HOCHSCHULE MITTWEIDA
UNIVERSITY OF APPLIED SCIENCES

MASTER THESIS

Mr.
Mehul Bafna

Mathematical Aspects and Challenges of the Algorand Blockchain

2022

Faculty of **Applied Computer Sciences and Biosciences**

MASTER THESIS

Mathematical Aspects and Challenges of the Algorand Blockchain

Author:
Mehul Bafna

Study Programme:
Applied Mathematics in Networking and Data Science

Seminar Group:
MA19w1-M

First Referee:
Prof. Dr. Klaus Dohmen

Second Referee:
Prof. Dr. Peter Tittmann

Mittweida, July 2022

Acknowledgement

I want to convey sincere and heartfelt gratitude to Prof. Dr. Klaus Dohmen and Prof. Dr. Peter Tittmann for their kind guidance, supervision, and valuable feedback. I also would like to thank my grandfather Mr. Bahadur Mal Dosi, parents Mr. Sanjay Bafna and Mrs. Sapna Bafna, brother Mr. Harsh Bafna, uncle Amit Dosi and colleague Ms. Shaghik Amirian for kind support in all my endeavors.

Bibliographic Information

Bafna, Mehul: Mathematical Aspects and Challenges of the Algorand Blockchain, 63 pages, 14 figures, Hochschule Mittweida, University of Applied Sciences, Faculty of Applied Computer Sciences and Biosciences

Master Thesis, 2022

Abstract

With the growing market of cryptocurrencies, blockchain is becoming central to various research areas relevant from a mathematical and cryptographic point of view. Moreover, it is capable of transforming the traditional methods involving centralized network operations into decentralized peer-to-peer functionalities. At the same time, it provides an alternative to digital payments in a robust and tamperproof manner by adding the element of cryptography, consequently making it traversable for an individual who is a part of the blockchain network. Furthermore, for a blockchain to be optimal and efficient, it must handle the blockchain trilemma of security, decentralization, and scalability constraints in an effective manner. Algorand, a blockchain cryptocurrency protocol intended to solve blockchain's trilemma, has been studied and discussed. It is a permissionless (public) blockchain protocol and uses pure proof of stake as its consensus mechanism.

I. Contents

II. List of Figures

III. List of Tables

1 Introduction

1.1 Background

Financial transactions have seen an evolution like nothing else. It has gone from the use of metal coins to the use of bitcoins. It has given rise to myriad other cryptocurrencies that form a part of the digital currency world. The standout aspect of the whole affair has been the rapid adoption of cryptocurrencies across the globe which prompts the governments and the tech community alike to proceed with innovative solutions for the shortcomings of these digital currencies. Moreover, these digital currencies use blockchain technology to facilitate a supremely secure and decentralized way to transact across geographies. They use two mechanisms to validate the transactions – Proof of Work (PoW) and Proof of Stake (PoS). Both work as a consensus mechanism to securely operate the network of users and validators to ensure only genuine participants add a transaction on the blockchain.

Blockchain plays a pivotal role in diverse areas, such as cloud computing, cybersecurity, fintech, healthcare, and non-fungible tokens (NFTs) [Dal21]. It relies on the fundamentals of peer-to-peer networking and decentralization. Use cases for blockchain technology are increasing, which calls for continuous improvements in the technology to keep the whole system robust and efficient. It brings us to one of the newer projects – Algorand. This project aims to elevate the utility of cryptocurrencies by increasing transaction speeds and reducing the time it takes for transactions to append to the network [Kra20]. Algorand intends to achieve these attributes by a sound methodology that varies from how conventional cryptocurrencies operate. A noteworthy aspect of the Algorand is its unbiased selection process in choosing candidates with respect to a block for verification processes, thus reducing selection bias. Moreover, it allows users to work with smart contracts and generate tokens representing both new and existing assets. Furthermore, the native cryptocurrency for Algorand is named ALGO. This thesis study provides various mathematical aspects and challenges around the Algorand blockchain.

1.2 Structure of the Thesis

The remaining report has been structured in the following manner:

- Chapter 2 gives advanced cryptographic primitives, including blockchain, hash functions, consensus mechanisms, and Edwards digital signature algorithm (EdDSA).

- Chapter 3 provides the basic structure of the Algorand public distributed ledger, verifiable random functions (VRFs), applying cryptographic sortition in the leader and committee selection procedure, secure flow of information through message sharing, and utility of look-back parameter for Algorand blockchain protocol.
- Chapter 4 explains the security and attack analysis for the Algorand blockchain.
- Chapter 5 provides a conclusion and some areas for future scope.
- The Appendix chapter concludes the report with the implementation of primitive roots, discrete logarithm solution, Byzantine agreement, and Twisted Edwards curve over a finite field $\mathbb{F}_p$ in Python.

2 Advanced Cryptography and Mathematical Prerequisites

2.1 Blockchain

Stuart Haber and Scott Stornetta first gave the idea of blockchain [HS91]. Later, it gained attention when it was employed for Bitcoin by Satoshi Nakamoto [Nak08]. It has various applications in diverse areas such as cryptocurrency, voting mechanisms, supply chain management, and non-fungible tokens (NFTs) [Dal21]. A blockchain is analogous to a database on a distributed basis. Hence, it is accessible by every block of the blockchain network and works as a peer-to-peer network. The main difference that strikes out between blockchain and other forms of technology is the ensured security of enormous data and assured trust without the involvement of a centralized authority. A blockchain is either permissionless (public) or permissioned (private). A blockchain is permissioned if any user is required to go through an authorization process for joining the blockchain. On the contrary, for a permissionless blockchain, any user is able to join without any authorization process. Moreover, Algorand is a permissionless blockchain [Alg22f]. Since Algorand blockchain is a cryptocurrency protocol, the blockchain gives a digital way to store transaction information that is computationally infeasible to alter or reverse. Therefore, it is a decentralized distributed ledger scheme. Furthermore, new blocks are generated based on a consensus mechanism, and on validation, it joins the verified blockchain using cryptographic methods. In general, each block comprises:

- Cryptographic hash: Consists of a hashed value of all transactions of the previous block in a single hash using the concept of Merkle tree[1]
- Timestamp: Stores the information about the block creation time
- Transaction data: Details related to each transaction are stored in transaction data

The first block is termed the Genesis block. Since each subsequent block is linked to the previous block via a secure cryptographic hash, a slight change in any of the above transaction information would lead to inconsistency in the succeeding blocks. Hashing, along with any of the below-discussed consensus mechanisms, makes the blockchain more secure.

[1] Merkle tree refers to a tree where each node (apart from leaf node) consists of concatenated recursive hashing of all its child nodes concatenated together. A leaf node contains the exact transaction data used to create the Merkle tree. The topmost node refers to the Merkle root of the tree comprising the hash of all nodes recursively hashed together [Mer82].

2.2 Hash Functions

Definition 2.1 A *hash function* H is described as a function that creates a map between data of arbitrary size to fixed-size values.

$$H : \{\{\text{Arbitrary string of any length}\}\} \longrightarrow \{\{\text{A fixed-size length output}\}\}$$

The values obtained after passing through the *hash function* are termed *hash values*.

Hash functions are one-way, i.e., it is faster to compute the output from input, but the converse is not computationally feasible. A hash function H is always collision-free if the range set's cardinality is greater or equal to the cardinality of the domain set. Considering a hash function such as SHA-512 produces an output of fixed-size length 512 for an arbitrarily input length. Nevertheless, it must generate one of 2^{512} outputs for each input out of a much larger set of inputs. A hash function with more inputs than outputs is bound to have collisions, and the pigeonhole principle ensures that some inputs hash to the same output. However, collisions are computationally hard to identify due to uniformity in the mapping from input to output. In other words, $H(x) = H(y)$ for some $x, y \in$ *domain(H)* but difficult to identify that for which x, y this holds true [SL07a]. A hash function is compact if it is faster to compute and is collision-resistant.

Cryptographic hash functions can be categorized into keyed and non-keyed cryptographic hash functions [Pre94]. Keyed cryptographic hash functions involve the usage of a secret key, while the latter does not require a secret key. Some majorly employed non-keyed cryptographic hash functions in various cryptographic protocols are SHA-256, SHA-512, and MD-5, with the fixed-size length output of 256, 512, and 128.

Majorly used non-keyed cryptographic hash functions				
Hash function	SHA-256	SHA-384	SHA-512	MD5
Output Length	256 bits	384 bits	512 bits	128 bits

Table 2.1: Hash functions

2.3 Consensus Mechanisms

2.3.1 Byzantine Agreement

Byzantine agreement problem, also referred to as Byzantine fault, occurs specifically in distributed computing systems. On failure of one or more components, it becomes difficult to determine whether a component is working due to the lack of sufficient information. To make a system more resilient against such failures and attacks, the concept of the Byzantine agreement was studied and given by M.C. Pease, R.E. Shostak, and L. Lamport [PSL80].

Definition 2.2 Given a set of k independent components with each component having a secret information I_j for all $j \in \{1, \cdots, k\}$ with the below assumptions:

- Point-to-point fail-free communication among all the components
- At most i components are faulty (malicious) such that $k \geq 3i+1$

is termed as *tolerant* towards Byzantine faults, if for a component X after communicating its secret information I_X with all other components it follows that:

1. If X is non-faulty (reliable) component, then all non-faulty components agree on the value I_X.
2. In any scenario, all reliable components agree on a common value.

The faulty component(s) controls the right to exchange its information. In order to determine the i faulty components, $i+1$ iterations of exchanging information suffice [PSL80]. Algorand employs Byzantine agreement by means of optimal cryptographic tweaking to generate a new block discussed in a later section. In Appendix A.2, a Python program based on the methodology of the Byzantine agreement has been implemented.

Example 2.3 $k = 4$ and $i = 1$.

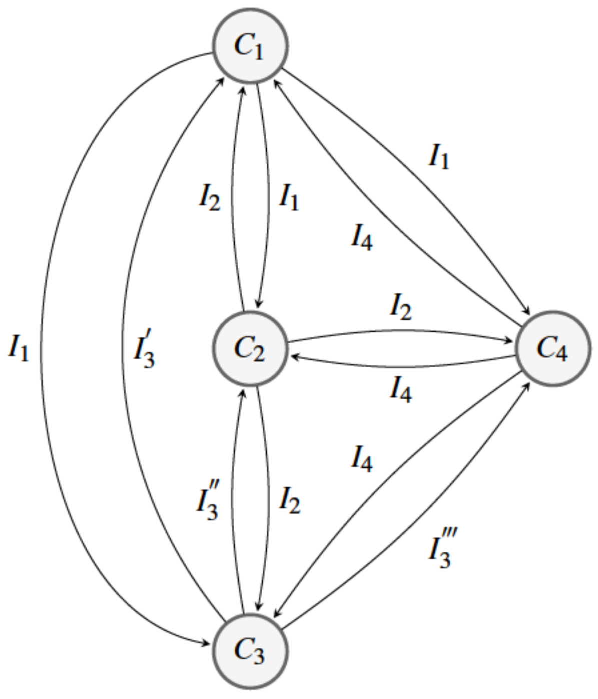


Figure 2.1: Communicating information

For $I_3^{'} \neq I_3^{''} \neq I_3^{'''}$, Table 2.2 is obtained

Agreed value of components as per majority				
	Value for C_1	Value for C_2	Value for C_3	Value for C_4
C_1	-	I_2	NIL	I_4
C_2	I_1	-	NIL	I_4
C_3	a	b	-	c
C_4	I_1	I_2	NIL	-

Table 2.2: Analyzing information - Case I (Byzantine agreement)

For any other possibilities, with either two or all of $I_3^{'}$, $I_3^{''}$, $I_3^{'''}$ having value x, Table 2.3 is rendered.

Agreed value of components as per majority				
	Value for C_1	Value for C_2	Value for C_3	Value for C_4
C_1	-	I_2	x	I_4
C_2	I_1	-	x	I_4
C_3	a	b	-	c
C_4	I_1	I_2	x	-

Table 2.3: Analyzing information - Case II (Byzantine agreement)

Since C_3 is faulty(malicious), a, b, c, x are random values. For every scenario the above mentioned two properties are satisfied.

2.3.2 Proof of Work (PoW)

Cynthia Dwork and Moni Naor initiated proof of work (PoW) [DN93]. Later it gained attention when it was employed as a decentralized consensus mechanism by Bitcoin, where members (miners) involved in the network solve an arbitrary computationally complex mathematical puzzle in order to authenticate transactions for processing and generating a new block [Nak08]. It is based on the difficulty of finding the original value based on the hash value, i.e., it is a one-way function. Until the status of a transaction is confirmed, it initiates as follows. First, transactions are merged into a block. Then, the miners check whether these transactions are authentic by hashing the block header of the candidate block. The first miner who obtains the correct solution or a solution closest to the correct one receives the monetary benefits and covers the individual miner's transaction fees. Finally, the validated transactions are attached to the blockchain in a verified block. The transactions are validated in a protected peer-to-peer network without the involvement of a trusted third party. Moreover, energy level surmounts as the number of miners joining the network increases.

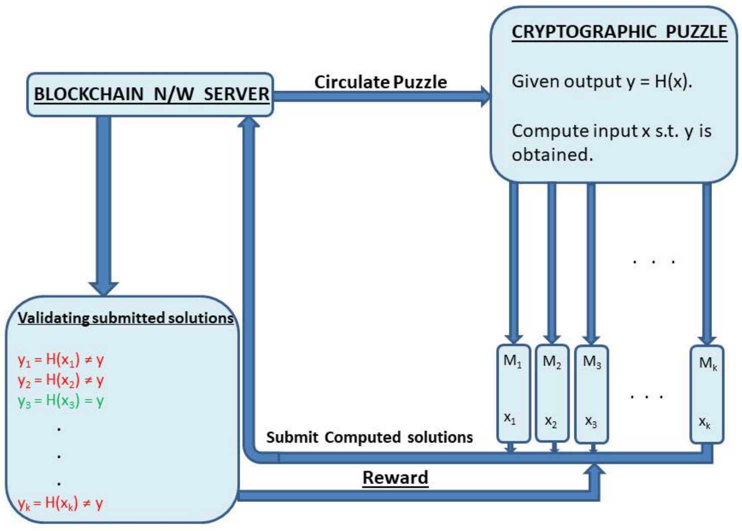


Figure 2.2: Proof of Work methodology
[Aca19]

2.3.3 Proof of Stake (PoS)

Proof of Stake (PoS) is a consensus technique created to address the shortcomings of proof of work. Sunny King and Scott Paul initially gave proof of stake for the peer-to-peer cryptocurrency Peercoin [KN12]. It was further adapted as a consensus mechanism for the Ethereum blockchain [But14]. The selection of candidates for verification of transactions and generation of a new block comprises a weighted random selection where the weights of the individual participants are obtained from the assets owned by a participant(the "stake"). After selection, transactions are verified, validated, and put together in a block. Verifiers end up with monetary benefits in the form of a transaction fee. In order to avoid inconsistency, the sum over all transactions fee rewarded to a validator must not exceed the individual stake. While there have been many variations with the proof of stake mechanism, Algorand operates on the pure proof of stake approach that combines the Byzantine consensus with the traditional proof of stake mechanism, making it more soundproof and resilient.

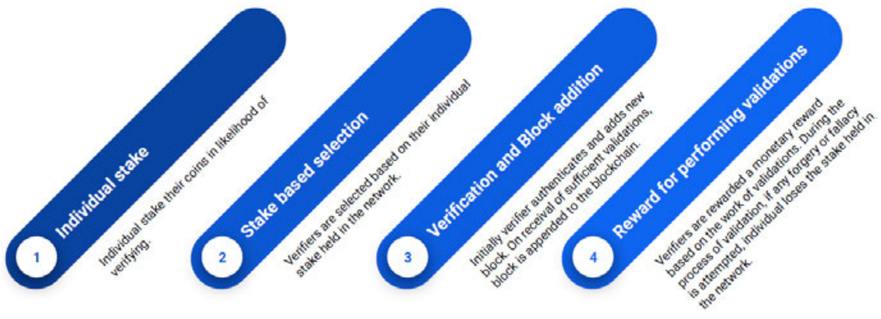

1 Individual stake
Individual stake their coins in likelihood of verifying.
2 Stake based selection
Verifiers are selected based on their individual stake held in the network.
3 Verification and Block addition
Initially verifier authenticates and adds new block. On receival of sufficient validations, block is appended to the blockchain.
4 Reward for performing validations
Verifiers are rewarded a monetary reward based on the work of validations. During the process of validation, if any forgery or fallacy is attempted, individual loses the stake held in the network.

Figure 2.3: Proof of Stake methodology

2.4 Digital Signature Scheme

The signature is physically a portion of the document or contracts to ensure that the signer is held responsible for the details mentioned within the document. However, since forgery is highly possible, it becomes difficult to consider such a physical signature process. Therefore, it is required to consider methods to sign messages over a digital medium that is functionally equivalent to a physical signature but more resilient to any forgery or malicious attacks. Therefore, digital signatures are used for signing messages in various cryptographic procedures.

A digital signature combines mathematical steps for validation on authentication of digital messages or documents. It ensures that the information is issued from the signer and not altered during the exchange of information. A digital signature is authentic, i.e., satisfying all necessary mathematical prerequisites and ensuring a significant trust between sender and receiver. As asymmetric cryptosystems, it employs a pair of public and private keys in signature generation and verification. Numerous digital signature schemes are based on the difficulty of computing discrete logarithms. Below are some definitions related to discrete logarithm [SL07b].

Definition 2.4 For a prime p, its *primitive root* α is defined as a number that satisfies

$$\{1,2,\cdots,p-1\}=\{\alpha^1,\alpha^2,\cdots,\alpha^{p-1}\}\ (mod\, p)$$

α is a *primitive root* of p, if and only if α is a generator of the cyclic group $\mathbb{Z}_p^*$ [2].

3	3	9	10	13	5	15	11	16	14	8	7	4	12	2	6	1
5	5	8	6	13	14	2	10	16	12	9	11	4	3	15	7	1
6	6	2	12	4	7	8	14	16	11	15	5	13	10	9	3	1
7	7	15	3	4	11	9	12	16	10	2	14	13	6	8	5	1
10	10	15	14	4	6	9	5	16	7	2	3	13	11	8	12	1
11	11	2	5	4	10	8	3	16	6	15	12	13	7	9	14	1
12	12	8	11	13	3	2	7	16	5	9	6	4	14	15	10	1
14	14	9	7	13	12	15	6	16	3	8	10	4	5	2	11	1

Table 2.4: Primitive roots of 17

Appendix A.1.1 implements a Python program for the computation of primitive roots.

Definition 2.5 Let p be a prime with α as its primitive root. For any $n \in \{1,2,\cdots,p-1\}$, there exist an $x \in \{1,2,\cdots,p-1\}$ that satisfies

$$n=\alpha^x\ (mod\ p)$$

[2] A group that can be produced by a single element (generator) is termed a *cyclic group*.

For every value of n, this value x is unique and is termed as *discrete logarithm* of n to base α modulo p and is denoted as

$$x = \log_{\alpha} n \ (mod\ p)$$

As an example, discrete logarithm of 9 to base 7 modulo 17, given as $\log_7 9 \ (mod\ 17)$ is 6 from table 2.4. In the Appendix A.1.2, the Python program for the computation of discrete logarithm is implemented.

In general, the digital signature scheme comprises three procedures as given below:

- Key-generation algorithms (G): Outputs a secret (private) key with its corresponding public key.
- Signing algorithm (S): Outputs a signature on the basis of the message, secret key, and hashing.
- Verification algorithm (V): Based on the signature obtained from S and the message and public key, the verifier responds 0 or 1. 0 represents a rejection and 1 an acceptance.

ElGamal digital signature scheme, Elliptic curve digital signature algorithm (ECDSA), Schnorr signature scheme, and Edwards-curve digital signature algorithm (EdDSA) are some of the commonly used signature algorithms [Bon05, JMS01, Sch89, BDL$^+$12]. Algorand blockchain employs highly secure Edwards-curve digital signature algorithm (EdDSA) signatures [Alg22b].

2.4.1 Edwards-curve Digital Signatures (EdDSA)

Edwards Curves

Harold M. Edwards proposed a different form for representing elliptic curves over an algebraically closed non-binary field K, i.e., char(K) $\neq 2$ and comprises each root of any non-constant polynomial in $K[x]$ [Edw07] :

$$x^2 + y^2 = c^2(1 + x^2y^2) \tag{2.1}$$

These curves were further studied and presented in a newer form termed *Edwards curves* by Daniel Bernstein and Tanja Lange as described in definition 2.6 [BL07].

Definition 2.6 An *Edwards curve* over a field K, with *char(K)* $\neq$ *2* is represented as

$$x^2 + y^2 = 1 + dx^2y^2 \tag{2.2}$$

where $d \in K \setminus \{0,1\}$.

For $d = 0$ and $d = 1$ a unit circle and 4 lines $x = \pm 1$ and $y = \pm 1$ are rendered respectively. Below are Edwards curves over $K = \mathbb{R}$ for $d = \pm 10$. The following script has been computed using Sage computer-algebra system to generate plots [DSJ+20].

```
R.<x, y> = QQ[]
C = Curve(x^2 + y^2 - 1 +10*x^2*y^2)
C.plot()
C = Curve(x^2 + y^2 - 1 -10*x^2*y^2)
C.plot()
```

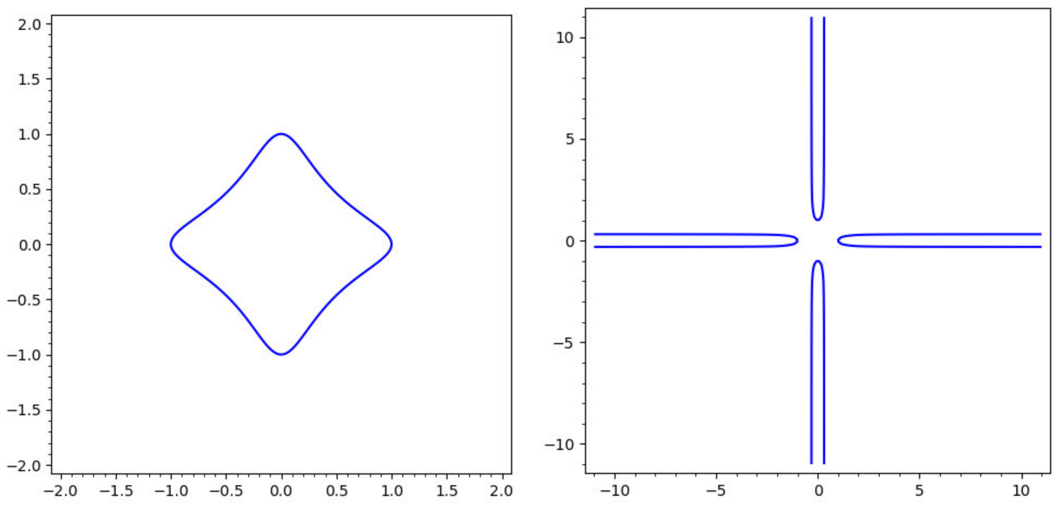


Figure 2.4: Edwards curve for d = -10 and d = 10

Definition 2.7 For integers n, p, if there exists $m \in \mathbb{Z}$ such that $n = m^2 \ (mod\ p)$, then n is described as *quadratic residue* modulo p.

Definition 2.8 For a prime y and integer x, the *Legendre symbol* for x over y is a function defined as,

$$\left(\frac{x}{y}\right) = \begin{cases} 1 & \text{if } x \text{ is a \textit{quadratic residue} modulo } y \text{ and } y \nmid x, \\ 0 & \text{if } y \mid x, \\ \text{-1} & \text{if } x \text{ is a \textit{non-quadratic residue} modulo } y. \end{cases}$$

Addition of Points on Edwards Curves

Suppose E_d represents a Edwards curve as -

$$E_d : x^2 + y^2 = 1 + dx^2y^2$$

Let $P_i = (x_i, y_i) \in E_d$ for $i \in \{1,2\}$ then $-P_i = (-x_i, y_i)$. Also $dx_1x_2y_1y_2 \neq \pm 1$, then the addition of the two points is given as:

$$(x_1,y_1) + (x_2,y_2) = (x_3,y_3) = (\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - x_1x_2}{1 - dx_1x_2y_1y_2})$$

$$2(x_i,y_i) = (\frac{2x_iy_i}{1 + dx_i^2y_i^2}, \frac{y_i^2 - x_i^2}{1 - dx_i^2y_i^2}) \text{ for } i = 1,2$$

Here, $(0,1)$ is the neutral element and $(x_3,y_3) \in E_d$.
It has been shown that for $\left(\frac{d}{p}\right) = -1$, addition formula always hold without any exceptions [BL07].

Example 2.9 $P_1 = (17,28)$ and $P_2 = (22,85)$. Compute P_1+P_2, $2P_1$, and $2P_2$ for $d = 30$ over $\mathbb{F}_{97}$.

$$P_1 + P_2 = (24 \cdot [83^{-1}]_{97}, 66 \cdot [16^{-1}]_{97}) = (26,89)$$

$$2P_1 = (79 \cdot [6^{-1}]_{97}, 10 \cdot [93^{-1}]_{97}) = (94,46)$$

$$2P_2 = (54 \cdot [46^{-1}]_{97}, 48 \cdot [53^{-1}]_{97}) = (56,43)$$

where $[a^{-1}]_b$ denotes modular inverse of a with respect to b.

Twisted Edwards Curves

Twisted Edwards curves are a generalized form of the curves defined in Definition 2.6 as it covers more curves over finite fields [BBJ$^+$08].

Definition 2.10 A *twisted Edwards curve* over a field K, with *char(K)* $\neq$ *2* is represented as

$$ax^2 + y^2 = 1 + dx^2y^2 \tag{2.3}$$

where *a,d* $\in K \setminus \{0\}$ *and a* $\neq$ *d.*

The curve reduces to the *Edwards curve* for $a = 1$. In the Appendix A.3, a Python program is implemented plotting all points lying on $E_{a,d}$ and also checks whether a point $P \in E_{a,d}$ or $P \notin E_{a,d}$ over a finite field $\mathbb{F}_p$ for p being a prime. A mathematical overview with respect to finite field has been provided in Appendix B.3.

Addition of Points on twisted Edwards Curves

Suppose $E_{a,d}$ represents a twisted Edwards curve as -

$$E_{a,d} : ax^2 + y^2 = 1 + dx^2y^2$$

Let $P_i = (x_i, y_i) \in E_{a,d}$ for $i \in \{1,2\}$ then $-P_i = (-x_i, y_i)$. On the condition that $dx_1x_2y_1y_2 \neq \pm 1$, then the addition of the two points is given as:

$$(x_1,y_1) + (x_2,y_2) = (x_3,y_3) = (\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2})$$

$$2(x_i,y_i) = (\frac{2x_iy_i}{1 + dx_i^2y_i^2}, \frac{y_i^2 - ax_i^2}{1 - dx_i^2y_i^2}) \text{ for } i = 1,2$$

Here, $(0,1)$ is the neutral element and $(x_3,y_3) \in E_{a,d}$.

Example 2.11 $P_1 = (1,8)$ and $P_2 = (7,3)$. Compute P_1+P_2, $2P_1$, and $2P_2$ for $a = 4, d = 5$ on $\mathbb{F}_{23}$.

$$P_1 + P_2 = (13 \cdot [13^{-1}]_{23}, 19 \cdot [12^{-1}]_{23}) = (1,15)$$

$$2P_1 = (16 \cdot [22^{-1}]_{23}, 14 \cdot [3^{-1}]_{23}) = (7,20)$$

$$2P_2 = (19 \cdot [21^{-1}]_{23}, 20 \cdot [4^{-1}]_{23}) = (2,5)$$

where $[a^{-1}]_b$ denotes modular inverse of a with respect to b.

The points considered in the numerical examples for addition with respect to Edwards curves and twisted Edwards curves have been computed from the Python program attached in appendix A.3.

Theorem 2.12 *[BN19] Suppose $E_{a,d}$ be a twisted Edwards curve defined over $\mathbb{F}_p$, where prime characteristic of $\mathbb{F}_p$ is greater than 2. Assume two points (x_1,y_1) and (x_2,y_2) lie on $E_{a,d}$. Then the addition of the two points given as*

$$(x_1,y_1) + (x_2,y_2) = (\frac{x_1y_2 + x_2y_1}{1 + dx_1x_2y_1y_2}, \frac{y_1y_2 - ax_1x_2}{1 - dx_1x_2y_1y_2})$$

is always defined without any exceptions if

- $\left(\frac{a}{p}\right) = 1$

- $\left(\frac{d}{p}\right) = -1$

Proof: Assume $E_{a,d}$ be a twisted Edwards curve in $\mathbb{F}_p$, where $char(\mathbb{F}_p) > 2$, $\left(\frac{a}{p}\right) = 1$ and $\left(\frac{d}{p}\right) = -1$. Suppose two points (x_1, y_1) and (x_2, y_2) lie on $E_{a,d}$ and on the contrary assume that addition law does not hold, i.e, $dx_1x_2y_1y_2 = \pm 1$. Therefore, $x_1x_2y_1y_2 \neq 0$. Since, (x_2, y_2) lies on $E_{a,d}$, it gives

$$\begin{aligned} ax_2^2 + y_2^2 &= 1 + dx_2^2y_2^2 \ (mod\ p) \\ &= (\pm 1)^2 + dx_2^2y_2^2 \ (mod\ p) \\ &= (dx_1x_2y_1y_2)^2 + dx_2^2y_2^2 \ (mod\ p) \\ &= dx_2^2y_2^2(dx_1^2y_1^2 + 1) \ (mod\ p) \end{aligned}$$

Also (x_1, y_1) lies on $E_{a,d}$, it gives

$$ax_2^2 + y_2^2 = dx_2^2y_2^2(ax_1^2 + y_1^2) \ (mod\ p)$$

Further we proceed as follows

$$\begin{aligned} (\sqrt{a}x_2 \pm y_2)^2 &= (\sqrt{a}x_2)^2 + y_2^2 \pm 2\sqrt{a}x_2y_2 \ (mod\ p) \\ &= ax_2^2 + y_2^2 \pm 2\sqrt{a}dx_1y_1x_2^2y_2^2 \ (mod\ p) \\ &= dx_2^2y_2^2(ax_1^2 + y_1^2) \pm 2\sqrt{a}dx_1y_1x_2^2y_2^2 \ (mod\ p) \\ &= dx_2^2y_2^2(ax_1^2 + y_1^2 \pm 2\sqrt{a}dx_1y_1x_2^2y_2^2) \ (mod\ p) \\ &= dx_2^2y_2^2(\sqrt{a}x_1 \pm y_1)^2 \ (mod\ p) \end{aligned}$$

Therefore, $d = \left(\frac{(\sqrt{a}x_2 \pm y_2)}{x_2y_2(\sqrt{a}x_1 \pm y_1)}\right)^2 \ (mod\ p)$

Assuming that $(\sqrt{a}x_1 \pm y_1) \neq 0 \ (mod\ p)$, since $x_1x_2y_1y_2 \neq 0$, a is non-zero as studied in the definition and as p is a prime, $[(x_2y_2\sqrt{a}x_1 \pm x_2y_2y_1)^{-1}]_p$ will always exist and is computed through extended euclidean algorithm. This renders $\left(\frac{d}{p}\right) = 1$ that contradicts our initial assumption of $\left(\frac{d}{p}\right) = -1$. Due to this contradiction, $(\sqrt{a}x_1 \pm y_1) \neq 0 \ (mod\ p)$ does not hold that gives $x_1 = 0 \ (mod\ p)$ and $y_1 = 0 \ (mod\ p)$ which contradicts $dx_1x_2y_1y_2 = \pm 1$. Hence, addition law always hold for $\left(\frac{d}{p}\right) = -1$ and $\left(\frac{a}{p}\right) = 1$ [BN19].

□

Lemma 2.13 *Let $E_{a,d}$ denote twisted Edwards curve over $\mathbb{F}_p$ where p is an odd prime. If the number of points on the curve is n, then $4|n$.*

Proof: Assume (u,v) lies on the $E_{a,d}$, i.e.,

$$au^2+v^2=1+du^2v^2(mod\ p)$$

Since, the exponent of x and y in $E_{a,d}$ are even, $(u,p-v)$, $(p-u,v)$ and $(p-u,p-v)$ will lie on the curve $E_{a,d}$ as well. This holds true for each such (u,v) where $u,v \in \mathbb{F}_p$.

$\square$

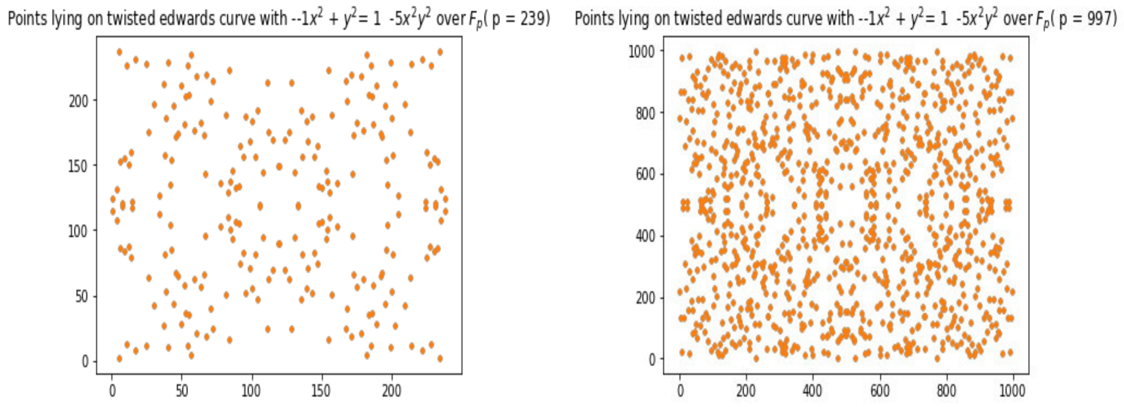


Figure 2.5: Twisted Edwards curve over $\mathbb{F}_{239}$ and $\mathbb{F}_{997}$

Process for Generating and Verifying Signatures

Secret and public keys are denoted as sk and pk. A random input, when passed in the key generation algorithm (G), produces pair (sk, pk) [MRV99].

Parameters used in the signature scheme are as follows:

- An integer $m \geq 10$
- A cryptographic hash function H rendering $2m\text{-}bit$ output
- A prime power p such that $4|(p-1)$
- An element $d \in \mathbb{F}_p$ such that $d \neq c^2\ (mod\ p)$ for any $c \in \mathbb{F}_p$
- A point K lying on the curve $E_{-1,d}$, other than the neutral element (0,1) over $\mathbb{F}_p$
- A prime q such that $2^{m-4} \leq q \leq 2^{m-3}$ and qK gives the neutral element of $E_{-1,d}$

EdDSA's sk is a $m\text{-}bit$ string t. Value of $H(t)$ is denoted as h. Let h_0 and h_1 denotes the first m-bits and last m-bits of h respectively. Let j denotes the integer derived from h_1. jK is considered as pk and h_0 is utilized for signing message M.

Signing

- Define $r = H(h_0 \mid\mid M)$
- Define $R = rK$
- Define $S = r + H(R \mid\mid pk \mid\mid M) \cdot j \ (mod\ q)$

(R, S) is the signature generated.

Verification

- If $R, pk \in E_{-1,d}$ is checked initially.
- On validation of initial condition, computes $v_1 = 8R + 8H(R \mid\mid pk \mid\mid M) \cdot pk$
- Then computes $v_2 = 8SK$

If $v_1 = v_2$, signature is validated.

Why is it correct?

$$v_1 = 8R + 8H(R \mid\mid pk \mid\mid M) \cdot pk$$

$$= 8rK + 8H(R \mid\mid pk \mid\mid M) \cdot jK$$

$$= 8K(r + H(R \mid\mid pk \mid\mid M) \cdot j)$$

$$= 8SK$$

$$= v_2$$

Algorand uses fast functioning, highly secure EdDSA as Ed25519 curves with specifications as given in the below table.

Specification	Value
m	256
H	SHA-512
p	$2^{255} - 19$
q	27742317777372353535851937790883648493
a	-1
d	$-121665 \cdot [121666^{-1}]_{2^{255}-19}$
K	$(x, 4 \cdot [5^{-1}]_{2^{255}-19})$

Table 2.5: Ed25519 specifications
[Ber06]

3 Mathematical Aspects of Algorand

Algorand is a permissionless cryptocurrency blockchain protocol given by Silvio Micali in 2016 [CM16]. It is designed in a manner to solve the Blockchain trilemma [Alg21]. The three factors involved in the blockchain trilemma are:

1. Decentralization: Ensuring the operations are performed without depending on a centralized node or server
2. Security: Network is resilient against attacks by individuals acquiring a significant chunk of blockchain network resources
3. Scalability: Assuring that the blockchain network remains consistent while accommodating a vast amount of transactions and users

The blockchain system is termed Algorand since there is an element of randomness associated with every step in the approach for end-to-end block generation. Moreover, the random computations performed are strongly resistant to mutability and remain unpredictable. Therefore, rather than the power being in some limited hands, the power resides with all individuals that constitute the Algorand blockchain. Furthermore, the occurrence of forking in the Algorand blockchain is highly negligible as each block is separately agreed upon by a committee of users and remains intact in the blockchain [Alg22e]. Since Proof of Work (PoW) is computationally expensive and extremely complex, Algorand relies on the pure proof of stake consensus mechanism, an amalgamation of Proof of Stake (PoS), and the Byzantine agreement protocol.

3.1 Basic Structure of Algorand's Public Distributed Ledger

Algorand is systematically performed in rounds denoted as r $(r > 0)$ [CM16]. Let pub^r denote the set of public keys at the beginning of round r and amt^r_j be the number of money units held by a public key in round r for $j \in pub^r$. The structure is explained in the following steps :

- The initial status S^0 of the public distributed ledger is known to everyone involved in the system and is represented as

$$S^0 = (pub_1, amt_1), \cdots, (pub_n, amt_n)$$

where n is the total initial public keys in the whole transaction system. When round

r begins, the public ledger status is represented as

$$S^r = \{(j, amt_j^r)\ \forall j \in pub^r\}$$

where a public ledger demonstrates the respective money units held by an individual and keeps a track of all the existing transactional records.

- Let m,n denotes two public keys at the beginning of round r with amt_m^r, amt_n^r money units held respectively such that $amt_m^r \geq amt_n^r \geq 0$. Then an authentic payment $\mathscr{P}$, is a digital signature corresponding to amt_m^r, represents the transfer of $x\ (\leq amt_m^r)$ units from m (sender) to n (beneficiary) appended with some additional information as:

$$\mathscr{P} = Sig_m(m,n,x,I,H(\mathscr{I}))$$

where I contains information that is useful but insensitive whereas $\mathscr{I}$ comprises useful critical information. Since $\mathscr{I}$ is sensitive information, it is hashed. For generating digital signatures, Edwards curve digital signature algorithm (EdDSA) is utilized [Alg22b].

- Digital signatures ensure that none of the information relevant to payment is counterfeited by any other user. Since H is an ideal hash function, it assures that the sensitive information $\mathscr{I}$ is not interpreted unless a brute force approach is applied, which is computationally infeasible. In other words, it is computationally hard to find a $\mathscr{I}^{'}$ such that $H(\mathscr{I}) = H(\mathscr{I}^{'})$
- The blockchain network aims to systemize all payments into an ordered arrangement of blocks that are connected via a cryptographic hash and always possess the below characteristics:

  1. As the block is verified and appended to the network, it becomes common knowledge in the blockchain network.
  2. All payments existing in any block are authentic concerning each payer's money units as per the initial status and payments in previous blocks.
  3. Every payment after being verified becomes simultaneously visible in a block.

- Blocks in the Algorand blockchain are relative to rounds. For a round r the respective block B^r is given as

$$B^r = (r, PAY^r, Q^{r-1}, H(B^{r-1}))$$

where PAY^r refers to set of all valid payments of round r, Q^r is a recursion based value whose computation will be studied in section 3.5 and $H(B^{r-1})$ represents the hash of the previous block. A block B^r is termed authentic in Algorand blockchain, if there exist a block certificate $CERT^r$ for the r^{th} block. It comprises of digital signatures of majority (through Algorand's Byzantine agreement protocol) of individuals of SV^r, where SV^r refers to set of verifiers for verifying the authenticity of block r. Afterwards, it becomes a proven block $\overline{B^r}$.

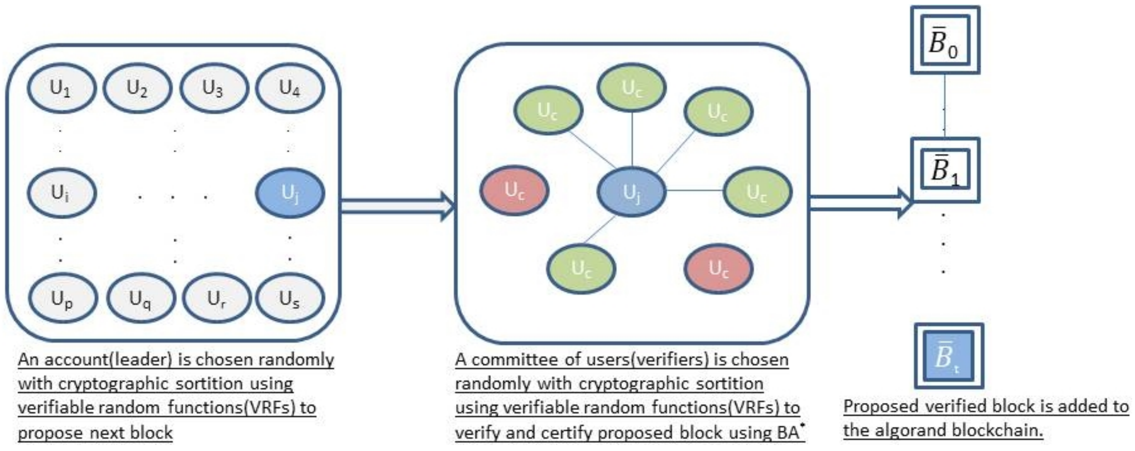

U1
U2
U3
U4
Ui
Uj
Up
Uq
Ur
Us
Uc
Uc
Uc
Uc
Uj
Uc
Uc
Uc
B̄0
B̄1
B̄t
An account(leader) is chosen randomly with cryptographic sortition using verifiable random functions(VRFs) to propose next block
A committee of users(verifiers) is chosen randomly with cryptographic sortition using verifiable random functions(VRFs) to verify and certify proposed block using BA*
Proposed verified block is added to the algorand blockchain.

Figure 3.1: Algorand viewpoint from higher level

3.2 Byzantine Agreement (BA*) in Algorand

Since many users operate on the Algorand blockchain, it is computationally inconsistent and unscalable to approach the traditional BA considering every existing user on the network. Therefore, the verification is undertaken by a considerably small set of randomly chosen users called verifiers via an enhanced Byzantine agreement denoted as BA* where these verifiers do not share any hidden, unknown information. Although, the initial conditions for running a BA remain unaltered for BA*, i.e., an honest majority ratio that is not less than 2/3. Applying the BA* protocol with a randomly selected set of verifiers ensures the scalability parameter is solved in a feasible manner [GHM$^+$17]. Moreover, the procedure enables higher security. Once a verifier has participated in the Byzantine agreement, the verifier becomes insignificant since BA* does not hold any private state, making it unbiased towards any user's participation. In other words, it follows user replacement, i.e., for each step, new committee members are chosen, demonstrating randomness and security within the process. BA* is performed via two stages. The first stage consists of the Graded Consensus (GC) protocol. The rendered output on the first stage's termination is employed as an input for running the second stage, i.e., Binary Byzantine Agreement (BBA*).

3.2.1 Graded Consensus (GC) Protocol

The Graded Consensus (GC) protocol was studied and given by Paul Feldman and Silvio Micali [FM89].

Definition 3.1 Suppose P is a cooperatively terminated protocol such that the set S of all users is universally known, and the identity for each user (the sender) is denoted as

i. In this protocol, only the sender carries a private value v. Then with every iteration involving n players in which at most t are fraudulent, P is an (n,t)-*graded consensus protocol* such that each benign user output a value-grade pair (v_u, g_u) where $g_u \in \{0,1,2\}$ satisfying the below properties [CM16]:

- Absolute difference between the grade of any two honest players is less than or equal to 1.
- If the grades of two honest players are positive, it implies that they have the same output value.
- If the sender is benign, then for every honest player u, the outputted value equals the shared private value of the sender, i.e., v and $g_u = 2$.

The GC protocol is performed as follows. In the first step of the graded consensus (GC) protocol, each user u involved shares its secret input v_u to all involved users. In the second step, the first step obtained value is circulated to other participating users. If greater than 2/3 of obtained values are identical on the second step distribution, this obtained value is distributed. Otherwise, no information is exchanged [FM89].

The output rendered on performing the steps of first stage is (v_u, g_u) for every user u is interpreted below:

- If the ratio of the number of users from whom user u has received some y from the second step with total users involved is greater than 2/3, then $v_u = y$ and $g_u = 2$.
- Moreover, if the ratio of the number of users from whom user u received some y in the second step with total users involved is between 1/3 and 2/3, then $v_u = y$ and $g_u = 1$.
- Otherwise, $v_u = w$ and $g_u = 0$, where w is a random distinguished string not belonging to $range(H)$.

Here, g_u represents a grading parameter in judging the certainty of a value that gradually increases as the ratio grows.

3.2.2 Binary Byzantine Agreement (BBA*)

The Binary Byzantine agreement (BBA*) protocol depends on the majority of honest users and quickly provides an output. In this protocol, individuals on repetitive basis exchange boolean values, and different individuals leave the protocol at different times conditioned on the majority of consensus obtained.

The BBA* is a 3-step protocol. It is a binary-based protocol; therefore, the values considered are either of the form 0 or 1. For each step, if the number of instances for any binary values is greater than two-thirds of the total instances, then initial input b_u is fixed to that value. After outputting o_u as the same binary value, stalls the protocol [CM16].

As per the output received from GC protocol, in BBA* protocol first step, user u assigns its private binary input b_u for binary BA* [3] as 0, if $g_u = 2$ and 1 otherwise. In the next step, every user u performs the remaining steps of binary BA* protocol, till binary output o_u is obtained. Output is interpreted as $o_u = v_u$ if $b_u = 0$ and $o_u = w$ if $b_u = 1$ [Mic17]. It always maintains the two fundamental characteristics:

1. There exists $o \in V \bigcup \{w\}$ where $o_u = o$ for all honest users u illustrating an agreement among all honest users.
2. If the initial value for each honest user is $v \in V$, then $o = v$, thus maintaining the consistency in the consensus of honest users.

where V, w denotes the set of the initial value of the users and a distinguished random string not belonging to $range(H)$ respectively.

In BA* protocol each participating user u performs graded consensus (GC) protocol in order to render v_u, g_u. This information is employed in performing BBA* to obtain a mutual consensus over the value. A BA* protocol utilizes the idea behind the traditional Byzantine agreement to optimally and feasibly overcome scalability issues.

In order to determine whether a user is chosen either as a leader or verifier in the block generation process, they are required to implement cryptographic sortition by themselves. Cryptographic sortition is applied using verifiable random functions (VRFs), studied in Section 3.3.

3.3 Verifiable Random Functions (VRFs)

Verifiable random functions (VRFs) were studied and introduced by Silvio Micali, Salil Vadhan, and Michael Rabin [MRV99]. It is employed to map a given input to a verifiable pseudorandom output. It finds applications in numerous cryptographic procedures and cybersecurity schemes. Moreover, with an element of pseudorandomness, it assures more security and robustness.

For a function f, in order to be a VRF, it always maintains the below properties:

- A consistent and direct representation ensures that f is calculated efficiently.
- A consistent and indirect representation assures that f is not calculated efficiently.

The first property ensures that the individual holder of the secret key easily evaluates

[3] Binary BA* is derived from probabilistic binary BA initiated by Silvio Micali and Paul Feldman [FM89].

the function and generates corresponding rendered proof. While the second property guarantees that the verifier with a linked public key and the proof can conclude whether the evaluated value is valid or not.

A VRF comprises three algorithms, $KeyGen$ to generate key pair, $Eval$ for calculating pseudorandom output with proof, and $Verify$ for verifying whether the outputted value is authentic or not, as explained below [Gor18]:

1. $KeyGen(\lambda)$ - Takes a random input as λ and renders a secret and verification key pair (sk, pk). pk and sk are linked; therefore, it is smoother to obtain pk from sk. On the contrary, the converse is computationally infeasible. sk is analogous to trapdoor information leading to pk. The length of the secret and verification key for the Algorand blockchain is 256 bits [Alg22b].

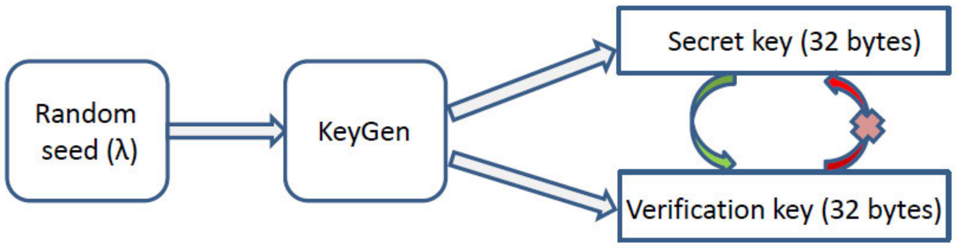


Figure 3.2: Key generation
[Alg22b]

2. $Eval(sk,m)$ - Takes message m alongwith secret key sk as input and generates a pseudorandom output o along with a proof ω.
3. $Verify(pk,m,o,\omega)$ - Takes the verification key pk, message m, output o and proof ω as input and gives the output as either 0 (invalid) and 1 (valid).

The three algorithms mentioned above constituting a VRF strengthen a cryptographic protocol in a blockchain as a result of the following key properties:

- For a given secret key sk and message m, the outputted value o and proof ω are always unique.
- The pseudorandomness of output o is analogous to the proof ω, i.e., it is random to anyone unaware of the corresponding proof ω.

Although traditional digital signature schemes are akin to VRFs, it is distinguishable based on the stronger element of pseudorandomness associated with VRFs. Algorand implements a cryptographic sortition algorithm based on VRFs for choosing the leader and committee.

3.4 Cryptographic Sortition

The method of sortition involves choosing a random set of officials from a larger set of competent and qualified candidates [Sto16]. It has major applications in lottery ticket schemes and electoral voting systems. Since, in Algorand, we require a set of verifiers chosen randomly from a large pool of active users to verify the proposed block, the sortition method is applied with cryptographic functioning. The cryptographic operations performed in the Algorand blockchain remain unpredictable until the last moment and are clear for each user in the network. Such a cryptographic sortition process assures Algorand to be more scalable and secure [GHM+17].

From a high-level point of view, Algorand depicts selecting the official payset PAY^r among all paysets of round r to a chosen set of verifiers who verify the proposed block via BA^*. Algorand also ensures that correct information reaches every selected verifier, and after the validation, the decisions are disclosed to every user in the blockchain network. Verifier selection is a mathematical process where a user implements cryptographic sortition via verifiable random functions. After each round, a new random committee of verifiers is chosen to ensure more resilience towards adversarial attacks. In order to verify a block proposed by a randomly chosen Algorand user (leader), a committee of randomly chosen verifiers is selected for each step in each round such that it satisfies the below properties:

- The collection of payments PAY^r is authentic and verified by the committee members.
- Each payment in PAY^r is always a round-r payment initiated by an honest user.

Cryptographic sortition uses a weight-based computation, i.e., each user u is assigned a weight w_u depending on the money units the user acquires in the blockchain network. W denotes the sum of all weights. For any user, the probability that user u is chosen for a role either as a leader or verifier is directly proportional to the ratio of the individual user weight w_u and W.

3.4.1 Leader Selection Process

As discussed in section 3.1, for a public distributed ledger, block B^r is given as

$$B^r = (r, PAY^r, Q^{r-1}, H(B^{r-1}))$$

Since the payset PAY^r is required to be maximal, it is critical to choose a leader for the block proposal; otherwise, it leads to a non-maximal payset because the individual block of each honest user is unlikely to be very identical. Hence, a leader l^r for round r is chosen to proceed initially [CM19]. Afterwards, the leader proposes and publicizes its block $B^r_{l^r}$. Furthermore, the set of chosen verifiers agrees to a consensus by BA^* on $B^r_{l^r}$.

We denote the fixed-length of collision-resistant function H as $lhash$. When the round r begins, each user has prior knowledge about the current status of the blockchain, i.e.,

$$\overline{B^0}, \cdots, \overline{B^{r-1}}$$

which also provides set of corresponding public keys utilized to each preceding round as

$$pub^1, \cdots, pub^{r-1}$$

The set of potential leaders L comprising of all users u represented as

$$L = \{u \mid \frac{H(Sig_u(r, 1, Q^{r-1}))}{2^{lhash}} \leq p\}$$

where $Sig_u(r, 1, Q^{r-1})$ depicts the signature generated by user, where the elements to be signed are $r, 1, Q^{r-1}$. The signature scheme utilized in Algorand is unique with respect to u and r and since H is a collision resistant hash function it thereby gives a random $lhash$ length output uniquely connected to u and r. For each user in the network it is checked whether the ratio of the hashed value of signature over $|range(H)|$ is lying in the range $[0, p]$ related to the money units user u holds in the network. Moreover, this process boosts the security since only u is capable of computing its signatures and hence is able to check for being a potential leader. Furthermore, on disclosure of its credential $\sigma_u^r \triangleq Sig_u(r, 1, Q^{r-1})$, u shows its credibility of a potential leader for round r.

In L, the set representing potential leaders, l^r is the leader satisfying

$$l^r = arg \min_{i \in L} H(\sigma_i^r)$$

,i.e., the leader's hash value $H(\sigma_{l^r}^r)$ is the least of all L's potential leaders' hash value. In addition to the outcome of cryptographic sortition, for a user u to be considered a leader or verifier, it is required that u is a part of the network for at least t rounds assuring Q^r and succeeding rounds Q values being tamperproof. This proposed block will now be authenticated by a significantly small fraction of total users via multiple steps, as explained in Section 3.4.2.

3.4.2 Committee Selection Process

In the first step of round r, a leader l^r is selected based on cryptographic sortition of hashed credentials. For each consequent step s of round r, the block proposed by l^r is undertaken by $V^{r,s}$ where $V^{r,s}$ is a set of verifiers at step s of round r [CM19]. User u is qualified as a verifier if its corresponding public key $pk_u \in pub^{r-t}$ and

$$H(Sig_u(r, s, Q^{r-1})) \leq p' \cdot 2^{lhash}$$

where $Sig_u(r,s,Q^{r-1})$ depicts the signature generated by user u with the elements to be signed are r,s,Q^{r-1}.

$$V^{r,s} = \{u \mid \frac{H(Sig_u(r,s,Q^{r-1}))}{2^{lhash}} \leq p'\}$$

$$V^r = \bigcup_{s \geq 1} V^{r,s}$$

Let $HV^{r,s}$, $DV^{r,s}$ represents set of honest committee members and dishonest committee members respectively. Then, it always satisfies the following:

$$HV^{r,s} \cup DV^{r,s} = V^{r,s}$$

$$HV^{r,s} \cap DV^{r,s} = \phi$$

Furthermore, there are two representations of Algorand denoted as $Algorand_1$ and $Algorand_2$ where $HV^{r,s}$ and $DV^{r,s}$ satisfy the below properties:

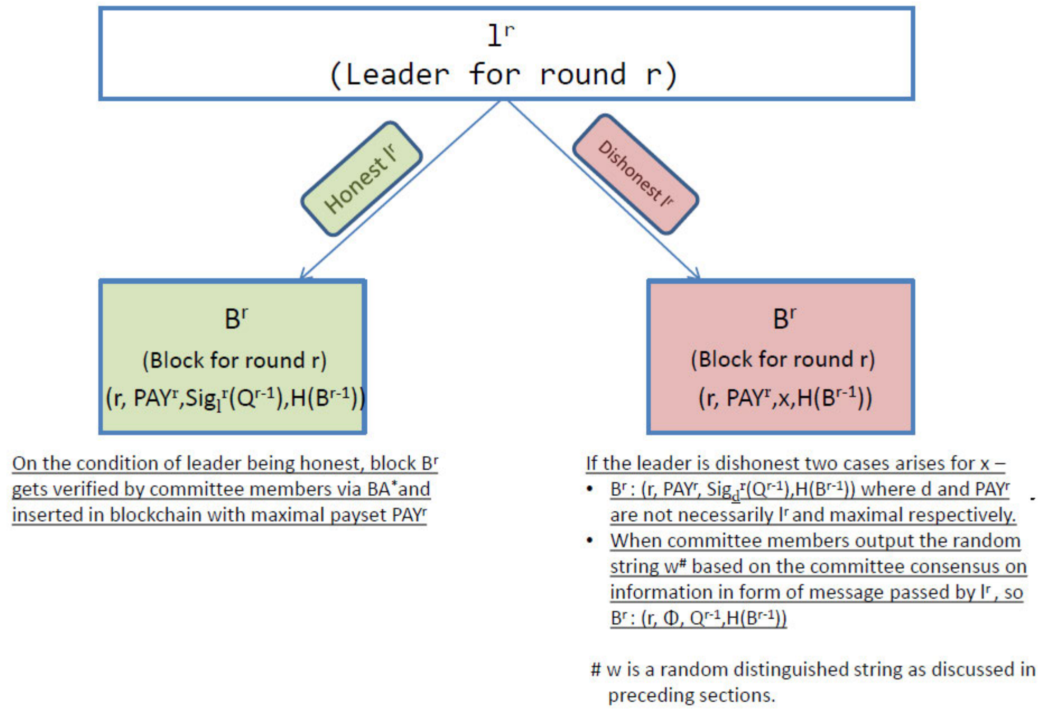
- $Algorand_1$
  1. Ratio of $|HV^{r,s}|$ with $|DV^{r,s}|$ is greater than 2.
  2. $|HV^{r,s}|$ + 4·$|DV^{r,s}|$ $<$ 2·$|V^{r,s}|$
- $Algorand_2$
  1. $|HV^{r,s}|$ is greater than e.
  2. $|HV^{r,s}|$ + 2·$|DV^{r,s}|$ $<$ 2·e
     where e is a particularized threshold value.

Solving the two inequalities for each of the versions, we concluded that the $2/3$ honest majority ratio necessary for BA^* protocol is satisfied.

Analogous to the leader selection process, only u is capable of computing its signatures and hence checks for being a member of the verifiers' committee. For every user in the network, it is inspected whether the hashed value of the signature is less or equal to the product of $|range(H)|$ and p', where p' is associated with the money units user u hold in the network. Likewise on publicizing its credentials $\sigma_u^{r,s} \triangleq Sig_u(r,s,Q^{r-1})$, u indicates its validation of being a potential verifier in step s for round r.

After the verification steps in round r have been conducted, depending on the block leader l^r being honest or malicious, the verified block falls under two categories as explained below in Figure 3.3.

The committee members communicate through a message-sharing protocol where messages are of the form $m_c^{r,s}$ as discussed in Section 3.6.2. c denotes a committee mem-



On the condition of leader being honest, block B^r gets verified by committee members via BA*and inserted in blockchain with maximal payset PAY^r

If the leader is dishonest two cases arises for x –

- $B^r : (r, PAY^r, Sig_d^r(Q^{r-1}), H(B^{r-1}))$ where d and PAY^r are not necessarily l^r and maximal respectively.
- When committee members output the random string w# based on the committee consensus on information in form of message passed by l^r, so $B^r : (r, \Phi, Q^{r-1}, H(B^{r-1}))$

w is a random distinguished string as discussed in preceding sections.

Figure 3.3: Leader cases

ber, and r, s represents the ongoing round and current executing step of round r respectively. $m_c^{r,s}$ carries information shared by c along with its credential $\sigma_c^{r,s}$. A mathematical explanation based on Bernoulli trials and binomial distribution for computation of probability p is attached in Appendix B.1.

3.5 Q^r-Value Computation Process

After verification by the committee members the new block that is appended to the blockchain is given by either of the two forms as below:

$$B^r : (r, PAY^r, Sig_i(Q^{r-1}), H(B^{r-1}))$$

or

$$B^r : (r, PAY^r, Q^{r-1}, H(B^{r-1}))$$

Both representations have a direct dependency on the value of Q^{r-1}. Cryptographic leader or committee selection relies on the Q^{r-1} quantity. It implies that the random value Q^r requires to be of common knowledge before round r is initiated. Therefore, it is critical to compute Q^r securely in order to maintain resilience against adversarial attacks. The process for computing non-forgeable Q^r-value is given below [CM16]:

$$Q^r \triangleq \begin{cases} H(Q^{r-1} || \, r), & \text{if } B^r = B^r_\varepsilon \\ H(Sig_{l^r}(Q^{r-1} || \, r)), & \text{otherwise.} \end{cases}$$

Suppose the chosen leader l^r is reliable, and Q^{r-1} is a valid independent selected value. Then by the definition of hash functions, the Q^r value is computed in an authentic way of fixed-size output length. Otherwise, after the termination of the BA^* protocol, if the agreed-upon value is w, then Q^{r-1} is directly considered for hashing. Furthermore, since the potential leaders and committee members are considered from round $r-t$, it is less probable for a malicious entity to manipulate the leader of round r. Hence, the look-back parameter t is termed as a security parameter [CM16]. Briefly, in Algorand with the below approach, cryptographic sortition using VRFs involving Q^r value is operated [Gor18]:

- Every user u present in the blockchain network utilizes its secret key sk_u and computes $Eval(sk_u, Q^{r-1})$ which gives o_u with its corresponding proof ω_u.
- It is checked whether $0 \leq o_u \leq p$, where p relies on the individual money units held by user u.
- While the output o_u lies in the range mentioned above, the user u confirms its credibility as a committee member by employing the proof ω_u.

3.6 Algorand Synopsis by Message Sharing Protocol

Since Algorand works in both synchronous and asynchronous settings, it is vital that the clock functions at the same speed for every user, whether participating or non-participating. For example, two users, u_1 and u_2 belong to different geographical locations and have different timings. Assuming that time at u_1 and u_2 location is 13:00 and 19:00, respectively. Then, without loss of generality after five minutes, the corresponding time at both locations should be 13:05 and 19:05, respectively. Below are some time-related additional notations for further analysis [CM16]:

- γ: Denotes an upper bound for time required to operate step 1, i.e., step for propagating a block for any round r corresponding to B^r in the network.
- Δ: Denotes an upper bound for time required to operate any further steps, i.e., verification steps via BA^* protocol, for any round r corresponding to B^r in the Algorand blockchain network.

3.6.1 Ephemeral Keys

Prior to discussing the cyclic process involved in an end-to-end block generation, it is required to discuss a critical factor termed ephemeral keys utilized for signing the messages. By a cyclic process, we refer to the exact process with the followed steps that stays common for each round r and its corresponding block B^r. To ensure more security and robustness, a user u employs the public/secret key pair to render its credentials, i.e., $\sigma_u^{r,s}$. On the contrary, u uses ephemeral key pair for rendering and processing the messages $m_u^{r,s}$ for transferring information regarding the BA^* protocol. For any user u, the total number of ephemeral key pairs generated for information exchange is the product of the number of rounds r involved with the corresponding round's steps s performed [CM19]. $esk_u^{r,s}$ and $epk_u^{r,s}$ denote ephemeral key pairs used for signing and verifying message $m_u^{r,s}$ respectively. After performing cryptographic sortition, u identifies whether it belongs to $V^{r,s}$ or not, and on the condition that $u \in V^{r,s}$, u signs $m_u^{r,s}$ with $esk_u^{r,s}$ and destroys the ephemeral key $esk_u^{r,s}$.

Ephemeral keys are processed by various mathematical procedures, with one of them being a recursive approach of Merkle trees as explained in figure 3.4.

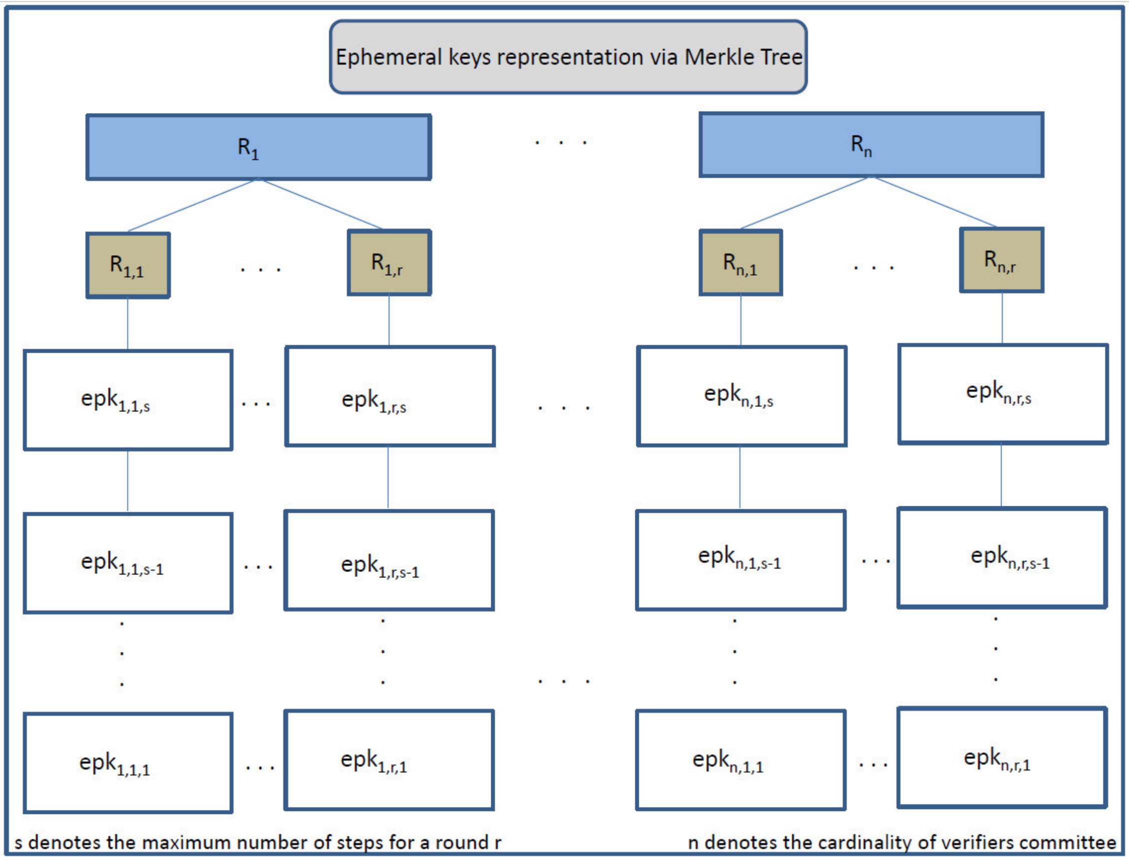


Figure 3.4: Ephemeral keys using Merkle Tree approach

The above diagram implies that for each user, $epk_u^{r,s} \triangleq (i, r, s)$. $R_{u,j}$ denotes the root value for a user u, round j at each step involved on execution of round r steps. While signing the message in message sharing protocol by $epk_u^{r,s}$, user u circulates the rendered signature with the required path. By path we refer to the route in the Merkle Tree traversing to $epk_u^{r,s}$ by utilizing root value R_u. The signatures generated for some value v by a user u employing ephemeral key $epk_u^{r,s}$ is denoted as $esig_u v$. The term $ESig_u v$ illustrates $(u, v, esig_u v)$. The ephemeral key $esk_u^{r,s}$ used during a step s of a round r by user u is promptly destroyed after signing a message of form

$$m_u^{r,s} = (u, \mathcal{V}, esig_u(H(\mathcal{V})), \sigma_u^{r,s})$$

for some value $\mathcal{V}$.

3.6.2 Information Exchange through Message Sharing Protocol

Below is the cyclic process for an end-to-end block B^r generation consisting of authentic payments through information exchange via messages:

- Every existing user u who participated in the Algorand blockchain network for atleast t rounds calculates previous round's Q^{r-1} value by fetching third element of the universally known proven previous block $\overline{B^{r-1}}$. After functioning cryptographic sortition, u confirms whether it is part of committee for step 1 of round r or not. As a participating member of verifiers committee $V^{r,1}$ u computes and proposes its own individual block

  $$B_u^r = (r, PAY_u^r, Sig_u(Q^{r-1}), H(B^{r-1}))$$

  Then, this information is signed with ephemeral key and shared via the message $m_u^{r,1}$ as:

  $$m_u^{r,1} = (B_u^r, esig_u(H(B_u^r)), \sigma_u^{r,1})$$

- Likewise, in step 1 of round r, user u, who is a part of the Algorand blockchain network for at least t rounds, calculates the previous round's Q^{r-1} value by fetching the third element of the universally known proven previous block $\overline{B^{r-1}}$. After functioning cryptographic sortition, u confirms whether it is a part of the committee for step 2 of round r or not. As a participating member of the verifiers' committee $V^{r,2}$ in an asynchronous environment, u waits for $\gamma + \Delta$ duration, i.e., time exhausted during block proposal and circulation of the message. At the end of the aforementioned time, u follows the first step of graded consensus (GC) protocol and determines l^r by the least credential value through valid round r step 1 messages. With l^r message given as

  $$m_{l^r}^{r,1} = (B_{l^r}^r, esig_{l^r}(H(B_{l^r}^r)), \sigma_{l^r}^{r,1})$$

is validated by u. Then, u fixes value $B^r_{l^r}$ and shares $m^{r,2}_u$

$$m^{r,2}_u = (u, B^r_{l^r}, esig_u(H(B^r_{l^r})), \sigma^{r,2}_u)$$

otherwise w a psuedorandom string is fixed as value and shared message is of the form

$$m^{r,2}_u = (u, w, esig_u(H(w)), \sigma^{r,2}_u)$$

- Till a user u confirms that $u \in V^{r,3}$ same process as mentioned in previous steps is followed. On the cryptographic sortition confirmation, and the passage of latent time duration $3\gamma + \Delta$, u for the second step of GC protocol analyses the step two messages received from $V^{r,2}$. We denote each user involved in $V^{r,s}$ $(s \geq 2)$ by k. If a message of the form

  $$(k, B^r_{l^r}, esig_k(H(B^r_{l^r})), \sigma^{r,2}_k)$$

  appears for more than 2/3 of the total verifiers $V^{r,2}$ then $m^{r,3}_u$ is circulated as

  $$m^{r,3}_u = (u, B^r_{l^r}, esig_u(H(B^r_{l^r})), \sigma^{r,3}_u)$$

  and if the mutual consensus value was set to w by $V^{r,2}$ verifiers, then $m^{r,3}_u$ is shared and given by

  $$m^{r,3}_u = (u, w, esig_u(H(w)), \sigma^{r,3}_u)$$

- Analogous to the previous steps $V^{r,4}$ is formed by applying cryptographic sortition and further employed to determine the output of GC protocol. A user $u \in V^{r,4}$ succeeding the time lapse of $5\gamma + \Delta$ sets the value v_u and grade g_u as $H(B^r_{l^r})$ and *2* respectively, if over 2/3 of the messages received from $V^{r,3}$ is identical to

  $$m^{r,3}_k = (k, B^r_{l^r}, esig_k(H(B^r_{l^r})), \sigma^{r,3}_k)$$

  Furthermore, if only 1/3 of such messages are received value remains the same but the corresponding grade alters to 1. For any other scenarios value and grade are fixed as $H(B^r_\varepsilon)$ and 0 respectively. B^r_ε represents an empty block. The assigned grade determines the input for BBA*. For g_u = 2, b_u is fixed to 0 else, b_u is fixed to 1. Finally the message $m^{r,4}_u$ is shared as

  $$m^{r,4}_u = (ESig_u(b_u), ESig_u(v_u), \sigma^{r,4}_u)$$

- As explained in each previous step, a user u who has been part of Algorand blockchain network for at least t rounds, initially verifies whether he is a participant of verifiers committee for conducting the further steps by the means of cryptographic sortition. The number of steps in BBA* relies on the cardinality of verifier set along with required sufficient $(> 2 \cdot |V^{r,s}|/3)$ signatures. It is assumed that the number of steps is upper-bounded by S in order to approve the leader's proposed block. Moreover, since four steps have already been discussed, so after waiting up to the time-limit of all consequent steps in BBA* summing as $(2s-3)\gamma+\Delta$, u performs as follows. If u receives $2 \cdot |V^{r,s}|/3+1$ authentic messages as

$$m_k^{r,s'-1} = (ESig_k(0), ESig_k(H(B_{l^r}^r)), \sigma_k^{r,s'-1})$$

s' denoting previous step, then it terminates the current step and fixes $B^r = B_k^r$ allocates to $CERT^r$ all such authentic messages $m_k^{r,s'-1}$. On the contrary, If u receives $2 \cdot |V^{r,s}|/3+1$ authentic messages as

$$m_k^{r,s'-1} = (ESig_k(1), ESig_k(H(w)), \sigma_k^{r,s'-1})$$

then it terminates the current step and fixes $B^r = B_\varepsilon^r$ allocates to $CERT^r$ all such authentic messages $m_k^{r,s'-1}$.

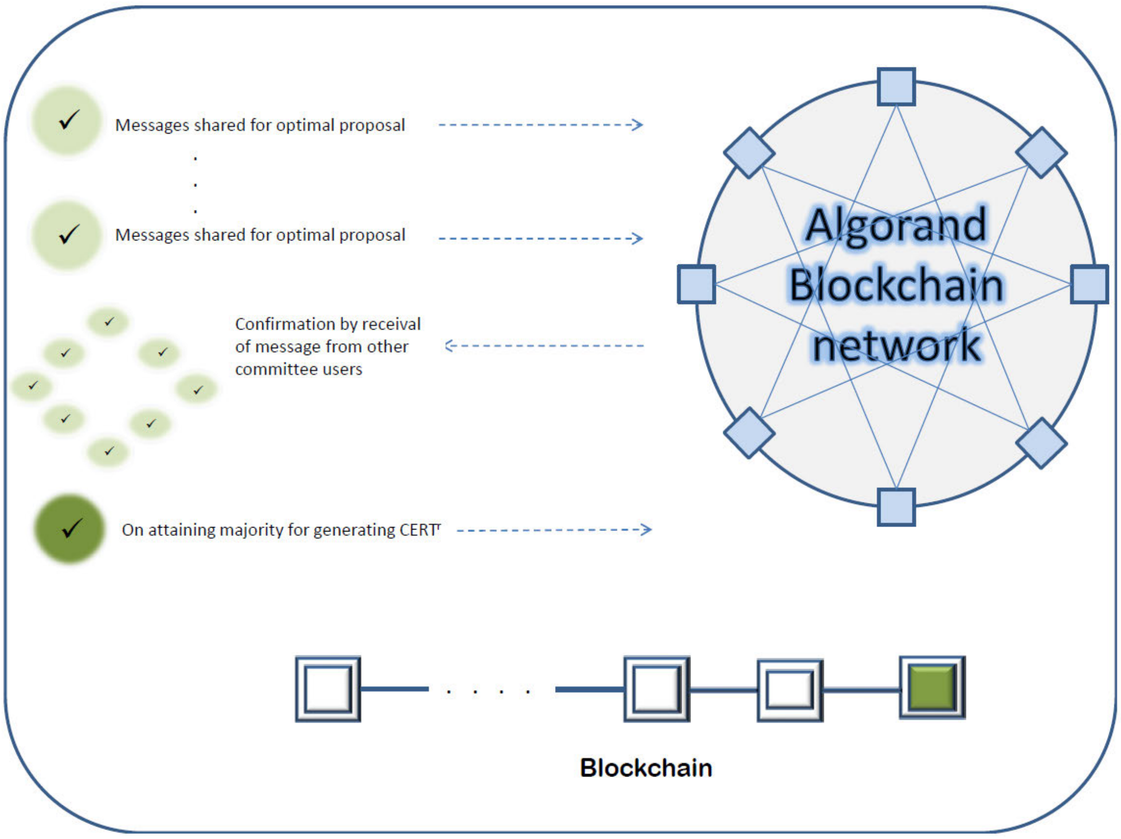

Messages shared for optimal proposal
Messages shared for optimal proposal
Confirmation by receival of message from other committee users
On attaining majority for generating CERT^r
Algorand Blockchain network
Blockchain

Figure 3.5: Message sharing protocol in Algorand blockchain [Alg22c]

Analogously, $Algorand_2$ employs ephemeral keys for signing messages during message sharing protocol. While in $Algorand_1$ it is required at each step at least 2/3 verifier's messages for undergoing the next stage, in $Algorand_2$ the considered cutoff value relies on e, a specific threshold value as discussed in Section 3.4.2. On confirmation of $CERT^r$ comprising of adequate quantity of digital signatures, block B^r transforms to proven block $\overline{B^r}$ [CM16].

3.7 Utility of Look-back Parameter t

As discussed in Section 3.4, there is a dependency on the Q-value in the process of generating the next successive block. Furthermore, to ensure that the current and all successive Q-values remain unaltered, we consider the look-back parameter t. This look-back parameter becomes more critical in the leader and committee selection process. A participating user requires to be associated with the Algorand blockchain network for at least t preceding rounds.

To proceed further requires undergoing the notion of a lazy-but-honest user. A user in the Algorand blockchain network falls under the category of lazy-but-honest if it follows the two properties:

- It conforms with every task allocated while undergoing an assigned role.
- It is needed for that user to join infrequently on being appropriately informed on a prior basis.

A block generation in Algorand blockchain network takes approximately 4.5 minutes [Alg22a]. On continuation with this block generation speed each day generally 320 blocks are appended to the blockchain network. If a user u is concerned with knowing in advance whether it is chosen as a verifier for the next x days, it is required that for a round r, verifiers are selected from round $r-t-320x$ users through cryptographic sortition on the basis of $Q^{r-320x-1}$ [CM19]. On round r initiation, user u is already aware about $320x$ round Q-values retrospectively. Therefore, u is eligible member in step s of j^{th} round verifiers committee if and only if it belongs to the set

$$V^{r+j,s} = \{u \mid \frac{H(Sig_u(r+j,s,Q^{r+j-320x-1}))}{2^{lhash}} \leq p'\}$$

where $j \in \{1, \cdots, 320x\}$.

Therefore, it is required for user u to generate its credential

$$\sigma_u^{j,s} = H(Sig_u(r+j,s,Q^{r+j-320x-1}))$$

and confirms that for which j's the ratio of this hash credential over 2^{lhash} is less than or equal to p. Since it is a probabilistic scenario, on applying cryptographic sortition, there is a possibility that u identifies the below cases:

- For the next consecutive j rounds, it is not a part of the verifiers' committee. Then u is allowed to be offline or online depending on u's choice.
- Otherwise u is selected for z rounds, where $1 \leq z \leq 320x$. Then u is prepared to be a verifier at the correct round.

Hence, such an approach tackles the honest users in an offline setting. This methodology also assures complete security in the blockchain without the regular participation of benign individuals.

3.8 Honest Majority of Money Approach

Until now, our study for both the models of Algorand, i.e., $Algorand_1$ and $Algorand_2$, have been confined to the assumption of the majority of benign users at every stage. This approach further extends to the Honest Majority of Money (HMM) methodology assuring a more robust and unbiased procedure. The majority of benign users indicate that the participants involved in each stage of the block generation process in the Algorand blockchain network follow a strict majority of honest participants. On the contrary, in the HMM approach, the money units belonging to the honest participants cover the majority of the money at stake.

Based on the assumptions of regular participation of users and their associated money units at the beginning of the round r, HMM is applied as follows [CM16]:

- Method is approached by restricting the public keys possessing money units upper bounded by P, where P is a pre-defined value for held money units.
- The ratio of P with overall money units involved in the system is significantly small. It assures that for a user u, pub_u is not utilized more than once in a round's execution for at least t rounds with higher probability.

The process for leader and verifiers committee selection for a user u involves cryptographic sortition but with dependency on P and respective amount amt_u^r at stake on round r. Mathematically it is applied as follows:

$$L = \{u \mid \frac{H(Sig_u(r,1,Q^{r-1}))}{2^{lhash}} \leq \frac{p \cdot amt_u^r}{P}\}$$

where L is the set demonstrating potential leader and from L leader l^r satisfies

$$l^r = arg\min_{i \in L} H(Sig_i(r,1,Q^{r-1}))$$

Similarly each user u involved in verifiers committee satisfies

$$\frac{H(Sig_u(r,s,Q^{r-1}))}{2^{lhash}} \leq \frac{p' \cdot amt_u^r}{P}$$

where $Sig_u(r,s,Q^{r-1})$ depicts the signature generated by user u with the elements to be signed are r, s, Q^{r-1}.

$$V^{r,s} = \{u \mid \frac{H(Sig_u(r,s,Q^{r-1}))}{2^{lhash}} \leq \frac{p' \cdot amt_u^r}{P}\}$$

The process for exchanging information through message sharing protocol is identical except with the above mathematical modification involving HMM assumption.

4 Security and Attack Analysis

4.1 Possible Attacks in Blockchain

4.1.1 51% Attack

In a 51% attack, a group of miners owns more than 50% of the network's computing power so that attackers are able to halt the confirmation process for queued transactions. It leads to counterfeiting of transaction details, making it prone to double-spending [SMG19].

Blockchain cryptocurrency protocols rely on a shared public distributed ledger. It comprises transaction records accessible to all existing users of the network, assuring that no money units are spent twice. Every block in the blockchain network contains information specific to transactions loaded at a certain timeframe. Block generation in a blockchain network is processed after the passage of a certain latent time specific to the blockchain where the process is performed. Therefore, modification after a block is confirmed upon agreement is inconsistent. However, during this latent time passage, if a certain malicious entity handles the majority of computing power, it leads to the manipulation of transaction records in the ledger. Thus, with the major computing power, blocks of the original blockchain are replaced with the malicious entity's generated blocks.

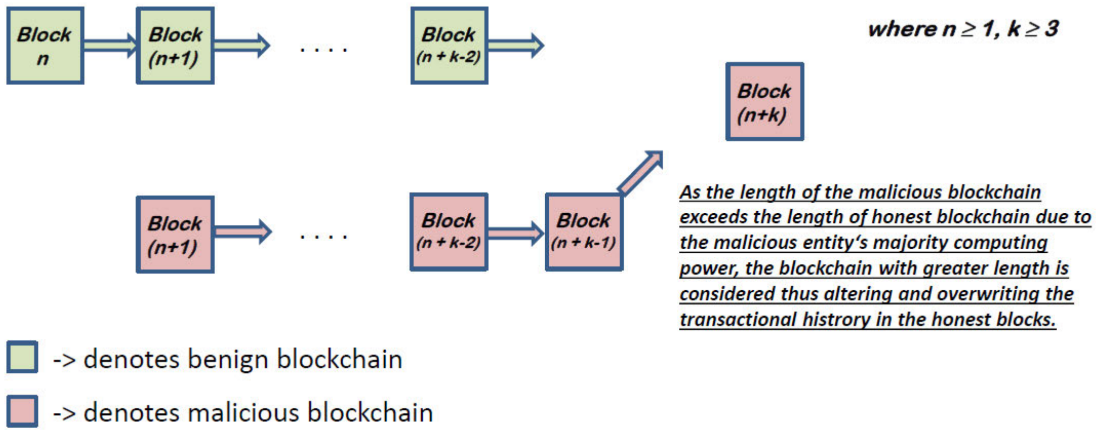

Block n
Block (n+1)
....
Block (n+k-2)
where $n \geq 1$, $k \geq 3$
Block (n+k)
Block (n+1)
....
Block (n+k-2)
Block (n+k-1)
As the length of the malicious blockchain exceeds the length of honest blockchain due to the malicious entity's majority computing power, the blockchain with greater length is considered thus altering and overwriting the transactional histrory in the honest blocks.
🟩 -> denotes benign blockchain
🟥 -> denotes malicious blockchain

Figure 4.1: 51% Attack
[But19]

Since Algorand utilizes the pure PoS consensus mechanism, a 51% attack is analogous to holding more than 50% of the stake. Generally, for any proof of stake reliant mechanism, the sum of all transaction fees rewarded to a validator does not exceed the individual stake. Hence, in case of any attempt towards counterfeiting, the majority held stake is lost.

4.1.2 Sybil Attack

A Sybil attack refers to an attack over a blockchain network where an individual attempts to control the network by executing operations through various fake nodes or public keys [Dou02]. The term "Sybil" dates back to a study and analysis around a woman named Shirley Mason diagnosed with Dissociative Identity Disorder, also referred to as Multiple Personality Disorder [MD13].

To conduct a Sybil attack in a blockchain network, an attacker or a group of attackers influences and overpowers the honest members of the network by infusing sufficient fake accounts (or Sybil accounts). Consequently, the series of actions involve the refusal to receive or transmit block requests, thus restraining other honest users in the network. A Sybil attack with a larger impact relates to a 51% attack as with a huge number of fake identities, a major fraction of computing power is obtained.

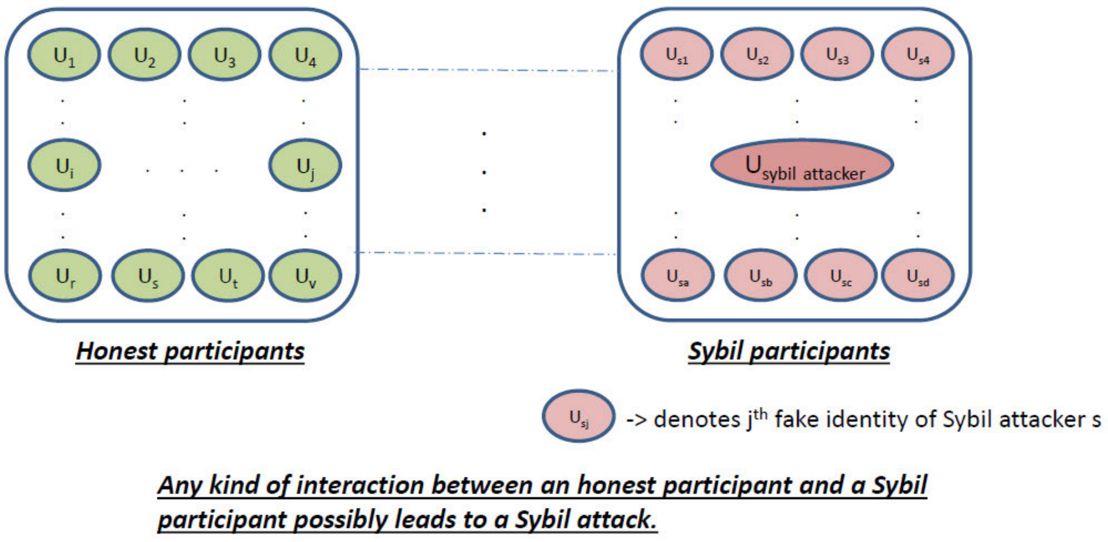


Figure 4.2: Sybil Attack
[JR22]

For the Algorand blockchain network, with HMM approach, Sybil attacks are restricted since, for a user u, pub_u is chosen as a leader or committee member for verification on applying successful self cryptographic sortition, that depends on amt_u^r held by pub_u for round r.

4.1.3 Partitioning Attack

A partitioning attack occurs in a blockchain network either by a natural calamity or a malicious entity. In a partitioning attack, the network is segmented into multiple networks, and intra-communication between participants belonging to one part of the network exists. However, inter-communication between participants belonging to distinct network parts is disabled. Under this attack, the malicious entity controls the exchange of information among the participants. Subject to the duration of such an attack, the malevolent entity assures that the users of distinguished parts agree to accept different blocks at the same length of the blockchain.

In the Algorand blockchain network, the probability of a fork occurrence is significantly negligible [CM16]. Therefore, for a malicious entity to make an agreement among users in various parts under a partitioning attack is computationally infeasible. Furthermore, in Algorand, while message sharing protocol is in progress, if there are no further advancements on the passage of latent time, then the network nodes progress to partition recovery mode. With this additional feature in the Algorand blockchain network, it recovers rapidly from the existing partitions [Alg22d].

4.2 Algorand's Compactness in Solving the Blockchain Trilemma

With the elevation in the use cases of blockchain in the domain of cryptocurrencies, it is vital that the performance parameters of blockchain perform efficiently and optimally. Therefore, the performance parameters resemble the three elements of the blockchain trilemma, i.e., decentralization, scalability, and security. While every blockchain network

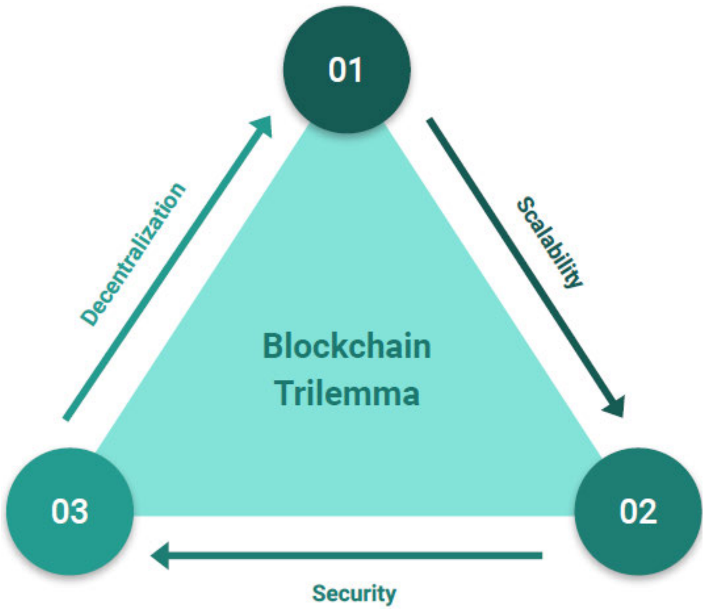


Figure 4.3: Blockchain Trilemma
[Nem22]

aims to solve the blockchain trilemma, benefits and repercussions are associated with maintaining each of the three performance parameters.
In the absence of a centralized authority, the consensus is reached, thus maintaining

trust. Furthermore, since the dependency on a single entity is circumvented, it reduces system failures. The scalability parameter is a measure demonstrating how many transactions a blockchain network holds feasibly. With large-scale users involved in a cryptocurrency protocol, satisfying the parameters of the blockchain trilemma is an arduous task, thus making way for trade-offs. Such trade-offs refer to a compromise with one of the blockchain trilemma parameters. In other words, optimizing any two parameters reduces the efficiency with respect to the third parameter.

Algorand tends to solve the blockchain trilemma and satisfies the three performance parameters simultaneously, as given below [Alg21]:

- Decentralization: By the self cryptographic sortition algorithm, verifiers are chosen randomly, removing the element of biasedness or dependency over a specific set of users in the network. In the Algorand blockchain, each user is selected for either proposing or verifying a block at a certain point in time, implying there is no reliance on a particular group of users.
- Scalability: On performing a fast self cryptographic sortition algorithm, an individual becomes aware of its participation in validating a block. Since the cardinality of the set of verifiers is a significantly smaller fraction of the total active users, transaction verification and block validation are rapidly performed, thus maintaining scalability.
- Security: Since the selection of candidates for verifying the transactions and validating the blocks is obtained through self cryptographic sortition algorithm, no one in the system is aware of the next set of validators. Hence, this property guarantees more robustness and assures that the performance parameter security is resolved effectively.

5 Conclusion

This thesis research provides an overview of mathematics and cryptography behind the Algorand blockchain network. It is observed that during each phase of operation in Algorand, there is a high degree of randomness linked to each activity, making it distinguishable from how other blockchains operate. Furthermore, the pure PoS consensus mechanism ensures that while mutually undergoing validation and verification, the occurrence of counterfeit or alteration in the information transferred through the communication is minimal. The pure PoS consensus mechanism involves computations through the GC and the BBA* protocols. Since considering all active users for the protocol is highly complex from a computational viewpoint, a significantly low proportion of randomly selected active users is considered for the protocol implementation. Rather than depending on a central authority for assigning any role or task, an individual runs cryptographic sortition based on VRFs on its own to be sure of whether it has a role to play in the whole cycle of block generation.

The key pair employed for signature purposes are sectioned into two categories ensuring more reliability and security. The first category of public-private keys is used perpetually to generate participating users' credentials at each step of every round. On the contrary, the other category, ephemeral key pair, is used to sign a message in the message sharing protocol.

A mathematical association between the honest majority of users and the Honest Majority of Money (HMM) approach has also been discussed. HMM is synonymous with the discrete participation of honest individuals without impacting the robustness of the blockchain network. Some common attacks relative to blockchain and how Algorand copes with it have also been studied. Figure 5.1 below summarizes various phases of the Algorand blockchain for an end-to-end secure block generation process.

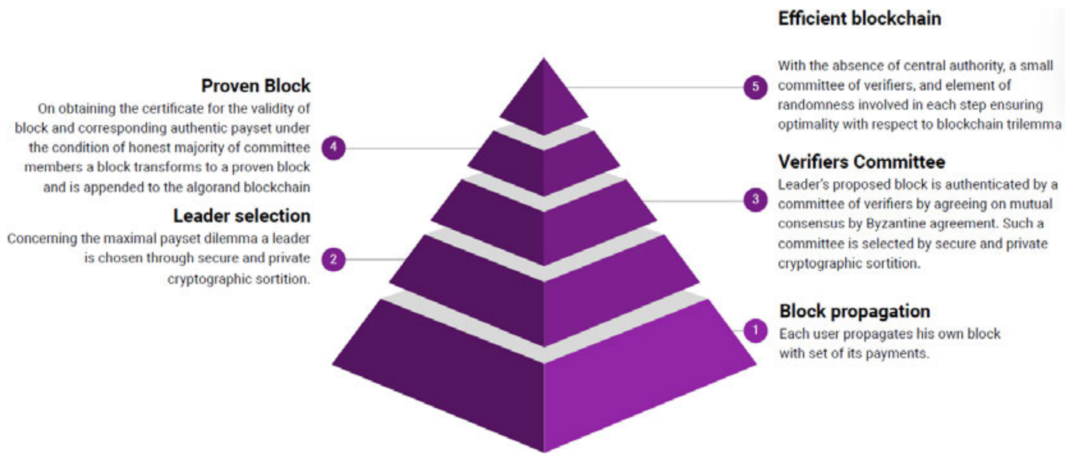

Efficient blockchain
With the absence of central authority, a small committee of verifiers, and element of randomness involved in each step ensuring optimality with respect to blockchain trilemma
5
Proven Block
On obtaining the certificate for the validity of block and corresponding authentic payset under the condition of honest majority of committee members a block transforms to a proven block and is appended to the algorand blockchain
4
Verifiers Committee
Leader's proposed block is authenticated by a committee of verifiers by agreeing on mutual consensus by Byzantine agreement. Such a committee is selected by secure and private cryptographic sortition.
3
Leader selection
Concerning the maximal payset dilemma a leader is chosen through secure and private cryptographic sortition.
2
Block propagation
Each user propagates his own block with set of its payments.
1

Figure 5.1: Algorand summary

Comparison with Bitcoin and Ethereum Blockchains

Below is a comparison of the Algorand blockchain with Bitcoin and Ethereum cryptocurrency blockchain protocol across various mathematical and computational parameters.

Aspect	**Bitcoin**	**Ethereum**	**Algorand**
Hash function	SHA-256	Keccak-256	SHA-256
Consensus mechanism	PoW	PoS	Pure PoS
Signature scheme	Schnorr	ECDSA	EdDSA
Native cryptocurrency	BTC	ETH	ALGO
Smart contracts	$\times$	✓	✓
Transaction/sec	3	13	1000
Transaction cost	1.1$	1.7$	0.0004$
Market Capitalization	370B $	130B $	2.2B $

Table 5.1: Blockchain comparison
[Alg22a, Sta22, Eth22, Ych22, Ran22]

Scope for Future Work

- Comparison of currently used EdDSA signature scheme with other applicable optimal schemes
- Robustness and security of the blockchain network can be escalated by incorporating Zero-Knowledge Proofs [GMW86]. A basic overview of Zero-Knowledge Proofs is provided in Appendix B.2.
- Machine learning-based time-series analysis and forecasting algorithms involving deep learning frameworks in monitoring critical blockchain aspects such as transaction per second (TPS), block generation time, and financial analysis

Bibliography

[Aca19] Ledger Academy. What is Proof-of-Work. https://www.ledger.com/academy/blockchain/what-is-proof-of-work, 2019. Accessed: 23/03/2022.

[Alg21] Algorand. How Does Algorand Solve the Blockchain Trilemma? - An Explanation by Silvio Micali on the Lex Fridman Podcast. https://www.algorand.com/resources/blog/silvio-micali-lex-fridman-algorand-and-the-blockchain-trilemma, 2021. Accessed: 26-04-2022.

[Alg22a] Algorand. Algorand Blockchain. https://www.algorand.com/, 2022. Accessed: 03-07-2022.

[Alg22b] Algorand. Algorand Developer Docs: Account Details. https://developer.algorand.org/docs/get-details/accounts/, 2022. Accessed: 14-04-2022.

[Alg22c] Algorand. Algorand Developer Docs: Algorand Consensus. https://developer.algorand.org/docs/get-details/algorand_consensus/, 2022. Accessed: 16-05-2022.

[Alg22d] Algorand. Algorand Security. https://www.algorand.com/technology/security, 2022. Accessed: 23-05-2022.

[Alg22e] Algorand. Algorand's Immediate Transaction Finality. https://www.algorand.com/technology/immediate-transaction-finality, 2022. Accessed: 10-01-2022.

[Alg22f] Algorand. Frequently Asked Questions. https://www.algorand.com/technology/faq, 2022. Accessed: 05-04-2022.

[BBJ+08] D.J. Bernstein, P. Birkner, M. Joye, T. Lange, and C. Peters. Twisted Edwards Curves. In *International Conference on Cryptology in Africa*, pages 389–405. Springer, 2008.

[BDL+12] D.J. Bernstein, N. Duif, T. Lange, P. Schwabe, and B.Y. Yang. High-Speed High-Security Signatures. *Journal of cryptographic engineering*, 2(2):77–89, 2012.

[Ber06] D.J. Bernstein. Curve25519: New Diffie-Hellman Speed Records. In *Inter-*

national Workshop on Public Key Cryptography, pages 207–228. Springer, 2006.

[BL07] D.J. Bernstein and T. Lange. Faster Addition and Doubling on Elliptic Curves. In *International conference on the theory and application of cryptology and information security*, pages 29–50. Springer, 2007.

[BL17] G. Birkhoff and S.M. Lane. *A Survey of Modern Algebra*. AK Peters/CRC Press, 2017.

[BN19] M. Boudabra and A. Nitaj. A new public key cryptosystem based on Edwards curves. *Journal of Applied Mathematics and Computing*, 61(1):431–450, 2019.

[Bon05] D. Boneh. *Elgamal Digital Signature Scheme*, pages 182–183. Springer US, 2005.

[But14] V. Buterin. Ethereum : A Next Generation Smart Contract and Decentralized Application Platform. 2014.

[But19] A. Butler. Ethereum classic attacked! how does the 51% attack occur? `https://medium.com/hackernoon/ethereum-classic-attacked-how-does-the-51-attack-occur-a5f3fa5d852e`, 2019. Accessed: 23-05-2022.

[CM16] J. Chen and S. Micali. Algorand, 2016.

[CM19] J. Chen and S. Micali. Algorand: A secure and efficient distributed ledger. *Theoretical Computer Science*, 777:155–183, 2019.

[Dal21] S. Daley. 34 Blockchain Applications and Real-World Use Cases Disrupting the Status Quo. `https://builtin.com/blockchain/blockchain-applications`, 2021. Accessed: 01-04-2022.

[DN93] C. Dwork and M. Naor. Pricing via Processing or Combatting Junk Mail. In *Advances in Cryptology — CRYPTO' 92*, pages 139–147. Springer Berlin Heidelberg, 1993.

[Dou02] J.R. Douceur. The Sybil Attack. In *Peer-to-Peer Systems*, volume 2429, pages 251–260. Springer Berlin Heidelberg, 2002.

[DSJ$^+$20] The Sage Developers, W. Stein, D. Joyner, D. Kohel, J. Cremona, and E. Burçin. Sagemath, version 9.0, 2020.

[Edw07] H. Edwards. A Normal Form for Elliptic Curves. *Bulletin of the American mathematical society*, 44(3):393–422, 2007.

[Eth22] Etherscan. The Ethereum Blockchain Explorer. `https://etherscan.io/`, 2022. Accessed: 03-07-2022.

[FM89] P. Feldman and S. Micali. An Optimal Probabilistic Algorithm For Synchronous Byzantine Agreement. In *International Colloquium on Automata, Languages, and Programming*, pages 341–378. Springer, 1989.

[GHM$^+$17] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich. Algorand: Scaling Byzantine Agreements for Cryptocurrencies. page 51–68. Association for Computing Machinery, 2017.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems (Extended Abstract). In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing, May 6-8, 1985, Providence, Rhode Island, USA*, pages 291–304. ACM, 1985.

[GMW86] O. Goldreich, S. Micali, and A. Wigderson. Proofs that Release Minimum Knowledge. In *Mathematical Foundations of Computer Science 1986*, pages 639–650, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.

[Gor18] S. Gorbunov. Algorand Releases First Open-Source Code: Verifiable Random Function. `https://medium.com/algorand/algorand-releases-first-open-source-code-of-verifiable-random-function-93c2960abd61`, 2018. Accessed: 15-04-2022.

[GS82] G.R. Grimmett and D.R. Stirzaker. Probability and random processes. 1982.

[HS91] S. Haber and W.S. Stornetta. How to Time-Stamp a Digital Document. In *Advances in Cryptology-CRYPTO' 90*, volume 537, pages 437–455. Springer Berlin Heidelberg, 1991.

[JMS01] D. Johnson, A. Menezes, and S.Vanstone. The Elliptic Curve Digital Signature Algorithm (ECDSA). *International journal of information security*, 1(1):36–63, 2001.

[JR22] G. Jethava and U.P. Rao. User behavior-based and graph-based hybrid approach for detection of Sybil Attack in online social networks. *Computers and Electrical Engineering*, 99:107753, 2022.

[KN12] S. King and S. Nadal. Ppcoin: Peer-to-Peer Crypto-Currency with Proof-of-Stake. *self-published paper, August*, 19(1), 2012.

[Kra20] Kraken. What is Algorand? (ALGO) A Beginner's Guide. `https://www.kraken.com/learn/what-is-algorand-algo`, 2020. Accessed: 20-03-2022.

[MD13] P.M. Coons MD. Sybil in Her Own Words: The Untold Story of Shirley Mason, Her Multiple Personalities and Paintings, by P. Suraci. *Journal of Trauma & Dissociation*, 14(3):359–361, 2013.

[Mer82] R.C. Merkle. Method of providing digital signatures, 1982. US Patent 4,309,569.

[Mic17] S. Micali. Fast and furious Byzantine agreement. In *Proceedings of the Innovations in Theoretical Computer Science (ITCS) Conference*, 2017.

[MRV99] S. Micali, M. Rabin, and S. Vadhan. Verifiable Random Functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[Nak08] S. Nakamoto. Bitcoin: A Peer-to-Peer Electronic Cash System. *Decentralized Business Review*, page 21260, 2008.

[Nem22] M. Nemack. Infographic: What is the Blockchain Trilemma? `https://www.coinpro.ch/infografik-was-ist-das-blockchain-trilemma/`, 2022. Accessed: 27-05-2022.

[Pre94] B. Preneel. Cryptographic Hash Functions. *European Transactions on Telecommunications*, 5(4):431–448, 1994.

[PSL80] M.C. Pease, R.E. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *J. ACM*, 27:228–234, 1980.

[Ran22] Randlabs. Why Algorand? `https://www.randlabs.io/algorand`, 2022. Accessed: 03-07-2022.

[Sch89] C.P. Schnorr. Efficient Identification and Signatures for Smart Cards. In *Conference on the Theory and Application of Cryptology*, pages 239–252, 1989.

[SL07a] M. Stamp and R.M. Low. Hash functions. In *Applied Cryptanalysis*, pages 193–264. John Wiley & Sons, Ltd, 2007.

[SL07b] M. Stamp and R.M. Low. *Public Key Attacks*, chapter 7, pages 315–360. John Wiley Sons, Ltd, 2007.

[SMG19] S. Sayeed and H. Marco-Gisbert. Assessing Blockchain Consensus and Security Mechanisms against the 51% Attack. *Applied Sciences*, 9(9), 2019.

[Sta22] Statoshi.info - Realtime Bitcoin Node Stats. `https://statoshi.info/?orgId=1`, 2022. Accessed: 03-07-2022.

[Sto16] P. Stone. Sortition, voting, and democratic equality. *Critical Review of International Social and Political Philosophy*, 19(3):339–356, 2016.

[Ych22] Ycharts. Ethereum Average Transaction Fee. `https://ycharts.com/indicators/ethereum_average_transaction_fee`, 2022. Accessed: 03-07-2022.

Appendix A: Python-Code

A.1 Modern Cryptography

A.1.1 Primitive Roots Computation

```
# Calculation of Primitive roots for a number
import numpy as np
import math

# Greatest common divisor of two numbers

class primitive:

    def __init__(self,N):
        self.N = N

    def gcd(a,b):
        if a == 0 or b==0:
            if a + b == 0:
                print('GCD does not exist')
            else:
                return a + b
        else:
            return primitive.gcd(b%a,a)

    #Euler Totient function
    def euler_phi(n):
        count = 0
        for i in range(1,n+1):
            if primitive.gcd(i,n)==1:
                count = count + 1
        return count

    # Function check for prime numbers
    def prime_check(n):
        if n<=1:
            return False
        for i in range(2,int(math.sqrt(n))+1):
            if primitive.gcd(n,i)==1:
                continue
            else:
                return False
                break
        return True

    # Computation of primitive roots
    def prim(self):
        n = abs(self.N)
        t = np.zeros((n-2,n-2))

        if primitive.prime_check(n)==False:
            # Creates a matrix where each row consists of (i^j)%n.
            for i in range(2,n):
                for j in range(2,n):
                    x = pow(i,j,n)
                    t[i-2][j-2] = x

            b = []
            y = []
            for k in range(0,n-2):
                # Check cardinality of unique elements in a row.
                b = t[k]
                if len(np.unique(b))==primitive.euler_phi(n):
                    if primitive.gcd(k+1,n)!=1:
                        y.append(k+2)
            return y

        else:
            # Creates a matrix where each row consists of (i^j)%n.
            for i in range(2,n):
                for j in range(2,n):
                    x = pow(i,j,n)
                    t[i-2][j-2] = x

            b = []
            y = []
            for k in range(0,n-2):
                # Check whether all elements in a row are unique.
                b = t[k]
                if len(np.unique(b))==n-2:
                    y.append(k+2)
            return y

if __name__=='__main__':
    x = int(input('Enter no : '))
    obj = primitive(x)
    print(obj.prim())
```

Listing A.1: Primitive roots computation

A.1.2 Discrete Logarithm Computation

```
#Computation of discrete logarithm using baby-step-giant-step-
    algorithm

import math
import numpy as np

class discrete_log:

    def __init__(self,p,k,n):
        self.p = p
        self.k = k
        self.n = n

    @staticmethod
    def is_prime(n):
        if n<=1:
            return False
        for i in range(2,int(math.sqrt(n))+1):
            if n%i!=0:
                continue
            else:
                return False
                break
        return True

    #Computing GCD
    @staticmethod
    def gcd(a,b):
        if a == 0 or b==0:
            if a + b == 0:
                print('GCD does not exist')
            else:
                return a + b
        else:
            return discrete_log.gcd(b%a,a)

    #Euler Totient function
    @staticmethod
    def euler_phi(n):
        count = 0
        for i in range(1,n+1):
            if discrete_log.gcd(i,n)==1:
                count = count + 1
        return count

    # Computation of primitive roots
    @staticmethod
    def prim(N):
        n = abs(N)
        t = np.zeros((n-2,n-2))
```

```python
        if discrete_log.is_prime(n)==False:
            return []

        elif discrete_log.is_prime(n)==False:
            # Creates a matrix where each row consists of (i^j)%n for j belongs to {2,...,n-1}.
            for i in range(2,n):
                for j in range(2,n):
                    x = pow(i,j,n)
                    t[i-2][j-2] = x
            b = []
            y = []
            for k in range(0,n-2):
                b = t[k]
                # Check whether cardinality of unique elements in a row is unique or not.
                if len(np.unique(b))==discrete_log.euler_phi(n):
                    if discrete_log.gcd(k+1,n)!=1:
                        y.append(k+2)
            return y

        else:
            # Creates a matrix where each row consists of (i^j)%n for j belongs to {2,...,n-1}.
            for i in range(2,n):
                for j in range(2,n):
                    x = pow(i,j,n)
                    t[i-2][j-2] = x

            b = []
            y = []
            for k in range(0,n-2):
                b = t[k]
                # Check whether all elements in a row is unique or not.
                if len(np.unique(b))==n-2:
                    y.append(k+2)
            return y

    def babystepgiantstep(self):

        if discrete_log.is_prime(self.p)==True and self.k in discrete_log.prim(self.p):
            m = math.ceil(math.sqrt(self.p-1))
            x = pow(self.k,m,self.p)

            # Creates Giant step list
            Q = []
            for i in range(0,m):
                q = pow(x,i,self.p)
                Q.append(q)

            R = []
            for i in range(1,self.p+1):
                if (i*self.k)%self.p==1:
                    z = i
                    break

            # Creates Baby step list
            for d in range(0,m):
                r = (self.n*(pow(z,d,self.p)))%self.p
                R.append(r)

            # For checking the values of q and r such that f(r) = h(q)
            for t in Q:
                if t in R:
                    return Q.index(t)*m + R.index(t)

if __name__=='__main__':
    p = int(input('Enter prime: '))
    k = int(input('Enter base(primitive root of above given prime): '))
    n = int(input('Enter number: '))
    obj = discrete_log(p,k,n)
    print(obj.babystepgiantstep())
```

Listing A.2: Discrete logarithm computation

A.2 Byzantine Agreement Implementation

```
from collections import Counter
from itertools import product
import random
from tabulate import tabulate
from fractions import Fraction as frac

class byzantine_agreement:

    def __init__(self,n):
        self.n=n

    def byzantine_agreement_r1(self):
        m = random.randint(0,int((self.n-1)/3))
        if self.n>=3*m+1:
            # List of permutations of length 2 for pairwise
   exchange of information.
            x = list(product(list(range(1,self.n+1)),repeat=2))
            # List denoting private information of each user.
            y = [0]*len(list(range(self.n)))
            for i in range(len(y)):
                w = random.randint(1,self.n+1)
                if w not in y:
                    y[i]=random.randint(1,self.n+1)
            # List for storing private information of user i
    corresponding to pairwise permutation.
            z = [0]*len(x)
            # Malicious or faulty users list
            M = []
            for i in range(m):
                if random.choice(list(range(1,self.n+1))) not in M:
                    M.append(random.choice(list(range(1,self.n+1)))
 )
            # Sharing of information on one-by-one basis.
            for i in range(len(z)):
                for j in range(0,self.n):
                    if x[i][0]==j+1:
                        z[i]=y[j]
                    if x[i][0] in M:
                        z[i]=random.randint(1,self.n)
            # List storing each information shared.
            R1 = []
            R1.append(['User i','User j','User i -> User j '])
            for i in x:
                for j in range(0,len(z)):
                    if x.index(i)==j:
                        R1.append([i[0],i[1],z[j]])

            for i in range(1,len(R1),self.n+1):
                R1[i][2]='Same user.'

```

```python
            print(' Communicating information ')
            print()
            print(tabulate(R1,headers='firstrow',tablefmt="pretty"))
            return R1,M
        else:
            return 'Byzantine agreement prerequisite not satisfied.'

    @staticmethod
    def byzantine_agreement_r2(L,n):

        if L =='Initial byzantine agreement prerequisite not satisfied.':
            return L
        else:
            print()
            print(' Shared information analysis ')
            print()
            # Creating vector of information obtained from each user
            # j by each user i and i!=j
            M = []
            M.append(['User i','Value of other user(s) denoted as [j,v] where user j has value v (as per user i)'])
            for j in range(1,n+1):
                N = []
                for i in L[0]:
                    if L[0].index(i)>0:
                        if i[1]==j and i[0]!=i[1]:
                            N.append([i[0],i[2]])
                M.append([j,N])

            print(tabulate(M,headers='firstrow',tablefmt="pretty"))
            P=[]

            for i in M:
                if M.index(i)>0:
                    for j in i[1]:
                        P.append(tuple(j))

            # Concluding final correct value obtained by majority
            # and thus rendering malicious users.
            R2 = []
            R2.append(('User','Agreed value','Fraction of users agreed with value'))
            Y = []
            W = []
            for k,v in Counter(P).items():
                if n==4:
                    if v/(n-1) >=2/3:
                        R2.append((k[0],k[1],frac(v-len(L[1]),(n-1)
   )))
                        W.append(k[0])
                    else:
                        if k[0] not in Y:
                            Y.append(k[0])

                elif n>4:
                    if v/(n-1) >2/3:
                        R2.append((k[0],k[1],frac(v-len(L[1]),(n-1)
   )))
                        W.append(k[0])
                    else:
                        if k[0] not in Y:
                            Y.append(k[0])
        print()
        Z = []
        for i in Y:
            if i not in W:
                Z.append('User '+str(i)+' is malicious.')
                R2.append((i,'NIL',1-frac(len(L[1])-v,(n-1))))

        print(tabulate(R2, headers='firstrow',tablefmt="pretty"))
        print()
        if len(Z)!=0:
            return Z
        else:
            return 'All users are honest.'

if __name__=='__main__':
    n = int(input('Enter number of users: '))
    print()
    if 0<n<3:
        print(str(n)+' are honest.')
    elif n<=0:
        print('Enter valid quantity.')
    else:
        obj = byzantine_agreement(n)
        x = obj.byzantine_agreement_r1()
        print(byzantine_agreement.byzantine_agreement_r2(x,n))
```

Listing A.3: BA agreement

A.3 Twisted Edward Curves with Finite Field

```
import math
import matplotlib.pyplot as plt

class ed_curves:
    def __init__(self,a,d,p):
        self.a = a
        self.d = d
        self.p = p

    def gcd(a,b):
            if a == 0 or b==0:
                if a + b == 0:
                    print('GCD does not exist')
                else:
                    return a + b
            else:
                return ed_curves.gcd(b%a,a)

    def prime_check(n):
            if n<=1:
                return False
            for i in range(2,int(math.sqrt(n))+1):
                if ed_curves.gcd(n,i)==1:
                    continue
                else:
                    return False
            return True

    def mod_inverse(n,p):
        if ed_curves.prime_check(p):
            for i in range(1,p):
                if (i*n)%p==1:
                    return i
        else:
            return str(p)+' is not prime.'

    def curve_plot(self):
        if ed_curves.prime_check(self.p) and self.a*self.d*(self.a-
    self.d)!=0:
            C = []
            if ed_curves.prime_check(self.p):
                for i in range(1,self.p):
                    for j in range(1,self.p):
                        if (self.a*pow(i,2)+pow(j,2)-1-self.d*pow(i
    ,2)*pow(j,2))%self.p==0:
                            C.append((i,j))
            if len(C)!=0:
                plt.scatter(*zip(*C),s=10)

            plt.title(f'Points lying on twisted edwards curve with
{self.a}$x^2$ + $y^2$= 1 + {self.d}$x^2y^2$ over $F_{{p}}$( p = {self.p})')
            return C

        else:
            return []

    def check_point(self,P):
        if ed_curves.prime_check(self.p) and self.a*self.d*(self.a-self.d)!=0:
            if len(P[0])==1:
                x = int(P[0][0])%self.p
                if len(P[1])==1:
                    y = int(P[1][0])%self.p
                    if (x,y) in ed_curves.curve_plot(self):
                        return True
                    else:
                        return False
                else:
                    nu = int(P[1][0])%self.p
                    de = int(P[1][1])%self.p
                    if ed_curves.gcd(d,self.p)==1:
                        y = (nu*ed_curves.mod_inverse(de,self.p))%self.p
                        if (x,y) in ed_curves.curve_plot(self):
                            return True
                        else:
                            return False
                    else:
                        return False
            else:
                nu = int(P[0][0])%self.p
                de = int(P[0][1])%self.p
                if ed_curves.gcd(de,self.p)==1:
                    x = (nu*ed_curves.mod_inverse(de,self.p))%self.p
                    if len(P[1])==1:
                        y = int(P[1][0])%self.p
                        if (x,y) in ed_curves.curve_plot(self):
                            return True
                        else:
                            return False
                    else:
                        nu = int(P[1][0])%self.p
                        de = int(P[1][1])%self.p
                        if ed_curves.gcd(d,self.p)==1:
                            y = (nu*ed_curves.mod_inverse(de,self.p))%self.p
                            if (x,y) in ed_curves.curve_plot(self):
                                return True
                            else:
                                return False
                        else:
                            False
                else:
                    return False
        else:
            return False


if __name__=='__main__':
    a = int(input('Enter a - '))
    d = int(input('Enter d - '))
    p = int(input('Enter prime - '))
    obj = ed_curves(a,d,p)
    print('Points on twisted edwards curve - ',obj.curve_plot())
    P = []
    for _ in range(2):
        x = tuple(input().split('/'))
        P.append(x)
    print(obj.check_point(P))
```

Listing A.4: Twisted Edwards curve

Appendix B: Mathematical Explanation

B.1 Computation of p in Leader and Committee Selection

Below are some relevant definitions [GS82].

Definition B.1 *Bernoulli's trial* is described as a random event E with the existence of only two outcomes, either a success or failure, such that whenever the event E occurs, the probability of either outcome remains identical.

Definition B.2 *Binomial distribution* concerning the parameters n and p is the probability distribution for the number of successes or failures for n-independent Bernoulli trials for an event E.

Assume there are a total of k users and L monetary units in the Algorand blockchain network. Let monetary units held by each user u is L_u. Then,

$$L_1 + \cdots + L_k = L$$

initially we conduct an experiment E with respect to these L monetary units as follows:

$$E : \text{Coin flipping experiment}$$

Without loss of generality, we assume that heads represent success and tails represent failure. So on each coin flip

$$p(success) = \frac{q}{L}$$

$$p(failure) = 1 - p(success)$$

where q is a specific parameter of the Algorand blockchain network utilized in the determination of cardinality of the verifiers' committee. Since the probability of success can be altered w.r.t the parameter q, it is analogous to a biased outcome.

Now we relate this with respect to each monetary unit existing in the system. In other words, for each monetary unit, we flip the coin and map its outcome o to the corresponding monetary unit l. It can be represented as $(l, o(E))$. As we discussed before, each monetary unit is held by a user in the network, i.e.,

$$\exists\, i \in \{1, \cdots, k\} \text{ such that } l \in L_i$$

Therefore, for any user u holding L_u monetary units, the total number of successful outcomes lies in the range [0,L_u]. For determining the probability of the total number of successes or failures, a random variable R is defined, illustrating the count of successes out of L_u trails. Hence,

$$R \sim Bin(L_u, p)$$

where $Bin(L_u, p)$ denotes binomial distribution with parameters L_u and p.

So, the probability for m successes for a user u is given as

$$P(R = m) = \binom{L_u}{m} p^m (1-p)^{L_u - m}$$

where $\binom{L_u}{m}$ represents binomial coefficient given as:

$$\binom{L_u}{m} = \frac{L_u!}{(L_u - m)! \cdot m!}$$

Since p is sufficiently small due to large L, the cryptographic sortition algorithm ensures optimum selection while choosing the leader and verifiers committee.

B.2 Zero-knowledge Proofs

Zero-knowledge proofs are an enhancement of interactive proof systems. These interactive proof systems were first studied and given by Shafi Goldwasser, Silvio Micali, and Charles Rackoff [GMR85]. In an interactive proof system, a prover demonstrates its credibility to a verifier by circulating messages. The credibility here refers to the probability of success or failure in proving a statement such that the below two properties hold:

- Completeness: The probability of success in proving a correct statement is significantly high.
- Soundness: The probability of success in proving an incorrect statement is significantly low.

Zero-knowledge proofs were studied and given by Oded Goldreich, Silvio Micali, and Avi Wigderson [GMW86]. A zero-knowledge proof is an efficient methodology where a prover demonstrates that a statement holds without disclosing any extra information apart from the statement's validity. Moreover, apart from the soundness and completeness property of interactive proof systems, it follows one additional significant property:

- Zero-Knowledge: The corresponding proof renders no extra information apart from the theorem's validity, but the verifier is still convinced about the statement's validity.

B.3 Finite Field

A finite field, also termed Galois field, is a field with a finite number of elements. The order of a finite field is either a prime or a prime power. For a prime number p and every positive integer k, there exists fields of order p^k, all of which are isomorphic. In other words, for p^k, there is a field of order p^k which is unique up to isomorphism. These fields are denoted either as $GF(p^k)$ or $\mathbb{F}_{p^k}$ [BL17].

Steps involved in formation of $GF(p^k)$:

1. Select an irreducible polynomial $f(x)$ of degree k with coefficients belonging to $\mathbb{Z}_p$.
2. Choose an element $\beta \notin \mathbb{Z}_p$ such that $f(\beta)$=0.
3. Obtain each element of the $GF(p^k)$ in the form of a polynomial with the degree less than k.

Example B.3 $GF(2^3)$

$$\text{Irreducible polynomial : } f(x) = x^3 + x^2 + 1 \in \mathbb{Z}_2[x]$$

$$\text{Select } \beta \notin \mathbb{Z}_2 \text{ such that } \beta^3 + \beta^2 + 1 = 0$$

$$\beta^3 \equiv \beta^2 + 1$$

$$\beta^4 \equiv \beta \cdot \beta^3 \equiv \beta \cdot (\beta^2 + 1) \equiv \beta^3 + \beta \equiv \beta^2 + \beta + 1$$
$$\beta^5 \equiv \beta^3 \cdot \beta^2 \equiv (\beta^2 + 1) \cdot \beta^2 \equiv \beta^4 + \beta^2 \equiv \beta + 1$$
$$\beta^6 \equiv \beta^3 \cdot \beta^3 \equiv (\beta^2 + 1) \cdot (\beta^2 + 1) \equiv \beta^4 + 1 \equiv \beta^2 + \beta$$

+	0	1	β	β^2	β^3	β^4	β^5	β^6
0	0	1	β	β^2	β^2+1	β^2 + β+1	β+1	β^2+β
1	1	0	1+ β	1+β^2	β^2	β^2 + β	β	β^2+β +1
β	β	β+1	0	β+ β^2	β^2 + β + 1	β^2 + 1	1	β^2
β^2	β^2	β^2+1	β+ β^2	0	1	β + 1	β^2 + β + 1	β
β^3	1+ β^2	β^2	1+ β+ β^2	1	0	β	β^2 + β	β+1
β^4	1+ β+ β^2	β^2 + β	1+ β^2	1 + β	β	0	β^2	1
β^5	1+ β	β	1	β^2 + β+1	β^2+ β	β^2	0	1+ β^2
β^6	β^2+ β	β^2 + β+1	β^2	β	1+β	1	β^2 +1	0

Table B.1: Addition operation - $GF(2^3)$

$\times$	0	1	β	β^2	β^3	β^4	β^5	β^6
0	0	0	0	0	0	0	0	0
1	0	1	β	β^2	$1+\beta^2$	$\beta^2+\beta+1$	$\beta+1$	$\beta^2+\beta$
β	0	β	β^2	$1+\beta^2$	$\beta^2+\beta+1$	$\beta+1$	$\beta^2+\beta$	1
β^2	0	β^2	$1+\beta^2$	$\beta^2+\beta+1$	$\beta+1$	$\beta^2+\beta$	1	β
β^3	0	$1+\beta^2$	$\beta^2+\beta+1$	$\beta+1$	$\beta^2+\beta$	1	β	β^2
β^4	0	$\beta^2+\beta+1$	$\beta+1$	$\beta^2+\beta$	1	β	β^2	β^2+1
β^5	0	$\beta+1$	$\beta^2+\beta$	1	β	β^2	β^2+1	$1+\beta+\beta^2$
β^6	0	$\beta^2+\beta$	1	β	β^2	β^2+1	$1+\beta+\beta^2$	$1+\beta$

Table B.2: Multiplication operation - $GF(2^3)$

Erklärung

Hiermit erkläre ich, dass ich meine Arbeit selbstständig verfasst, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt und die Arbeit noch nicht anderweitig für Prüfungszwecke vorgelegt habe.

Stellen, die wörtlich oder sinngemäß aus Quellen entnommen wurden, sind als solche kenntlich gemacht.

[redacted]

Mittweida, im July 2022