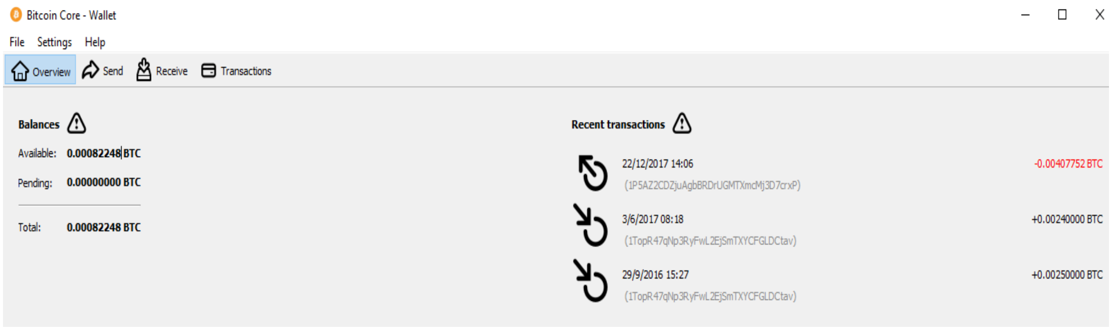
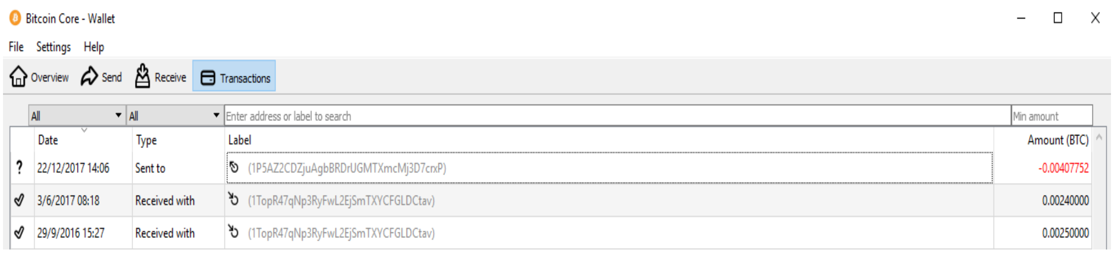


Figure 4.3: Transactions of Bitcoin Core-Wallet in mainet



“Speaking about testnet, we could say that this test network runs in parallel with mainnet, except that the value of the coins are negligible. It exists to experiment with a blockchain that won’t harm the mainnet blockchain, e.g. with new features to Bitcoin Core. It also has the intention to be easier to mine blocks, and therefore it is essentially free to acquire testnet bitcoins” [29] (Kaszuba, 2018). For the purposes of this project there was no need for using the testnet network.

Figure 4.4: Overview of Bitcoin Core-Wallet in testnet

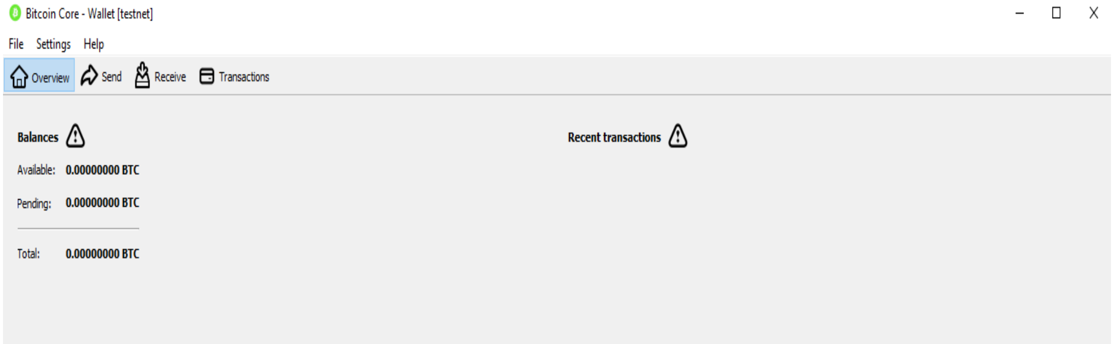
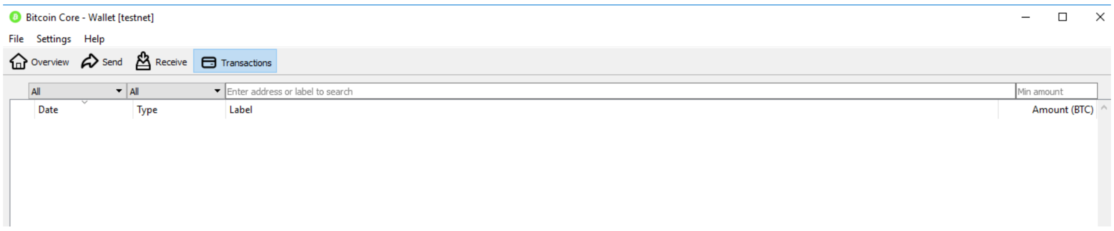


Figure 4.5: Transactions of Bitcoin Core-Wallet in testnet



Finally, there is the regtest or regression network which can provide full independency. The developer or the user can create its own private blockchain without interacting with random peers and without having to store huge amount of data in his local storage. Moreover, the user can send value of bitcoins whenever he wants without waiting for confirmation by someone. In this project, despite the fact that we have not succeed in storing our draw results in the Bitcoin blockchain, we experiment with some remote procedure calls on the regression test and we think that is important to present the most usual when someone starts the network for the first time.

Table 4.6: RPC calls on regtest network

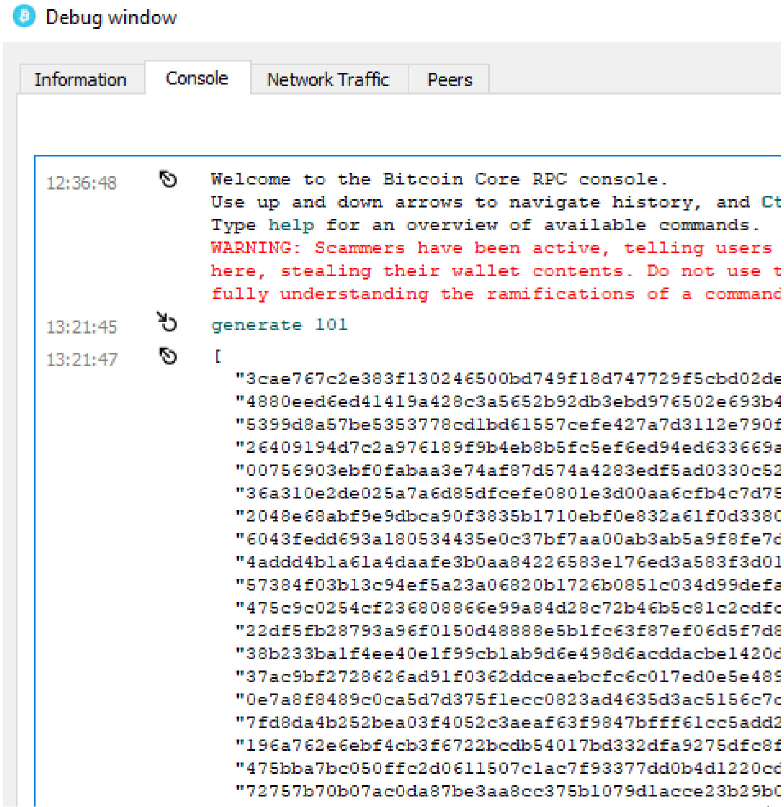
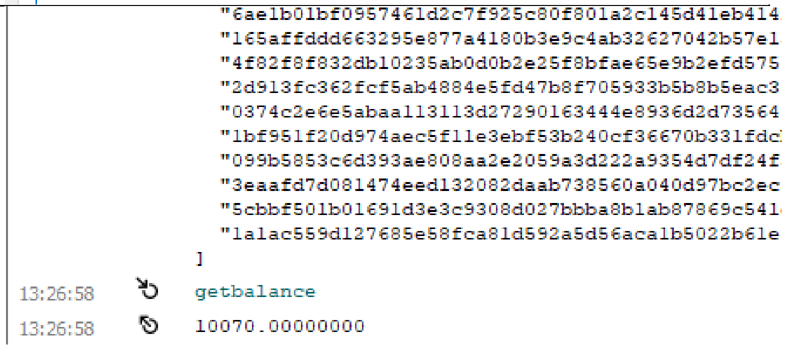
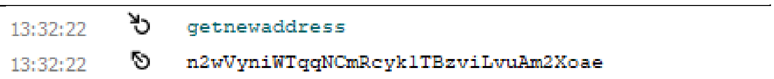
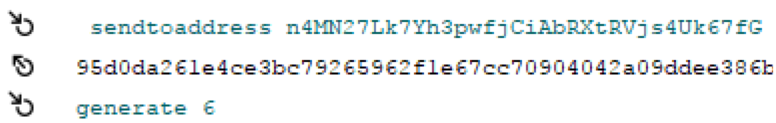
<table>
<tr><th>A/A</th><th>RPC</th><th>Description</th><th>Figure</th></tr>
<tr><td>1</td><td>generate 101</td><td>Generate 101 blocks to get access to the coinbase transaction from block #1</td><td>
</td></tr>
<tr><td>2</td><td>getbalance</td><td>Returns the balance of the user’s wallet</td><td>
</td></tr>
<tr><td>3</td><td>getnewaddress</td><td>Create a new test address</td><td>
</td></tr>
<tr><td>4</td><td>sendtoaddress address amount</td><td>Sending to the testing address the require amount of bitcoins</td><td>
</td></tr>
</table>

Figure 4.6: Overview of Bitcoin Core-Wallet in regtest

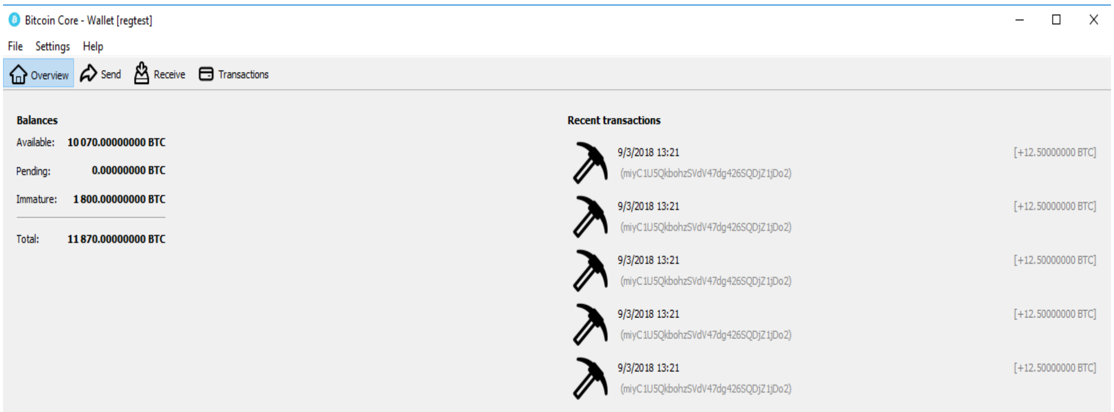
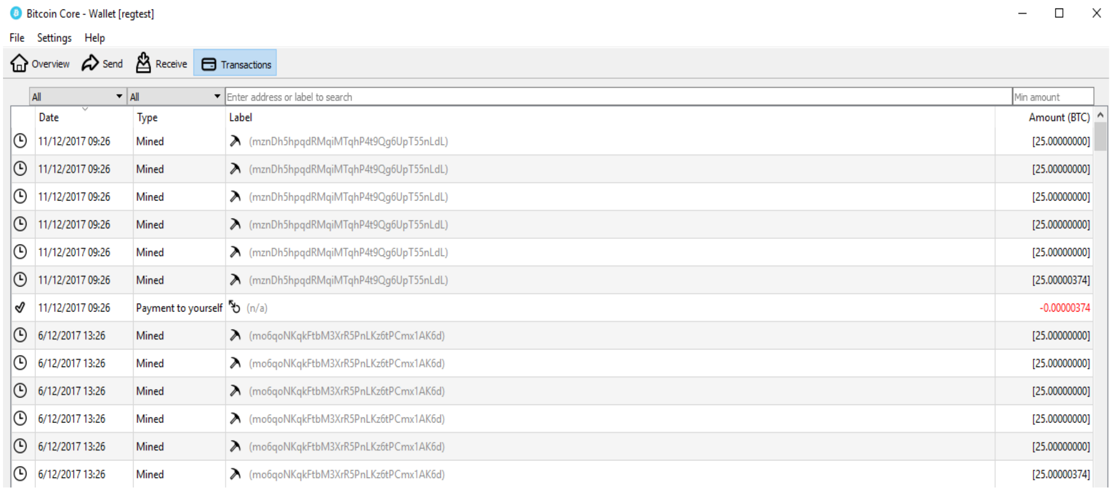


Figure 4.7: Transactions of Bitcoin Core-Wallet in regtest

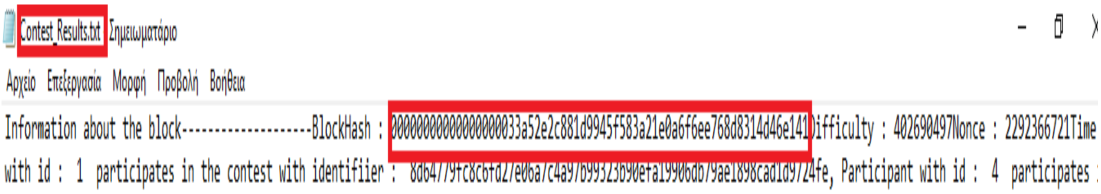


4.5 Validating the output of the OP_RETURN Transaction

The validation of the two times hashed value of the file containing information concerning the contest is a procedure which must be done in order to check that the output bytes of the OP_RETURN transaction is the same with this of the two times hashed value. The hash function which is used in our case is the SHA256 hash function. Let's try to see the validation process in three steps.

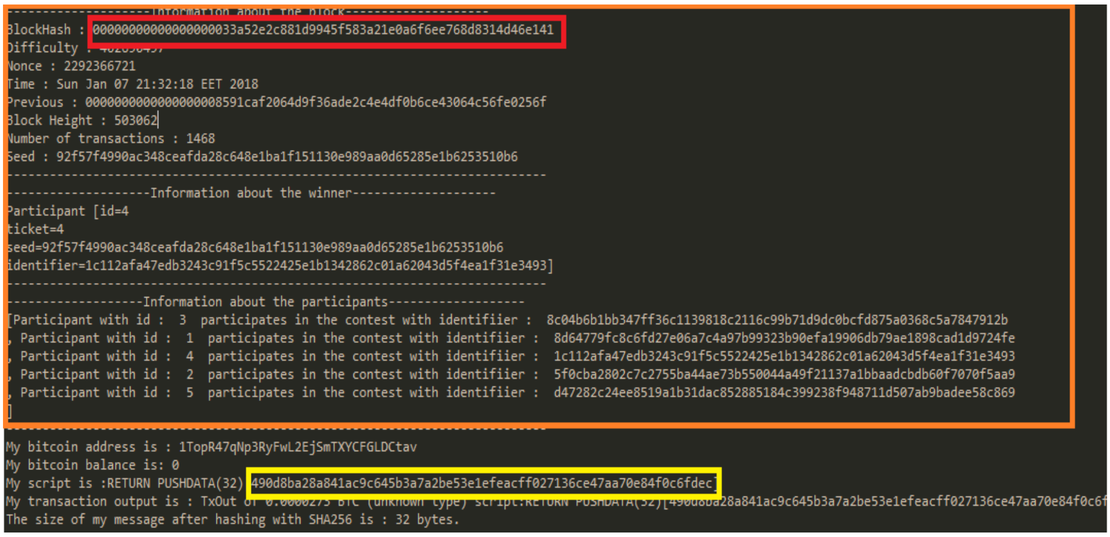
Step 1: The user is checking that the appropriate information concerning the contest and the draw are stored in a file in the local directory of the user.

Figure 4 8: Information in the file



Step 2: The user can check that the information of the file is successfully printed in the console after running the application. Moreover, apart from that, the user can see the result of hashing two times the information of the file as output of the OP_RETURN transaction (32 bytes message). In our case the information have to do with the block with hash:
000000000000000000033a52e2c881d9945f583a21e0a6f6ee768d8314d46e141

Figure 4.9: Information in the console



Step 3: To be sure that we talk about the same hashing value of the file, we can visit an online SHA256 calculator [23] and perform two times the hashing procedure. The first time we are going to calculate the hash value of the information of the file (SHA256(file)) and the second time we are going to calculate the hash value of the hash value of the file (SHA256(SHA256(file))).

In our example, we can see that the hashing values are: SHA256(File)= cb948c627dcf535707efcae41736db06904f962c1f46c34c1a488823db1d8802

SHA256(SHA256(File))=
490d8ba28a841ac9c645b3a7a2be53e1efeacff027136ce47aa70e84f0c6fdec. The second hashing value is the output value imported in the OP_RETURN transaction which is going to be stored in the Bitcoin blockchain and used as evidence of the contest.

Figure 4.10: Calculating SHA256(file)

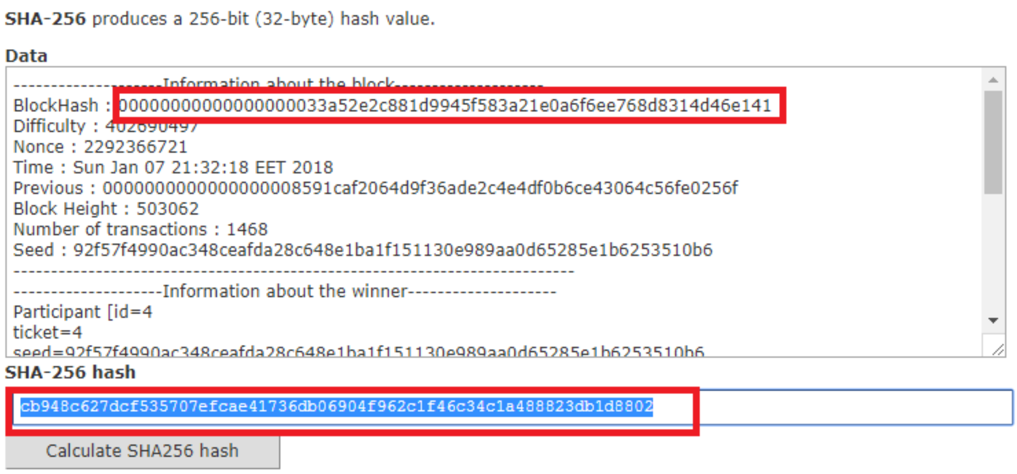


Figure 4.11: Calculating Sha256(Sha256(file))

4.6 Summary

In this chapter we calculated the executional time of our lottery algorithm ending up in important results about the scalability. Moreover, we validate with the aid of an online service the hash result of the output of the OP_RETURN transaction which is going to be stored to the Bitcoin's blockchain and remained there behaving as a proof of existence. In the upcoming and final chapter of the project we are going to present the conclusions of the whole process of trying to build a system like this. Moreover, we are going to examine if we catch the initial goals that we present to chapter one.

Chapter 5 Introduction

5.1 Introduction

In this chapter we will present our conclusions in regard to how they answer our research objective. Our main research objective was the development of a service that uses decentralized digital currency technology to solve an existing problem.

We can argue that we managed to build a service depending on decentralized technology to solve an existing need. More in particular our service, despite the fact that was difficult to store the data to the blockchain, managed to create an algorithm for electronic draws which counts its functionality on the entropy taken from the hashes of the blocks of the Bitcoin network in order to create high levels of randomness.

We have demonstrated that a full decentralized application can be developed and launched with minimal effort to the Bitcoin network. Moreover, the application we have developed is much more than a trivial project and it's fully decentralized which adds great value.

5.2 Results and Contributions

As we mentioned before, the application we built involves features such as capturing entropy from open and big data for ensuring randomness, public verifiability and transparency of lottery results with the aid of blockchain and security based on the immutability of hash function results. All these features in traditional approaches are too complex and very difficult to solve.

Based on our technical implementation and on the research in the field of decentralization we could say it is able to solve problems concerning centralization of power, security, transparency and verifiability. The potential of applications like this one that we built is big because as we said it can solve the above issues.

A service like this one of electronic lotteries might be used by companies, organizations, governments, institutions and individuals who would like to perform electronic lotteries. The advantages over competition are important since there is no need for central authority, no ping pong ball physics cheats, no corruption on isolated engines, no video replay and scam attacks, no notary per raffle but only once for setting the initial rules and finally cost effective as it can run with no special hardware, every 10 minutes concerning the implementation based on Bitcoin network.

5.3 Limitations and future Work

This project stands between building a decentralized service implementing electronic draws and investigating the technical inner working of BitcoinJ library of Java. However, there is a great amount of research to be done in order future projects to solve the problems of security, centralization, transparency and verifiability with more efficient ways.

First of all, for the needs of this project we only used a subset of all the classes and features offered by BitcoinJ library. One could look much deeper into the API (Application Programming Interface) of the library and use also other features and tools available to leverage even more the power of a decentralized network as Bitcoin or the functionality of a digital wallet. Moreover, since only the maintest implementation was used in order to simulate a SPV [30] (Bitcoin.org, 2018) node, we could also get involved with other client implementations of Bitcoin core, the testnet and the regtest.

Apparently, we can build upon the already developed algorithm to overcome its limitations on storing the outcomes of the electronic lottery with the usage of OP_RETURN transaction. Furthermore, we could change the term/rule that the winner is the participant whose the outcome of SHA256(SHA256(blockHash || ticket)) is the smallest among all the other. Finally, we could complement the service with a proper web-based friendly user interface which would allow an easy verification of the draw results.

In a more technical concept, there could be the possibility for improving the algorithm in a sense of making the draw algorithm faster, with the usage of a faster sorting algorithm for example. An improvement like this may give the opportunity of supporting lotteries with more than 100000 participants, which is the testing limit of our application. Furthermore, we could have more accurate results concerning the execution time of the algorithm by running the test cases with different amount of participants as input, more than ten times for every experiment.

Lastly, blockchain can be the basis of great research. By developing a number of different applications, for example regarding proof of concept, proof of existence or proof of work, one could design a number of very interesting and useful services with the aid of decentralized attributes of blockchain technology.

References

[1] Saferandom.com. (2018). SafeRandom » Fair and Transparent Raffles and Statistical Sampling. [online] Available at: http://saferandom.com/ [Accessed 25 Jan. 2018].

[2] Firelotto.io. (2018). Token sales Fire Lotto blockchain lottery. [online] Available at: https://firelotto.io/ [Accessed 25 Jan. 2018].

[3] RANDOM.ORG. (2018). Fraud Warning about the List Randomizer. [online] Available at: https://giveaways.random.org/warning [Accessed 9 Feb. 2018].

[4] En.wikipedia.org. (2018). 1980 Pennsylvania Lottery scandal. [online] Available at: https://en.wikipedia.org/wiki/1980_Pennsylvania_Lottery_scandal [Accessed 9 Feb. 2018].

[5] Hastley, R., Allen, J., White, M., Taylor-Reynolds, J., Turner, J., Allen, J. and Allen, J. (2018). 5 of the Biggest Lottery Scandals. [online] Business Pundit. Available at: http://www.businesspundit.com/5-of-the-biggest-lottery-scandals/ [Accessed 26 Feb. 2018].

[6] Mail Online. (2018). Serbian lottery under fire after winning ball appears before its drawn. [online] Available at: http://www.dailymail.co.uk/news/article-3180480/Number-s-bungling-lottery-organisers-Viewers-uproar-winning-ball-Serbian-lottery-appears-screen-drawn-machine-sparking-accusations-fix.html [Accessed 9 Feb. 2018].

[7] Bitcoinj.github.io. (2018). bitcoinj. [online] Available at: https://bitcoinj.github.io/ [Accessed 26 Jan. 2018].

[8] En.wikipedia.org. (2018). Entropy (information theory). [online] Available at: https://en.wikipedia.org/wiki/Entropy_(information_theory) [Accessed 19 Mar. 2018].

[9] Clark, J. and Hengartner, U. (2010). On the Use of Financial Data as a Random Beacon∗. [ebook] Available at: https://eprint.iacr.org/2010/361.pdf [Accessed 2 Feb. 2018].

[10] Finance.yahoo.com. (2018). KO Historical Prices | Coca-Cola Company (The) Stock - Yahoo Finance. [online] Available at: https://finance.yahoo.com/quote/KO/history?period1=1250802000&period2=1282251600&interval=1d&filter=history&frequency=1d [Accessed 30 Jan. 2018].

[11] Bitinfocharts.com. (2018). Bitcoin (BTC) statistics - Price, Blocks Count, Difficulty, Hashrate, Value. [online] Available at: https://bitinfocharts.com/bitcoin/ [Accessed 2 Feb. 2018].

[12] Web2.0calc.com. (2018). Web 2.0 scientific calculator. [online] Available at: https://web2.0calc.com/ [Accessed 2 Feb. 2018].

[13] Bitinfocharts.com. (2018). Litecoin (LTC) statistics - Price, Blocks Count, Difficulty, Hashrate, Value. [online] Available at: https://bitinfocharts.com/litecoin/ [Accessed 2 Feb. 2018].

[14] En.bitcoin.it. (2018). Proof of work - Bitcoin Wiki. [online] Available at: https://en.bitcoin.it/wiki/Proof_of_work [Accessed 11 Feb. 2018].

[15] Firelotto.io. (2018). Token sales Fire Lotto blockchain lottery. [online] Available at: https://firelotto.io/ [Accessed 16 Feb. 2018].

[16] Cryptovest. (2018). Fire Lotto Builds International Lottery Over Ethereum Blockchain - Cryptovest. [online] Available at: https://cryptovest.com/news/fire-lotto-builds-international-lottery-over-ethereum-blockchain/ [Accessed 17 Feb. 2018].

[17] GitHub. (2018). bug: owner can arbitrarily choose winner · Issue #1 · firelotto/firelotto. [online] Available at: https://github.com/firelotto/firelotto/issues/1 [Accessed 17 Feb. 2018].

[18] Quora.com. (2018). What is modulo bias? - Quora. [online] Available at: https://www.quora.com/What-is-modulo-bias [Accessed 17 Feb. 2018].

[19] Anon, (2018). [online] Available at: https://www.linkedin.com/pulse/why-firelottos-blockchain-based-random-engine-fair-kostas-chalkias/?trackingId=%2BwjRDWOjK4qGAFYGSPLnnw%3D%3D [Accessed 17 Feb. 2018].

[20] En.wikipedia.org. (2018). Unified Modeling Language. [online] Available at: https://en.wikipedia.org/wiki/Unified_Modeling_Language [Accessed 1 Mar. 2018].

[21] Bitcoinj.github.io. (2018). bitcoinj 0.14.3 API. [online] Available at: https://bitcoinj.github.io/javadoc/0.14.3/index.html?overview-summary.html [Accessed 1 Mar. 2018].

[22] Docs.oracle.com. (2018). javax.swing (Java Platform SE 7). [online] Available at: https://docs.oracle.com/javase/7/docs/api/javax/swing/package-summary.html [Accessed 2 Mar. 2018].

[23] Xorbin.com. (2018). SHA-256 hash calculator. Online SHA-256 hash generator. Mining Bitcoin. [online] Available at: http://www.xorbin.com/tools/sha256-hash-calculator [Accessed 2 Mar. 2018].

[24] Bitcoin.org. (2018). Developer Guide - Bitcoin. [online] Available at: https://bitcoin.org/en/developer-guide#simplified-payment-verification-spv [Accessed 2 Mar. 2018].

[25] En.bitcoin.it. (2018). Wallet import format - Bitcoin Wiki. [online] Available at: https://en.bitcoin.it/wiki/Wallet_import_format [Accessed 6 Jan. 2018].

[26] Docs.oracle.com. (2018). System (Java Platform SE 6). [online] Available at: https://docs.oracle.com/javase/6/docs/api/java/lang/System.html#nanoTime%28%29 [Accessed 18 Jan. 2018].

[27] Bitcoin.org. (2018). *Download - Bitcoin*. [online] Available at: https://bitcoin.org/en/download [Accessed 9 Mar. 2018].

[28] Bitcoin.org. (2018). Bitcoin Developer Reference - Bitcoin. [online] Available at: https://bitcoin.org/en/developer-reference#remote-procedure-calls-rpcs [Accessed 9 Mar. 2018].

[29] Kaszuba, G. (2018). *Creating your own experimental Bitcoin network | Gerald Kaszuba*. [online] Geraldkaszuba.com. Available at: https://geraldkaszuba.com/creating-your-own-experimental-bitcoin-network/ [Accessed 9 Mar. 2018].

[30] Bitcoin.org. (2018). *SPV, Simplified Payment Verification - Bitcoin Glossary*. [online] Available at: https://bitcoin.org/en/glossary/simplified-payment-verification [Accessed 13 Mar. 2018].

Appendix 1: Source code

All the Source Code developed for the needs of this project,
including the Java classes can be found in the following github repository.
https://github.com/TopFlankerKiller/BlockInfo-MScProject