Bachelor Thesis

Blockchain Consensus and Transaction layer adaptation for Inter-domain Security Applications

Emanuele Cesari

Supervised by
Alberto Cabellos Aparicio
Jordi Paillissé Vilanova

Final Project Report

UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH

Facultad de Informática de Barcelona
June 2018

Abstract

Generating public randomness without a trusted party is often desirable, especially in decentralized systems such as blockchain, where many users may wish to participate. Producing and using randomness in a distributed setting presents many issues and challenges, such as how to combine random outputs from multiple beacons without permitting bias by an active adversary. [10]
This document analyzes how blockchain technology can be used to generate randomness decentralized beacons, in a distributed way, to secure the allocation, delegation and binding to topological information of the IP address space.

Contents

Chapter 1

Introduction

The generation of random numbers has fascinated man since ancient times, nowadays there is still a great demand of random numbers, in various sectors from gambling to cryptography. For these purposes different methods for the generation of random numbers were invented: the use of dice, roulette wheels, coin-flipping, statistical methods, computer algorithms and the use of quantum mechanic principles. These kinds of traditional random generators have all the same problems, the lack of decentralization and verifiable fairness; although a real random number is not verifiable fair by his nature.
People always wants a random number generator system with higher fairness. Nowadays blockchain systems are getting more and more popular for their decentralized properties, and they also provide a nature basis for the verifiable fair random numbers.[9]

1.1 Objective

The objective of our research was develop a random numbers generator for a blockchain based on proof-of-stake algorithm. In a proof-of-stake protocol, randomness is important to avoid participants cheat the network by choosing a malicious node in order to damage the current blockchain. If malicious nodes are selected continuously, they can behave in a way in which the new created blocks are incorrectly created, slowing the network each time they became selected. For this reason is important to achieve a random consensus selection to hinder the option of slow down the system by malicious participants

1.2 Scope

The scope of this document is to show how to safely save a list of ip addresses inside the bloackchain and to achieve this I need a proof of stake algorithm.
However, none of the existing algorithms meet our requirements so we decided to design a new one. In the following sections of this document we will present our algorithm and describe in details, how create a decentralized randomness beacon generator, based on a blockchain systems, using a distributed key generator algorithm (DKI) and a threshold signature scheme.

Chapter 2

State of the Art

2.1 Overview

Bitcoin and the Blockchain technology have started at 2009 with a lot of controversy around different areas of study, e.g., security, environment, efficiency, etc. One of the most discussed aspects of Bitcoin is the proof of work or PoW.
As Bitcoin uses PoW as its way of choosing a new block signer, nodes in the network have to mine,compute a complex mathematical problem in order to add a block, thus demanding more and more computational effort.
Regarding the prototype, we have opted for a different block signer election, it is called proof of stake or PoS.
In the following sections we will explain in detail the concepts mentioned here.

2.2 Blockchain

A blockchain is a distributed, secure and trustless database. It can also be regarded as a state machine with rules that clearly state which transitions can be performed. Participants in the blockchain communicate through a P2P network. The smallest data unit of a blockchain is a transaction. Users attach data to a transaction along with its signature and their associated public key. Usually, the attached data is an asset or a token, something that is unique and should not be replicated (e.g., coins in Bitcoin).
Then they broadcast this transaction to the other participants. The rest of the nodes in the network store temporarily this transaction. At some fixed intervals in time, one of the nodes takes a set of these transactions and groups them in a block. It then broadcasts this block back to the network. When the other nodes receive this block they verify it, remove the transactions contained in the block from the temporary storage and add it after the previous block, thus creating a chain of blocks. It should be noted that all nodes store the entire blockchain locally. Figure 2.1 presents an overview of the most common elements in a block. Two basic mechanisms are used to protect the chained data: a chain of signatures and a consensus algorithm.

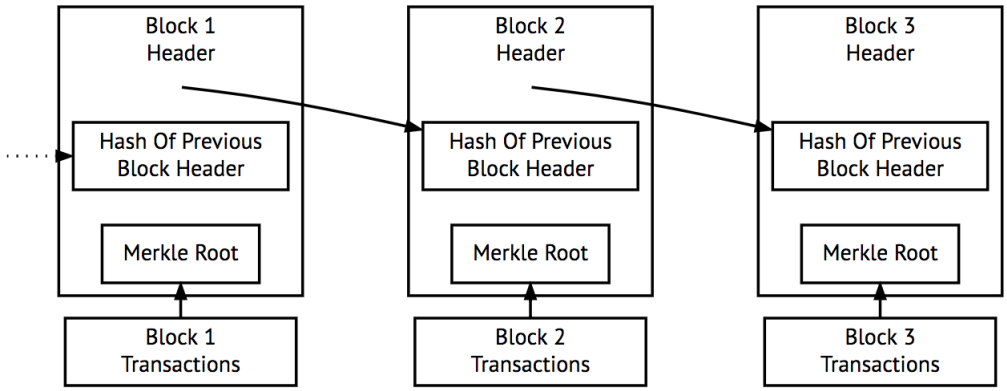

Figure 2.1: simplified bitcoin blockchain

2.3 Chain of signatures

The chain of signatures operates at transaction level. Consider the sender and receiver of a token, each with its public-private keypair. To change the owner of a token, the sender signs the data and the receiver's public key. It then puts together its public key, the signature, the data and the hash of the receiver's public key to form a transaction.

2.4 Consensus algorithm

The consensus algorithm is the central part of blockchain and it controls the chaining of data blocks. The main role of the algorithm is to provide a set of well-defined rules so that participants agree on a consistent view of the database. For this it has the following main functions. First, forks (multiple chains) can exist, this may happen for instance due to varying network latency among participants.
In this case the participants must agree on which is the valid chain. And second, another important function of the consensus algorithm is to determine which participants are allowed to add a new data blocks. The two more popular consensus algorithms are: Proof of Work and Proof of Stake.

2.5 Proof of Work

As we mentioned before in the Proof of Work nodes have to solve a complex mathematical problem to add a block, thus requiring some computational effort, this is commonly know as mining.For example in Bitcoin the problem is to find a hash starting with a fixed amount of zeroes, the only known way to solve this problem is by brute force.
The valid chain is the one with most accumulated computing power, this chain is also the more expensive in terms of computing power to modify.
This is because modifying a block going N blocks back from the tip of the chain would require redoing the computations for all these N blocks.As a result, an attacker should have more computational power than the power required to create the N blocks to be able to modify the chain. Overall, it is commonly assumed that if more than half of the nodes are honest the blockchain is considered as secure. PoW offers relevant features, adding new blocks requires an external resource (CPU power) that has an economical cost. However this also results in some relevant drawbacks:

- **Risk of overtaking**: The security of PoW is entirely based on computation power. This means that if an entity has access to more than half of the total blockchain's computing power it can control the chain. As a result and in order to keep blockchain secure, the incentive of taking control of the chain must be lower than the cost of acquiring and operating the hardware that provides the equivalent to half of the participants computing power. This is hard to guarantee since the economy of the blockchain and the economy of the required hardware are independent. As an example an attacker can acquire the required hardware and operate it, take control of the blockchain to obtain an economical benefit and finally sell the hardware to reduce the final cost of the attack.

- **Energy inefficiency**: PoW requires large amounts of energy to perform the computations (e.g.,miningfarm).

- **Hardware dependency**: Bitcoin automatically increases -over time- the complexity of the mathematical problem that needs to be solved in order to add a block.This is done to account for Moore's law. As a result the community has designed mining specific hardware(ASICs) that provides a competitive advantage.In this context blockchain becomes less democratic, since the cost of participating in it increases.

2.6 Proof of Stake

The main idea behind Proof of Stake is that participants with more assets (or stake) in the blockchain are more likely to add blocks. With this, the control of the chain is given to entities who own more stake. For each new block, a signer is selected randomly from the list of participants typically weighted according to their stake.A fundamental assumption behind PoS is that such entities have more incentives for honest behavior since they have more assets in the chain.
Proof of Stake is seen as an alternative to PoW. At the time of this writing major players in the blockchain environment such as Ethereum are preparing a shift towards PoS, moreover several blockchains based on PoS already exist.
The main reason behind this paradigm shift is that PoS addresses some of PoW's main drawbacks:

- It does not require special hardware nor computationally or energy-expensive calculations.
- An attacker must get hold of a significant part of the assets in order to gain control of the blockchain. As opposed to PoS the investment required to gain control of the chain lies within the chain, and does not involve using external resources.

On the other side, Proof of Stake introduces new sources of attacks:

- In Proof of Stake the signer is selected randomly among the stakers.In this context attackers can manipulate the source of randomness to sign more blocks and ultimately gain control over the chain.
- As opposed to PoW, creating forks is very inexpensive, since no computational power is required. The PoS must provide means to select the valid chain, which is typically the longer one.
- Collusions of high-stakers can create alternate chains which can appear to be valid.

2.7 Methods for generating random bits

According to the NIST "Recommendation for Random Number Generation" [1], there are two fundamentally different methods for generating random bits.
The first is to produce bits non-deterministically, where every bit of output is based on a physical process that is unpredictable; this categories of random bit generators (RBGs) is commonly known as non-deterministic random bit generators (NRBGs).
The second method is to compute bits deterministically using an algorithm; this class of RBGs is known as Deterministic Random Bit Generators (DRBGs).

2.8 Types of random numbers

Random numbers can be divided into three categories: true random numbers, pseudorandom numbers and quasi random numbers.

- **Pseudorandom numbers**: are a sequence of numbers which are algorithmically produced and they are not really random.If the initial conditions and algorithm are known the sequence can be repeated.
- **Quasi random numbers**: act as random numbers in some sort of simulations, but are well-ordered in some other types, they are designed to be highly correlated, in a way such that they will fill space fairly evenly. The three most famous methods to generate this type of random numbers are:
  - Richtmeyer-sequences
  - Van der Corput-sequences
  - Sobol sequences
- **True random numbers**: the process of generation of this type of random numbers is not predictable, because is completely base on an underlying random physical process.
  Unlike the pseudorandom number there is no internal state kept in the generator and the output is based only on the physical process and not any previously produced bits [4]

Chapter 3

Project management

3.1 Stake Holders

The stakeholders involved in this project are described below.

- **Developer** The main function of the developer in this project is to do research on the multiple available algorithms for the generation of random number , develop the selected algorithm and integrate it with the rest of the parts in the blockchain system.

- **Director and co-director** Their role is the guidance and supervision of the work done by the developer. They also help on the comprehension of the technology and the implementation on the network.

- **End-Users** As the main goal of the project is to secure the Internet, anyone who uses Internet nowadays, will be beneficiary of my project, i.e. companies, users, government entities etc.

3.2 Methodology

I will develop my project with my thesis Director Albert Cabellos and my thesis Co-director Jordi Paillissé. We both agree to use for this project an Agile software development methodology, where communication among team member is crucial, working on code is supposed to be mandatory and responding to change is supposed to be rapidly applied.
According to the Agile methodology, there will be weekly meetings between us. The Objective of these meetings will be to summarize the work done during the week and to discuss relevant aspects for the good performance of the prototype

3.3 Intensive customer Feedback

While the project doesn't have a real client, this concept from agile methodologies could still be applied : By letting the person that is ultimately responsible for evaluating the project, Professor Albert Cabellos, checks the project state as frequently as possible to provide feedbacks about the project

Figure 3.1: Agile methodology process

3.4 Developement tools

To develop the project and to achieve a correct integration with all the parts of the project, I used the following tools:

- **Visual Studio Code**: is a code editor redefined and optimized for building and debugging modern web and cloud applications. I will use this tool to write the code of my Algorithm.
- **Git**:is a version control system for tracking changes in computer files and coordinating work on those files among multiple people.
- **Thunderbird**: is an email client I used to comunicate with my cordinators.
- **Wmare Workstation**: Is a virtualization software that I used to create different virtual machines for test and validate my project.

3.5 Validation

In order to validate my project, I will use a beta network to test the project. This network is very similar to the real world environment, so this will give me a valid and consistent result.

3.6 Temporal planning

The estimated project duration was approximately 5 months. The project started on Febrary 15th, 2018 and it finished in the end of June.
It must be noticed that, due to the characteristics of this project, the planning was subject to modifications depending on how the project was being developed. Furthermore, during March 2018 the project stopped due to Easter holidays

3.7 Project planning and flexibility

Five months (Erasmus period) can seem like a long time, but a bad scheduling can make it insufficient.
To avoid this problem, a very rigid but realistic timetable with weekly meetings with the project tutors will be set to ensure the project is on schedule, or to apply the appropriate corrections otherwise. The coverage of the GEP course can be divided in this stages:

- Context and scope of the project (10h)
- Project planning (5h)
- Budget and sustainability (5h)
- First oral presentation (10h)
- Specification (5h)
- Oral presentation and final document (15h)

All this stages will be done by the Project Manager under the supervision of the Co-director and Director of the project. By the end of the thesis the planning is the following:

- Project planning (40h)
- Research about blockchains (50h)
- Analysis of requirements (30h)
- Research on Random number generator (90h)
- Design of the algorithm (90h)
- Algorithm development (50h)
- Module tests (40h)
- Documentation of the final document (40h)
- Defense preparation (20h) TOTAL (450h)

3.8 Project analysis

This section will cover the analysis of the project and is implementations. Before starting the development of the project, there is a need to understand what are the technologies involved in it, the objectives to achieve and the requirements to have. In the case of this scenario, the technology is the most important part, as it is new and hard to understand,the technology involved in the project is called the blockchain. In order to achieve the state of the art knowledge of the technology, I must read a lot of papers to understand what a blockchain is, how it works and how it should be developed. Only when I fully understand the technology, I will be able to design and analyze a plan in order to develop the project.

3.9 Project Design

This part consists on creating the software architecture in order to get a working setup and the correct performance expected for the completion of the project. As I mentioned before there are a lot of blockchains developed in the last years and this can be helpful to have a general view of how a blockchain should be implemented. To filter the search of a good prototype, I have opted for Bitcoin and Ethereum. They are very different but both have characteristic that can be very useful for my project

3.10 Algorithm Implementation

The core part of the project is the implementation of the algorithm. After the hard work involved on understanding each part of the Blockchain,it is time to shape my thoughts in the code.

3.11 Action Plan

As I mentioned before I developed my project with my thesis director and my thesis co-director.
We used the Agile methodology as a development method so it allowed me to adapt the timings in a proper way.If something required more time than expected. Moreover, in the meetings, my Director and Co-director checked all the timings in order to decide whether I was in time or not.
If they saw that I was in time, everything was good, and we didn't take any additional actions. If we saw that I was not in time, we tried to find which part was taking the extra time with the objective of reducing the waste of time. For this purpose, we had a meeting every week, where we analyzed the situation and we took extra actions if was required.

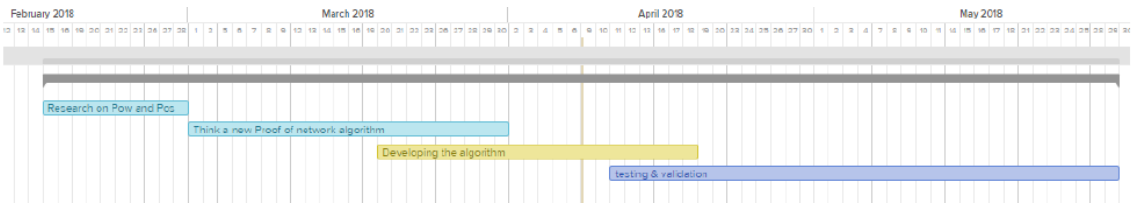

Figure 3.2: Gantt Chart

3.12 Economic Sustainability

Taking into account the amount of hours needed and the number of computers dedicated to develop the project, is difficult to do a similar project with a lower general cost. It was difficult to say beforehand if we will have the time to finish the TFG according with the deadline, but this is the objective, so no extra costs will be needed in theory.
In order to reduce the cost of the project, we used free software tools for the development of it. Moreover, with this kind of actions we are supporting the free software community, which is very important for the development of the software world.

3.13 Social Sustainability

The aim of this project is to decentralize the assignment of Internet IP addresses in order to make the Internet more free and more accessible to people.
With this project and the final arrival of the IPv6 protocol, I could dream in a world where each entity, person or whatever can have its own public IP without the need of third parties. The only entity affected by our project is the IANA institution.
They have the nowadays power to determine who have which IP address, so they will want to keep their power

3.14 Risk management

As I will be using open source software for the development of the project, I have the risk that they could being deprecated in the future. If the developers do not continue with the development we can be affected. This possibility is not so real as there are a lot of contributors working in the software that I've chosen for the implementation.
It is true that licensed software can be a good choice if I need support. But as I know the tools and there a lot of documentation, I won't have any problem for the development of the project with the software chosen.
As a risk I also have the management of more than one virtual machines. It is not trivial to manage them, as they are hardware simulated by software, so sometimes it can be tricky to make everything work as expected.

3.15 Relation of the project with technical competences

- To demonstrate comprehension, apply and manage the reliability and security of the computer systems : This competence was chosen because the main objective of the project is to secure the Internet in the end, so I need to understand and manage the security that is around the network system
- **To design communications software** :Behind the Blockchain there is the communication protocol, the P2P mentioned above. This is a crucial part, so in order to develop the rest of the project, I need to fully understand how it works.

Chapter 4

Cryptographic background

4.1 Overview

Before presenting the different solutions already available in the market, and my algorithm for the generation of random numbers, we will briefly explain some cryptographic background that are fundamentals to clearly understand the following chapters.

4.2 Distributed Key Generation

A Distributed Key Generation Protocol (DKG) is a cryptographic protocol in which multiple entities contribute to calculating a public key and a set of private keys where each private key belongs to one of the entities. This prevents any participants from having access to secret information belonging to another participants.
The first DKG protocol was specified by Torben Pryds Pedersen in *"Non-interactive and information-theoretic secure verifiable secret sharing"* [8].
It was based on the security of the Joint-Feldman Protocol for verifiable sharing of secrets. Few year later Professor Rosario Gennaro and others published a series of tests that showed that the Joint-Feldman Protocol was vulnerable and malicious contributions to the protocol proposed by Pedersen were vulnerable.
The same group proposed in *"Secure distributed key generation for discrete-log based cryptosystems"* [3] an update of the Pedersen protocol that protects against malicious contributions.
Nowadays the distributed keys generation algorithm (DKG) are used in different sectors and for different purpose, but is main implementation is in the setup session of a threshold signature scheme.

4.3 Threshold Signature Scheme

In a (t,n)-threshold signature scheme, n parties together set up a public group key, and each party keep an individual secret key. After this setup, t out of the n parties are required and sufficient for generating a group signature that could be validate against the public group key generated before. The aim of distributing the functionality by having different entities cooperate is usually due to the following two motivations:

- **Fault tolerance**: it is necessary to compromise the security of several entities to break the security of the system and on the other hand, establishing a threshold lower than the number of participants allows some of the entities to stop operating, thus increasing the availability of the functionality
- **Distribution of responsibilities**: For example, in an encryption system with a threshold to obtain clear content, it is necessary for several servers to cooperate in order to perform the action. In a threshold signature system it is necessary that the two entities, which may have different views of the problem, agree to sign. Is worth mentioned that a Distributed key generation algorithm is much more than a secret sharing protocol,because in a secret sharing protocol the shares can be used to recover the group secret just once, and after everyone has learned the group secret the shares cannot be reused. In a distributed key generation protocol the secret shares can be used for an unlimited number of group signatures without ever recovering the group secret key.

Different Threshold versions of signature schemes can be built for many public encryption schemes such as:

- RSA
- ElGamal
- Damgård–Jurik cryptosystem
- Paillier cryptosystem

According to whether a verifier can trace the signatories we can distinguish two different types of threshold signatures scheme.

- **Anonymous signers**:no one can reveal the identity of the parties involved.
- **Traceable signers**: the identity of the parties involved is public.

4.4 Shamir secret sharing

Shamir's secret sharing system is a cryptographic algorithm. It is a form of secret sharing where a secret is divided into parts and each participant is given only one: all or part of them are necessary to reconstruct the secret.
The purpose of the algorithm is to divide a secret D (for example, a key) into N parts so that:

- The knowledge of K or more D parts makes easily computable the secret D.
- Knowledge K *-1* or less D parts makes D indeterminate, in the sense that all its possible values are equally likely to be true.

4.5 Verifiable Random Functions

The Verifiable random functions were invented by Mical, Vadhan and Rabin in *"Verifiable random functions"* [6] to solve the problem that a pseudo random oracle is not verifiable without knowledge of the seed or other information.
It's a pseudo-random function that provides for his output a fully verifiable proofs of correctness.

Chapter 5

Existing alternatives

In this chapter we will present in details the different solutions already available on the market for generate true random numbers.

5.1 NIST generator

NIST is offering a public source of randomness, this service use two independent commercially available sources of randomness, each with an independent hardware entropy source and SP 800-90-approved components.[7]

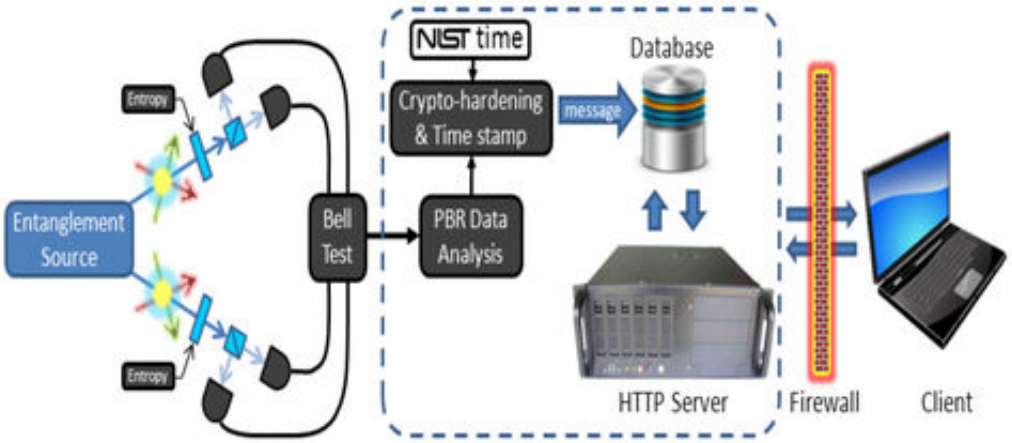

Figure 5.1: NIST beacon scheme

The NIST Beacon service is designed to provide:

- **Unpredictability** means that users cannot algorithmically predict bits before they are made available by the source.
- **Autonomy** means that the source is resistant to attempts by outside parties to alter the distribution of the random bits.
- **Consistency** means that the exact sequence will be shown when being accessed by a group of users

5.2 Random.org

Random.org is a website that offers true random numbers to anyone on the Internet. The randomness comes from atmospheric noise, and as we explained in the previous chapter this is better than the pseudo-random number algorithms typically used in computer programs.
The random numbers generate by randaom.org are used in different way such as holding drawings, lotteries and sweepstakes, to drive online games, for scientific applications and for art and music.

5.3 Algorand

Algorand [5], despite at the beginning is not easy to understand, is another good alternative solution for the generation of randomness. Algorand uses Verifiable Random Functions (VRFs) to build a Common Coin, which acts as a God's dice, and when the nodes cannot research consensus, the hesitant nodes can react according to the result given by the Common Coin.
In the Algorand algorithm the users communicate through a gossip protocol. Messages are signed using the private key of the sender, and peer selection is weighted depending on how much money a peer has.
This gossip protocol is also used by users to submit new transactions, and every user collects a block of pending transactions that they hear about. Algorand runs on different rounds, and in each of these rounds a set of users are chosen randomly, in a completely decentralised way, using cryptographic sortition, to propose a new block for the blockchain.

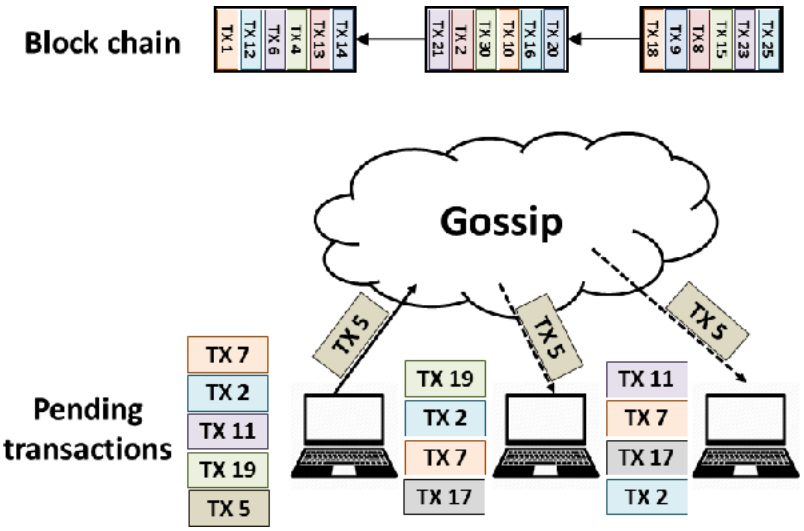

Figure 5.2: An overview of transaction flow in Algorand.

5.4 Randao

Randao [9] is a protocol that offers a public random number generator, based on two different schemes: Commit Reveal and BLS.

5.4.1 Commit Reveal

The Commit Reveal algorithm is designed to operate on the famous Etherum blockchain and it uses the Etherum smart contracts, the commit reveal scheme is compose by three main steps:

- **Collecting ETH and *sha3(s)***: All the nodes that want participate to the protocol should send the sha3(s) of a random number s and m amount of ETH to the smart Contract C.

- **Collecting *s***: All the partecipants that successfully submitted the *sha3(s)* send the random number s to smart contract C.
  C checks if the random number s matches with the *sha3(s)*received in the first phase, if yes then the Smart Contract C saves *(*s) into the seeds of the function that he will use to generate the random number.

- **Generating the random number**: This is the last steps of the Randao Commit Reveal scheme, in this phase the smart Contract C generates the random number, using the fucntion F,comunicates it to all the partecipants, and it returns the amounts of m ETH received in the first step of the protocol. It is worth mentioning, that if a partecipant sends the wrong random number s in the second phase of the protocol,the m of ETH that were deposited in the first step will not be returned.

5.4.2 BLS

The second scheme proposes in the Randao white paper [9] is based on the Boneh Lynn Shacham (BLS) signature scheme, that allows a user to verify that a signer is authentic.
The main idea to generate randomness using a BLS signature scheme is that all the participants are divided into different groups, the first group generates a random number,sign it, and send to another group that will do the same process. The final random number will be the last group signature of the message.

5.5 Dfinity

DFINITY [2] is another solution to generate randomness, is it a decentralized randomness beacon generator that use a verifiable random function (VRF),a distributed key generation algorithm (DKG) and a threshold signature scheme, all of this will be explained with more details in the next chapter.
Because of the threshold signature scheme each random output of the DFINITY algorithm is unpredictable by anyone until the it becomes available to everyone.
The threshold signature scheme used in the DFINITY protocol is non-interactive and uniqueness

- **Non-Interactive**: the creation mechanism of the group signature is compose only by one single round of one-way communication for each of the t participating parties.
  Usually in this type of process, each node generates a signature share using his individual signature and then he sends it to a third party. When the third party has received t valid shares it can rebuild the group signature without other further interactions.

- **Uniqueness**; a signature scheme is unique if for every public key and for every message there is only one valid signature.

5.6 RandHerd

RandHerd is a protocol that allow a potentially large group of servers to form a distributed public randomness beacon, which proactively generates a regular series of public random outputs.[10]
Ranherd aims to offer a public randomness beacon with the following security properties:

- **Availability**: Given an honest leader, the protocol successfully completes and produces the final random output Z with high probability.

- **Unpredictability**: Nobody knows anything about the final random output Z,until the final group responses are revealed.

- **Unbiasability**: The collective randomness Z represents an unbiased, uniformly random value, except with negligible probability.

- **Verifiability**: The final randomness output randomness Z is third-party verifiable as a collective Schnorr signature under X

5.7 Evaluation of the different solutions

In order to evaluate the different random number generator mentioned before we used these criteria suggested in *Randao: Verifiable Random Number Generation* [9].

- **Unpretictable**: Nobody can predict the possible value of the next random number based on the previous result or other historical data, not even a tiny increase in the success rate prediction.
- **Unconcealable**: After the random number is generated the users cannot refuse to disclose it.This also means that after being generated a random number will be public available and cannot be hidden or withdrawn.
- **Conspiracy-resistant**: If some of the network nodes privately join together to exchange their own private information, they cannot affect the final output or have other competitive advantages.
- **Response rate**: The process for the random number generation should be as fast as possible
- **No possibility to know in advance**: All the parties involved in the protocol for the random number generator discover the generated random number at the same time.No one can know the final result ahead of time.
- **Open partecipation**: The random number generation process should be open to public partecipation, in order to reduce or eliminate the threshold to participate.
- **Unselectable**:The nodes cannot withhold any results or replace one with another one.
- **Auditable**: The whole process of the random number generation should be fully auditable.
- **Low cost**: The cost for random number generation should be as low as possible.
- **Tamper-resistant**: After the generation of the random number no one can modify it.

Chapter 6

Algorithm proposal

6.1 Overview

Since none of the solutions mentioned above meets our requirements in terms of scalability, safety and performance, I decided, in agreement with my thesis director and co-director, to implement a new solution that would better cover our requirements.
The algorithm we designed for the generation of a random beacon in a distributed way, is based on D-infitiy and Randherd since it always uses a DKG algorithm and a threshold signature scheme.
However, our algorithm has a substantial and innovative difference compared to the two algorithms previously mentioned, the use of blockchain in the process of generating random numbers to avoid the high numbers of messages between the nodes.
The algorithm is composed mainly of the two following phases: *setup phase* and *random number generation*.

6.2 Setup

In the setup phase, the nodes belonging to the peer to peer network are randomly grouped into groups.
Once created the groups, begins the phase of key generation that is achieved through distributed key generation (DKG) algorithm invented by the professor MIchali [3] which is a safer improvement of the famous Petersen algorithm [8].
Once the keys are generated, each node will have a secret piece of a private key, and the shared key of the group will be saved in the blockchain to be available to everybody. Once the key generation is complete, the second phase of the Algotihm will begin.

6.3 Random Number Generation

Each participating node will calculate the hash of the previous blockchain block, sign it with its private key and save the signature in the blockchain; everyone can verify this signature using the public group signature available in the blockchain.
Once there are enough signatures in the blockchain any node can use a "recover" function to calculate the final public group signature, and save it into the blockchain. However, it is worth remembering that given the security properties of the threshold signature scheme, it is irrelevant which signatures are selected to generate the final group signature.
The random number at the end will be determined by the hash of the final group signature, which will then be used to generate an ip address and the owner of this ip address will be the one who will signs the block.

6.4 Security Properties

In this section we will describe and explain the security properties of our algorithm:

6.4.1 Unpredictability

Through the security properties of the threshold signature scheme our algorithm ensures that nobody can predict the random number in advance until the final group signature is calculated and saved into the blockchain.

6.4.2 Verifiability

Thanks to the properties of the blockchain,DKG algorithm and threshold signature scheme, all the phases of the algorithm can be verified by a third party. More in details:

- **Hash signatures** of the previous block can be verified using the public group key saved in the blockchain.
- **Final group signature** can be recalculated by everyone through a recover function using as input the block signatures saved in the blockchain.
- **Random number** can be easily verified, just calculate the hash of the final group signature saved in the blockchain.

6.4.3 Availability

Our algorithm guarantees a high probability that the protocol will end in a positive way generating a random number even in the presence of an adversary who behaves in a malicious way.
The availability is guaranteed mainly by the threshold signature scheme which allows that n out of t nodes are required to generate the final group signature.

6.4.4 Unbiasiability

The algorithm prevents the adversary from biasing the value of the random number. As we explained before it requires that at least one node is honest and the algorithm will product a true random number.

Chapter 7

Performance evaluation

7.1 Overview

This section experimentally evaluates our prototype implementations of the algorithm for the distributed beacon generation.
Given the absence of an open source cryptographic library to implement quickly and easily the distributed key generation algorithm of Micali [6] and the threshold signature scheme, and in agreement with my director and co-director I decided to use for the evaluation phase the Randherd algorithm previously explained. we decided to use this algorithm for two main reasons:

- **Similitude** Randherd is a solution very similar to my algorithm because it uses the same components: distributed key generation and thresold signature scheme, except for blockchain so the results I got in this simulation phase are the same valuable and significant.
- **Open Source**:is the only one whose source code is publicly available online and also has a very good documentation that has allowed me to understand how the code works and allowed me to easily make some changes to the source code to better adapt him to my needs.

7.2 Implementation

For the testing and analysis of the algorithm we used as a starting point an open-source code available on Github, and we modified the following script that we used to generate the various nodes of the peer to peer network, test the exchange of keys for the DKG and the generation of the random beacon. In detail we added the possibility to set the number of the network nodes and we inserted a function to print on the console the execution time of every step to record the data for the graphs.

```
#!/bin/bash
start=$(date +%s)
N=6
BASE="/tmp/drand"
if [ ! -d "$BASE" ]; then
    mkdir $BASE
fi
unameOut="$(uname -s)"
case "${unameOut}" in
    Linux*)     TMP=$(mktemp -p "$BASE" -d);;
    Darwin*)
        A=$(mktemp -d -t "drand")
        mv $A "/tmp/$(basename $A)"
        TMP="/tmp/$(basename $A)"
    ;;
esac
GROUPFILE="$TMP/group.toml"
IMG="dedis/drand:latest"
DRAND_PATH="src/github.com/dedis/drand"
DOCKERFILE="$GOPATH/$DRAND_PATH/Dockerfile"
NET="drand"
SUBNET="192.168.0."
PORT="80"

function checkSuccess() {
    if [ "$1" -eq 0 ]; then
        return
    else
        echo "TEST <$2>: FAILURE"
        cleanup
        exit 1
    fi
}


function convert() {
    return printf -v int '%d\n' "$1" 2>/dev/null
}

if [ "$#" -gt 0 ]; then
    #n=£(convert "£1")
    if [ "$1" -gt 4 ]; then
        N=$1
    else
        echo "./run_local.sh <N> : N needs to be an integer > 4"
        exit 1
    fi
fi
```

```
49
50  ## build the test travis image
51  function build() {
52      echo "[+] Building the docker image $IMG"
53      docker build -t "$IMG" .  > /dev/null
54  }
55
56
57  function run() {
58      echo "[+] Create the docker network $NET with subnet ${SUBNET}0/24"
59      docker network create "$NET" --subnet "${SUBNET}0/24" > /dev/null 2> /dev/null
60
61      sequence=$(seq $N -1 1)
62      #sequence=£(seq £N -1 1)
63      # creating the keys and compose part for each node
64      echo "[+] Generating all the private key pairs..."
65      for i in $sequence; do
66          # gen key and append to group
67          data="$TMP/node$i/"
68          addr="${SUBNET}2$i:$PORT"
69          addresses+=($addr)
70          mkdir -p "$data"
71          #drand keygen --keys "£data" "£addr" > /dev/null
72          public="key/drand_id.public"
73          volume="$data:/root/.drand/:z"
74          allVolumes[$i]=$volume
75          docker run --rm --volume ${allVolumes[$i]} $IMG keygen "$addr" > /dev/null
76              #allKeys[£i]=£data£public
77          cp $data$public $TMP/node$i.public
78          ## all keys from docker point of view
79          allKeys[$i]=/tmp/node$i.public
80          echo "[+] Generated private/public key pair $i"
81      done
82
83      ## generate group toml
84      #echo £allKeys
85      docker run --rm -v $TMP:/tmp:z $IMG group --out /tmp/group.toml "${allKeys[@]}" > /dev/null
86      echo "[+] Group file generated at $GROUPFILE"
87      echo "[+] Starting all drand nodes sequentially..."
88      for i in $sequence; do
89          # gen key and append to group
90          data="$TMP/node$i/"
91          groupFile="$data""drand_group.toml"
92          cp $GROUPFILE $groupFile
93          dockerGroupFile="/root/.drand/drand_group.toml"
94          #drandCmd=("--debug" "run")
95          drandCmd=("--debug" "run" "--period" "2s")
96          detached="-d"
```

```
97          args=(run --rm --name node$i --net $NET  --ip ${SUBNET}2$i --volume ${allVolumes[$i]} -d)
98          #echo "--> starting drand node £i: £{SUBNET}2£i"
99          if [ "$i" -eq 1 ]; then
100             drandCmd+=("--leader" "--period" "2s")
101             if [ "$1" = true ]; then
102                 # running in foreground
103                 echo "[+] Running in foreground!"
104                 unset 'args[${#args[@]}-1]'
105             fi
106             echo "[+] Starting the leader of the dkg"
107         else
108             echo "[+] Starting node $i"
109         fi
110         drandCmd+=($dockerGroupFile)
111         docker ${args[@]} "$IMG" "${drandCmd[@]}" > /dev/null
112         sleep 0.1
113         detached="-d"
114     done
115 }
116
117 function cleanup() {
118     echo "[+] Cleaning up the docker containers..."
119     docker stop $(docker ps -a -q) > /dev/null 2>/dev/null
120     docker rm -f $(docker ps -a -q) > /dev/null 2>/dev/null
121 }
122
123 cleanup
124
125 ## END OF LIBRARY
126 if [ "${#BASH_SOURCE[@]}" -gt "1" ]; then
127     echo "[+] run_local.sh used as library -> not running"
128     return 0;
129 fi
130
131 ## RUN LOCALLY SCRIPT
132 trap cleanup SIGINT
133 build
134 run false
135
136 end=$(date +%s)
137 echo "[+] Waiting 3s to get some beacons..."
138 sleep 230
139 while true;
140 do
141     rootFolder="$TMP/node1"
142     distPublic="$rootFolder/groups/dist_key.public"
143     serverId="/key/drand_id.public"
144     drandVol="$rootFolder$serverId:$serverId"
```

```
145     drandArgs=("--debug" "fetch" "private" $serverId)
146
147     runtime=$(python -c "print '%u:%02u' % ((${end} - ${start})/60, (${end} - ${start})%60)")
148     echo "Runtime was $runtime"
149     echo "-----------------------------------------"
150     echo "              Private Randomness         "
151     docker run --rm --net $NET --ip ${SUBNET}11 -v "$drandVol" $IMG "${drandArgs[@]}"
152     echo "-----------------------------------------"
153     checkSuccess $? "verify randomness encryption"
154     echo "-----------------------------------------"
155     echo "               Public Randomness         "
156     drandPublic="/dist_public.toml"
157     drandVol="$distPublic:$drandPublic"
158     drandArgs=("--debug" "fetch" "public" "--public" $drandPublic "${addresses[1]}")
159     docker run --rm --net $NET --ip ${SUBNET}10 -v "$drandVol" $IMG "${drandArgs[@]}"
160     checkSuccess $? "verify signature?"
161     echo "-----------------------------------------"
162     sleep 3
163 done
```

7.3 Proof of concept

In this section we will provide an example of the output produced by our script when it is executed. The script runs with the following command: *./script.sh 10* where 10 represents the number of nodes we want to create. Once the script is executed the screen shown in *Figure 7.1* will appear.

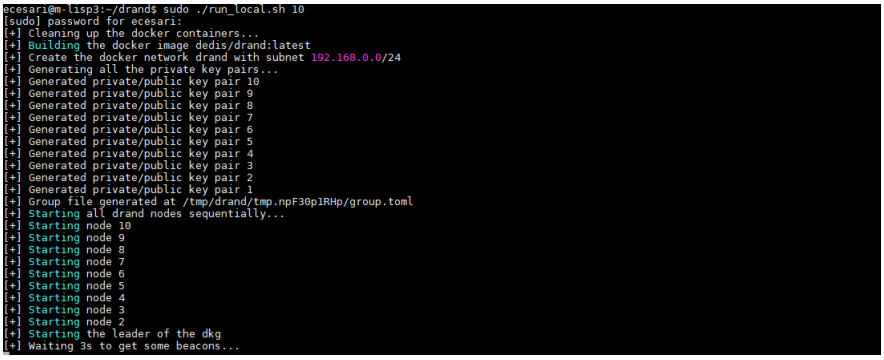

Figure 7.1: Setup phase of the algorihtm

As we can easily see the script generates 10 container dockers which simulate each a different node of the p2p network and each of them produces a private and public key and the public key of the group will be saved in a file called *group.toml*. Then the random beacon generation shown in the *Figure 7.2* will begin.
The script produces a random beacon completely verifiable every five seconds.

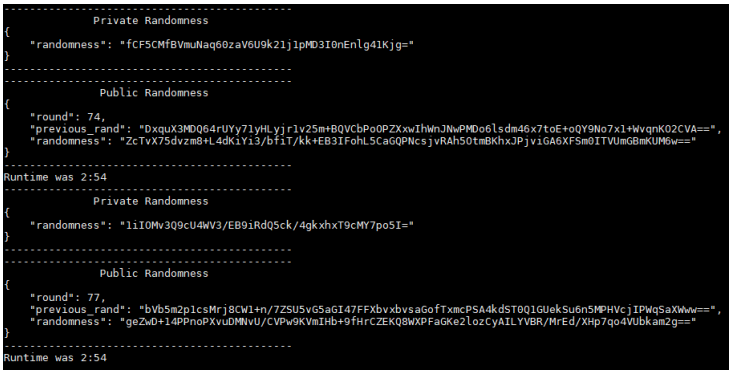

Figure 7.2: Example of the random beacon output

7.4 Performance measurements

7.4.1 Experimental setup

we ran all our experiments and tests on a server from UPC, equipped with:

- Intel Xeon dual core E5-2620 v4
- 64 Gb of RAM
- 2 gb nics

7.4.2 Setup time analysis

Below in Figure 7.1 we will report an analysis on the time needed to generate the shared key, for the DKG algorithm, depending on the number of nodes, to verify the performance and the scalability of my algorithm.
As we can easily see from the graph in Figure 7.1 for a low number of nodes, the algorithm takes reasonable time for the generation of public and private keys, but as the number of nodes increases, the time for the generation of keys grows exponentially, thus highlighting a possible problem of scalability in the future.

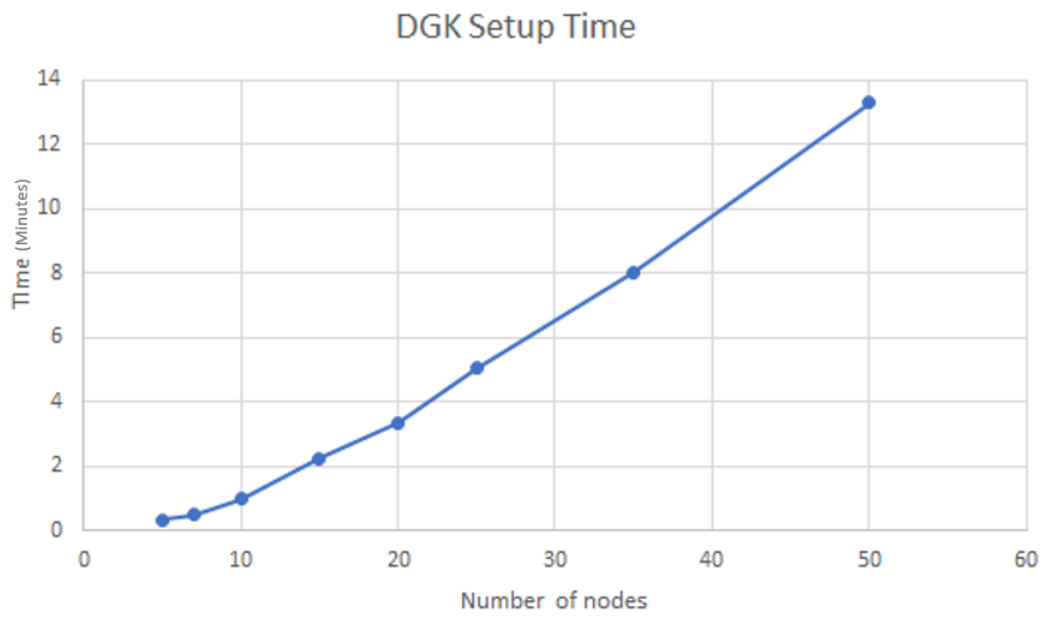

Figure 7.3: Analysis of DGK setup time

7.4.3 Beacon generation time analysis

However it is worth remembering that the setup phase mentioned above must be performed only once, and not at each round, then the times for the generation of random numbers remain constant without depending on the number of nodes as shown in the graph below .

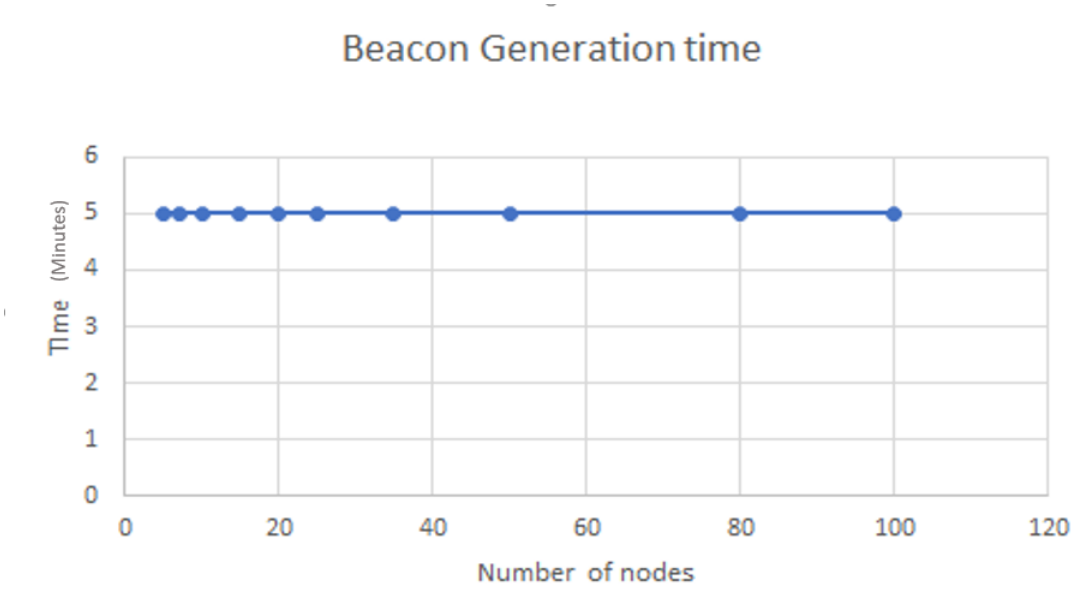

Figure 7.4: Analysis of beacon generation time

This is a positive result to take into consideration and could balance the long setup times needed in the first phase of the algorithm.

Chapter 8

Conclusion

8.1 Final results

The results shown in the two graphs above are meaningful and to be taken into account for the future implementation of my algorithm. The long setup times required to generate the keys for the Randherd algorithm are a problem that can be solved using the blockchain as theorized in my algorithm. The constant time to generate the random beacon instead is a positive factor that adds a great value to the algorithm.
However, the fact that every time a node wants to join the network it is necessary to re-execute the protocol of key generation with its long times is a significant problem, which led me to the conclusion that my algorithm is more suitable for systems where the number of nodes of the peer to peer network changes very rarely, or to allow only nodes with a certain amount of stakes to join the process, so the algorithm have to re-generate the keys less times.
Our algorithm is also a perfect solution for other systems that require public randomness such as tor node protection, selection of encryption parameters for elliptical curves, electronic voting systems, lottery, gambling. In all of these services my algorithm can be useful for generating third-party verifiable randomness.
For example our solution could be integrated into the Nist beacon generator to provide the same service but in more distributed way.

8.2 Future work

In this section we briefly explain the work to be done in the future. After the test phases using the Randherd [10] algorithm described in the previous chapter, we demonstrated the great potential of the algorithm we designed.
So for the future we have to implement our own algorithm, trying to use part of the RanHerd code for the common parts in order to reduce the development time and not reinvent the wheel, and write the code of the part of the algorithm that interacts with the blockchain.

Acknowledgements

I would like to thank my thesis director Professor Albert Cabellos and my thesis co-director Jordi Paillissé for supporting me during these past five months and for their advices and support throughout the thesis work.
A special thank to the Universitat Politècnica de Catalunya and FIB department for accepting me and allowing me to do my thesis as an international student.
Last but not least, I would like to thank my family, my parents Ambrogio and Maria and my sister Erika, for their unconditional support,encouragement and love without which I would not have have come this far.

Bibliography

[1] Elaine Barker et al. *Recommendation for Random Number Generation Using Deterministic Random Bit*. 2013.

[2] dfinity.org. *DFINITY Technology Overview Series Consensus System*. Jan. 2018. URL: https://dfinity.org/pdf-viewer/pdfs/viewer?file=../library/dfinity-consensus.pdf.

[3] Rosario Gennaro et al. "Secure distributed key generation for discrete-log based cryptosystems". In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1999, pp. 295–310.

[4] Paul Kohlbrenner and Kris Gaj. "An embedded true random number generator for FPGAs". In: *Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*. ACM. 2004, pp. 71–78.

[5] Silvio Micali. "ALGORAND: The Efficient and Democratic Ledger". In: *CoRR* abs/1607.01341 (2016). arXiv: 1607.01341. URL: http://arxiv.org/abs/1607.01341.

[6] Silvio Micali, Michael Rabin, and Salil Vadhan. "Verifiable random functions". In: *Foundations of Computer Science, 1999. 40th Annual Symposium on*. IEEE. 1999, pp. 120–130.

[7] NIST.org. *NIST Randomness Beacon*. Dec. 2017. URL: https://www.nist.gov/programs-projects/nist-randomness-beacon.

[8] Torben Pryds Pedersen. "Non-interactive and information-theoretic secure verifiable secret sharing". In: *Annual International Cryptology Conference*. Springer. 1991, pp. 129–140.

[9] Randao.org. *Randao: Verifiable Random Number Generation*. Sept. 2017. URL: http://randao.org/.

[10] E. Syta et al. "Scalable Bias-Resistant Distributed Randomness". In: *2017 IEEE Symposium on Security and Privacy (SP)*. May 2017, pp. 444–460. DOI: 10.1109/SP.2017.45.