

# Blockchain Application for Academic Documents Verification



By

**Asad Hayat**

00000119891

Supervisor

**Dr. Jamil Ahmad**

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF THE  
REQUIREMENTS FOR THE DEGREE OF MS COMPUTATIONAL  
SCIENCE AND ENGINEERING

Research Center for Modelling and Simulation (RCMS)  
National University of Sciences and Technology (NUST)  
Islamabad, Pakistan

September 2018

# Declaration

I, *AsadHayat* declare that this thesis titled "DocChain: Academic Documents Verification on Blockchain" and the work presented in it are my own and has been generated by me as a result of my own original research.

I confirm that:

1. This work was done wholly or mainly while in candidature for a Master of Science degree at NUST
2. Where any part of this thesis has previously been submitted for a degree or any other qualification at NUST or any other institution, this has been clearly stated
3. Where I have consulted the published work of others, this is always clearly attributed
4. Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work
5. I have acknowledged all main sources of help
6. Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself

---

Asad Hayat,  
00000119891

This thesis is dedicated to *my beloved Ammi and Abbu, Daji and Chachi, my sisters, brothers and my fiancée.*

Thank you for your everlasting prayers, love, sacrifices and warm encouragement throughout my life. I couldn't have reached this far without your support.

# Acknowledgments

I would like to thank Almighty ALLAH who is the greatest of all, who has provided me with strength to achieve this milestone. I would like to show my deepest gratitude to my supervisor Dr. Jamil Ahmad for his support, guidance and mentor-ship throughout this project. I sincerely appreciate the efforts of my GEC members Dr. Mian Ilyas Ahmad, Dr. Sana Ajmal and last but not the least Dr. Muhammad Tariq Saeed for giving their insightful input at every step.

I am highly obliged to my all family members, without their unending support, tolerance and prayers it was impossible to complete this work.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Introduction to Blockchain . . . . .	1
1.2	Applications of Blockchain . . . . .	1
1.2.1	Cryptocurrency Applications . . . . .	2
1.2.2	Non Cryptocurrency Applications . . . . .	3
1.3	Documents Forgery and Fake Degrees . . . . .	4
1.4	Problem Statement . . . . .	5
1.5	Aim and Objectives . . . . .	5
<b>2</b>	<b>Overview of Blockchain in Context of Bitcoin</b>	<b>7</b>
2.1	Centralized, Decentralized and Distributed Systems . . . . .	7
2.2	Cryptography Concepts . . . . .	8
2.2.1	Cryptographic Hash Function . . . . .	8
2.2.2	Collision in Cryptographic Hash Functions . . . . .	9
2.2.3	Public Key Cryptography . . . . .	9
2.2.4	Byzantine General Problem . . . . .	11
2.3	Bitcoin . . . . .	11
2.3.1	Transactions . . . . .	12
2.3.2	Block . . . . .	14
2.3.3	Genesis Block . . . . .	16

## CONTENTS

2.3.4	Blockchain	16
2.3.5	Merkle Tree	16
2.3.6	Bitcoin Mining	20
2.3.7	Forks and Side Chains	21
2.4	Ethereum	22
2.5	Hyperledger	23
<b>3</b>	<b>Hyperledger Fabric</b>	<b>25</b>
3.1	Hyperledger Fabric Network	25
3.1.1	Transactions	26
3.1.2	State	26
3.1.3	Ledger	26
3.1.4	Nodes	27
3.2	Channel	28
3.3	Chaincode	29
3.4	Certificate Authority	29
<b>4</b>	<b>Blockchain Solution</b>	<b>31</b>
4.1	Development Environment	31
4.1.1	Installing Fabric Binaries and Docker Images	32
4.1.2	Generating Artifacts	33
4.1.3	Channel Configuration Transaction	34
4.1.4	Starting Network	35
4.1.5	Creating Channel	35
4.1.6	Joining Channel	35
4.1.7	Updating Anchor Peers	36
4.1.8	Installing and Instantiating Chaincode	37
<b>5</b>	<b>Development and Testing of Blockchain Solution</b>	<b>38</b>

## CONTENTS

5.1	Participants and Use Case . . . . .	38
5.1.1	Governing Body . . . . .	38
5.1.2	Organizations (Universities) . . . . .	39
5.1.3	Degree Signers . . . . .	40
5.1.4	Employer/Verification Point . . . . .	41
5.2	Chaincode Development . . . . .	42
5.2.1	University Structure . . . . .	42
5.2.2	Department Structure . . . . .	43
5.2.3	Program Structure . . . . .	44
5.2.4	Specialization Structure . . . . .	44
5.2.5	Degree Structure . . . . .	45
5.2.6	CNICDegree Structure . . . . .	45
5.3	Development of Web Application . . . . .	45
5.3.1	University Admin Routes . . . . .	46
5.3.2	Exam Admin Routes . . . . .	46
5.3.3	Signing Routes . . . . .	46
5.4	Web Application Interface . . . . .	46
5.4.1	University Admin . . . . .	46
5.4.2	Exam Admin . . . . .	51
5.4.3	Degree Signing . . . . .	51
5.5	Testing of Blockchain Application . . . . .	57
5.5.1	University Admin . . . . .	57
5.5.2	Exam Admin . . . . .	62
5.5.3	Signing Degree . . . . .	65
5.5.4	Verification of Degree . . . . .	68
<b>6</b>	<b>Discussion, Conclusion and Future Work</b>	<b>69</b>

## CONTENTS

6.1 Discussion . . . . .	69
6.2 Conclusion . . . . .	70
6.3 Future Work . . . . .	71
<b>A Cryptographic Configuration</b>	<b>75</b>
<b>B Configuration Transaction</b>	<b>77</b>
<b>C Sample Network Configurations for docker-compose</b>	<b>81</b>
C.1 YAML code for Docker Compose . . . . .	82
C.2 Starting the network . . . . .	96



# List of Figures

2.1	(A) Centralized System, (B) Decentralized System and (C) Distributed System [17]. . . . .	8
2.2	View of real Bitcoin transaction. . . . .	12
2.3	Transactions for a specific Address . . . . .	13
2.4	An illustration to view how blocks are linked in the blockchain [22] . . . . .	17
2.5	Graphical representation of Merkle Tree [22] . . . . .	19
2.6	Balancing of Merkle Tree with odd number of nodes. [22] . . . . .	19
2.7	Illustration of using Merkle Path to verify a transaction. [22] . . . . .	20
2.8	Illustration of Proof of Work. [23] . . . . .	21
2.9	Illustration of chain split. [23] . . . . .	22
2.10	Longest chain is the official chain. [23] . . . . .	23
3.1	Illustration of Transaction flow in Hyperledger Fabric . . . . .	28
3.2	High level overview of a typical Hyperledger Fabric Network. . . . .	30
5.2	Use Case Diagram: Exam Admin. . . . .	39
5.1	Use Case Diagram: University Admin. . . . .	40
5.3	Use Case Diagram: Principal. . . . .	41
5.4	Use Case Diagram: Registrar. . . . .	41
5.5	Use Case Diagram: Rector. . . . .	41
5.6	Chaincode Class Diagram . . . . .	43

## LIST OF FIGURES

5.7	Sequence Diagram - Register University . . . . .	47
5.8	Sequence Diagram - Add Department . . . . .	48
5.9	Sequence Diagram - Add Program . . . . .	49
5.10	Sequence Diagram - Add Specialization . . . . .	50
5.11	Sequence Diagram - Add Degree. . . . .	52
5.12	Sequence Diagram - View Degree. . . . .	53
5.13	Sequence Diagram - Signing Degree by Principal. . . . .	54
5.14	Sequence Diagram - Signing Degree by Registrar. . . . .	55
5.15	Sequence Diagram - Signing Degree by Rector. . . . .	56
5.16	Snapshot - Login View. . . . .	57
5.17	Snapshot - University Admin Dashboard at first login. . . . .	58
5.18	Snapshot - University Admin, registering university. . . . .	59
5.19	Snapshot - University Admin Dashboard, after university is registered. . . . .	59
5.20	Snapshot - University Admin, Department details input. . . . .	60
5.21	Snapshot - University Admin, Departments list. . . . .	60
5.22	Snapshot - University Admin, Add Program. . . . .	61
5.23	Snapshot - University Admin, Add Specialization. . . . .	61
5.24	Snapshot - Exam Admin, Home Page. . . . .	62
5.25	Snapshot - Exam Admin, Add Degree. . . . .	63
5.26	Snapshot - Exam Admin, Add Degree, Success Response. . . . .	64
5.27	Snapshot - Exam Admin, Add Degree, Error Response . . . . .	64
5.28	Snapshot - Exam Admin, View Degree . . . . .	65
5.29	Snapshot - Exam Admin, Degree not found error. . . . .	65
5.30	Snapshot - Principal, Signing List. . . . .	66
5.31	Snapshot - Registrar, Signing List. . . . .	66
5.32	Snapshot - Rector, Signing List. . . . .	66
5.33	Snapshot - Degree Issued. . . . .	67

# List of Tables

2.1	Major fields in a Bitcoin block . . . . .	14
2.2	Structure of Block Header . . . . .	15
C.1	List of Peers, Orderers and Certificate Authority in each organization. . . . .	82

# List of Abbreviations and Symbols

## Abbreviations

<b>BTC</b>	Bitcoin
<b>ETH</b>	Ethereum
<b>HL</b>	Hyperledger
<b>HLF</b>	Hyperledger Fabric
<b>LF</b>	Linux Foundation
<b>CC</b>	Chaincode
<b>CA</b>	Certificate Authority
<b>POW</b>	Proof of Work
<b>SHA</b>	Secure Hash Algorithm

# Abstract

Blockchain is an innovative technology that has the potential to overhaul our approach to many problems and issues which previously did not seem possible. Blockchain is distributed ledger technology in which record of different transactions are stored on continuously growing ledger. This ledger of transactions are kept on different peers of network in a way that make it tamper proof. Data on blockchain is highly secured by the use of cryptography and network structure of blockchain makes it almost impossible to hack. Moreover, transactions on blockchain are immutable, auditable and traceable. Fake and forged academic documents is a major issue that organizations face worldwide. Particularly in Pakistan, verification of academic documents is a time consuming process that may require up to several days which prolong administrative processes of organizations. In this research we design and implement an approach to issue academic degrees on blockchain, which can be verified by an employer by using Computerized National Identity Card (CNIC) number of an individual. The proposed solution for academic documents verification is a developed using Hyperledger Fabric. The chaincode (smart contract) is developed in Go programming language. Web Interface of the application is implemented by using official NodeJS Software Development Kit (SDK) for Hyperledger Fabric.

This system allows examination section to create degrees on the blockchain network which is then signed by different officials. After creation of a degree, first it needs to be signed by Principal, then by University Registrar and finally by University Rector. After all the sign transactions, the degree is made available for verification to any organization that have obtained proper network enrollment certificates from Higher Education Commission (HEC) or University.

## CHAPTER 1

# Introduction

### 1.1 Introduction to Blockchain

The term "Blockchain" first appeared in the white paper [1], presented by the inventor of Bitcoin, Satoshi Nakamoto. In 2008, Satoshi Nakamoto sent an email to cryptography mailing list, in which he proposed peer to peer electronic cash system with a solution to double spend problem. The proposed cash system was designed to operate without the requirement of a central authority to clear and validate transactions. Few months later, in november 2008, Satoshi Nakamoto shared link to Bitcoin White-paper [1]. On 9 January, 2009, Satoshi Nakamota released Bitcoin v0.1 on cryptography mailing list[2]. Bitcoin network was created in january 2009, when Satoshi Nakamoto mined the first block (genesis block) of the chain.

Transactions record of bitcoin are grouped in form of blocks. Each new formed block is linked with previous block by storing hash of previous block in newly formed block. This series of cryptographically linked blocks forms a chain, called Blockchain. Each full node on the network maintain a copy of blockchain which is updated continuously as new blocks are added by the network.

### 1.2 Applications of Blockchain

Blockchain is one of the most important innovation of this century. Although it is primarily know for its association with cryptocurrencies, mainly bitcoin, blockchain has the potential to alter many industries to its core.

### 1.2.1 Cryptocurrency Applications

Blockchain is driving almost all decentralized cryptographic currencies. There are about two thousands cryptocurrencies with combined market capitalization of more than 200 billion dollars <sup>1</sup>. Few major cryptocurrencies are:

- **Bitcoin:** Bitcoin is the first and largest cryptocurrency by market capitalization. Transactions are grouped in blocks, which are verified by network through mining. Algorithm used for mining by Bitcoin network is called Proof of Work. On average, a block is generated every 10 minutes. With every block mined, specific number of bitcoins is awarded to miner. This block reward is halved every four years until total number of bitcoins on the network is reached to its upper cap, 21 million bitcoins. Bitcoin core software is open source and is maintained by group of developers, called Bitcoin Core Developers[1].
- **Ethereum:** Ethereum is open source, distributed blockchain platform with support of smart contract functionality. Ether is cryptocurrency on ethereum platform, which ,like bitcoin, is used to transfer credit between accounts and reward miners. Ethereum also use Proof of Work as consensus algorithm.  
  
Smart contracts for ethereum platform are written in special programming language called Solidity. Solidity is a java script styled programming language and is developed specially for developing smart contracts for ethereum platform [3].
- **Bytecoin:** Bytecoin is open source cryptocurrency that uses CryptoNote technology. Bytecoin is different from bitcoin and ethereum, as it ensures anonymous cash settlement. Bytecoin was developed independently from bitcoin as a true private cryptocurrency based on CryptoNote technology[4].
- **Ripple:** Ripple is a decentralized payment protocol designed for instant, secure and nearly free global transactions of any size. Ripple network uses cryptocurrency called XRP. Ripple is continuously expanding and has been adopted by many banks and settlement companies across the globe[5].

---

<sup>1</sup>CoinMarketCap - [www.coinmarketcap.com](http://www.coinmarketcap.com)

### 1.2.2 Non Cryptocurrency Applications

Blockchain can be used in different industries to improve transparency and services, and reduce cost. Below are few scenarios where blockchain can be applied.

- **Digital Identity:** As blockchain, unlike traditional management systems, do not require accounts and permission related to these accounts for managing digital assets. On blockchain system, ownership of digital certificates or cryptographic keys represents ownership of digital assets. Blockchain is continuously growing by addition of new transactions and each transaction is signed by a cryptographic keys to prove ownership of assets involved in transaction. This provide an efficient and secure way to manage digital identities that avoid user to share personal information[6][7].

- **Financial Services**

Blockchain has tremendous potential to improve various financial systems. Current systems are slow, error prone and intermediaries are required to clear transactions and resolve conflicts. Blockchain systems does not require intermediaries for clearing transactions and conflict resolution. Trade processing and settlements, Insurance claims processing and Cross border payment are few Use Cases[8].

- **Supply Chain Management** Blockchain has huge potential to increase transparency and reduce costs of global supply chain. About 90% of goods in global trade are carried by shipping. Processing documents and information for a container shipment costs twice that of actual shipment<sup>2</sup>.

Every intermediary in the system maintains their own record independent from others. This slow down the process and make the system prone to documentation errors. Blockchain provide same ledger for all participants, thus greatly helps in reducing documentation errors and speeding up the process.

Blockchain can be integrated with internet of things to record complete details of conditions in which goods are shipped on the blockchain. This can greatly improve the transparency for quality inspection to provide better end products to consumer[9].

---

<sup>2</sup><https://youtu.be/dcddYatMCGQ?t=44>



### 1.3 Documents Forgery and Fake Degrees

Even in this age of advanced digital communication, paper documents still holds very important position. Advanced printing technologies provides various ways to print documents with special security features, but the same technology can also be accessed by malicious users [10].

Document forgery is a process by which authorized documents are modified or unauthorized documents are fabricated for illegal usage[11]. Several techniques are used to identify authentic documents. In [12], the author discusses the use of digital watermarking for authentication of digital documents. In [13], the author propose the usage of specialized fonts in documents to tackle documents forgery. Hidden information can be embedded digital documents for verification purposes. This concept of using hidden information in digital documents is known as steganography [14] [15]. Although these techniques makes forgery a difficult task but due to easy and cheap access to technologies, these techniques can be used maliciously.

Forged academic degrees or diplomas have become a worldwide problem due to its increasing commercial value. Academic achievements is used for employment access, promotion, professional recognition and safe passage through immigration. Educational records are also used as bargaining tool for better positions and salaries[16].

Today, where access to technology is cheaper than ever, forging an academic document is not very difficult. With tens of thousands of educational institutes globally, verifying the authenticity of an academic degree could also be a challenge. There are mainly two points that needs to be verified in order to assure the authenticity of an academic document. Firstly, it is assured that the issuing institute is legitimate and secondly, the document at hand is actually issued by that particular institute and not forged maliciously.

Another problem is that a legitimate document might be obtained illegally. In this type of scenario the malicious actors have high level access to the issuing institute. In such scenarios there is very low chance of tracing and detecting the illegal entry and audit of the record might not be possible.

## 1.4 Problem Statement

Academic Degree is a document of proof issued by an educational institute stating that the individual has studied a certain academic program at that particular institute. Academic institutes issue degrees on paper with some unique features, like institute's seal, and signatures of officials.

The problem with the paper degree is that it can be illegally forged outside the institute by malicious actors in society. These documents can be forged with such excellence that it cannot be detected by common observation. Cases of using fake degree for employment has been on the news several times <sup>3</sup> in which applicants had successfully secured employment position. The degree can be verified by contacting the issuing institute but the process is slow and verification may require up to few days. Some institute also offer verification portal, which is a quick way for degree verification. Although quick and efficient, such portals maintain a central database which is a single point of failure. Such portals are also prone to hacking and record can be altered without any detection.

In this thesis we are taking advantage of decentralized nature of blockchain to develop a degree issuing and verification platform to address the issue of documents forgery. Since the record on blockchain can be audited all the way to its origin, any alteration of records can be detected and traced.

## 1.5 Aim and Objectives

Aim of this research is to develop a blockchain application for the verification of academic documents. Objectives to develop this application are:

- Design a permissioned blockchain network for issuance and verification of academic degrees.
- Design and model a blockchain solution for issuance and verification of academic degrees.

---

<sup>3</sup> <https://www.thenews.com.pk/print/271370-more-than-1-100-pakistanis-paid-for-axact-degrees>  
<https://tribune.com.pk/story/1660677/1-659-pia-employees-found-holding-fake-degrees-na-told/>  
<https://tribune.com.pk/story/1706726/1-24-active-pia-pilots-fake-degrees-caa-informs-sc/>  
<https://tribune.com.pk/story/674780/bogus-documents-ogdcl-suspects-109-officials-have-fake-degrees>  
<https://www.thenews.com.pk/print/270174-axact-offered-fake-degree-to-bbc-reporter>

## CHAPTER 1: INTRODUCTION

- Develop and Deploy a smart contract that enables users to issue and verify academic degrees.
- Design and develop web user interface for interaction with blockchain network.
- Testing of network and application using docker containers.
- Deployment of application on physical network.

# Overview of Blockchain in Context of Bitcoin

## 2.1 Centralized, Decentralized and Distributed Systems

Computer Network Systems can be classified as centralized systems, decentralized systems and distributed systems based on arrangement of connectivity among different nodes. These systems can be described briefly as:

- **Centralized Systems:**

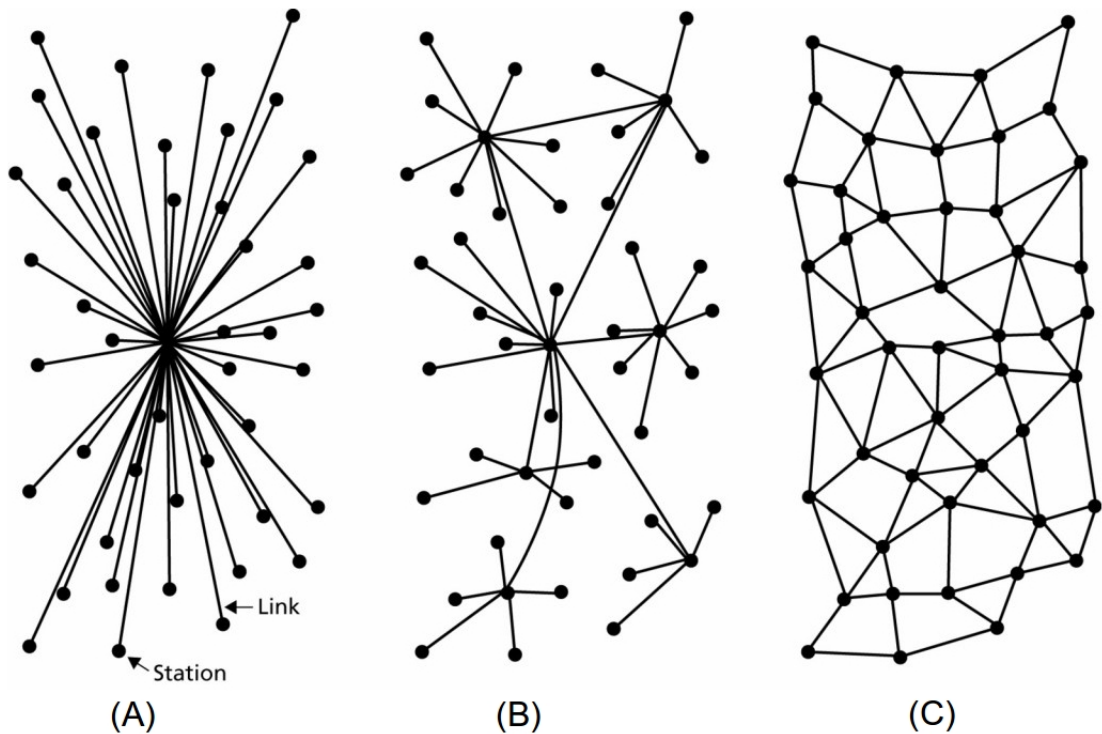
In Centralized Systems there is a central node to which every other node are connected. Centralized systems have central entity which controls and manage the whole system. If central entity is compromised the whole system goes down.

- **Decentralized Systems:**

Decentralized systems do not have a single central point of control, thus no single point of failure. If one node is compromised, it does not compromise the whole network but few nodes that are connected through the compromised node.

- **Distributed Systems:**

In Distributed Systems every node is connected to one or more adjacent nodes. There is no single point of failure. If a node is compromised it does not affect the network. The nodes that were connected to compromised node may already be connected to other nodes or can form new connections to stay connected in the network.



**Figure 2.1:** (A) Centralized System, (B) Decentralized System and (C) Distributed System [17].

## 2.2 Cryptography Concepts

### 2.2.1 Cryptographic Hash Function

A cryptographic hash function is a one to one mapping function that takes any arbitrary input and map it to a fixed length alphanumeric output. The alphanumeric output is called hash value, fingerprint, digest or signature of the input [18]. There are many types of cryptographic hash functions, but the one that is of interest in this discussion is SHA256. SHA stands for Secure Hash Algorithm. There are three different families of SHA algorithms, SHA-0, SHA-1 and SHA-2. There are many hash functions in SHA-2 family, which uses same algorithms with different numbers of output bits. For example SHA-256 produces hash of 256 bits and SHA-512 outputs 512 bits hash[19].

Hash functions are one way functions, we can easily compute hash of any arbitrary input, but given a hash value, it is computationally infeasible to find the corresponding input.

### 2.2.2 Collision in Cryptographic Hash Functions

The take away from above paragraph is that if we feed any arbitrary input to a particular cryptographic hash function we will get a fixed length hash string. No matter how many times we provide a specific input, we will get same output. As the output is of fixed length, no matter what the size of input data is, we have a finite set of outputs that a particular hash function can produce. The variation of input data that we can provide to a hash function is infinite. So without any doubt we can say that, there exists same output for two different inputs. This same hash of two different inputs is called collision.

This problem of collision can be stated as,

Find two different inputs  $m1$  and  $m2$ ,  
such that

$$m1 \neq m2$$

$$\text{hash}(m1) = \text{hash}(m2) = h$$

where

$$m1, m2 \in M, \quad h \in H$$

.

In this statement  $M$  is an infinite sized state space, where as  $H$  has  $2^n$  elements, where  $n$  is the number of bits in output of hash function.

Here we can easily infer that  $H \subset M$ , because we can also provide hash value of specific message as input to hash function. So it is possible to find  $m1$  and  $m2$ , such that  $\text{hash}(m1) = \text{hash}(m2)$ .

For example, SHA-256 produces 256 bit long alphanumeric hash value. So the total number of hashes that SHA-256 can produce is  $2^{256}$ . So we can certainly say that there would be at least one collision for  $2^{256} + 1$  different inputs.

Although, theoretically, we can prove that a collision exists, but practically finding a collision is computationally infeasible.

### 2.2.3 Public Key Cryptography

Public Key Cryptography is a cryptographic system that uses pair of keys, public key and private key. Public key, as its name indicates, can be shared publicly, while private key

is meant to keep private. Although this keys pair is mathematically related, computing private key from public key is computationally infeasible.

This Private/Public keys pair is used to authenticate that a specific message is signed by a specific sender. First a message is signed by sender using its private key. Then the sender share the message, message signature and public key with receiver. Now the receiver can verify the message using message, message signature and public key[20].

Without going into low level details of this process, at high level two main operations involved in this message sharing are.

- **Signing of Message** As stated the message is signed by sender using message and private key using  $sign()$  operation. The prototype of  $sign()$  operation is as following.

$$sign(Message, PrivKey) = Signature$$

The signature generated by  $sign()$  operation is different for different messages. This ensures that no one can just take a signature of one message and attach with another.

Now the sender shares Message, Signature and corresponding Public Key with receiver.

- **Verification of Message** The receiver receives the Message, Signature and Public Key of the sender. He can verify the message using  $verify()$  operation to make sure that the message has not been tampered during its transmission from the sender. The prototype of  $verify()$  operation is as following.

$$verify(Message, Signature, PublicKey) = T/F$$

As evident from the prototype, the verify operation is a boolean operation. It returns True if the message is not tampered and input combination is correct. If the message is tampered or the input combination is not correct, it will return False.

The inside details of these operations are above the scope of this work, but the take away from this discussion is that it is computationally infeasible to find a

valid signature without knowing the private key.

#### 2.2.4 Byzantine General Problem

Consider a city surrounded by several divisions of Byzantine Army. Each division is headed by a General some of which are traitors. The strategy is that all divisions must attack or retreat together. The city can not defeat Byzantine army if all divisions attack at once, but have enough resources to defeat them if one or more divisions retreat. These divisions of Byzantine Army communicate through messengers, who have to pass through the city under-attack, to transfer messages between different divisions and city's defence can intercept and tamper these messages.

The problem is to devise a mechanism in this trust-less environment, so that all division are in consensus about whether to attack or retreat.

Byzantine General's problem is used in context of a fault tolerant system, particularly distributed system, where components might fail and the information about status of these components is not authentic. This cause confusion in failure detection system, because in order to shut down a component, failure detection system must be in consensus regarding the status of component[21].

There are different solutions presented to different variants of Byzantine General's Problem. In Bitcoin network Proof of Work 2.3.6 is used to reach consensus among peers.

### 2.3 Bitcoin

Bitcoin is the cryptographic currency conceptualized and developed by Satoshi Nakamoto. It is the first cryptographic currency among more than fifteen hundreds others. Users can perform transactions by transferring bitcoins from their wallet to receiver wallet by sending it to receiver's bitcoin address. Bitcoin can be obtained either by mining or by buying from other bitcoin user. In this section we will discuss bitcoin and bitcoin protocol in detail.

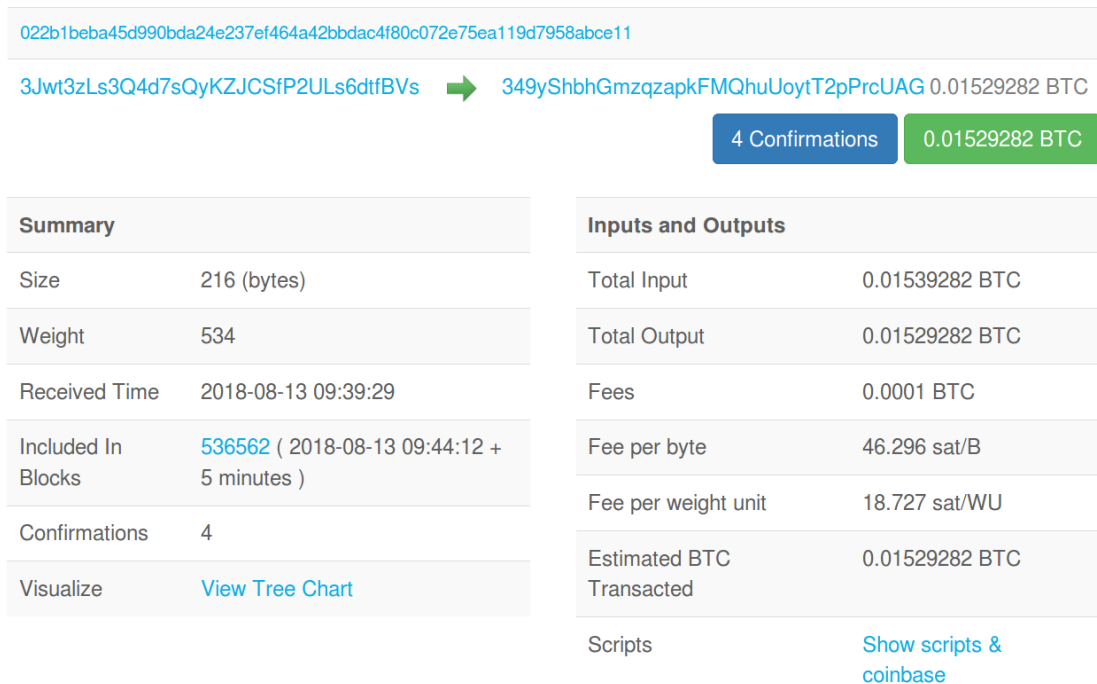


### 2.3.1 Transactions

In simple terms, transaction refer that the owner of some bitcoins has allowed the transfer of some of the the bitcoins, that he owned, to another owner. After successful transaction the new owner can now perform transaction by transferring some of bitcoins, he got from first owner, to another owner and so on.

Transaction has an input and an output. Input bitcoins are the bitcoins that a sender want to sends to a receiver and the output bitcoins are the bitcoins that receiver receives. The number of input bitcoins are not necessarily adds up to number of output bitcoins. Instead the output bitcoins are slightly less in number. The difference of input and output bitcoins are received by miner as transaction fees, who perform computational work to include transaction in the ledger. Lets dig a little deeper by inspecting a real bitcoin transaction.

## Transaction View information about a bitcoin transaction



**Figure 2.2:** View of real Bitcoin transaction.

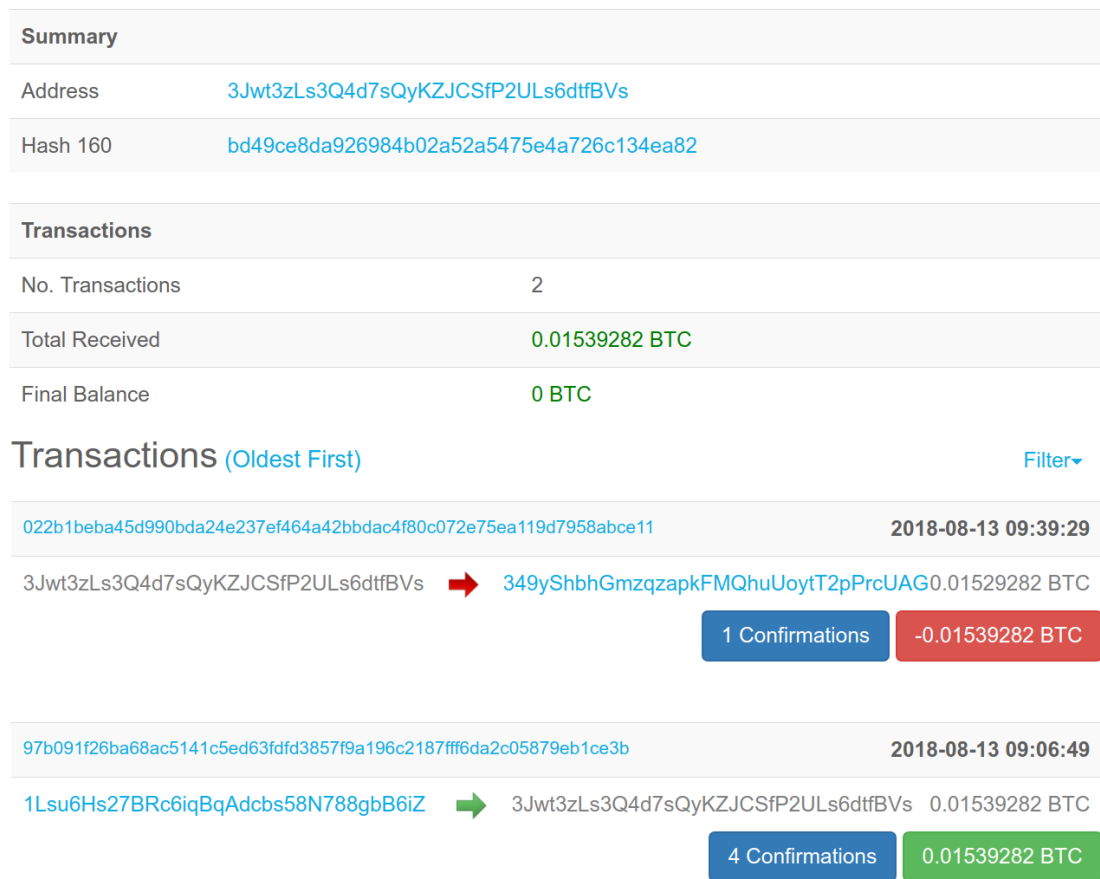
Above screen-shot shows data of a real bitcoin transaction. The sender with address `3Jwt3zLs3Q4d7sQyKZJCSfP2ULs6dtfBVs` sends `0.01529282 BTC` to receiver with address `349yShbhGmzqzapkFMQhuUoytT2pPrcUAG`. But how does the bitcoin protocol

ensures that the sender actually have the amount he is sending to receiver. As stated every transaction is recorded on the ledger, so by inspecting the ledger for transactions involving this specific sender. Final net balance of this sender must be greater or equal to the amount he is sending (0.01529282 BTC). The following screen-shot shows actual transactions involving this specific sender, 3Jwt3zLs3Q4d7sQyKZJCSfP2ULs6dtfBVs.

We can inspect the ledger to verify the balance of a specific sender. Fig 2.3 shows that the sender received 0.01539282 BTC from 1Lsu6Hs27BRc-6iqBqAdcbs58N788gbB6iZ just about half an hour before he send it to 349yShbhGmzqzapkFMQhuUoytT2pPrcUAG. So in Bitcoin Network, there is a chain of transaction where output of one transaction is the input of another.

## Bitcoin Address

Addresses are identifiers which you use to send bitcoins to another person.



**Figure 2.3:** Transactions for a specific Address

There is a problem, a big problem, in the scenario that we have built so far. The problem

is that how can we prevent someone to spent on behalf of another user, since the ledger is public and anyone can add transactions to the ledger. This is where public/private key cryptography comes in play, Section 2.2.3. To put in simple terms, the sender signs the transaction using his private key and broadcast the signed transaction to the network. Other participants (miners) on the network receives the transaction, verify it using the public key of the sender, verify that sender has sufficient balance by inspecting chain of transaction on the ledgers and if all goes well, the transaction is validated and added to the block. This process of validation of transaction and addition of transaction to ledger is not so straight forward. Transactions are bundled in blocks and are added to ledger by special nodes called miners. Miners compete each other to solve a special puzzle, difficult to solve and easy to verify, and the winner gets to add the block to ledger. The winner is rewarded with few freshly created bitcoins and transaction fees of transactions in the block. Since all nodes on the network must maintain same ledger, there must be consensus among nodes about transactions and order of transactions to be included in the ledger.

Before discussing about consensus and mining we need to discuss the structure of ledger maintained by participants in the bitcoin network.

### 2.3.2 Block

Blockchain is list of blocks of transactions. On average, new block is created every 10 minutes on bitcoin network. Each new block formed is linked with previous block by referring to SHA-256 hash of previous block.

A typical block has four major fields.

Field	Description
Block Size	Size of block in bytes
Block header	Multiple header fields
Transaction Counter	Number of transactions included in this block
Transactions	Transactions included in this block

**Table 2.1:** Major fields in a Bitcoin block

Let us discuss these fields one by one in details.

Block Size is simple and self explanatory. It stores the size of block in bytes, excluding itself.

### Block Header

Header of Block contains several fields, like, Version, Previous Block Hash, Merkle Root, Timestamp, Difficulty Target and Nonce.

Field	Description
Version	Version number of software/protocol used
Previous Block Hash	Hash of previous block in the chain
Merkle Root	Hash of root of Merkle-Tree of transactions
Timestamp	Time of creation of this block
Difficulty Target	POW Algorithm difficulty target
Nonce	A number the miners have to find

**Table 2.2:** Structure of Block Header

The version number is used to track software/protocol updates. Previous Block Hash links this block with parent block by using SHA-256 hash of previous block. Merkle Root refers to the the hash of root of Merkle-Tree of transactions. Timestamp refers to the approximate time of creation of this block, in seconds from Unix Epoch. Timestamp, Difficulty Target and Nonce is related to mining and will be discussed in Bitcoin Mining, Section 2.3.6.

Cryptographic hash of a block is its primary identifier. A block can be independently identified by any node using block hash without any ambiguity.

Another identifier of a block is its position in the blockchain, called block height. The first block of the blockchain have block height of zero. Unlike block hash, block height is not unique identifier because a single block height can refer to more than one block. Blockchain can grow side chain where blockchain splits in multiple chains. This commonly happens when there is a major upgrade in bitcoin protocol or two independent miners mined same block independently. Forks and side chain are discussed in Section 2.3.7, Forks and Side Chains [22].

### 2.3.3 Genesis Block

Genesis Block is the first block of the blockchain. It does not have any parent block and its Previous Block Hash field's value is "00". Genesis block of bitcoin was mined by Bitcoin's creator Satoshi Nakamoto at 2009-01-03 18:15:05.

### 2.3.4 Blockchain

Nodes on bitcoin maintain a local copy of blockchain. Local copy have all transactions ever occurred on the network starting from genesis block. As new transactions are performed, new blocks are formed. Nodes continuously updates local copy of ledger by adding these new blocks to remain in synchronization with other nodes.

As the node receive new block, it validates it by validating the transactions in the block and check that proof of work has been performed on the block. Now the newly received block has to be linked after the last block of the local copy. The node will inspect the "previous block hash" field in the header of new block and compare it with hash of last block from local copy of ledger. If the two hash values are same, the new block is child of the last block and are linked cryptographically. These blocks link to form a chain of blocks called **Blockchain** (Figure 2.4).

### 2.3.5 Merkle Tree

Merkle Tree provides an efficient way to verify that a specific transaction exist in the block. Each block uses Merkle Tree to store summary of all transactions.

Merkel Tree, like Binary Tree, is a Binary Hash Tree, which is used to verify the integrity of large data sets. It is formed by the recursive hashing of pairs of nodes until until there is a single node called root node. In bitcoin double SHA-256 is used in Merkel Trees. Complexity and computation time of Merkel trees are same as that of Binary Trees, which makes it an efficient approach to verify if a transaction is included in block. To see how Merkel trees are used in bitcoin, let us apply it on sample scenario of four transactions, Transaction A,B,C and D.  $H_A, H_B, H_C$  and  $H_D$  denotes double hashes of transactions A, B, C and D respectively.

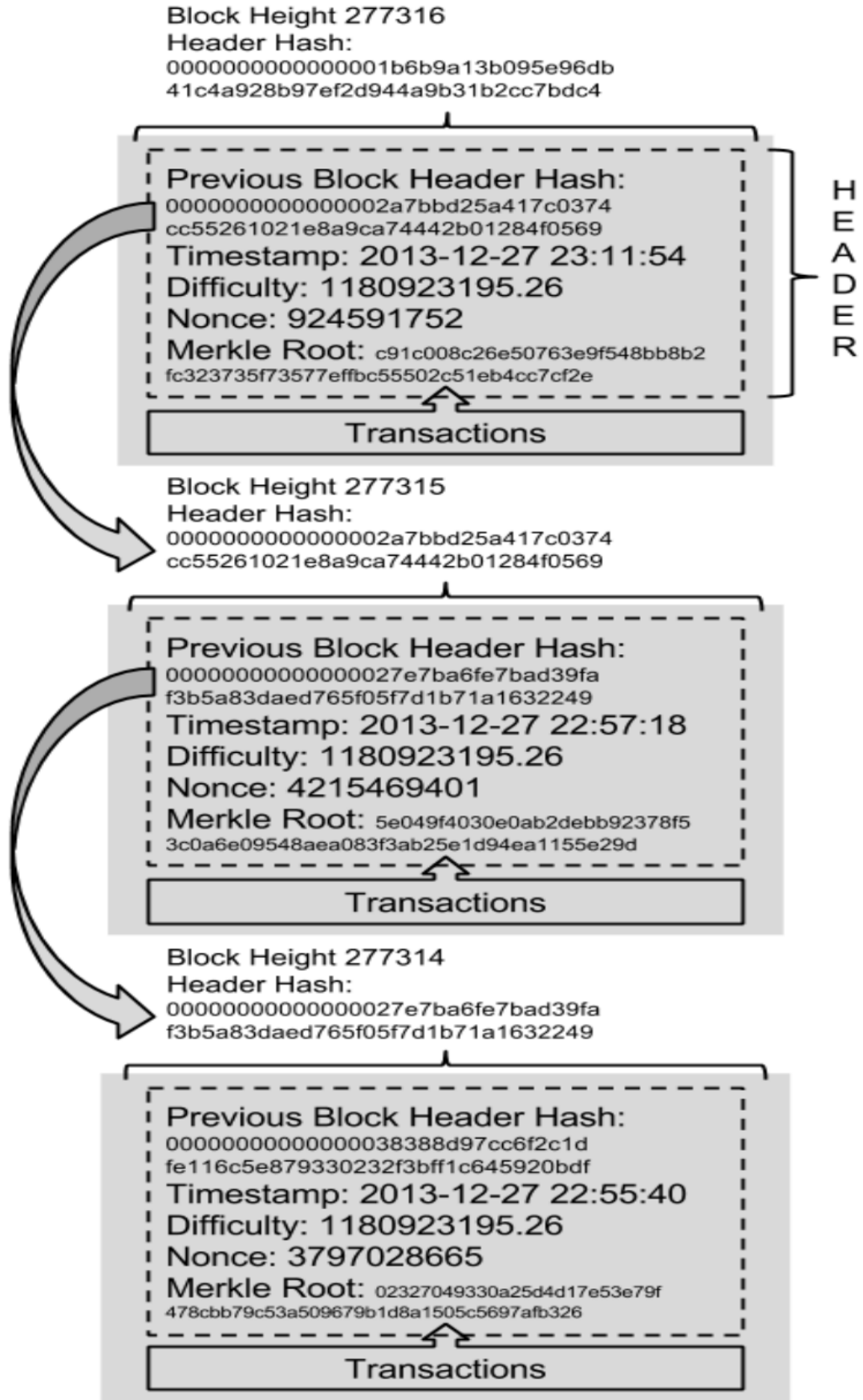


Figure 2.4: An illustration to view how blocks are linked in the blockchain [22]

$$H_A = \text{SHA256}(\text{SHA256}(\text{TransactionA}))$$

$$H_B = \text{SHA256}(\text{SHA256}(\text{TransactionB}))$$

$$H_C = \text{SHA256}(\text{SHA256}(\text{TransactionC}))$$

$$H_D = \text{SHA256}(\text{SHA256}(\text{TransactionD}))$$

The parent node is formed by concatenating hashes of two consecutive pairs and then hashing them together.

For example:

$$H_{AB} = \text{SHA256}(\text{SHA256}(H_A + H_B))$$

This process is recursively performed until there is a single node, called Merkle Root. Example is illustrated in Figure 2.5.

Merkle Tree needs even number of nodes, since it is a binary tree. If there are odd number of transactions in a bitcoin block, the last transaction is duplicated to make total number of nodes even. This process is called balancing of tree and is illustrated in figure 2.6.

To verify that a specific transaction is included in a block, we need to produce only  $\text{Log}_2(N)$  hashes of merkle path that connects a specific transaction to merkle root.

In example, illustrated in figure 2.7, to verify that "Transaction K" is included in block, we need a Merkle Path. In diagram (figure 2.7), the Merkle Path consists of four hashes,  $H_L, H_{IJ}, H_{MNOP}$  and  $H_{ABCDEFGH}$  as indicated by blue nodes. With Merkle Path provided we can verify that Transaction K is included in block by computing  $H_K, H_{KL}, H_{IJKL}, H_{IJKLMN}$  and the the Merkle Root. If the Merkle Root computed is identical to that stored in block then Transaction K is included in block.

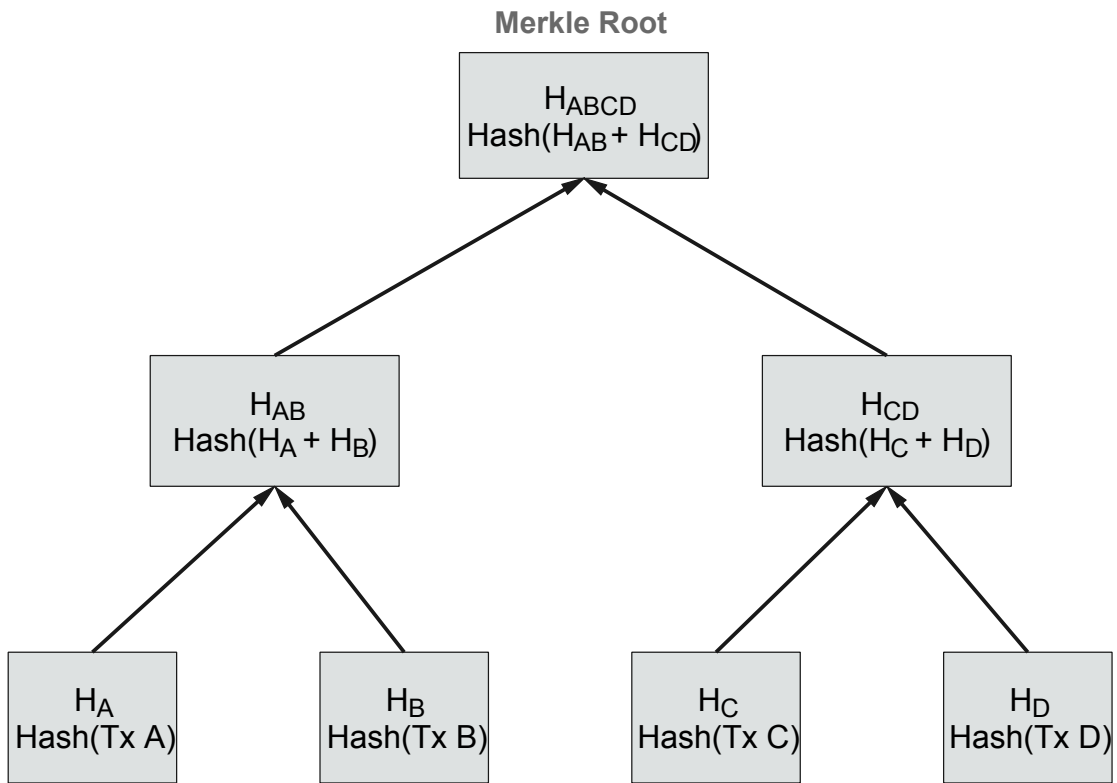


Figure 2.5: Graphical representation of Merkle Tree [22]

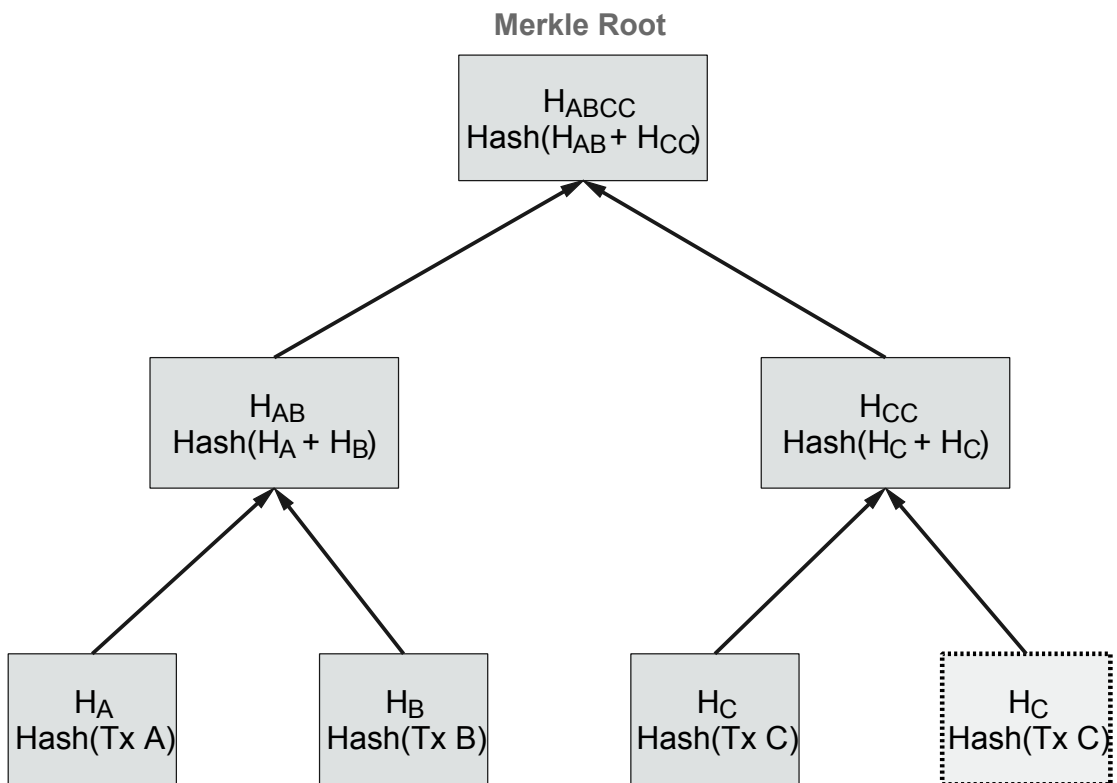


Figure 2.6: Balancing of Merkle Tree with odd number of nodes. [22]



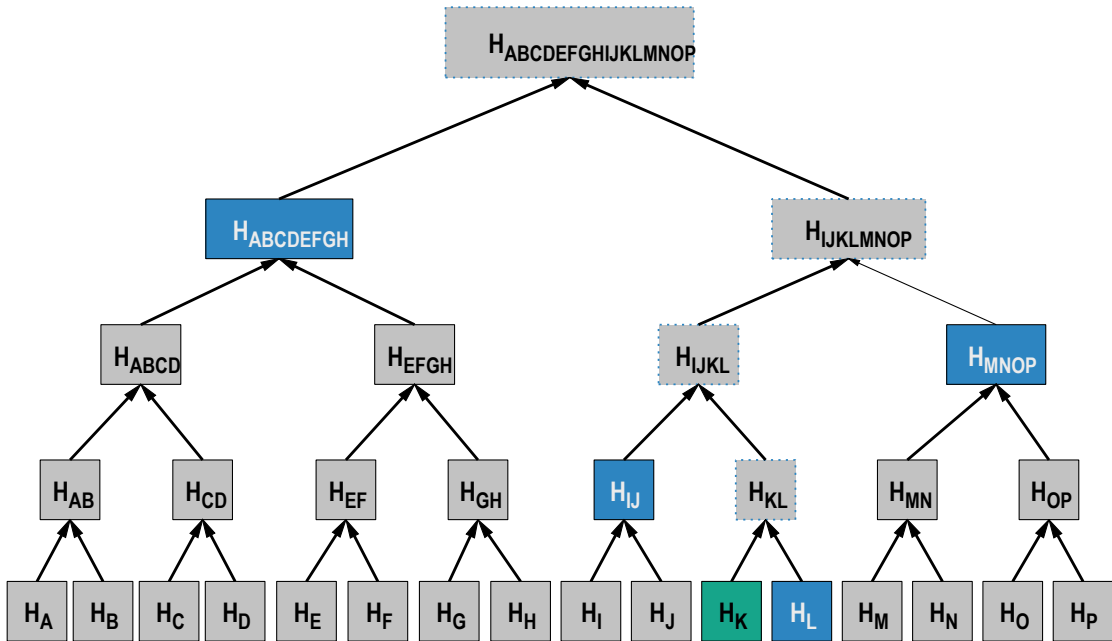


Figure 2.7: Illustration of using Merkle Path to verify a transaction. [22]

### 2.3.6 Bitcoin Mining

Mining is the process in which new bitcoins are generated by the miners. Each miner on the network compete to generate new block. To generate new block, miners have to solve a puzzle, called proof of work. The first miner to solve the puzzle wins the competition and are rewarded with newly created bitcoins and all the transaction fees associated with transactions included in the block.

Proof of work is a competition among miners in which they have to find SHA-256 hash of the block that possesses a certain property. As SHA-256 output is large 256 bits hexadecimal value, the miner's goal is to find a hash value that is less than a specific hash value, called threshold value. The first miner to find this specific value wins the competition and his version of block, after verification by other miners, is added to blockchain by other nodes. A hash function produces same hash value for same input. So how does we vary hash of block, for same block as input output must be same. To variate hash value of block we vary a special field in block header, called Nonce as shown in table-2.2. To find a hash that is less than threshold value, miner change value in nonce. When nonce changes so does hash of the block. Since the output of hash function is not predictable, the miner has no choice but to increment nonce and check block hash against threshold. Each miner has different contents in the block, transactions and its

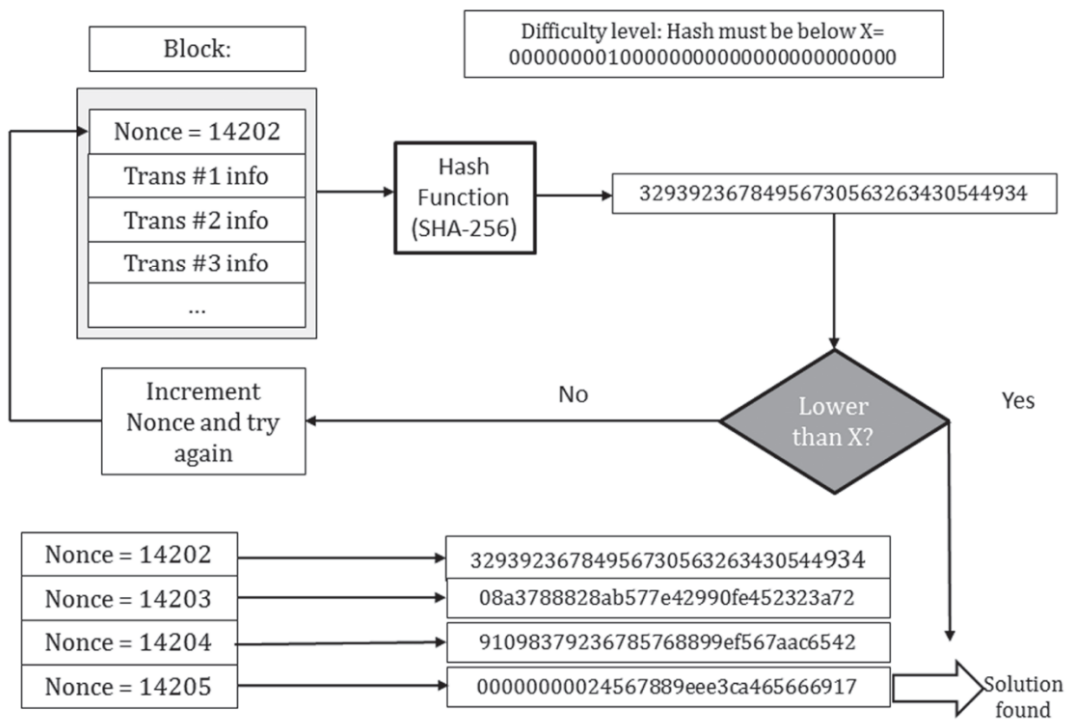


Figure 2.8: Illustration of Proof of Work. [23]

order is different and reward address is different along with other other fields, so the hash of block against a specific nonce value is different for each miner.

The bitcoin protocol automatically adjusted difficulty level of problem (threshold value) so that the miners solve a block in about 10 minutes average. If the miners is solving the puzzle in less than 10 minutes, the difficulty is increased and if the block is solved in more than 10 minutes, the difficulty is decreased.

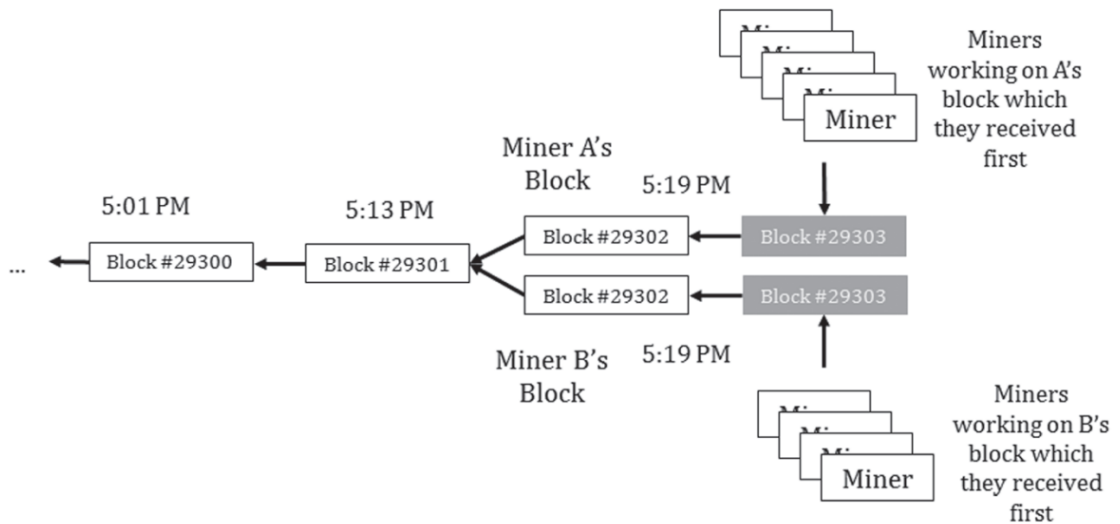
To summarize, proof of work is finding a value of nonce in about 10 minutes, so that the hash value of block is less than the threshold value dynamically set by bitcoin protocol.

### 2.3.7 Forks and Side Chains

As stated bitcoin network relies heavily on consensus to function properly. So what happens when two miners solves a block in about same time? When this happens both miners broadcast their blocks to the network. Other miners receive both blocks but their work on finding solution of next block will be based on whatever block they received

first.

Miner A and Miner B are two miners that have solved same block in about same time. Both miners, Miner A and Miner B, broadcasts their blocks to the network. Assume that 50% of miners receive block from Miner A first and other 50% receive block from Miner B first. This situation is illustrated in Figure 2.9

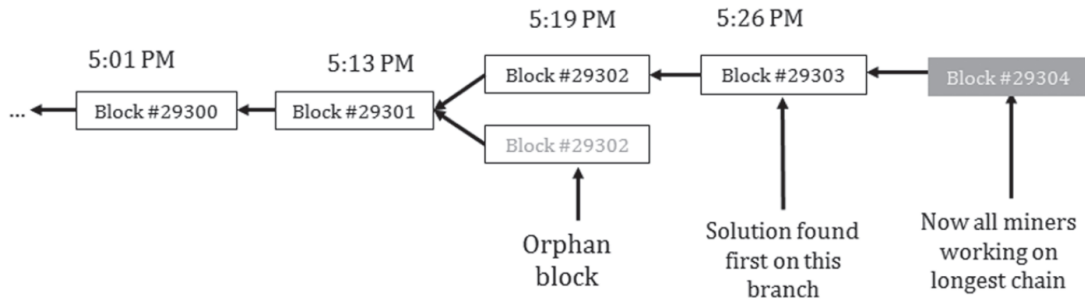


**Figure 2.9:** Illustration of chain split. [23]

At this point there are two versions of blockchain on the bitcoin network. Half miners have Miner A's block and other half have Miner B's block. Only one of these versions will survive so as to maintain consensus on the network. The survival of a version depends upon the next block to be mined. If next block is mined by a miner who is maintaining Miner A's block first, then the blockchain with Miner A's block will be the longest. Similarly if a miner maintaining Miner's B block mines the next block first, Miner B's version will be the longest. At the end the longest chain is official version of blockchain. The miners who are maintaining other version drops the block and adopt the official version. This block, dropped by miners is called orphan block. This process is illustrated in Figure 2.10.

## 2.4 Ethereum

Ethereum is a blockchain platform specifically designed to support smart contracts. Smart contracts are programs that lives on ethereum blockchain. It can send and



**Figure 2.10:** Longest chain is the official chain. [23]

receive funds. Source code of smart contract is public and immutable, so a properly designed smart contract can act as a trusted third party. Ethereum smart contracts are written solidity. Solidity is a special, JavaScript like, programming language developed for developing smart contracts on ethereum platform.

Ethereum platform has base cryptocurrency, called ether, that can be exchange on the network. Ethereum also has concept of gas. Transactions needs gas to be executed on ethereum network. Computational costs on the network are charged in form of gas and enough gas should be paid by originator of transaction, for the transaction to be executed on the network. Amount of gas to be paid depends on the amount of computation needed for a transaction. Greater the amount of computation, greater gas is charged. Smart contracts allow users to implement custom applications on ethereum network. These applications are called Decentralized Applications or DApps in short.

## 2.5 Hyperledger

Hyperledger is an umbrella of projects hosted by Linux Foundation. Hyperledger hosts several open source, enterprise grade blockchain projects which focus on providing blockchain solutions to advance cross industry collaboration. Hyperledger projects also focuses on improving performance and reliability of systems [24]. As of this writing, Hyperledger is hosting five projects, named, Hyperledger Fabric, Hyperledger Sawtooth, Hyperledger Burrow, Hyperledger Indy, and Hyperledger Iroha.

- **Hyperledger Fabric**

Hyperledger Fabric is being actively developed by IBM. It is a permissioned

blockchain and provide modular architecture. Its modular nature delivers high degree of resilience and flexibility both in design and implementation. Hyperledger fabric support chaincode (Smart Contracts on Fabric) to be written on any language. Currently support Go, NodeJS, Java and many others are in the line to be supported. Being permissioned blockchain, Hyperledger Fabric provide high degree of confidentiality, as only registered users can join the network. Hyperledger Fabric also support channels, providing medium of private transactions among multiple organizations.

Hyperledger Fabric is discussed in depth in chapter-3.

- **Hyperledger Sawtooth**

Hyperledger Sawtooth provides a modular platform for building and deployment of distributed ledgers. Sawtooth is also a permissioned blockchain with support of smart contracts. Sawtooth is highly modular and aims to keep smart contracts safe for enterprise use.

- **Hyperledger Burrow**

Hyperledger Burrow provides permissioned and modular blockchain client with interpreter partially developed for Ethereum Virtual Machine(EVM).

- **Hyperledger Indy**

Hyperledger Indy is a blockchain project that focuses on decentralized identity solutions. These decentralized digital identity is inter-operable between different domains and organizations.

- **Hyperledger Iroha**

Hyperledger Iroha is blockchain solution designed to be easily integratable with other infrastructure projects. Iroha is also designed to be simple and easy to manage.

# Hyperledger Fabric

Hyperledger Fabric is a distributed system that constitute of many components. We can broadly divide Hyperledger Fabric in three main systems.

1. Hyperledger Fabric
2. Hyperledger Fabric Certificate Authority
3. Software Development Kits (SDKs)

Despite availability of default services, Hyperledger Fabric provides what is called plug and play architecture. We can integrate any alternate service with hyperledger fabric to deploy a working hyperledger fabric network. This allows the flexibility to modify the system for a particular use case [25].

## 3.1 Hyperledger Fabric Network

In a nutshell Hyperledger Fabric blockchain is a distributed system consisting of many peers that interact with each other. Hyperledger Fabric execute special program called chaincode (smart contract in general), maintain state and ledger database and execute transactions.Chaincodes are vital element as transactions are performed by special operations on chaincode, called invocation of chaincode. Transaction needs to be endorsed by other peers in the network and only endorsed transactions are committed to ledger and have effect on state database. There are also special chaincodes, called system chaincodes, for management of functions and parameters.

### 3.1.1 Transactions

There are two main types of transactions performed on Hyperledger Fabric network.

- **Deploy Transaction**

Deploy Transaction install a new chaincode on the blockchain network. Deploy Transaction takes chaincode program as input parameter. Upon success it install new chaincode on the blockchain network.

- **Invoke Transaction**

Invoke Transaction perform an operation in context of already deployed chaincode. It performs a specific operation by executing a specific function from chaincode. Invoke transaction take a function name and arguments as input in transaction request. Upon success invoke transaction may update ledger and state data.

Deploy transaction is special case of invoke transaction on system chaincode, which installs new chaincode on the network.

### 3.1.2 State

State represent the latest state of assets/data on blockchain. State is modelled as versioned Key Value Store (KVS). State is manipulated by chaincode using *get* and *put* operations. *get(k)* reads blockchain state for key *k*, and returns its value to chaincode. *put(k,v)* add or update value *v* against key *k*.

Since transaction will be storing data persistently on blockchain, every updates are logged on the blockchain ledger. This makes it possible to track various states of assets (represented by KVS) all the way back to its inception on blockchain.

State database is maintained by peers and not by clients and orderers.

### 3.1.3 Ledger

Ledger is the verifiable record of all successful state change in hyperledger fabric. Ledger is constructed by ordering service as ordered hash chain for blocks. Each block contains array of ordered transactions. Ledger is maintained by all peers and optionally by subset of orderers. Ledger maintained by peer is called PeerLedger, in context of peer, and ledger maintained by orderer is called OrdererLedger, in context of orderer.

Orderers maintain ledger for fault tolerance. OrdererLedger have both valid and invalid transactions submitted to network, thus providing complete history of transaction on the network. Moreover, orderer may decide any time to prune ledger, since there is no point of keeping invalid transactions forever.

Peers in addition to PeerLedger may maintain vLedger. vLedger is subset of PeerLedger that contains only valid transactions. Blocks of vLedger are called vBlocks. A vBlock can have zero transaction.

### 3.1.4 Nodes

Nodes are the entities that communicate each other on the blockchain network. Node is just a logical term as a single physical service can run different types of nodes simultaneously.

There are three main type of nodes, namely client, peer, and orderer.

#### Client

Client is the endpoint where end user interact with the blockchain network. Client submit transaction to endorsers (peers) and broadcast endorsed transaction to ordering service. Client communicate with peers and orderers. Client create and thereby invoke transactions.

#### Peer

Peers receive ordered state updates from orderer in the form of blocks and maintain ledger and state database. Peers play an additional role of endorser. Peer can receive transaction proposal from client, simulate it w.r.t particular chaincode and return endorsed response to client. Chaincode may specify endorsement policy that state necessary and sufficient conditions valid transaction endorsement.

#### Orderers

Orders provide ordering service to the blockchain network. It can be implemented in many ways. For development environment, a single centralized orderer node can be



used but for production multiple, distributed ordering nodes must be used to avoid single point failure.

Orderers provides a communication channel shared by clients and peers. Clients connect to this channel and broadcast messages(transactions) on the channel which are then delivered to all peers. This channel provide atomic delivery of messages to guarantee total order in the system. Peers then include these messages in blockchain state.

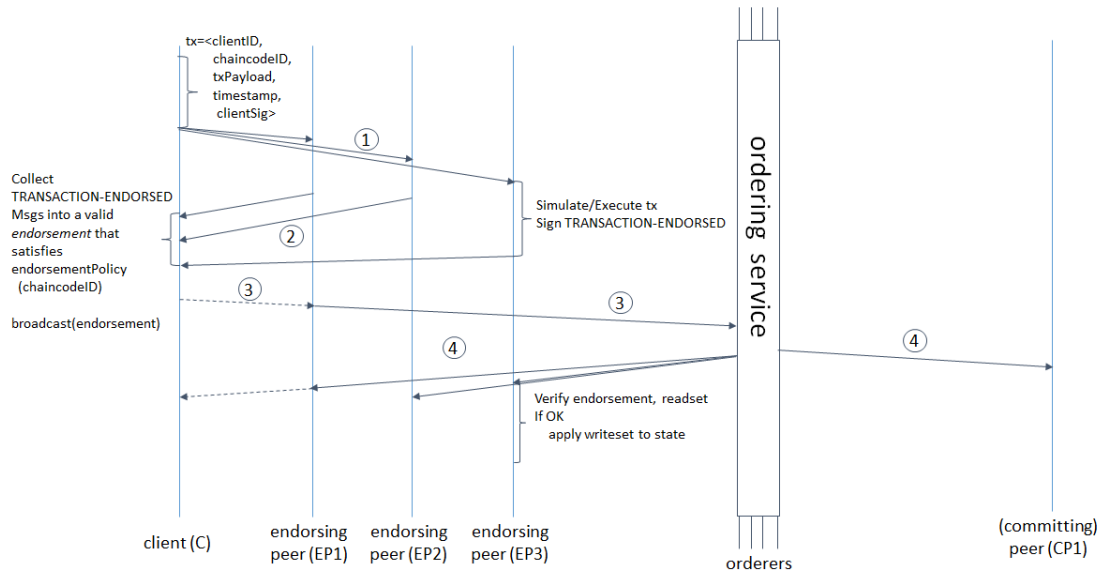


Figure 3.1: Illustration of Transaction flow in Hyperledger Fabric

### 3.2 Channel

Hyperledger is specially designed for enterprises to do business transactions on blockchain. Usually business transactions among two organizations are confidential. Putting such confidential data on a shared blockchain is not a good idea. To support such type of confidential transactions, hyperledger is designed to support "Channels". A channel is a private communication medium between two or more organization on the blockchain network. Each channel maintain a separate ledger. So if a peer is member of more than one channel will maintain multiple ledgers. Every organization that are part of a channel have defined peer members, that can access transaction data on the ledger. Channel can have multiple chaincodes. Each organization have to define an anchor peer in the channel, through which cross organization communication is performed on the channel.

Although one peer can be part of multiple channels and maintain multiple ledgers, data from one ledger can not pass to other channel.

### 3.3 Chaincode

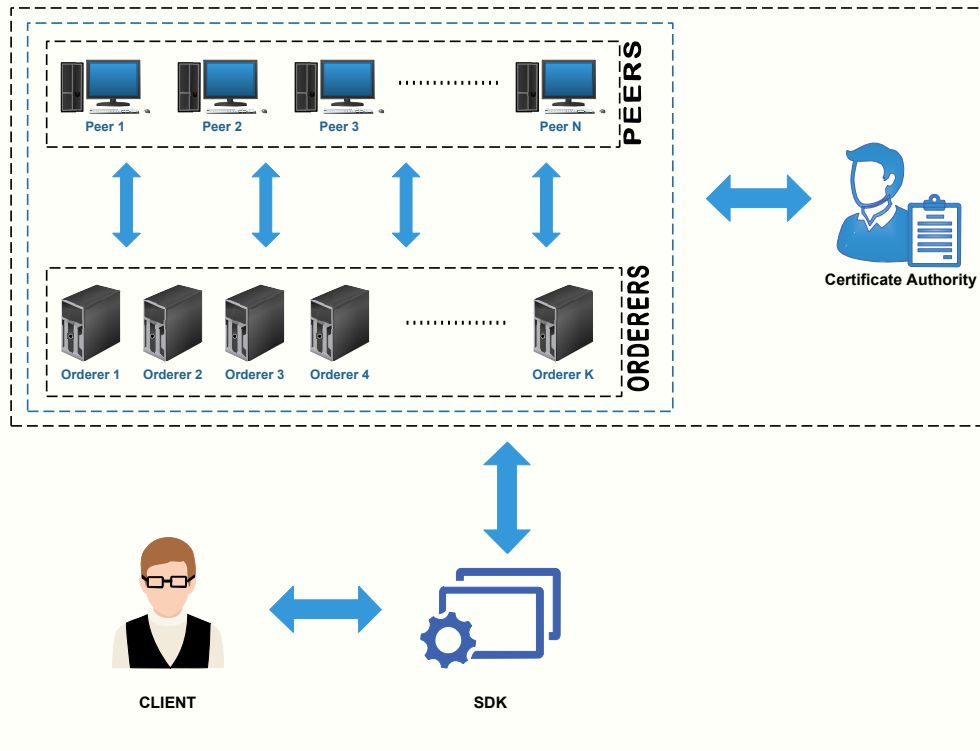
Chaincode is smart contract that implements business logic agreed upon on by member of the network. Chaincode, currently, can be written in Go and NodeJS and eventually in other programming languages such as Java. Chaincodes provides prescribed interface. Chaincode run in docker container which keep it isolated from other processes of endorsing peer. Chaincode manages transactions and ledger state on hyperledger fabric network. State for a chaincode is exclusively scoped and other chaincodes cannot access these states directly. However with appropriate permissions a chaincode can be invoked by another chaincode to access its state.

Hyperledger Fabric offers two perspectives for a chaincode, chaincode developer and chaincode operator. Chaincode developer develops chaincode for proper implementation of required business logic. Chaincode Operator package, sign, install and initialize chaincode on the network. Chaincode operator is also responsible for any subsequent management of chaincode on the network.

### 3.4 Certificate Authority

Hyperledger Fabric provide an out of the box certificate authority, called Hyperledger Fabric CA. Hyperledger Fabric CA is used to issue, revoke and manage participants certificates on the network. Beside Hyperledger Fabric CA, any certificate authority capable of issuing and managing ECDSA (Elliptic Curve Digital Signature Algorithm) certificates can be used.

A high level schematic of a typical Hyperledger Fabric network is shown in Fig 3.2.



**Figure 3.2:** High level overview of a typical Hyperledger Fabric Network.

# Blockchain Solution

In this chapter we will be building blockchain solution for issuance and verification of academic degrees.

After analyzing different blockchain platforms, we have selected hyperledger fabric as development platform for our blockchain application. Primary reasons for selecting hyperledger fabric as development platform are:

- Permissioned Blockchain
- Certificate Authority
- Organization Based Access
- Support of Channels

## 4.1 Development Environment

Hyperledger Fabric provides docker images for its different components and provide multiple options for developing chaincode. Hyperledger Fabric currently provide chaincode development support in NodeJS and Go. Hyperledger Fabric also provides Software Development Kit (SDK) for NodeJS and Java, for developing client applications.

We choose to develop chaincode in Go and NodeJS SDK for client application, as these were only choices at the starting time of this development. Pre-requisite tools for development environment for Hyperledger Fabric v1.2 are<sup>1</sup>:

---

<sup>1</sup><https://hyperledger-fabric.readthedocs.io/en/release-1.2/prereqs.html>

- Docker CE 17.06.2 or greater
- Docker Compose v1.14.0 or greater
- Go v1.10.x
- NodeJS v8.9.x
- Node Package Manager (npm) v5.6.0

For development of chaincode in Go, standard go development environment setup is required. We followed official guidelines for installation and configuration of Go development environment<sup>2</sup>.

#### 4.1.1 Installing Fabric Binaries and Docker Images

Hyperledger provides bootstrap Unix shell script that downloads all necessary binaries and docker images required for running local Fabric network. List of tools downloaded by the bootstrap script are as follow:

- **cryptogen or Crypto Generator** is used to generate cryptographic materials (x509 certificates and signing keys) for various network entities. These cryptographic materials represents identities of various network participants and allow participants to sign and verify transactions.
- **configtxgen or Configuration Transaction Generator** is used to generate some important configuration artifacts.
  - Orderer genesis block
  - Channel configuration transaction
  - Anchor peer transaction
- **configtxlator** is used to translate between JSON and Protobuf version Fabric data structures.
- **peer** has five commands which allows administrators to perform specific functions related to peer. These commands are:

---

<sup>2</sup><https://golang.org/doc/install>

- **peer chaincode** - for chaincode management related functions.
  - **peer channel** - for channel management related functions.
  - **peer logging** - for controlling and managing log stream.
  - **peer node** - for starting and checking status of a peer node.
  - **peer version** - prints version various Fabric components used by peer.
- **orderer** is used for starting ordering service.
  - **fabric-ca-client** is used for managing identities and certificates.
  - **fabric-ca-server** is used for starting certificate authority server.

### 4.1.2 Generating Artifacts

Configurations for required cryptographic certificates and signing keys are given in Appendix-A. We write these cryptographic configurations in a text file, maintaining valid *yaml* syntax. Now we feed this configurations file to *cryptogen* tool to generate required certificated and signing keys. The *cryptogen* tool will generate cryptographic materials in a separate folder, named *crypto-config*.

```
cryptogen generate --config=./crypto-config.yaml
```

Above bash command will create a folder named *crypto-config* with all the certificates and signing keys for configurations defined in *crypto-config.yaml* file as specified by *-config* flag in the command.

After creating cryptographic materials, we generated orderer genesis block using *configtxgen* tool. Configurations for orderer genesis block are given in Appendix-B and are maintained in a file named *configtx.yaml*. Now we navigated to the directory where *configtx.yaml* is located and executed following commands on the terminal.

```
export FABRIC_CFG_PATH=$PWD #Path to derectory of configtx.yaml
configtxgen -profile DocChainOrdererGenesis -outputBlock
↪ ./channel-artifacts/genesis.block
```

*configtxgen* uses *FABRIC\_CFG\_PATH* environment variable to read *configtx.yaml* configuration file and generate genesis block at path specified by *-outputBlock* option.

### 4.1.3 Channel Configuration Transaction

Now we need to create configuration transaction artifacts for the channel that we will create on our blockchain network.

```
configtxgen -profile DocChainOrgsChannel -outputCreateChannelTx
→ ./channel-artifacts/channel.tx -channelID docchain
```

This command will create a channel configuration file, named *channel.tx* in folder *channelartifacts*, alongside orderer genesis block we have created in previous section. This command uses channel id, *docchain*, which will be the name of channel we are creating. This command also uses *configtx.yaml* file by leveraging *\$FABRIC\_CFG\_PATH* environment variable.

Now we need to define anchor peer for each organization. Organizations are also defined in *configtx.yaml* file and we will generate a configuration transaction file for each anchor peer.

```
configtxgen -profile DocChainOrgsChannel -outputAnchorPeersUpdate
→ ./channel-artifacts/NUSTMSPanchors.tx -channelID docchain -asOrg
→ NUSTMSP
```

This command define an anchor peer for organization NUST. Following commands defines anchor peers for GIKI and LUMS.

```
configtxgen -profile DocChainOrgsChannel -outputAnchorPeersUpdate
→ ./channel-artifacts/GIKIMSPanchors.tx -channelID docchain -asOrg
→ GIKIMSP
```

```
configtxgen -profile DocChainOrgsChannel -outputAnchorPeersUpdate
→ ./channel-artifacts/LUMSMSPanchors.tx -channelID docchain -asOrg
→ LUMSMSP
```

These are all the configuration artifacts we need. Next we will be starting our Hyperledger Fabric Network.

#### 4.1.4 Starting Network

Network structure and configuration are discussed with detail in Appendix-C. We will start our network as described in Appendix-C. After starting the network we will be using *cli* container to create channel, add organizations to channel, define anchor peers and install chaincode on the channel. To avoid mounting each container to have access to configuration artifacts, we will mount the *cli* container to the folder that contains our configuration artifacts and orderer genesis block. We will update the the environment variables of *cli* container to run commands on behalf of different peers.

After we log in to *cli* container we will update its environment variables to run commands on behalf of *peer0* of organization *NUST* to create the channel.

#### 4.1.5 Creating Channel

After starting the network and updating environment variables of *cli* container, we will create channel named *docchain* on our network.

```
peer channel create -o orderer.nust.edu.pk:7050 -c docchain -f
→ ./channel-artifacts/channel.tx --tls --cafile
→ /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
→ ordererOrganizations/nust.edu.pk/orderers/orderer.nust.edu.pk
→ /msp/tlscacerts/tlsca.nust.edu.pk-cert.pem
```

This command returns a genesis block file, named *docchain.block*, with configurations defined in *channel.tx* file. *docchain.block* is used by peers to join this particular channel. The `-cafile` flag refers to the orderer certificate.

The channel ,*docchain*, is now created on the network.

#### 4.1.6 Joining Channel

After successful creation of the channel, we need to join all peers of all organizations to join the channel.

As we are in the *cli* container, we need to update environment variables one after other for each peer and then join the peer to the channel.



```
peer channel join -b docchain.block
```

This command will make a particular peer join the channel depending upon the environment variables of *cli* container.

#### 4.1.7 Updating Anchor Peers

At this point our blockchain network is running, channel is set up and organizations have joined the channel. Now we need to specify anchor peer for each organization. Anchor peer enables cross organization communication for an organization.

NUST:

```
peer channel update -o orderer.nust.edu.pk:7050 -c docchain -f
→ ./channel-artifacts/NUSTMSPanchors.tx --tls --cafile
→ /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
→ ordererOrganizations/nust.edu.pk/orderers/orderer.nust.edu.pk
→ /msp/tlscacerts/tlsca.nust.edu.pk-cert.pem
```

GIKI:

```
peer channel update -o orderer.nust.edu.pk:7050 -c docchain -f
→ ./channel-artifacts/GIKIMSPanchors.tx --tls --cafile
→ /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
→ ordererOrganizations/nust.edu.pk/orderers/orderer.nust.edu.pk
→ /msp/tlscacerts/tlsca.nust.edu.pk-cert.pem
```

LUMS:

```
peer channel update -o orderer.nust.edu.pk:7050 -c docchain -f
→ ./channel-artifacts/LUMSMSPanchors.tx --tls --cafile
→ /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
→ ordererOrganizations/nust.edu.pk/orderers/orderer.nust.edu.pk
→ /msp/tlscacerts/tlsca.nust.edu.pk-cert.pem
```

Note that we are using solo ordering service maintain by organization *NUST* at *orderer.nust.edu.pk*.

### 4.1.8 Installing and Instantiating Chaincode

At this point we are running a Hyperledger Fabric blockchain network with a channel joined by all organizations. The final thing these organization require is a smart contract, or chaincode, that they can invoke to perform transactions on the network. We will be installing and instantiating a our chaincode for issuing and verification degrees. By leveraging this chaincode any organization can issue degrees on this blockchain network.

```
peer chaincode install -n docchain -v 1.0 -p
  → github.com/chaincode/docchain/go/
```

This command installs the *docchain* chaincode on the peer. After installing, the chaincode needs to be initialized by some initial state.

```
peer chaincode instantiate -o orderer.example.com:7050 --tls --cafile
  → /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/
  → ordererOrganizations/example.com/orderers/orderer.example.com/
  → msp/tlscacerts/tlsca.example.com-cert.pem -C docchain -n docchain -v
  → 1.0 -c '{"Args":[""]}' -P "AND
  → ('NUSTMSP.peer', 'GIKIMSP.peer', 'LUMSMSP.peer')"
```

This command initialize the *docchain* chaincode on the channel and also specifies membership policy. The membership policy defined by this command specifies that a transaction must be validated by at least one peer from all three organizations.

At this point we have a running Hyperledger Fabric network through which on-network organizations can issue and verify degrees through command line. To make our application more user friendly, we have developed a front end web interface for our network through which users can leverage our blockchain application. Next chapter discusses development of chaincode and and web interface for our blockchain application.

# Development and Testing of Blockchain Solution

## 5.1 Participants and Use Case

We designed architecture of our network in light of Hyperledger Fabric. Participants of our network are assigned specific roles as supported by Hyperledger Fabric. Each University is viewed as an organization with a certificate authority. Each department/school of a university is department inside organization where at least one peer node is maintained. Although this is not a necessity, but its included to strengthen the network. As mentioned each university is viewed as an organization in our blockchain network, so a university has to maintain few peers, multiple orderers and a certificate authority. Function and purposes of each of these components is discussed with details in chapter-3. Each university has to join a common channel. The common channel is used by universities to issue degrees that can be viewed by other universities and organizations for verification purposes.

The main players in the network are:

### 5.1.1 Governing Body

Network is started by the governing body with creation of channel with proper chaincode installed and initiated. Being a permissioned blockchain network, governing body can add and remove organizations to and from the network. This allow governing body to

stop unrecognized organizations from joining the network, which is the main reason for choosing hyperledger fabric as our blockchain framework.

### 5.1.2 Organizations (Universities)

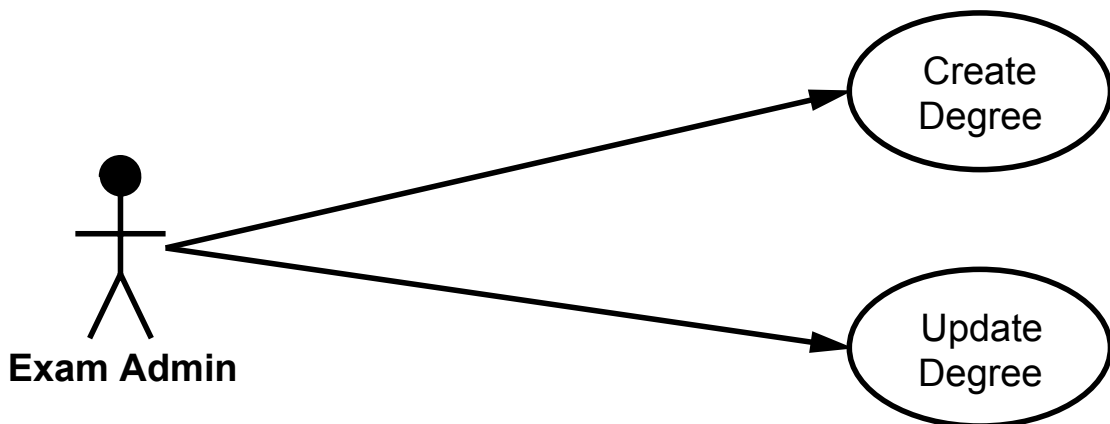
Every organization has a certificate authority, which issues signed certificates to other participants of the university. These participants can be university admin, other intermediate certificate authorities or department's examination section. For simplicity of our network we have only considered examination section of each department, where degrees for corresponding department are issued.

#### University Admin

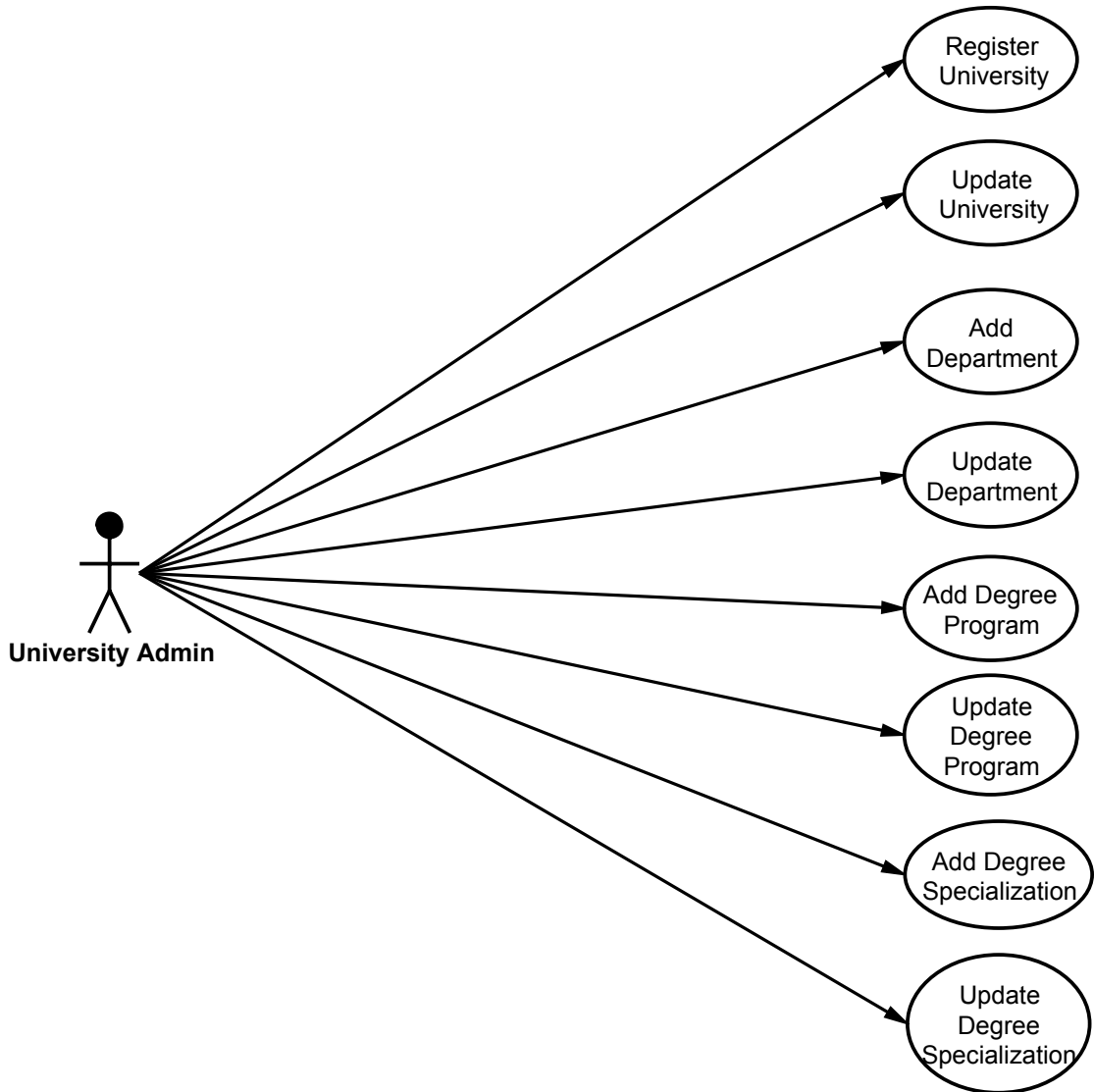
University Admin uses client certificate signed by root certificate of organization with an attribute "university\_admin=true". University is registered through this certificate on blockchain. After registration, this certificate is used to register departments, programs and specializations offered by each department. This structure is later used by Exam Admin to issue degrees.

#### Exam Admin

Exam Admin uses client certificate signed by root certificate of organization with two specific attributes, "exam\_admin=true" and "department=Department Name". The department attribute used in this certificate allows the admin to issue degrees for a specific department.



**Figure 5.2:** Use Case Diagram: Exam Admin.



**Figure 5.1:** Use Case Diagram: University Admin.

### 5.1.3 Degree Signers

After a degree is created by exam admin, it needs to be signed by three official bodies of the university. After successful signing by all three officials, degree is viewed as verified degree. These officials are:

- Department Principal/Dean
- University Registrar
- University Rector/Vice Chancellor

All three signers have client certificates with special attributes. Principal/Dean can only

sign degrees issued by his/her department. After Principal/Dean sign the degree, it is shown to Registrar in pending list for signing. After signing of registrar, degree is added to pending list for signing of Rector/Vice Chancellor, where he/she can sign degree. After signing of Rector/Vice Chancellor, a degree gains a status of valid degree on the blockchain. Use case diagrams for Principal, Registrar and Rector are given.

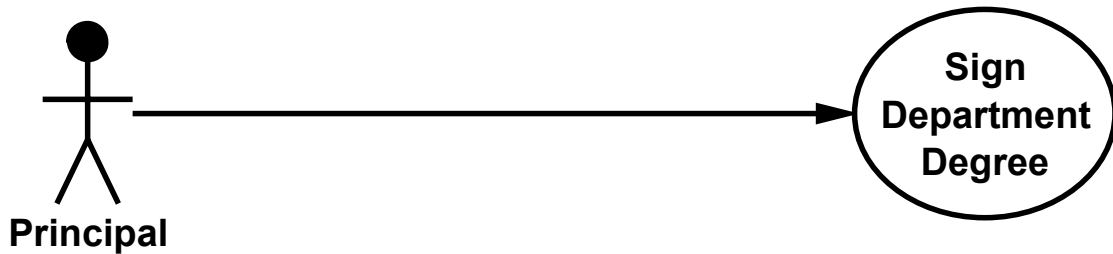


Figure 5.3: Use Case Diagram: Principal.

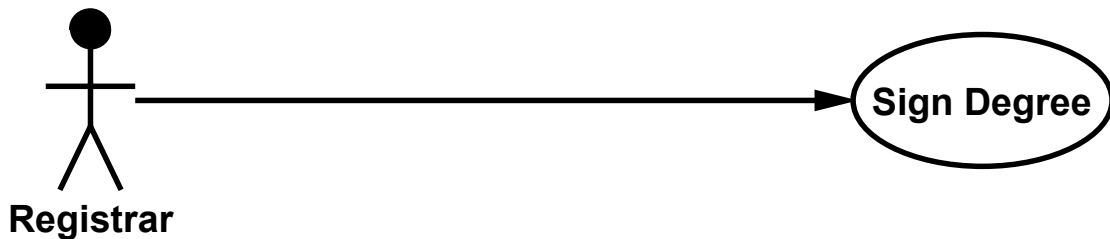


Figure 5.4: Use Case Diagram: Registrar.

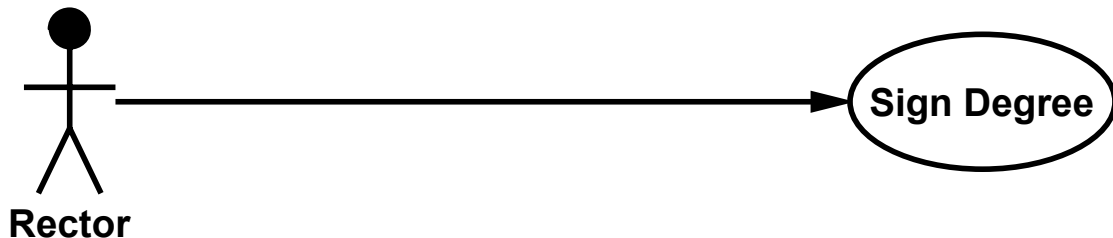


Figure 5.5: Use Case Diagram: Rector.

#### 5.1.4 Employer/Verification Point

This is also a client certificate signed by either a registered organization or governing body. This certificate has "verifier=true" attribute. This certificate provides access to degree verification portal.

## 5.2 Chaincode Development

Smart Contracts are called chaincode when working with Hyperledger Fabric Network. Chaincode is programming logic deployed on Hyperledger Fabric network. Chaincode implements two main methods, `Init()` and `Invoke()`. `Init` method is executed when an initialize or upgrade operation is performed on chaincode. `Invoke` method is executed when an invoke or query operation is performed on the chaincode. Interface of *Init* and *Invoke* methods are as follow.

```
func (asset *CCAsset) Init(shim.ChaincodeStubInterface) peer.Response
```

```
func (asset *CCAsset) Invoke(shim.ChaincodeStubInterface) peer.Response
```

These are the entry points to chaincode and any operation that we need the chaincode to perform will be triggered through these entry points. *Init* is only called when the chaincode is initialized or upgraded and is mainly used to initialize initial states of assets. *Invoke* is mainly called on *query* or *invoke* operation. In *query* operation any updates in the state database or ledger is only simulated to compute final response and do not result in any real updates of state database or ledger. While in *invoke* operation, any updates to the state database are reflected throughout the channel and changes are appended to ledger as a transaction. In this section we will be discussing our approach to the development of chaincode for issuing and verification of academic degree.

Chaincode structure is illustrated in class diagram Figure 5.6. As evident from class diagram (Figure 5.6), complete structure for a degree program is recorded on blockchain.

### 5.2.1 University Structure

A university is register by using Membership Service Provider ID (MSPID) as key. So any certificate issued by a particular MSP will be treated as clients certificate of same university. This functionality can be achieved by *MSPID* attribute in University structure.

There are two methods associated with University structure, *AddNewMSP()* and *GetUniversity()*. When university organization is added to channel, the university admin has to register the details of university on the blockchain. *AddNewMSP()* method is used to register

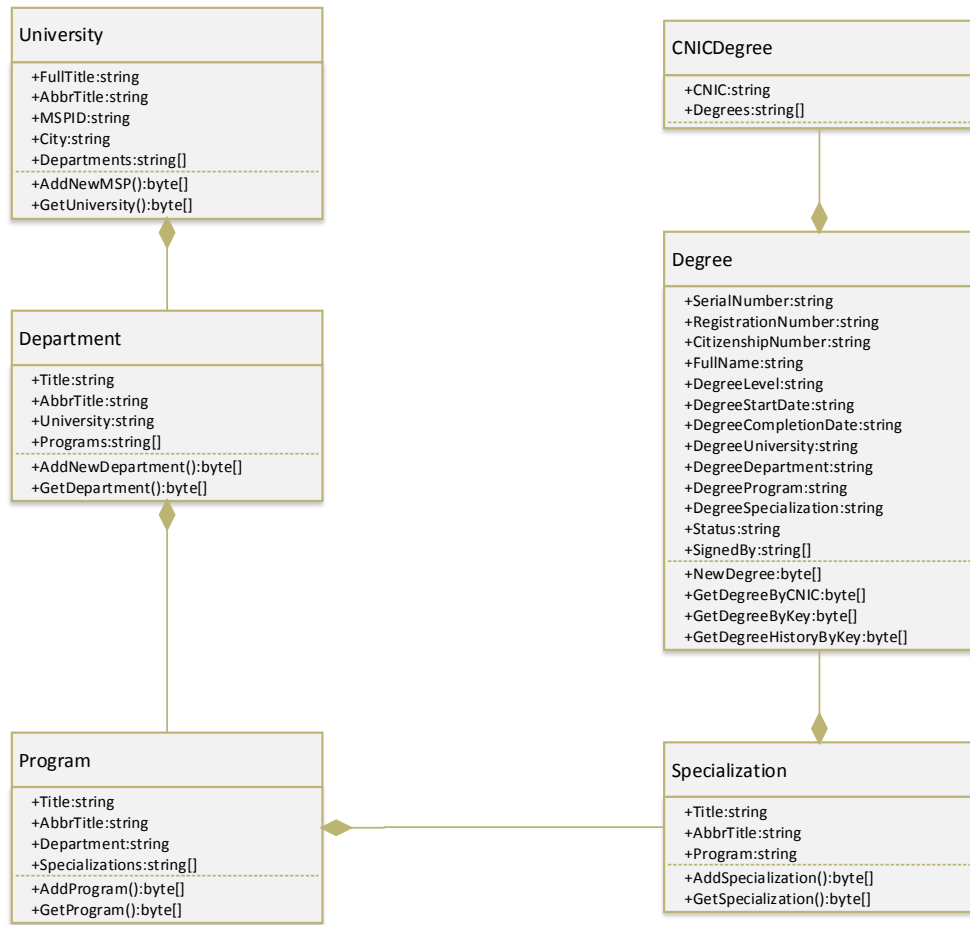


Figure 5.6: Chaincode Class Diagram

university for blockchain. *GetUniversity()* return the details of the university by using MSPID of the client certificate.

University structure also stores the ledger keys of all its departments in the *Departments[]* slice.

### 5.2.2 Department Structure

After university is registered on blockchain, university admin can manage departments of the university. University admin is also required to add the departments of the university on the blockchain. Attributes for a department are given in class diagram, fig-5.6.

There are two methods associated with Department structure, *AddNewDepartment()*



and *GetDepartment()*. *AddNewDepartment()* is used to add new department in the university and is only allowed to be executed by university admin. *AddNewDepartment()* also updates *Departments[]* slice in University structure by adding ledger key of new added department. Department structure also keep record of its university by maintaining ledger key of university. In this way a direct relation is maintained between a university and department.

### 5.2.3 Program Structure

A department may offer different academic programs. So after adding a department, programs can be added to that department. Attributes for a degree program are given in class diagram, fig-5.6. After a department is added, university admin can add programs offered at particular department.

Program structure has two methods, *AddProgram()* and *GetProgram()*. *AddProgram* adds new program for a particular department and add program ledger key to *Programs[]* slice of Department structure. Program structure also maintains an attribute *Department*, to keep record of the department it belongs to. In this way direct relation is maintained between a department and program. An indirect relation between university and program also exists through department.

### 5.2.4 Specialization Structure

A particular academic program can offer many specialization streams. After adding a program, university admin can add specialization streams to that particular program. Attributes for specialization are given in class diagram, fig-5.6.

Specilization structure has two methods, *AddSpecialization()* and *GetSpecialization()*. *AddSpecialization()* adds new specialization to program by adding specialization details on blockchain and appending ledger key of specialization to *Specializations[]* slice of Program structure. In this way specialization maintains a direct relationship with Program, and indirect relationship with Department and university. *GetSpecialization()* return details of specialization.

### 5.2.5 Degree Structure

Degree structure provides interface to store degree on blockchain. Attributes related to a degree are given in class diagram, fig-5.6. Degree structure provides four methods.

- *NewDegree()* adds degree to blockchain. A degree can be added by exam admin. Degree structure has attributes to maintain direct relationship with university, department, program and specialization. A degree, when created, is required to be signed by principal, registrar and rector. *Status* attribute maintains current status of degree.
- *GetDegreeByCNIC()* receives CNIC number as input and return all degrees earned by the CNIC holder.
- *GetDegreeByKey()* receives degree ledger key and returns degree against that particular key.
- *GetDegreeHistoryByKey()* receives degree ledger key and return complete history of transaction for the degree registered against that particular ledger key.

### 5.2.6 CNICDegree Structure

CNICDegree Structure is used to maintain degree records against CNIC. This structure provide quick access to all degrees earned by an individual.

## 5.3 Development of Web Application

Web Application is developed using NodeJS server that will be running locally on client's computer. NodeJS server leverages Hyperledger Fabric NodeJS Software Development Kit (Node SDK) to access blockchain network with ease. Moreover client can allow other clients access to use NodeJS server.

For better structure and arrangement of web application we are using ExpressJS, a NodeJS framework. The main routes (URL access points) of the application are divided in following sections.

### 5.3.1 University Admin Routes

These routes are prefixed by *university\_admin*. Routes in this section can only be accessed by university admin and provide interface for managing university, departments, programs and specializations.

### 5.3.2 Exam Admin Routes

These routes are prefixed by *exam\_admin*. Routes in this section can only be accessed by exam admin and provide interface for creating degree.

### 5.3.3 Signing Routes

These routes are prefixed by *principal, registrar and rector*. These routes can be accessed by principal, registrar and rector respectively to view and sign degrees.

## 5.4 Web Application Interface

Client can access application at a particular address and port that NodeJS server is running on. At start client is presented with a Login form that require only username. The web application automatically search for certificates at predefined path. If web application found certificate, it determines user type associated with particular certificate and route user to dashboard for that user type. From dashboard a user can perform various operations as intended.

### 5.4.1 University Admin

University Admin registers and manages university on the blockchain. Firstly, university admin register a university on the block chain. Registering of university is a one time operation. Flow of registering a university is illustrated in fig-5.7. After university is registered, university admin can add departments, programs offered at these departments, and specializations offered at each program. Sequence diagram illustrated in fig-5.8 show the flow of adding a department, fig-5.9 shows flow of adding a program to a specific department and fig-5.10 shows flow of adding a specialization to a specific program.

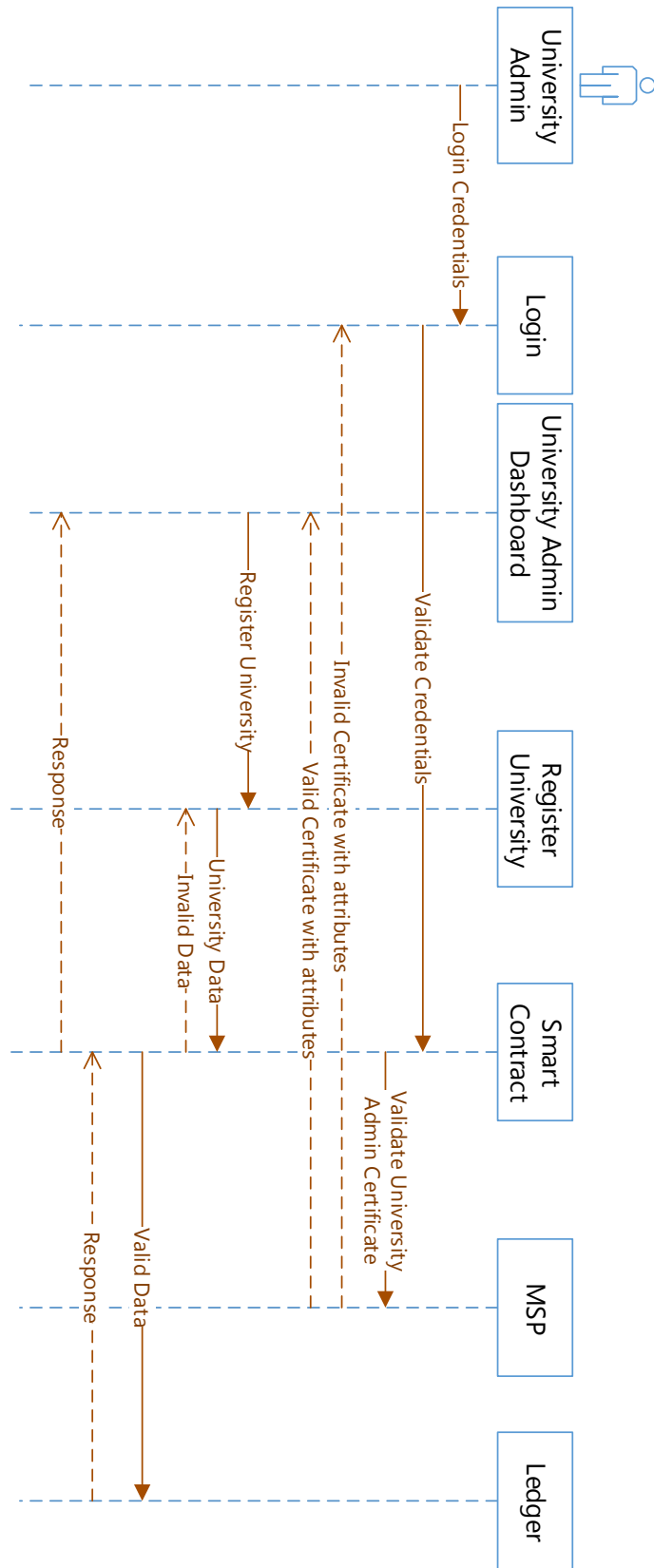


Figure 5.7: Sequence Diagram - Register University

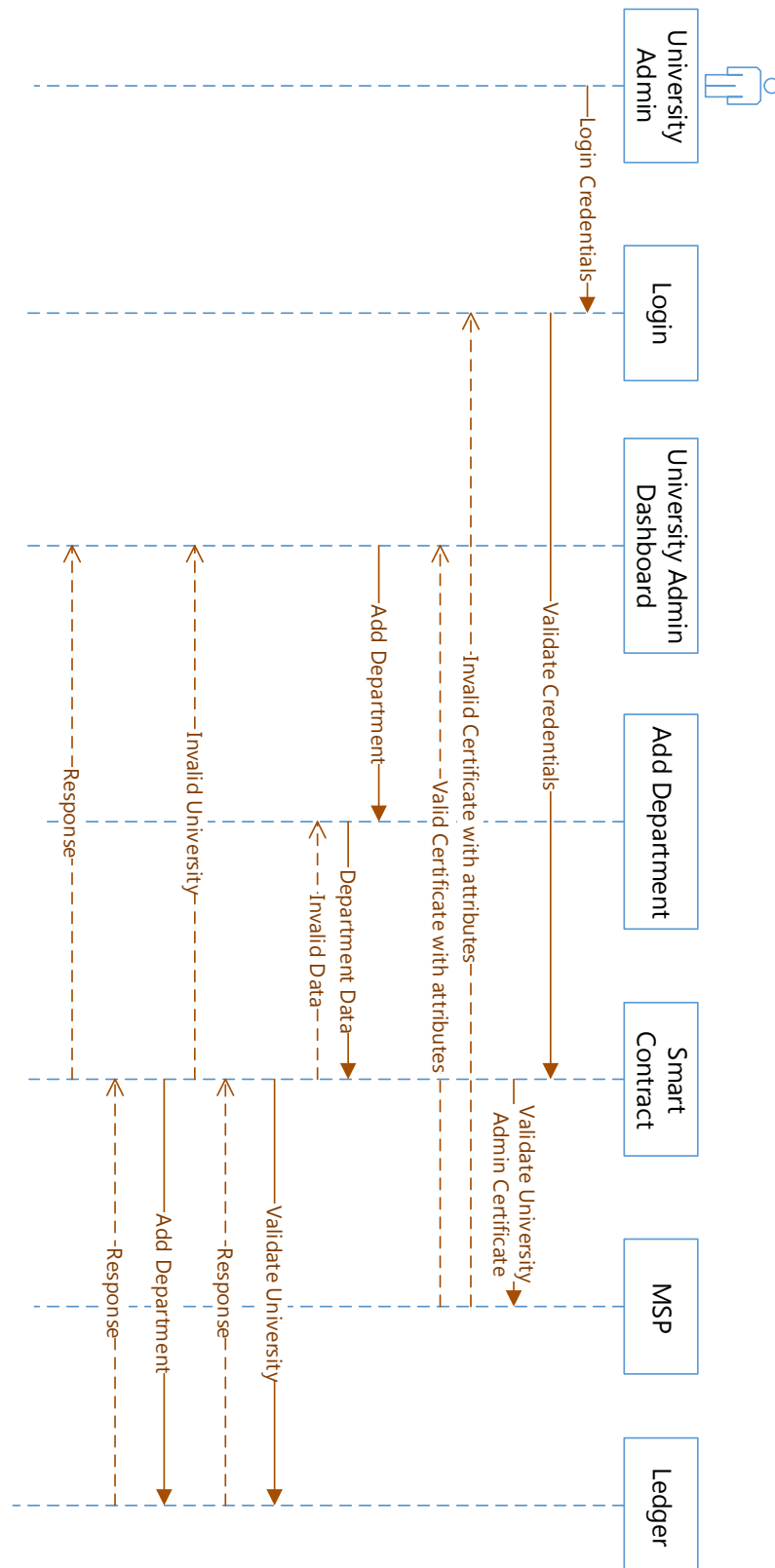


Figure 5.8: Sequence Diagram - Add Department

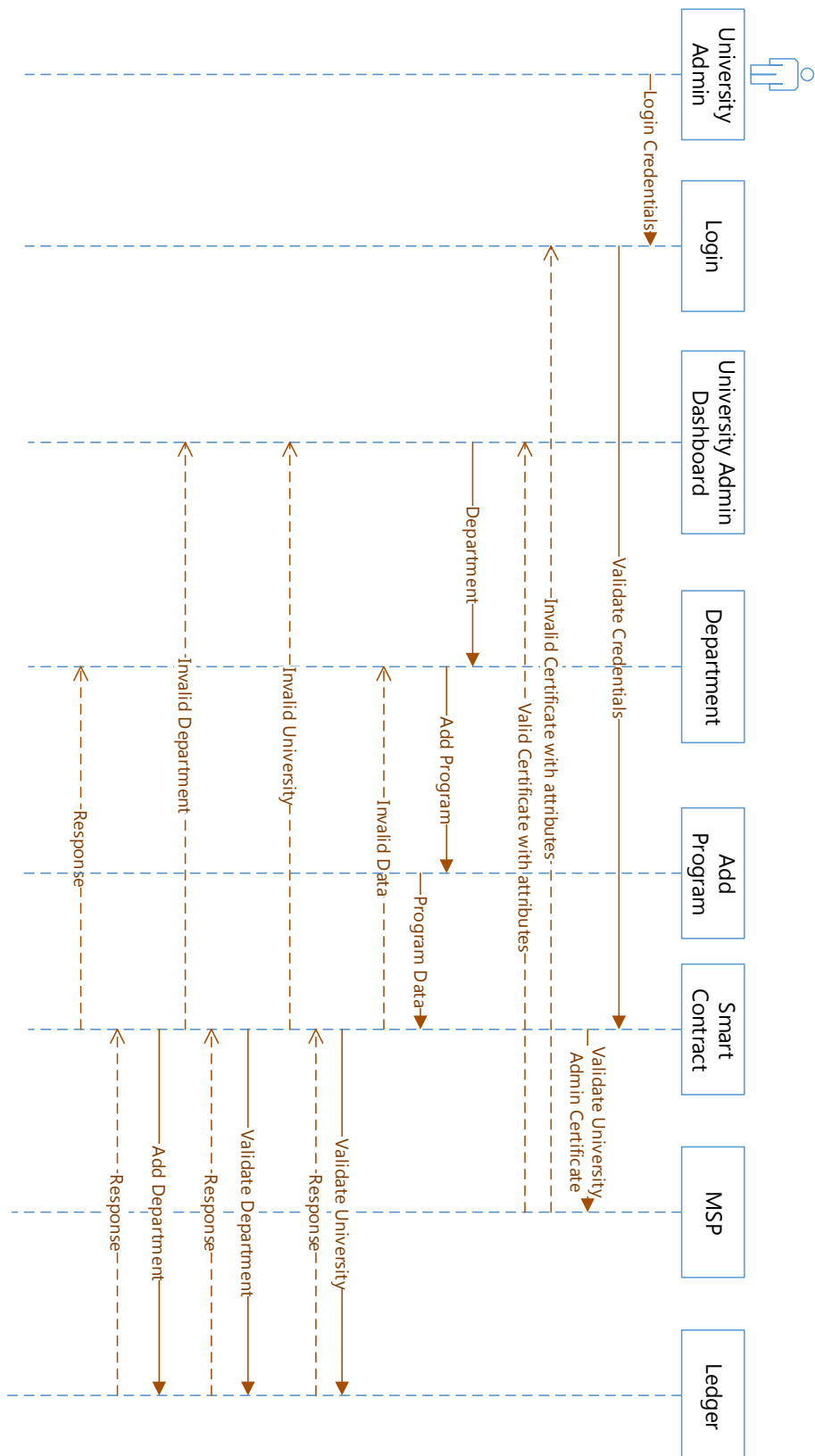
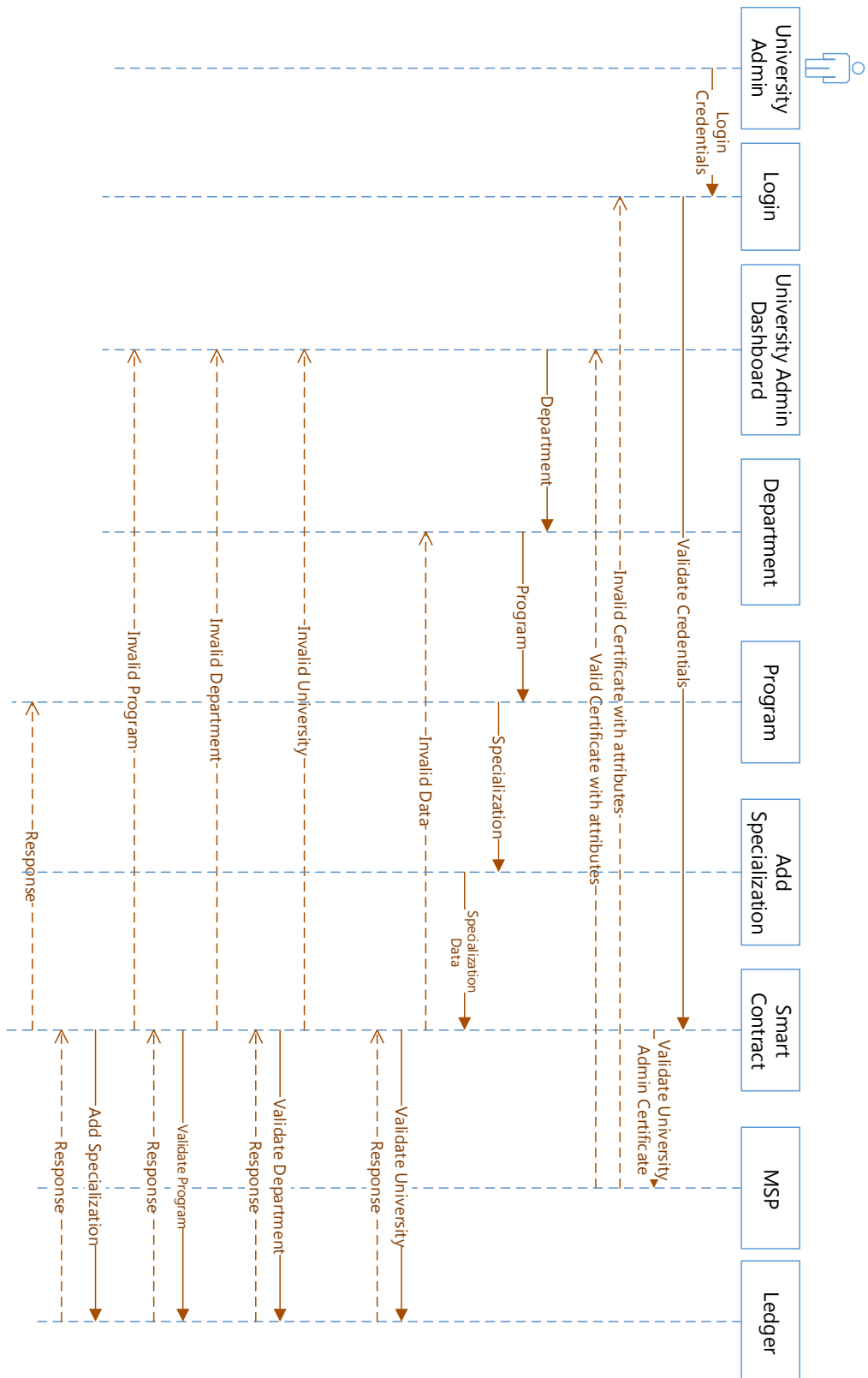


Figure 5.9: Sequence Diagram - Add Program



**Figure 5.10:** Sequence Diagram - Add Specialization

### 5.4.2 Exam Admin

As stated exam admin can view and create degrees. The process of creating degree is illustrated in sequence diagram in fig-5.11 and process viewing a degree is illustrated in fig-5.12.

### 5.4.3 Degree Signing

As per our application design, after a degree is created by exam admin it needs to be signed by Principal, Registrar and Rector to gain a verified status on blockchain. The process of signing degree by Principal is illustrated in sequence diagram fig-5.13, signing degree by Registrar is illustrated in sequence diagram fig-5.14 and signing degree by Rector is illustrated in sequence diagram fig-5.15.



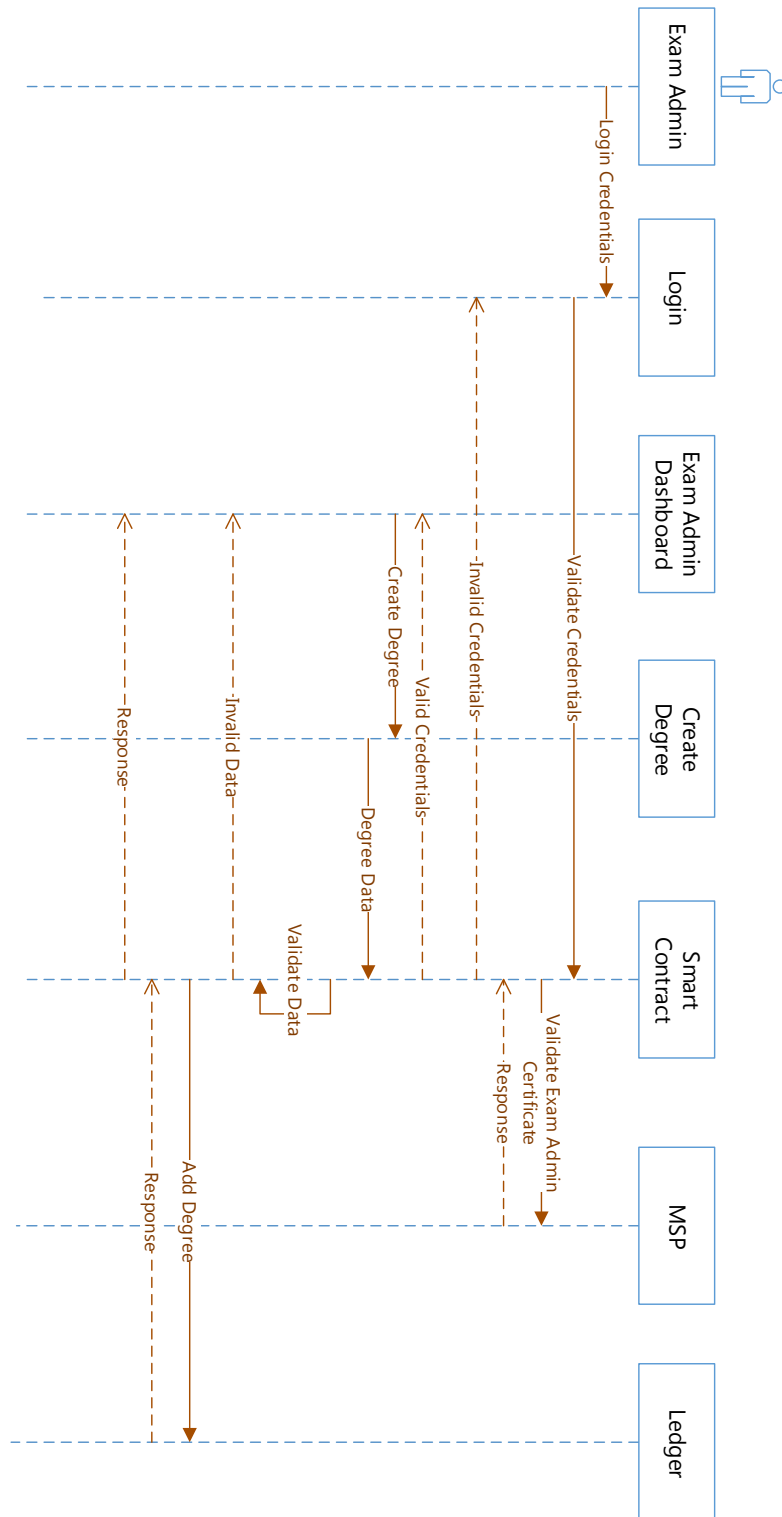


Figure 5.11: Sequence Diagram - Add Degree.

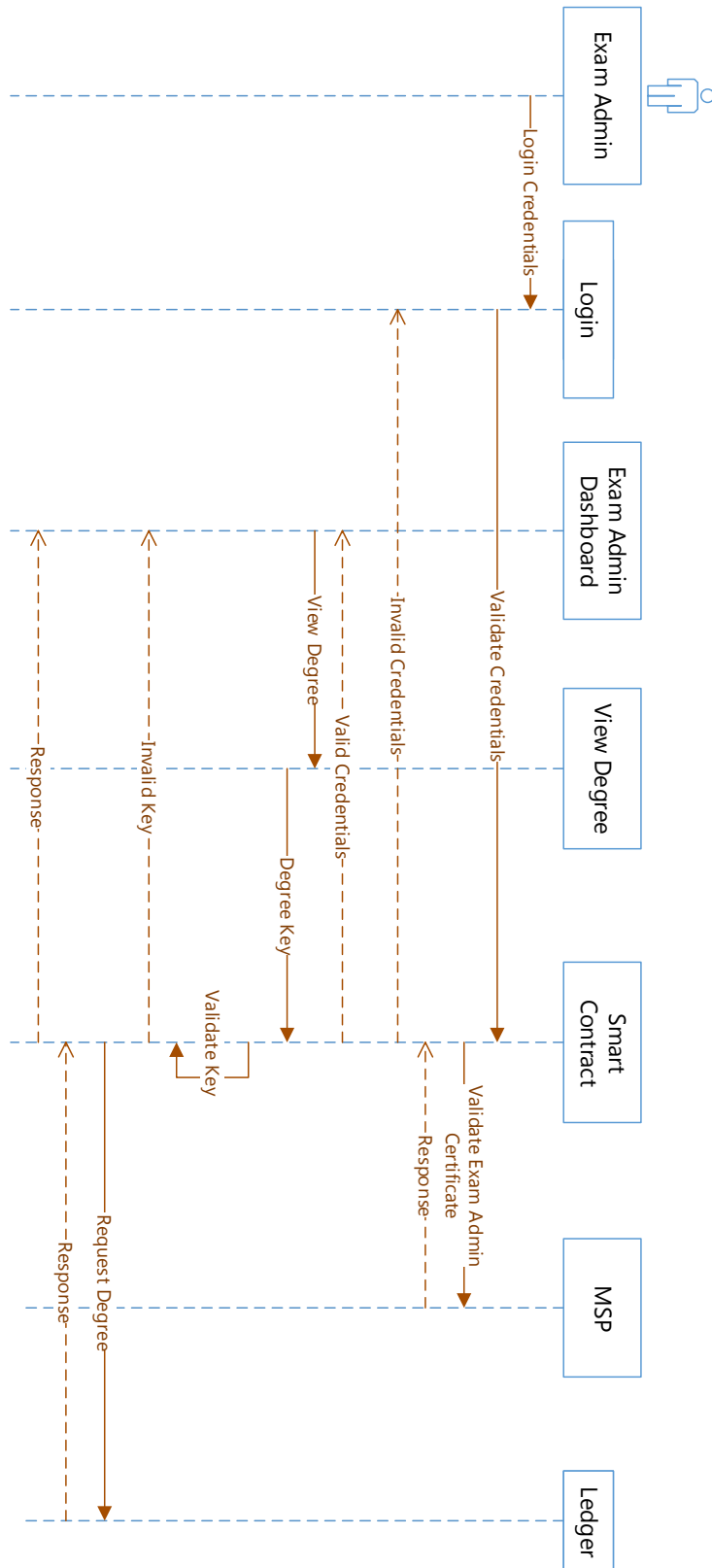


Figure 5.12: Sequence Diagram - View Degree.

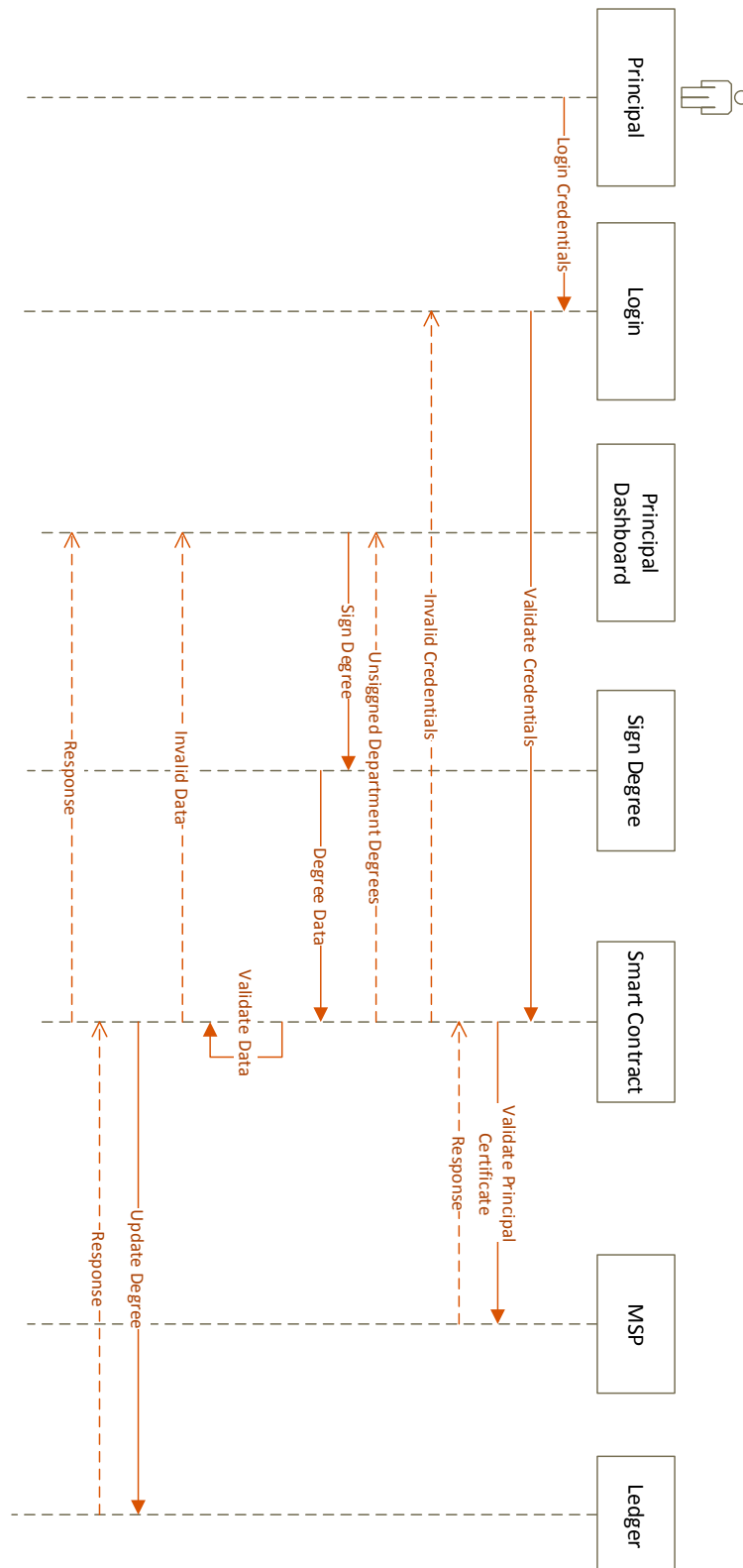


Figure 5.13: Sequence Diagram - Signing Degree by Principal.

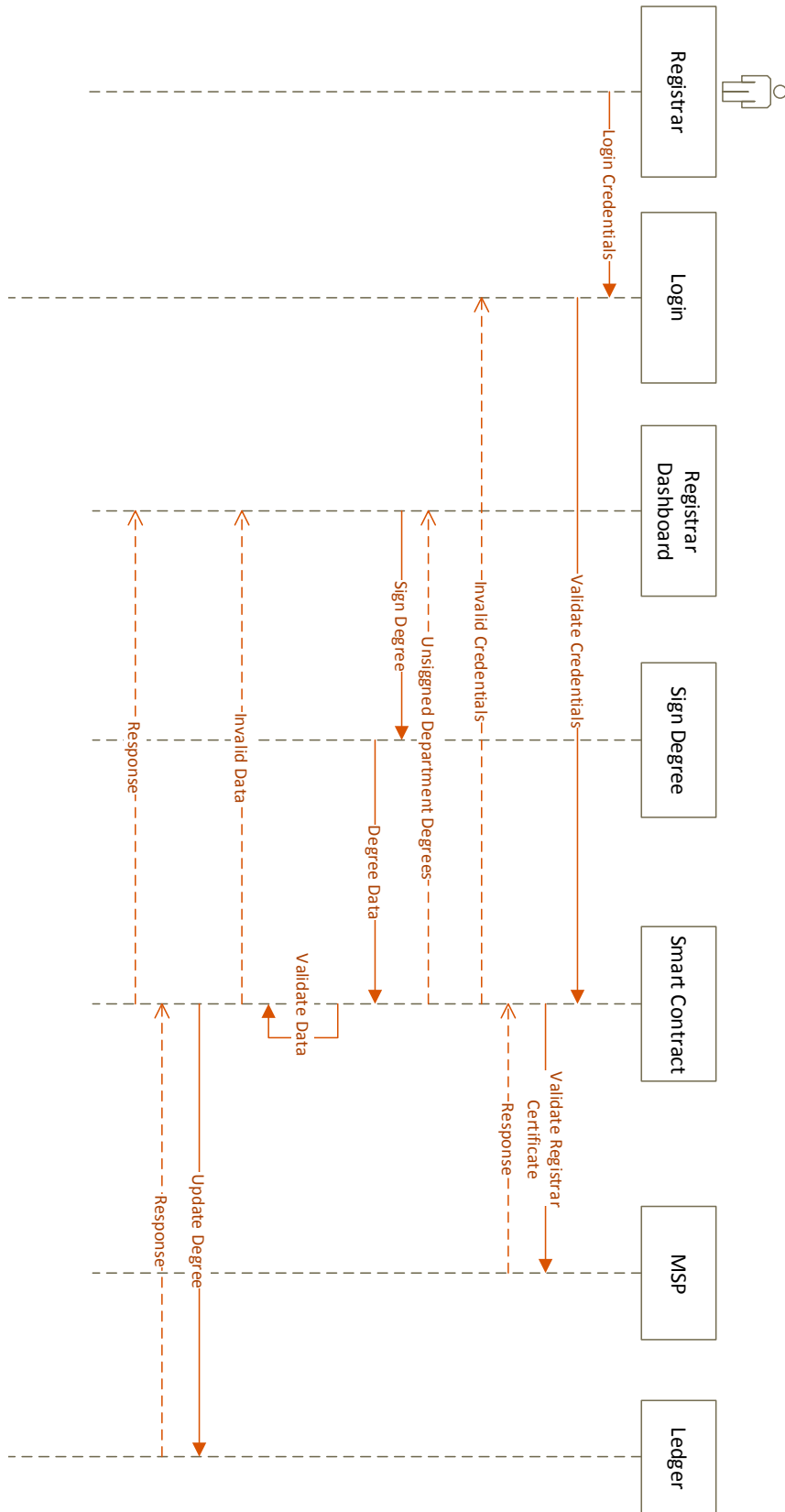


Figure 5.14: Sequence Diagram - Signing Degree by Registrar.

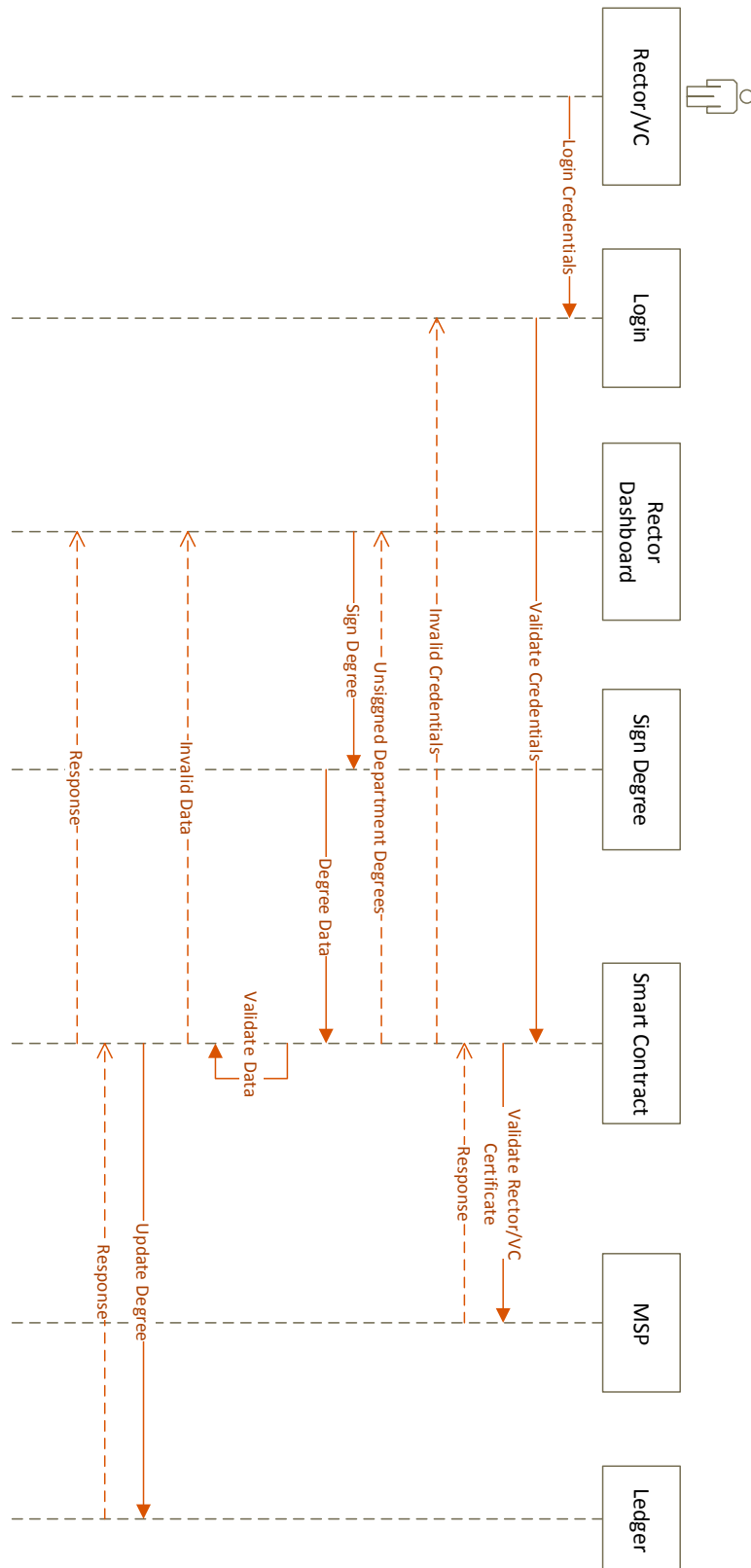
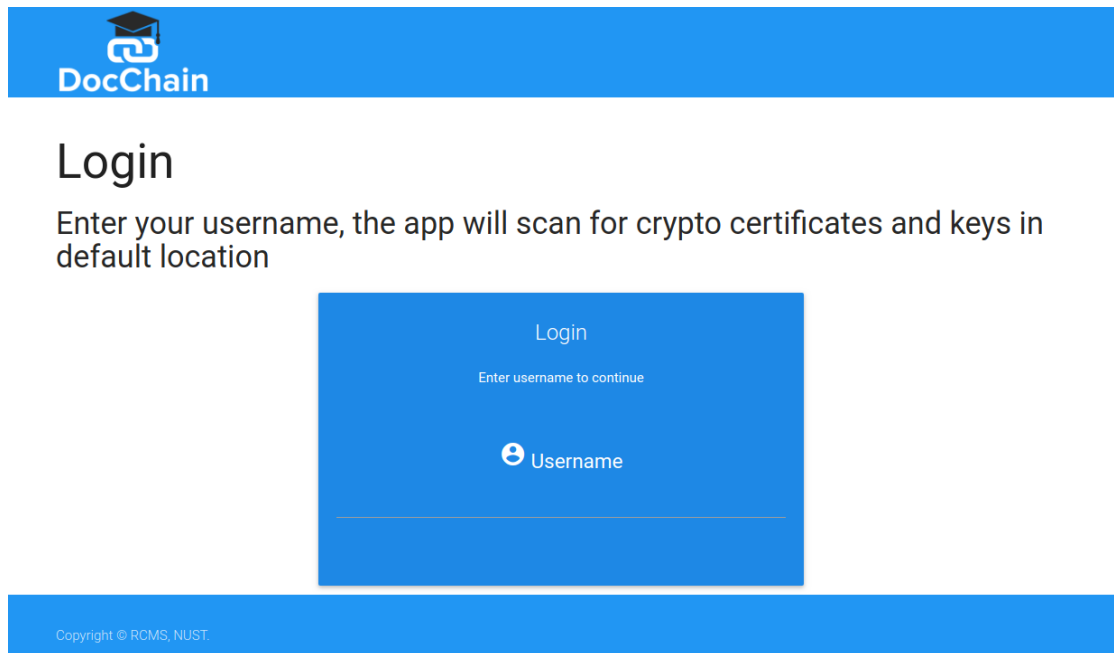


Figure 5.15: Sequence Diagram - Signing Degree by Rector.

## 5.5 Testing of Blockchain Application

When client opens the web application, he is automatically redirected to login page. On login page client needs to enter his username and local node server will automatically load required cryptographic certificates from a specified location on local computer. Snapshot of login view is given in fig-5.16.



**Figure 5.16:** Snapshot - Login View.

### 5.5.1 University Admin

We will be testing our application with university admin certificate first to register university and setup department, program and a specialization.

#### Login

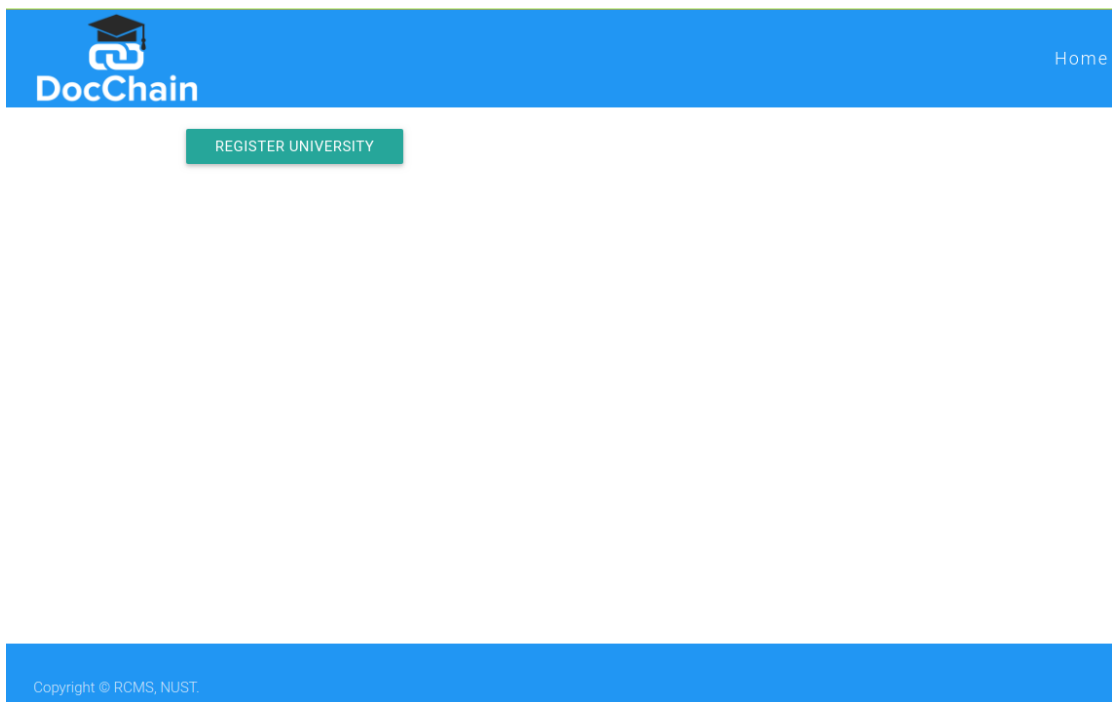
We are logging university admin for the first time. The current status of the system are listed as follow.

**Username:** user11

**User Type:** University Admin

**Special Attribute:** university\_admin=true

**University Status:** Not Registered



**Figure 5.17:** Snapshot - University Admin Dashboard at first login.

### **Register University**

University admin has logged in for the first time and his university is not registered on the blockchain. In this scenario university admin is presented with "Register University" button (fig-5.17) which provides interface for registering a university.

Fig-5.18 shows interface to input university details. Once university is registered, admin dashboard must show "Add Department" button and not "Register University" button, fig-5.18.

The screenshot shows a web form titled "Register University". At the top, a message states: "There is no University mapping to your MSPID. Enter your University Details to register one." The form contains three input fields: "University Name" with the value "National University of Science and Technology", "Name Abbreviation" with the value "NUST", and "City" with the value "Islamabad". A "REGISTER" button is located at the bottom right of the form.

**Figure 5.18:** Snapshot - University Admin, registering university.

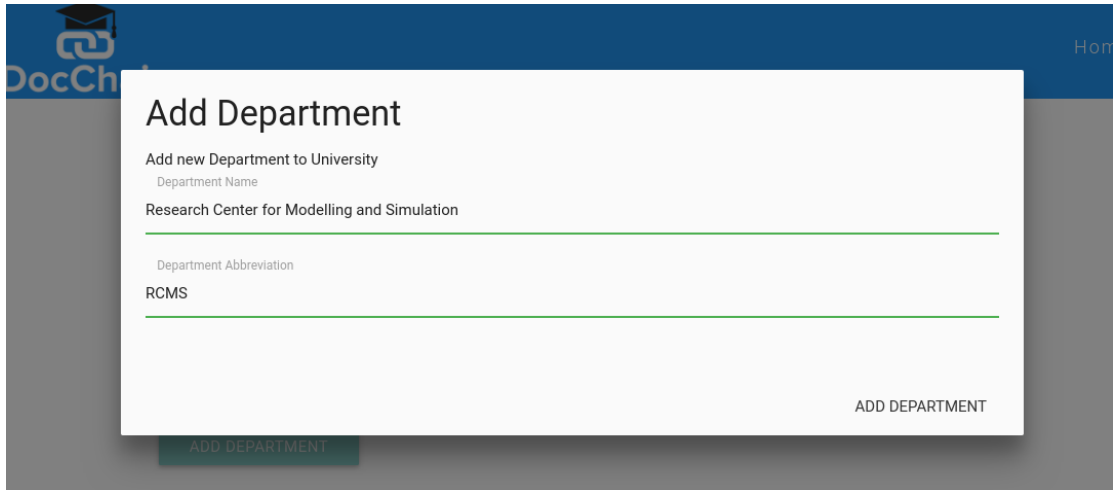
The screenshot displays the "University Admin Dashboard" for DocChain. The header includes the DocChain logo and a "Home" link. The main content area shows the registered university details: "National University of Science and Technology", "NUST", and "Islamabad". Below this information is a green button labeled "ADD DEPARTMENT". The footer contains the text "Copyright © RCMS, NUST."

**Figure 5.19:** Snapshot - University Admin Dashboard, after university is registered.



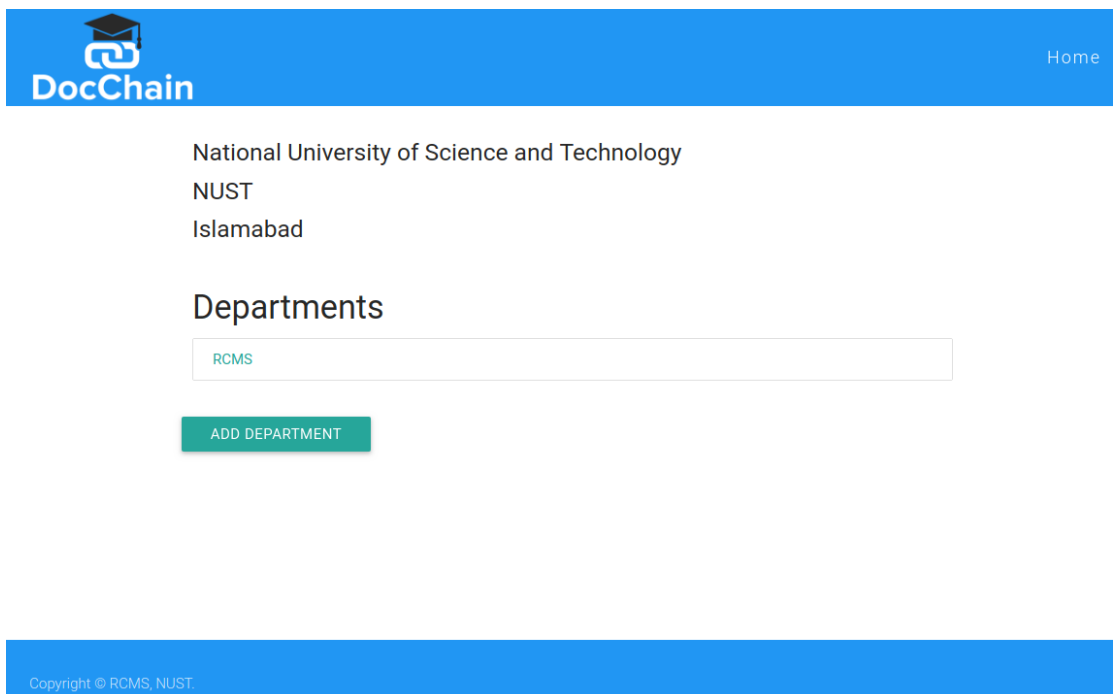
## Add Department

After university is registered, university admin can add departments by using "Add Department" button. "Add Department" button opens interface to input department details, as shown in fig-5.20.



The screenshot shows a modal window titled "Add Department" with the subtitle "Add new Department to University". It contains two input fields: "Department Name" with the value "Research Center for Modelling and Simulation" and "Department Abbreviation" with the value "RCMS". A green "ADD DEPARTMENT" button is located at the bottom right of the modal. The background shows a blurred view of the DocChain interface with a blue header and a "Home" link.

**Figure 5.20:** Snapshot - University Admin, Department details input.



The screenshot shows the "Departments" section of the DocChain interface. The header includes the DocChain logo and a "Home" link. Below the header, the university name "National University of Science and Technology" and "NUST Islamabad" are displayed. The "Departments" section features a search bar containing "RCMS" and a green "ADD DEPARTMENT" button. The footer contains the text "Copyright © RCMS, NUST."

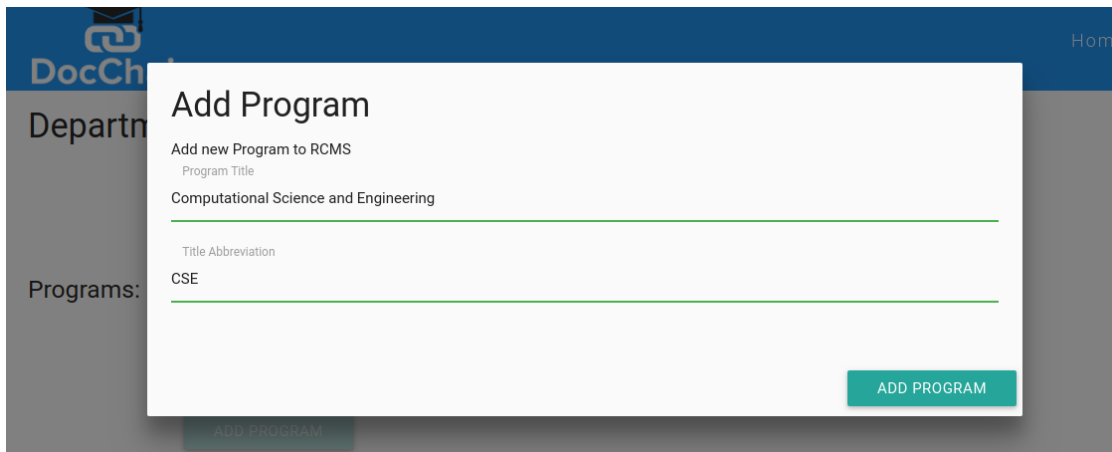
**Figure 5.21:** Snapshot - University Admin, Departments list.

After a department is added, it appears in list of departments at dashboard of university

admin, fig-5.21. Each department is linked to a department page, where Programs offered by department can be added.

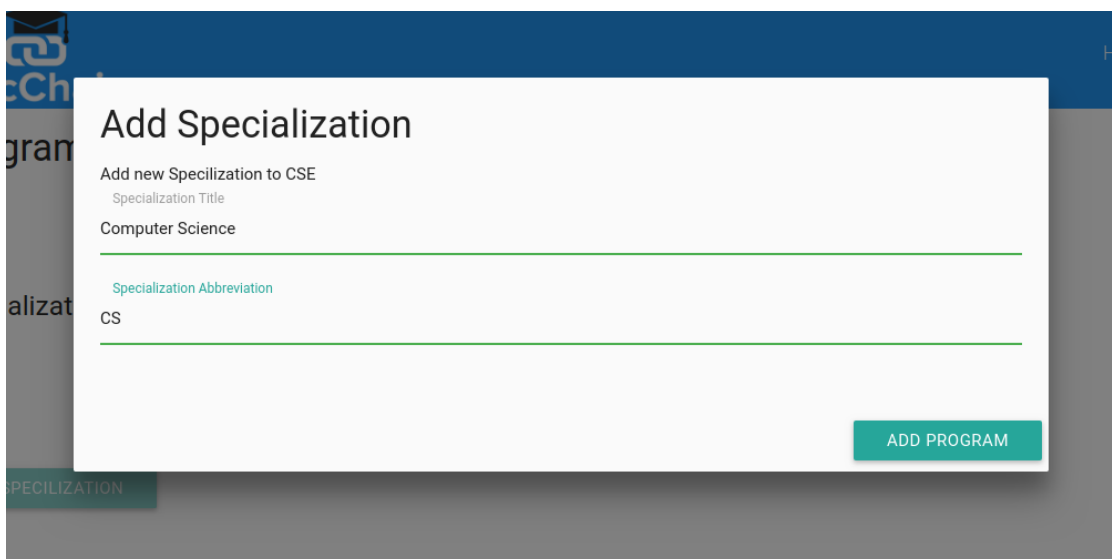
### Add Program

Department page provide "ADD PROGRAM" button, which opens interface to input program details, fig-5.22 and add particular program to the department.



**Figure 5.22:** Snapshot - University Admin, Add Program.

After a program is added, it appears in list of programs on the department page as link to program page.



**Figure 5.23:** Snapshot - University Admin, Add Specialization.

### Add Specialization

Program page provides "ADD SPECIALIZATION" button which open interface to input specialization details, fig-5.23. After a specialization is added it appears in list of specialization on program page.

#### 5.5.2 Exam Admin

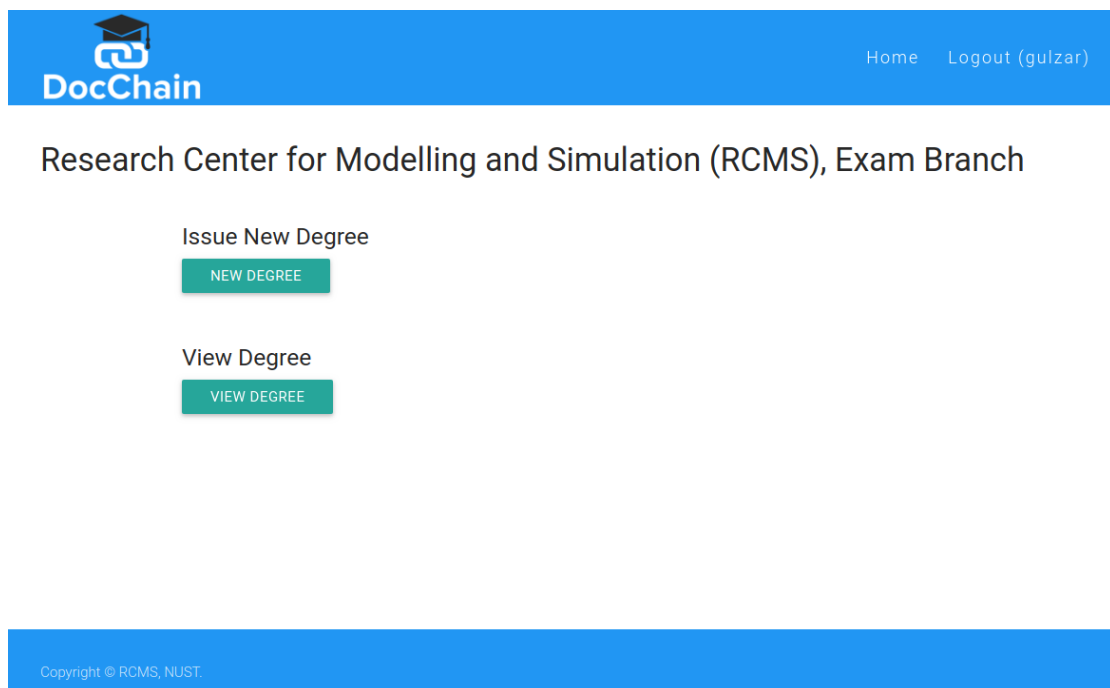
After university admin has setup basic platform, exam admin can now add degrees. Every department has different exam admin, so for an exam admin to add degree, his department must be registered by university admin. Since we have added RCMS department, we will be using RCMS exam admin certificate to log in. Specification are as follow.

**Username:** gulzar

**User Type:** Exam Admin

**Special Attributes:** exam\_admin=true, department=RCMS

**Department Status:** Registered

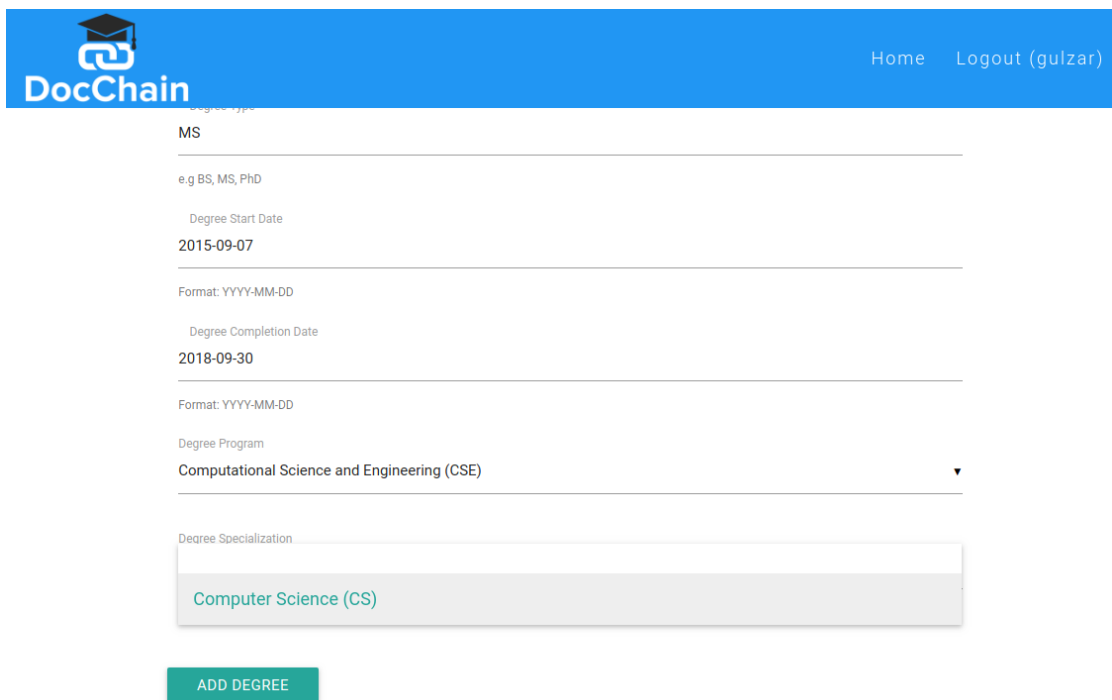


**Figure 5.24:** Snapshot - Exam Admin, Home Page.

After login with exam admin certificate, client is automatically redirected to exam admin home page, fig-5.24. Exam admin home page have two buttons, "NEW DEGREE" and "VIEW DEGREE".

### Adding Degree

NEW DEGREE button provide interface to create new degree by providing degree details, fig-5.25. On Successful addition of degree, we are responded with a success message, fig-5.26. If degree with same serial and registration number is already issued by department, an error message is returned, fig-5.27.



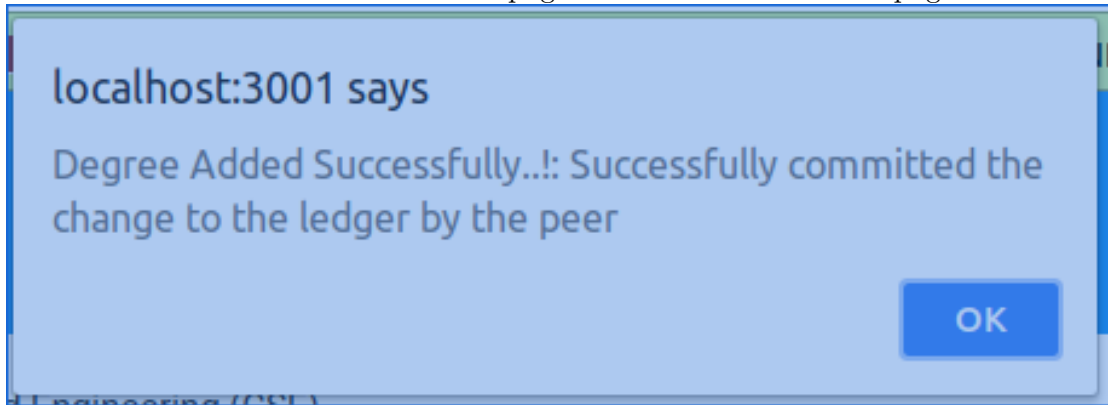
The screenshot shows the 'Add Degree' form in the DocChain system. The form is titled 'DocChain' and includes a navigation bar with 'Home' and 'Logout (gulzar)'. The form fields are:

- Degree Type:** MS
- Degree Start Date:** 2015-09-07 (Format: YYYY-MM-DD)
- Degree Completion Date:** 2018-09-30 (Format: YYYY-MM-DD)
- Degree Program:** Computational Science and Engineering (CSE)
- Degree Specialization:** Computer Science (CS)

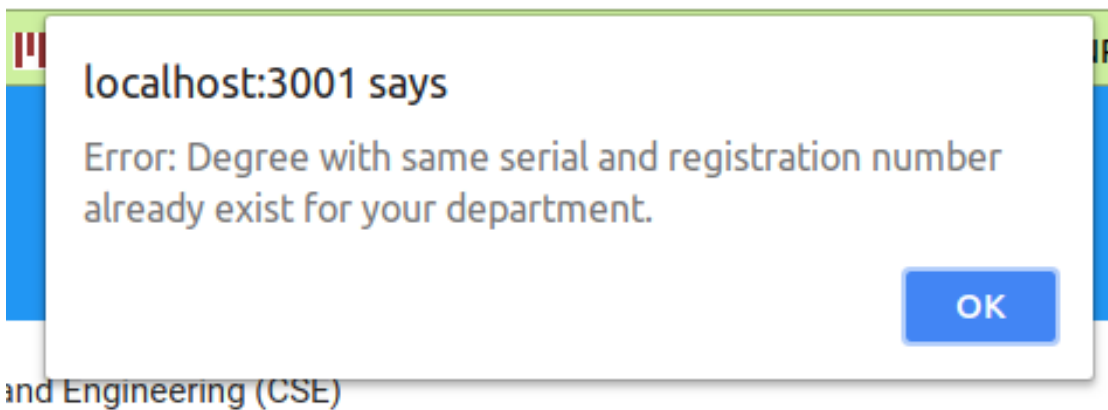
An 'ADD DEGREE' button is located at the bottom of the form.

**Figure 5.25:** Snapshot - Exam Admin, Add Degree.

from 2018-09-14 14-16-51.png from 2018-09-14 14-16-51.png



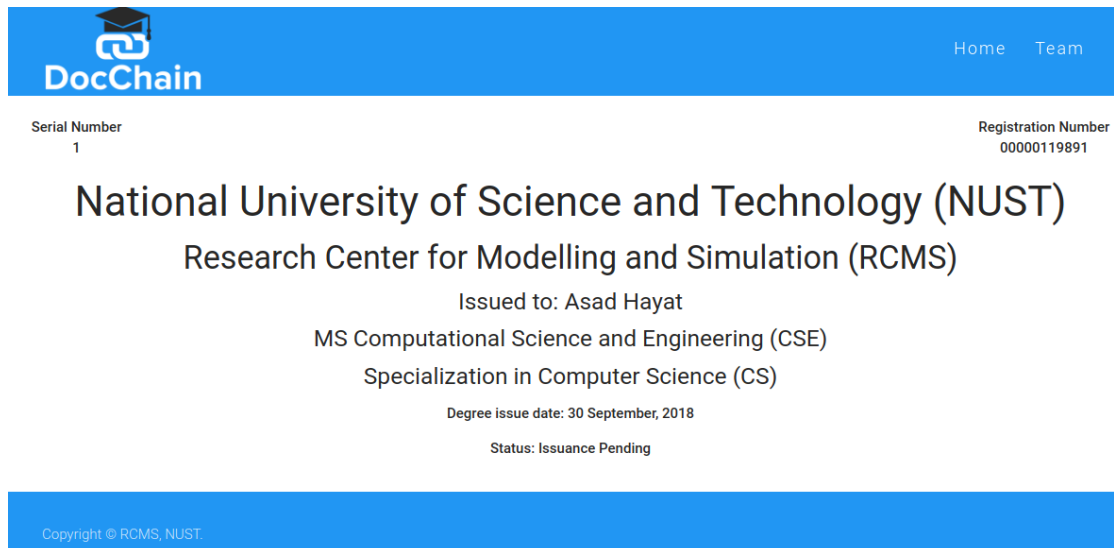
**Figure 5.26:** Snapshot - Exam Admin, Add Degree, Success Response.



**Figure 5.27:** Snapshot - Exam Admin, Add Degree, Error Response

### Viewing Degree

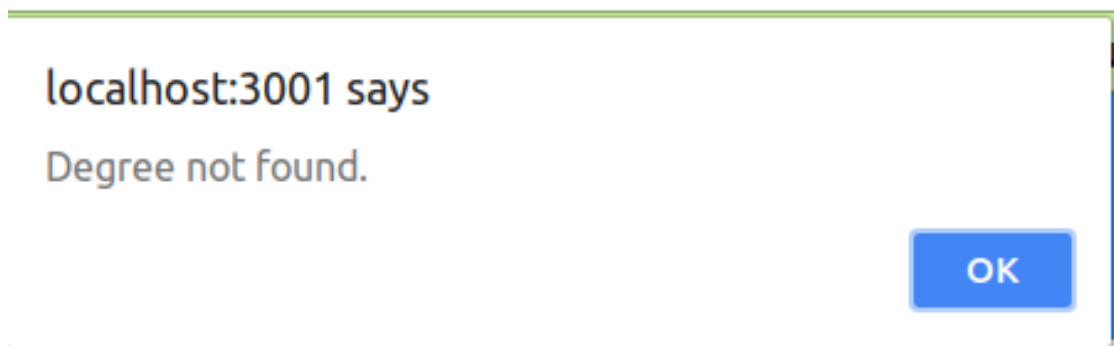
Exam Admin can view degree by using degree ledger key. VIEW DEGREE provides interface to input ledger key to be viewed. If a degree is added against the provided ledger key, client is moved to the page showing corresponding degree details, fig-5.28.



**Figure 5.28:** Snapshot - Exam Admin, View Degree

As evident from fig-5.28, degree status is "Pending" since it needs to be signed by Principal, Registrar and Rector.

If degree against the provided ledger key is not registered, an error message is returned, fig-5.29.

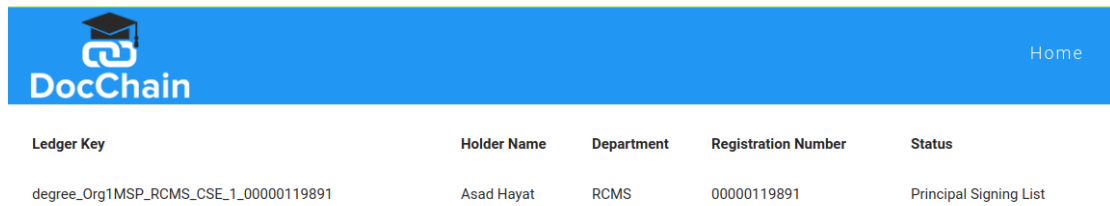


**Figure 5.29:** Snapshot - Exam Admin, Degree not found error.

### 5.5.3 Signing Degree

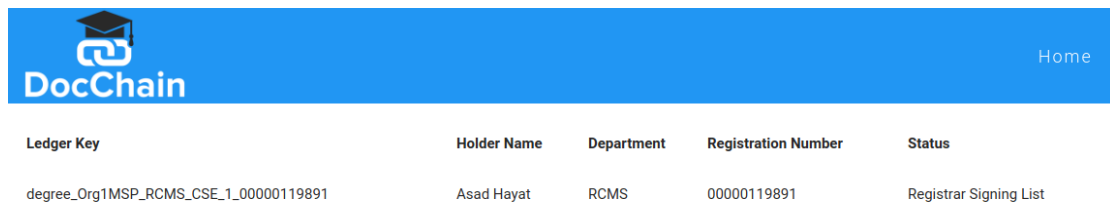
After exam admin has created degree on the blockchain, it is appeared in the "Pending signing list" of principal, where principal can sign degrees, fig-5.30. After principal has signed a degree, it is added to "Pending signing list" of Registrar, fig-5.31, and after a degree is signed by registrar it is added to "Pending signing list" of Rector, fig-5.32. After rector has signed a degree, its issuance is completed and is available for verification by

other organizations.



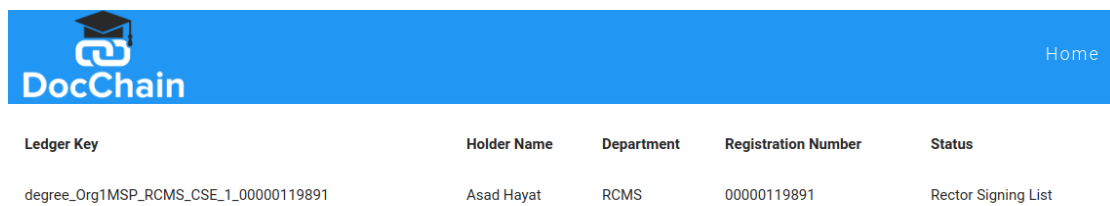
Ledger Key	Holder Name	Department	Registration Number	Status
degree_Org1MSP_RCMS_CSE_1_00000119891	Asad Hayat	RCMS	00000119891	Principal Signing List

**Figure 5.30:** Snapshot - Principal, Signing List.



Ledger Key	Holder Name	Department	Registration Number	Status
degree_Org1MSP_RCMS_CSE_1_00000119891	Asad Hayat	RCMS	00000119891	Registrar Signing List

**Figure 5.31:** Snapshot - Registrar, Signing List.



Ledger Key	Holder Name	Department	Registration Number	Status
degree_Org1MSP_RCMS_CSE_1_00000119891	Asad Hayat	RCMS	00000119891	Rector Signing List

**Figure 5.32:** Snapshot - Rector, Signing List.

When a signer clicks on a degree in the list a pop up appears which shows details of degree and a button "SIGN DEGREE". When signer click on SIGN DEGREE button, a request for degree signing is initiated by the respective signer and after successful verification, degree is signed and removed from current signing list.

After a degree is signed by all three signers, degree status is updated to "issued", fig-5.33.

The screenshot displays a digital document interface. At the top left, the 'DocChain' logo is visible, featuring a stylized 'e' and a graduation cap. To its right are navigation links for 'Home' and 'Team'. Below the logo, the 'Serial Number' is listed as '1'. On the right side, the 'Registration Number' is '00000119891'. The main content area is centered and contains the following text: 'National University of Science and Technology (NUST)', 'Research Center for Modelling and Simulation (RCMS)', 'Issued to: Asad Hayat', 'MS Computational Science and Engineering (CSE)', 'Specialization in Computer Science (CS)', 'Degree issue date: 30 September, 2018', and 'Status: Issued'. At the bottom left, a copyright notice reads 'Copyright © RCMS, NUST'.

Figure 5.33: Snapshot - Degree Issued.



#### 5.5.4 Verification of Degree

**Username:** verifier

**Special Attributes:** verifier=true

A verifier can verify degrees earned by a person by using CNIC/Citizenship Number of the person. Verifier home page has single input CNIC/Citizenship Number field, that returns all ledger keys of degrees earned by CNIC holder. Fig-5.33 shows degree view of an issued degree.

# Discussion, Conclusion and Future Work

## 6.1 Discussion

Blockchain is a distributed peer to peer system where record is replicated through out the network. It was first conceptualized by creator of Bitcoin, Satoshi Nakamoto in 2008. Nakamoto's idea was implemented in 2009 in the shape of Bitcoin, the first decentralized cryptographic currency.

Blockchain has variety of use cases in several industries and has the potential to transform the core of these industries. Beside cryptographic currencies, blockchain can be applied to supply chain management, digital identity, fast and cheap transactions settlement, goods tracking and hundreds of other use cases.

Smart contracts are deployed on the blockchain to automate different processes. Smart contracts are programmed set of rules and regulations deployed on the blockchain. It enables us to implement various uses cases on blockchain. Ethereum is the first smart contract platform. Linux Foundation have started Hyperledger project in December 2015. Different blockchain platforms are being developed under umbrella of hyperledger. Hyperledger Fabric is one of the project developed under Hyperledger. Hyperledger is permissioned, enterprise grade cross organizations based blockchain platform. Participants on Hyperledger Fabric platform are known and are granted permissions by their respective organization to join the network. Hyperledger Fabric also supports smart contracts. Smart contracts on hyperledger can be written in Go, NodeJS or Java as

of this writing and support for other programming languages are in progress. Each organization maintains a certificate authority that manages certificates for the organization. Hyperledger Fabric is an efficient blockchain platform as compared to bitcoin and ethereum and its support of implementing smart contracts in general programming languages makes application development more convenient.

Documents forgery is a major problem worldwide. To tackle this problem several techniques are used, such as specialized fonts, seals, embossed areas on paper et cetera. But easy and cheap availability of printing technology have enabled malicious bodies to forge different type documents. There are several cases in which candidates have used forged academic degrees for different purposes, such as securing employment position, negotiating better salaries or running for public offices.

## 6.2 Conclusion

We have designed and developed a blockchain application for academic documents verification. Each university on the network is a separate organization. A university has different types of participants to perform various type of operations. We have developed chaincode in Go and make use of Hyperledger Fabric NodeJS Software Development Kit (NodeJS SDK) to implement user interface for our application.

In our application for documents verification, a university has several types of user with different level of roles. University admin registers and setup university academic structure on the application. Exam admin creates academic degrees which needs to be signed by three required signers, principal, registrar and rector. When exam admin creates the degree, its initial status is "Issuance Pending". After principal, registrar and rector sign the degree, the status of the degree is updated as "Issued". A degree with "Issued" status can be verified by any verifier with valid certificates and permissions.

Traditional systems maintains record on a centralized server which is single point of failure and have low security. An inside or outside attacker can modify the records in the system without detection. Which may miss lead the system to verify forged information as authentic.

Blockchain systems on the other hand is more secure than centralized systems. Any updates in the record is logged on the ledger with complete history of information. This

provides an easy way to audit record and detect any unauthorized updates to data. Since each transaction on the blockchain is signed by cryptographic certificates of the client, we can detect who updated what and when.

This system, Blockchain application for academic documents verification, can efficiently address the problem of documents forgery. Adoption of this system will make the process of degree verification fast, easier and inexpensive.

### **6.3 Future Work**

The system we have developed can be extended by implementing additional features. Following are the list of few features that will make the system more important.

- Addition of semester wise academic records to add feature for verification of student transcript with complete timestamped history.
- Support for addition of academic record by boards of intermediate and secondary education.
- Support for school and college to issue Detailed Marks Certificates (DMCs) and other documents.

Implementation of these features will make the entire academic history verifiable at single application.

# References

- [1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.
- [2] ———, “Bitcoin v0.1 released,” 2009. [Online]. Available: <https://www.mail-archive.com/cryptography@metzdowd.com/msg10142.html>
- [3] V. Buterin *et al.*, “Ethereum white paper, 2014,” URL <https://github.com/ethereum/wiki/wiki/White-Paper>, 2013.
- [4] N. van Saberhagen, “Cryptonote v 2.0, 2013,” URL: <https://cryptonote.org/whitepaper.pdf>. *White Paper. Accessed*, pp. 04–13, 2018.
- [5] D. Schwartz, N. Youngs, A. Britto *et al.*, “The ripple protocol consensus algorithm,” *Ripple Labs Inc White Paper*, vol. 5, 2014.
- [6] O. Jacobovitz, “Blockchain for identity management,” *The Lynne and William Frankel Center for Computer Science Department of Computer Science. Ben-Gurion University, Beer Sheva Google Scholar*, 2016.
- [7] A. Ebrahimi, “Identity management service using a blockchain providing certifying transactions between devices,” Aug. 1 2017, uS Patent 9,722,790.
- [8] S. Underwood, “Blockchain beyond bitcoin,” *Communications of the ACM*, vol. 59, no. 11, pp. 15–17, 2016.
- [9] K. Korpela, J. Hallikas, and T. Dahlberg, “Digital supply chain transformation toward blockchain integration,” in *proceedings of the 50th Hawaii international conference on system sciences*, 2017.
- [10] S. Shang, N. Memon, and X. Kong, “Detecting documents forged by printing and copying,” *EURASIP Journal on Advances in Signal Processing*, vol. 2014, no. 1, p. 140, 2014.

## REFERENCES

- [11] A. B. Hassan and Y. A. Fadlalla, “A survey on techniques of detecting identity documents forgery,” in *Computer Science and Information Technology (SCCSIT), 2017 Sudan Conference on*. IEEE, 2017, pp. 1–5.
- [12] S. Ibrahim, M. Afrakhteh, and M. Salleh, “Adaptive watermarking for printed document authentication,” in *Computer Sciences and Convergence Information Technology (ICCIT), 2010 5th International Conference on*. IEEE, 2010, pp. 611–614.
- [13] R. Bertrand, O. R. Terrades, P. Gomez-Kramer, P. Franco, and J.-M. Ogier, “A conditional random field model for font forgery detection,” in *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*. IEEE, 2015, pp. 576–580.
- [14] S. Bhattacharyya, “A survey of steganography and steganalysis technique in image, text, audio and video as cover carrier,” *Journal of global research in computer science*, vol. 2, no. 4, 2011.
- [15] L. H. Newman. (2017) Hacker lexicon: What is steganography? [Online]. Available: <https://www.wired.com/story/steganography-hacker-lexicon/>
- [16] T. Buttress, *Fraud: a growing problem in education, and how to guard against it*, August 2012. [Online]. Available: <https://uknaric.org/2012/08/17/how-to-spot-a-fraudulent-education-document>
- [17] P. Baran, “On distributed communications networks,” *IEEE transactions on Communications Systems*, vol. 12, no. 1, pp. 1–9, 1964.
- [18] N. Sklavos and O. Koufopavlou, “On the hardware implementations of the sha-2 (256, 384, 512) hash functions,” in *Circuits and Systems, 2003. ISCAS’03. Proceedings of the 2003 International Symposium on*, vol. 5. IEEE, 2003, pp. V–V.
- [19] R. P. McEvoy, F. M. Crowe, C. C. Murphy, and W. P. Marnane, “Optimisation of the sha-2 family of hash functions on fpgas,” in *Emerging VLSI Technologies and Architectures, 2006. IEEE Computer Society Annual Symposium on*. IEEE, 2006, pp. 6–pp.
- [20] R. Housley, W. Ford, W. Polk, and D. Solo, “Internet x. 509 public key infrastructure certificate and crl profile,” Tech. Rep., 1998.

## REFERENCES

- [21] L. Lamport, R. Shostak, and M. Pease, "The byzantine generals problem," *ACM Transactions on Programming Languages and Systems (TOPLAS)*, vol. 4, no. 3, pp. 382–401, 1982.
- [22] A. M. Antonopoulos, *Mastering Bitcoin: unlocking digital cryptocurrencies*. " O'Reilly Media, Inc.", 2014.
- [23] P. Champagne, "The book of satoshi: The collected writings of bitcoin creator satoshi nakamoto," *E53*, 2014.
- [24] P. Sajana, M. Sindhu, and M. Sethumadhavan, "On blockchain applications: Hyperledger fabric and ethereum."
- [25] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the Thirteenth EuroSys Conference*. ACM, 2018, p. 30.

## APPENDIX A

# Cryptographic Configuration

Following *yaml* structure defines configurations for generating cryptographic certificates for orderer and participants of organizations. There are three organizations and a single orderer in the following sample. A single orderer, OrdererNUST, and three organizations NUSTMSP, GIKIMSP and LUMSMSP.

OrdererOrgs:

- Name: OrdererNUST
- Domain: nust.edu.pk
- Specs:
  - Hostname: orderer

PeerOrgs:

- Name: NUSTMSP
- Domain: nust.edu.pk
- Template:
  - Count: 3
- Users:
  - Count: 2
- Name: GIKIMSP
- Domain: giki.edu.pk
- Template:
  - Count: 2
- Users:
  - Count: 1



## APPENDIX A: CRYPTOGRAPHIC CONFIGURATION

- Name: LUMSMSP
  - Domain: lums.edu.pk
  - Template:
    - Count: 2
  - Users:
    - Count: 1

## APPENDIX B

# Configuration Transaction

Following *yaml* structure presents configuration for generating orderer genesis block, channel configuration transaction, anchor peers transaction et cetera.

Profiles:

```
DocChainOrdererGenesis:
  Capabilities:
    <<: *ChannelCapabilities
  Orderer:
    <<: *OrdererDefaults
  Organizations:
    - *OrdererNUSTOrg
  Capabilities:
    <<: *OrdererCapabilities
  Consortiums:
    DocChainConsortium:
      Organizations:
        - *NUST
        - *GIKI
        - *LUMS
  DocChainOrgsChannel:
    Consortium: DocChainConsortium
  Application:
```

## APPENDIX B: CONFIGURATION TRANSACTION

```
<<: *ApplicationDefaults
Organizations:
  - *NUST
  - *GIKI
  - *LUMS
Capabilities:
  <<: *ApplicationCapabilities
Organizations:
  - &OrdererNUSTOrg
    Name: OrdererNUST
    ID: OrdererNUST
    MSPDir: crypto-config/ordererOrganizations/nust.edu.pk/msp

  - &NUST
    Name: NUSTMSP
    ID: NUSTMSP

    MSPDir: crypto-config/peerOrganizations/nust.edu.pk/msp

    AnchorPeers:
      - Host: peer0.nust.edu.pk
        Port: 7051

  - &GIKI
    Name: GIKIMSP
    ID: GIKIMSP
    MSPDir: crypto-config/peerOrganizations/giki.edu.pk/msp

    AnchorPeers:
      - Host: peer0.giki.edu.pk
        Port: 7051

  - &LUMS
```

## APPENDIX B: CONFIGURATION TRANSACTION

Name: LUMSMSP

ID: LUMSMSP

MSPDir: crypto-config/peerOrganizations/lums.edu.pk/msp

AnchorPeers:

- Host: peer0.lums.edu.pk

- Port: 7051

Orderer: `&OrdererDefaults`

OrdererType: solo

Addresses:

- orderer.nust.edu.pk:7050

BatchTimeout: 2s

BatchSize:

MaxMessageCount: 10

AbsoluteMaxBytes: 99 MB

PreferredMaxBytes: 512 KB

Kafka:

Brokers:

- 127.0.0.1:9092

Organizations:

- `*OrdererNUSTOrg`

Application: `&ApplicationDefaults`

Organizations:

- `*NUST`

- `*GIKI`

- `*LUMS`

Capabilities:

Global: `&ChannelCapabilities`

V1\_2: true

Orderer: `&OrdererCapabilities`

V1\_2: true

Application: `&ApplicationCapabilities`

## APPENDIX B: CONFIGURATION TRANSACTION

V1\_2: true

# Sample Network Configurations for docker-compose

Following *.yaml* represents structure and configurations of a sample docker virtual network. This virtual network is used to test development of our application, DocChain. An overview of the network is given in table [C.1](#).

There is no upper or lower limit on the number of peers an organization can maintain. In these configuration Organization NUST maintains 3 peers, a single orderer and a single certificate authority. Organization GIKI maintains two peers, one orderer and one certificate authority. Organization LUMS maintains two peers and no orderer or certificate authority.

Certificate Authority is essential for any organization for issuing new certificates or revoking already issued certificates. However, in our configurations, organization LUMS do not have any certificate authority. This is an example of keeping certificate authority completely out of the network. Keeping certificate authority out of the network will not affect certificates in any way and will reduce security threats. The network can still validate certificates even if no certificate authority is connected to network.

Organization LUMS also do not have any orderer. In this situation, where an organization have no orderer, any ordering service of other organizations can be used. Even if all organization have running ordering services, to validate a transaction all orderers must be in consensus.

Organization	Peers	Orderers	Certificate Authority
GIKI	peer0.giki.edu.pk peer1.giki.edu.pk	orderer.giki.edu.pk	ca.giki.edu.pk
NUST	peer0.nust.edu.pk peer1.nust.edu.pk peer2.nust.edu.pk	orderer.nust.edu.pk	ca.nust.edu.pk
LUMS	peer0.nust.edu.pk	-	-

**Table C.1:** List of Peers, Orderers and Certificate Authority in each organization.

## C.1 YAML code for Docker Compose

```

version: '2'

networks:
  docchain:

services:
  #=====
  ca.giki.edu.pk:
    image: hyperledger/fabric-ca
    environment:
      - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server-config
      - FABRIC_CA_SERVER_CA_NAME=ca.giki.edu.pk
      - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/fabric-ca-server-
        ↪ config/ca.giki.edu.pk-cert.pem
      - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/fabric-ca-server-
        ↪ config/3d1e52f62ee46c900d56e37be4394642c8da77faaa632
        ↪ 4b8f26ff26434fea9af_sk
    ports:
      - "7054:7054"
    command: sh -c 'fabric-ca-server start -b admin:adminpw'
    volumes:

```

## APPENDIX C: SAMPLE NETWORK CONFIGURATIONS FOR DOCKER-COMPOSE

```
- ./crypto-config/peerOrganizations/giki.edu.pk/ca/:etc/  
  ↪ hyperledger/fabric-ca-server-config  
container_name: ca.giki.edu.pk  
networks:  
  - docchain
```

#=====

```
ca.nust.edu.pk:  
  image: hyperledger/fabric-ca  
  environment:  
    - FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server  
    - FABRIC_CA_SERVER_CA_NAME=ca.nust.edu.pk  
    - FABRIC_CA_SERVER_CA_CERTFILE=/etc/hyperledger/  
      ↪ fabric-ca-server-config/ca.nust.edu.pk-cert.pem  
    - FABRIC_CA_SERVER_CA_KEYFILE=/etc/hyperledger/  
      ↪ fabric-ca-server-config/6d480a567fe47bd257f2bb46da3e8c651c2  
      ↪ 8ddd614da596e2f2cdc44ff12cdaa_sk  
  ports:  
    - "8054:7054"  
  command: sh -c 'fabric-ca-server start -b admin:adminpw -d'  
  volumes:  
    - ./crypto-config/peerOrganizations/nust.edu.pk/ca/:etc/  
      ↪ hyperledger/fabric-ca-server-config  
  container_name: ca.nust.edu.pk  
  networks:  
    - docchain
```

#=====

```
ca.lums.edu.pk:  
  image: hyperledger/fabric-ca
```



environment:

- FABRIC\_CA\_HOME=/etc/hyperledger/fabric-ca-server
- FABRIC\_CA\_SERVER\_CA\_NAME=ca.lums.edu.pk
- FABRIC\_CA\_SERVER\_CA\_CERTFILE=/etc/hyperledger/fabric-ca-server-  
   ↪ config/ca.lums.edu.pk-cert.pem
- FABRIC\_CA\_SERVER\_CA\_KEYFILE=/etc/hyperledger/fabric-ca-server-  
   ↪ config/69662f4cdd3161168ca754c5e201b2676d32e7fa5f887bda1ebb8de9  
   ↪ 538fa5ff\_sk

ports:

- "9054:7054"

command: sh -c 'fabric-ca-server start -b admin:adminpw -d'

volumes:

- ./crypto-config/peerOrganizations/lums.edu.pk/ca/:/etc/  
   ↪ hyperledger/fabric-ca-server-config

container\_name: ca.lums.edu.pk

networks:

- docchain

#=====

#=====

orderer.nust.edu.pk:

container\_name: orderer.nust.edu.pk

image: hyperledger/fabric-orderer

environment:

- ORDERER\_GENERAL\_LOGLEVEL=INFO
- ORDERER\_GENERAL\_LISTENADDRESS=0.0.0.0
- ORDERER\_GENERAL\_GENESISMETHOD=file
- ORDERER\_GENERAL\_GENESISFILE=/var/hyperledger/orderer/  
   ↪ orderer.genesis.block
- ORDERER\_GENERAL\_LOCALMSPID=OrdererNUST
- ORDERER\_GENERAL\_LOCALMSPDIR=/var/hyperledger/orderer/msp

```

    # enabled TLS
    - ORDERER_GENERAL_TLS_ENABLED=true
    - ORDERER_GENERAL_TLS_PRIVATEKEY=/var/hyperledger/orderer/tls/
      ↪ server.key
    - ORDERER_GENERAL_TLS_CERTIFICATE=/var/hyperledger/orderer/tls/
      ↪ server.crt
    -
      ↪ ORDERER_GENERAL_TLS_ROOTCAS=[/var/hyperledger/orderer/tls/ca.crt]
working_dir: /opt/gopath/src/github.com/hyperledger/fabric
command: orderer
volumes:
- ./channel-artifacts/genesis.block:/var/hyperledger/orderer/
  ↪ orderer.genesis.block
- ./crypto-config/ordererOrganizations/nust.edu.pk/orderers/
  ↪ orderer.nust.edu.pk/msp:/var/hyperledger/orderer/msp
- ./crypto-config/ordererOrganizations/nust.edu.pk/orderers/
  ↪ orderer.nust.edu.pk/tls:/var/hyperledger/orderer/tls
- ./data/orderer.edu.pk:/var/hyperledger

ports:
  - 7050:7050

networks:
  - docchain

peer0.nust.edu.pk:
  container_name: peer0.nust.edu.pk
  image: hyperledger/fabric-peer
  environment:
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    # the following setting starts chaincode containers on the same
    # bridge network as the peers
    # https://docs.docker.com/compose/networking/
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=docchain_docchain
    - CORE_LOGGING_LEVEL=INFO

```

## APPENDIX C: SAMPLE NETWORK CONFIGURATIONS FOR DOCKER-COMPOSE

```
#- CORE_LOGGING_LEVEL=DEBUG
- CORE_PEER_TLS_ENABLED=true
- CORE_PEER_GOSSIP_USELEADERELECTION=true
- CORE_PEER_GOSSIP_ORGLEADER=false
- CORE_PEER_PROFILE_ENABLED=true
- CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
- CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
- CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
- CORE_PEER_ID=peer0.nust.edu.pk
- CORE_PEER_ADDRESS=peer0.nust.edu.pk:7051
- CORE_PEER_GOSSIP_BOOTSTRAP=peer1.nust.edu.pk:7051
- CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.nust.edu.pk:7051
- CORE_PEER_LOCALMSPID=NUSTMSP

volumes:
  - /var/run/:/host/var/run/
  - ./crypto-config/peerOrganizations/nust.edu.pk/peers/
    ↪ peer0.nust.edu.pk/msp:/etc/hyperledger/fabric/msp
  - ./crypto-config/peerOrganizations/nust.edu.pk/peers/
    ↪ peer0.nust.edu.pk/tls:/etc/hyperledger/fabric/tls
  - ./data/peer0.nust.edu.pk:/var/hyperledger
  - ./chaincodes:/var/chaincodes

working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
command: peer node start

ports:
  - 7051:7051
  - 7053:7053

networks:
  - docchain

peer1.nust.edu.pk:
  container_name: peer1.nust.edu.pk
  image: hyperledger/fabric-peer
  environment:
```

## APPENDIX C: SAMPLE NETWORK CONFIGURATIONS FOR DOCKER-COMPOSE

```
- CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
# the following setting starts chaincode containers on the same
# bridge network as the peers
# https://docs.docker.com/compose/networking/
- CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=docchain_docchain
# - CORE_LOGGING_LEVEL=INFO
- CORE_LOGGING_LEVEL=DEBUG
- CORE_PEER_TLS_ENABLED=true
- CORE_PEER_GOSSIP_USELEADERELECTION=true
- CORE_PEER_GOSSIP_ORGLEADER=false
- CORE_PEER_PROFILE_ENABLED=true
- CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
- CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
- CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
- CORE_PEER_ID=peer1.nust.edu.pk
- CORE_PEER_ADDRESS=peer1.nust.edu.pk:7051
- CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer1.nust.edu.pk:7051
- CORE_PEER_GOSSIP_BOOTSTRAP=peer2.nust.edu.pk:7051
- CORE_PEER_LOCALMSPID=NUSTMSP

volumes:
  - /var/run/:/host/var/run/
  - ./crypto-config/peerOrganizations/nust.edu.pk/peers/
    ↪ peer1.nust.edu.pk/msp:/etc/hyperledger/fabric/msp
  - ./crypto-config/peerOrganizations/nust.edu.pk/peers/
    ↪ peer1.nust.edu.pk/tls:/etc/hyperledger/fabric/tls
  - ./data/peer1.nust.edu.pk:/var/hyperledger
  - ./chaincodes:/var/chaincodes

working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
command: peer node start

ports:
  - 8051:7051
  - 8053:7053

networks:
```

```
- docchain
```

```
peer2.nust.edu.pk:
```

```
  container_name: peer2.nust.edu.pk
```

```
  image: hyperledger/fabric-peer
```

```
  environment:
```

```
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
      # the following setting starts chaincode containers on the same
      # bridge network as the peers
      # https://docs.docker.com/compose/networking/
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=docchain_docchain
      # - CORE_LOGGING_LEVEL=INFO
    - CORE_LOGGING_LEVEL=DEBUG
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_PROFILE_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
    - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
    - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
    - CORE_PEER_ID=peer2.nust.edu.pk
    - CORE_PEER_ADDRESS=peer2.nust.edu.pk:7051
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer2.nust.edu.pk:7051
    - CORE_PEER_GOSSIP_BOOTSTRAP=peer0.nust.edu.pk:7051
    - CORE_PEER_LOCALMSPID=NUSTMSP
```

```
  volumes:
```

```
    - /var/run/:/host/var/run/
    - ./crypto-config/peerOrganizations/nust.edu.pk/peers/
      ↪ peer2.nust.edu.pk/msp:/etc/hyperledger/fabric/msp
    - ./crypto-config/peerOrganizations/nust.edu.pk/peers/
      ↪ peer2.nust.edu.pk/tls:/etc/hyperledger/fabric/tls
    - ./data/peer2.nust.edu.pk:/var/hyperledger
```

## APPENDIX C: SAMPLE NETWORK CONFIGURATIONS FOR DOCKER-COMPOSE

```
    - ./chaincodes:/var/chaincodes
working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer
command: peer node start
ports:
  - 9051:7051
  - 9053:7053
networks:
  - docchain

#####
#####
#
#  MUST PEER'S ENDS
#  GIKI PEER'S STARTS
#
#####

peer0.giki.edu.pk:
  container_name: peer0.giki.edu.pk
  image: hyperledger/fabric-peer
  environment:
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    # the following setting starts chaincode containers on the same
    # bridge network as the peers
    # https://docs.docker.com/compose/networking/
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=docchain_docchain
    # - CORE_LOGGING_LEVEL=INFO
    - CORE_LOGGING_LEVEL=DEBUG
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_PROFILE_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
```

## APPENDIX C: SAMPLE NETWORK CONFIGURATIONS FOR DOCKER-COMPOSE

- CORE\_PEER\_TLS\_KEY\_FILE=/etc/hyperledger/fabric/tls/server.key
- CORE\_PEER\_TLS\_ROOTCERT\_FILE=/etc/hyperledger/fabric/tls/ca.crt
- CORE\_PEER\_ID=peer0.giki.edu.pk
- CORE\_PEER\_ADDRESS=peer0.giki.edu.pk:7051
- CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT=peer0.giki.edu.pk:7051
- CORE\_PEER\_GOSSIP\_BOOTSTRAP=peer1.giki.edu.pk:7051
- CORE\_PEER\_LOCALMSPID=GIKIMSP

volumes:

- /var/run/:/host/var/run/
- ./crypto-config/peerOrganizations/giki.edu.pk/peers/  
↪ peer0.giki.edu.pk/msp:/etc/hyperledger/fabric/msp
- ./crypto-config/peerOrganizations/giki.edu.pk/peers/  
↪ peer0.giki.edu.pk/tls:/etc/hyperledger/fabric/tls
- ./data/peer0.giki.edu.pk:/var/hyperledger
- ./chaincodes:/var/chaincodes

working\_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer

command: peer node start

ports:

- 10051:7051
- 10053:7053

networks:

- docchain

peer1.giki.edu.pk:

container\_name: peer1.giki.edu.pk

image: hyperledger/fabric-peer

environment:

- CORE\_VM\_ENDPOINT=unix:///host/var/run/docker.sock
- # the following setting starts chaincode containers on the same*
- # bridge network as the peers*
- # <https://docs.docker.com/compose/networking/>*
- CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE=docchain\_docchain
- # - CORE\_LOGGING\_LEVEL=INFO*

## APPENDIX C: SAMPLE NETWORK CONFIGURATIONS FOR DOCKER-COMPOSE

- CORE\_LOGGING\_LEVEL=DEBUG
- CORE\_PEER\_TLS\_ENABLED=true
- CORE\_PEER\_GOSSIP\_USELEADERELECTION=true
- CORE\_PEER\_GOSSIP\_ORGLEADER=false
- CORE\_PEER\_PROFILE\_ENABLED=true
- CORE\_PEER\_TLS\_CERT\_FILE=/etc/hyperledger/fabric/tls/server.crt
- CORE\_PEER\_TLS\_KEY\_FILE=/etc/hyperledger/fabric/tls/server.key
- CORE\_PEER\_TLS\_ROOTCERT\_FILE=/etc/hyperledger/fabric/tls/ca.crt
- CORE\_PEER\_ID=peer1.giki.edu.pk
- CORE\_PEER\_ADDRESS=peer1.giki.edu.pk:7051
- CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT=peer1.giki.edu.pk:7051
- CORE\_PEER\_GOSSIP\_BOOTSTRAP=peer0.giki.edu.pk:7051
- CORE\_PEER\_LOCALMSPID=GIKIMSP

volumes:

- /var/run/:/host/var/run/
- ./crypto-config/peerOrganizations/giki.edu.pk/peers/  
↪ peer1.giki.edu.pk/msp:/etc/hyperledger/fabric/msp
- ./crypto-config/peerOrganizations/giki.edu.pk/peers/  
↪ peer1.giki.edu.pk/tls:/etc/hyperledger/fabric/tls
- ./data/peer1.giki.edu.pk:/var/hyperledger
- ./chaincodes:/var/chaincodes

working\_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer

command: peer node start

ports:

- 11051:7051
- 11053:7053

networks:

- docchain

#####

#####

#

# GIKI PEER'S ENDS



```

# LUMS PEER'S STARTS
#
#####

peer0.lums.edu.pk:
  container_name: peer0.lums.edu.pk
  image: hyperledger/fabric-peer
  environment:
    - CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
    # the following setting starts chaincode containers on the same
    # bridge network as the peers
    # https://docs.docker.com/compose/networking/
    - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=docchain_docchain
    # - CORE_LOGGING_LEVEL=INFO
    - CORE_LOGGING_LEVEL=DEBUG
    - CORE_PEER_TLS_ENABLED=true
    - CORE_PEER_GOSSIP_USELEADERELECTION=true
    - CORE_PEER_GOSSIP_ORGLEADER=false
    - CORE_PEER_PROFILE_ENABLED=true
    - CORE_PEER_TLS_CERT_FILE=/etc/hyperledger/fabric/tls/server.crt
    - CORE_PEER_TLS_KEY_FILE=/etc/hyperledger/fabric/tls/server.key
    - CORE_PEER_TLS_ROOTCERT_FILE=/etc/hyperledger/fabric/tls/ca.crt
    - CORE_PEER_ID=peer0.lums.edu.pk
    - CORE_PEER_ADDRESS=peer0.lums.edu.pk:7051
    - CORE_PEER_GOSSIP_EXTERNALENDPOINT=peer0.lums.edu.pk:7051
    - CORE_PEER_GOSSIP_BOOTSTRAP=peer1.lums.edu.pk:7051
    - CORE_PEER_LOCALMSPID=LUMSMSP
  volumes:
    - /var/run/:/host/var/run/
    - ./crypto-config/peerOrganizations/lums.edu.pk/peers/
      ↪ peer0.lums.edu.pk/msp:/etc/hyperledger/fabric/msp

```

## APPENDIX C: SAMPLE NETWORK CONFIGURATIONS FOR DOCKER-COMPOSE

- ./crypto-config/peerOrganizations/lums.edu.pk/peers/
  - ↪ peer0.lums.edu.pk/tls:/etc/hyperledger/fabric/tls
- ./data/peer0.lums.edu.pk:/var/hyperledger
- ./chaincodes:/var/chaincodes

working\_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer

command: peer node start

ports:

- 12051:7051
- 12053:7053

networks:

- docchain

peer1.lums.edu.pk:

container\_name: peer1.lums.edu.pk

image: hyperledger/fabric-peer

environment:

- CORE\_VM\_ENDPOINT=unix:///host/var/run/docker.sock
- # the following setting starts chaincode containers on the same*
- # bridge network as the peers*
- # <https://docs.docker.com/compose/networking/>*
- CORE\_VM\_DOCKER\_HOSTCONFIG\_NETWORKMODE=docchain\_docchain
- # - CORE\_LOGGING\_LEVEL=INFO*
- CORE\_LOGGING\_LEVEL=DEBUG
- CORE\_PEER\_TLS\_ENABLED=true
- CORE\_PEER\_GOSSIP\_USELEADERELECTION=true
- CORE\_PEER\_GOSSIP\_ORGLEADER=false
- CORE\_PEER\_PROFILE\_ENABLED=true
- CORE\_PEER\_TLS\_CERT\_FILE=/etc/hyperledger/fabric/tls/server.crt
- CORE\_PEER\_TLS\_KEY\_FILE=/etc/hyperledger/fabric/tls/server.key
- CORE\_PEER\_TLS\_ROOTCERT\_FILE=/etc/hyperledger/fabric/tls/ca.crt
- CORE\_PEER\_ID=peer1.lums.edu.pk
- CORE\_PEER\_ADDRESS=peer1.lums.edu.pk:7051

## APPENDIX C: SAMPLE NETWORK CONFIGURATIONS FOR DOCKER-COMPOSE

- CORE\_PEER\_GOSSIP\_EXTERNALENDPOINT=peer1.lums.edu.pk:7051
- CORE\_PEER\_GOSSIP\_BOOTSTRAP=peer0.lums.edu.pk:7051
- CORE\_PEER\_LOCALMSPID=LUMSMSP

volumes:

- /var/run/:/host/var/run/
- ./crypto-config/peerOrganizations/lums.edu.pk/peers/  
↪ peer1.lums.edu.pk/msp:/etc/hyperledger/fabric/msp
- ./crypto-config/peerOrganizations/lums.edu.pk/peers/  
↪ peer1.lums.edu.pk/tls:/etc/hyperledger/fabric/tls
- ./data/peer1.lums.edu.pk:/var/hyperledger
- ./chaincodes:/var/chaincodes

working\_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer

command: peer node start

ports:

- 13051:7051
- 13053:7053

networks:

- docchain

cli:

container\_name: cli

image: hyperledger/fabric-tools

tty: true

stdin\_open: true

environment:

- GOPATH=/opt/gopath
- CORE\_VM\_ENDPOINT=unix:///host/var/run/docker.sock
- CORE\_LOGGING\_LEVEL=DEBUG

# - CORE\_LOGGING\_LEVEL=INFO

- CORE\_PEER\_ID=cli

- CORE\_PEER\_ADDRESS=\_\_

- CORE\_PEER\_LOCALMSPID=\_\_

- CORE\_PEER\_TLS\_ENABLED=true

## APPENDIX C: SAMPLE NETWORK CONFIGURATIONS FOR DOCKER-COMPOSE

```
- CORE_PEER_TLS_CERT_FILE=__
- CORE_PEER_TLS_KEY_FILE=__
- CORE_PEER_TLS_ROOTCERT_FILE=__
- CORE_PEER_MSPCONFIGPATH=__

working_dir: /opt/gopath/src/github.com/hyperledger/fabric/peer

command: /bin/bash

volumes:
  - /var/run/:/host/var/run/
  - /home/asad/projects/src/github.com/hyperledger/fabric:
    ↪ /opt/gopath/src/github.com/hyperledger/fabric/
  - ./crypto-config:/opt/gopath/src/github.com/hyperledger/
    ↪ fabric/peer/crypto/
  - ./chaincodes:/opt/gopath/src/github.com/chaincodes
  - ./scripts:/opt/gopath/src/github.com/hyperledger/fabric/
    ↪ peer/scripts/
  - ./channel-artifacts:/opt/gopath/src/github.com/hyperledger/
    ↪ fabric/peer/channel-artifacts

depends_on:
  - orderer.nust.edu.pk
  - peer0.nust.edu.pk
  - peer1.nust.edu.pk
  - peer2.nust.edu.pk
  - peer0.giki.edu.pk
  - peer1.giki.edu.pk
  - peer0.lums.edu.pk
  - peer1.lums.edu.pk

networks:
  - docchain
```

## C.2 Starting the network

To start network with these configuration follow, make sure that **Docker** and **docker-compose** are installed on your system and follow these steps:

- Copy these configurations to a text file **docker-compose-3-orgs.yaml**.
- Navigate to the directory of **docker-compose-3-orgs.yaml** on your command line.
- Now execute **docker-compose -f docker-compose-3-orgs.yaml up**.

If the script is being run for the first time, all the required docker images will be downloaded first and then the network will be started.