12-2019

A Study of Blockchain Framework–Hyperledger Fabric and Implementation as Educational Network

Venkata Ayyappa Devarasetty
ayyappa.venkat87@gmail.com

A Study of Blockchain Framework–Hyperledger Fabric and Implementation as Educational Network

by

Venkata Ayyappa Devarasetty

A Starred Paper

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Information Assurance

December, 2019

Abstract

Blockchain, the foundation for Bitcoin, has gained lots of attention recently. Blockchain works as a distributed ledger technology that allows information exchange to take place in a distributed way, and ledger is immutable. Blockchain database removes the necessity of the centralized system; therefore, applications based on Blockchain are getting high in number. This paper covers an discuss in detail of blockchain technology, and its consensus algorithms along with workflow, how trust has will be upon a system having no centralized system. This paper also studies various frameworks being built upon the blockchain systems and how they are helpful in solving many organizational issues and Developing of an application on an existing blockchain framework which is an access based system, has information regarding academic records, certifications and eligibility requirement examination records belong to a person, who can share with any organization, eliminating the need of physical documents.

Keywords: Distributed Ledger, Bitcoin, Consensus, blockchain

Table of Contents

List of Tables

List of Figures

Chapter I: Introduction

This study investigates the current challenges, feasibility, benefits, and risks of working with blockchain technology in the educational certification system. Certificates act as proof for a student when reaching out to education and employment companies, which play a crucial role in a person's professional career. Therefore availability and immutability are important aspects here.

Blockchain technology offers us these characteristics. It helps us to store information where all the history will be maintained, and all the data stored in it is secure, transparent, immutable in every way. Until today, whenever a person graduated from any university, did any certification from a private institution has been receiving the certificates in a physical format, and this certification process is not digitized. In this paper, the application of blockchain technology on issuing, maintaining, monitoring, and verification of certificates by surveying most popular blockchain concepts, such as Ethereum and Hyperledger Fabric.

Process automation is taken care of, by the concept of smart contracts, which runs on a blockchain. This application will represent digital certificates for paper certificates, and their digital fingerprints stored on the blockchain, and Since the architecture is self-maintaining and open-source, It will be a great application added to the network.

Problem Statement

Every school/university around has its way of managing or maintaining its student records and transcripts. Of course, many of them usually do not share student information, such as transcripts for privacy reasons. Typically, in the case of international students, when one student tries for an admission in a foreign country, the student must get his transcripts evaluated by a third-party evaluator such as WES, an International evaluator.

For one reason, if somehow these transcripts are in a foreign language, there needs a translator and, most of the time requires to approach a third-party evaluator. When a person had started applying for a university in the United States, The university required him to first apply all the transcripts from his school and then had them evaluate through an external evaluator to match the grading system between the two countries, and it takes an average of 2-3 weeks to get the evaluation report. Table 1 is the third-party evaluator's cost summary.

So, a cryptographic database solution for recording the academic certificates will help to solve all these issues. By making all the official and unofficial transcripts of the student store in a blockchain, which are accessible through all over the world and can share to any Employer or a University And the transcripts stored on a blockchain system are immutable, therefore preserves the integrity of data.

Table 1: Academic transcript evaluation list

Company Name	Time	General	Course by Course	Trans
Validential	5 bus. days	75	139	24.99
A2Z Evaluations, LLC	3 weeks	195	335	
Academic Credentials Evaluation Institute,	7 bus. days	259	349	$59+
Academic Evaluation Services, Inc.	2-3 weeks	180	450	50
American Education Research Corporation	3-4 weeks	175	275	
Educational Credential Evaluators, Inc.	7 bus. days	165	275	
Educational Perspectives, nfp	1 week	125	185	
Educational Records Evaluation Service	3 months	340	490	$100+
Evaluation Service, Inc.	7-10 bus. days	145	230	
Foreign Academic Credentials Service	Unposted	310	460	
Foreign Credits, Inc.	7 bus. days	175	260	50
Foreign Credential Evaluations, Inc.	5-7 bus. days	175	275	
Foundation for International Services	2-4 weeks	140	350	$50+
Global Credential Evaluators, Inc.	3-4 weeks	200	270	
Global Services Associates	4 weeks	165	235	
Globe Language Services, Inc.	2 weeks	165	265	$50+
International Academic Credential Evaluat	2 weeks+	250	355	
International Consultants of Delaware, Inc.	4-6 weeks	225	325	85
International Education Research Foundati	3 weeks	170	250	
International Evaluation Services	2-4 weeks	140	250	
Josef Silny & Associates	2 weeks	130	230	
Lisano International	2-4 weeks	250	450	

Nature and Significance of the Problem

> Axact from Pakistan and University Degree Program (UDP) run by an American in Romania—have accounted for many hundreds of thousands of sales to Americans, and that is just the tip of the iceberg. More than half of all new PhDs in the U.S. each year are fake. (Gibson, 2017)

Academic transcripts are being generated falsely, and many agencies are producing false documents. Due to the lack of proper verification systems, many physical documents are being forged and are being distributed. On taking account of these issues, I would like to state that the need to develop a global system which is a trusted, tamper-proof system.

Objective of the Study

To develop an application on a blockchain framework, which is a role-based access system, takes inputs and stores all the changes performed on it, which supports an authentication system and high scalability.

Study Questions/Hypotheses

1. Is there a better framework other than HLF to develop this framework?
2. Cryptographic network, Is it secure enough?
3. How stable will be the network in handling upgrades and significant development changes?

Limitations of the Study

Since the cryptocurrency networks are still under development, a new framework or application based on blockchain technology is coming on to light/deployed every day. There is a deficient number of successful applications, and in turn, very few peer-reviewed articles available. Furthermore, a few of these frameworks require a new programming language to work on them.

Definition of Terms

A Distributed Ledger is one of the different types of data storage methods, which is shared and distributed between the participants of the decentralized network. This network can store transactions such as currency, property information, and any data that can be converted to bytes.

Blockchain is a distributed network of peer, where users can be participants and can work on complex algorithms to confirm the transactions.

Ethereum is an open software platform based on blockchain technology that enables developers to build and deploy decentralized applications (Higgins, 2017).

Hyperledger Fabric is a permissioned distributed blockchain infrastructure, initially contributed by IBM and hosted by Linux Foundation, It brings elements of confidentiality, privacy, and trust. It is an extensive scalable system that supports smart contracts.

Summary

This chapter discusses the basic introduction of the sudden hype of cryptocurrencies, how it can help develop cost-effective solutions. It talks about what the research is about and what are the limitations to it. The next chapter helps to understand more about blockchain technology's workflow and its frameworks in more detail.

Chapter II: Background and Review of Literature

Introduction

In this chapter, we are interested in different mechanisms, taxonomy, and various applications of blockchain technology. This chapter also deals with in detail explanation of blockchain frameworks which are, Ethereum and Hyperledger Fabric. This chapter also talks about the literature that we had previous literature related to the research area and provides an opinion about those works.

Background Related to the Problem

A Distributed Ledger is a technical implementation of a kind of data storage system, which has a distributed nature; all the peers in the network hold onto a copy of the ledger or in some cases, a partial copy of the ledger. This network can handle transactions supported by Proof of Work and consensus mechanisms.

Decentralization is a crucial concept in blockchain implementation. With this technology, many people can write records into this decentralized database, and a community of an honest user will control the record of information into the Distributed ledger. All the records write on to a ledger will be distributed among its nodes, and Every node is continuously updating the copy of the ledger on their database.

Blockchain helps store information about transactions in a distributed implementation. Some computers in the network are called nodes. They own a full copy of the blockchain. There will not be a central authority to distribute the information to the nodes. Overall, A distributed ledger is a database held and updated independently by each node in a network. The information will get distributed uniquely. The information is maintained and distributed by every participating node in the network.

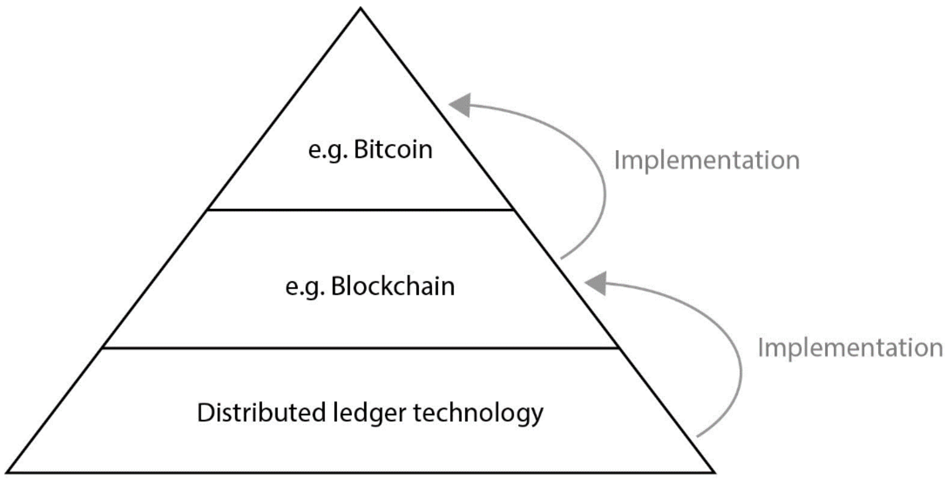


*Figure 1***:** Bitcoin as Blockchain 2.0 (Stevens, 2018)

Each node/stakeholder in the network function as per consensus, which will be specific to each network, which they agree on without any third-party interference. Consensus in the network is what brings in trust among the nodes. Every record in the distributed ledger consists of a timestamp, hash value of corresponding transactions, and a unique signature, which makes the transactions immutable in the ledger.

Blockchain is a decentralized network which connects multiple nodes, and the peers in the network manage all the transaction on the ledger.

How Blockchain works on a distributed ledger. Blockchain stores information about transactions in a distributed manner. Some computers in the network are called nodes, and they own a full copy of the blockchain. There will not be a central authority to distribute the information to the nodes. Figure 2 explains how a blockchain system and its potential applications. A distributed ledger is a database hosted and updated by each node in a network. The information distribution among the nodes is unique.

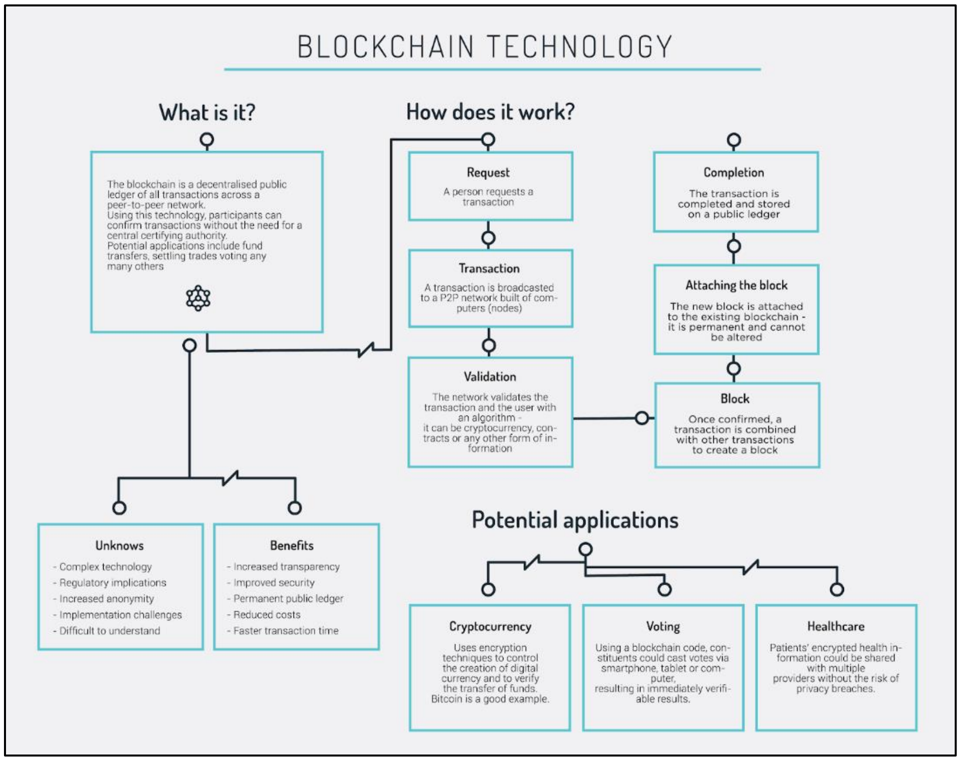

BLOCKCHAIN TECHNOLOGY
What is it?
The blockchain is a decentralised public ledger of all transactions across a peer-to-peer network.
Using this technology, participants can confirm transactions without the need for a central certifying authority.
Potential applications include fund transfers, settling trades voting any many others
How does it work?
Request
A person requests a transaction
Transaction
A transaction is broadcasted to a P2P network built of com-puters (nodes)
Validation
The network validates the transaction and the user with an algorithm - it can be cryptocurrency, con-tracts or any other form of in-formation
Block
Once confirmed, a transaction is combined with other transactions to create a block
Attaching the block
The new block is attached to the existing blockchain - it is permanent and cannot be altered
Completion
The transaction is completed and stored on a public ledger
Unknows
- Complex technology
- Regulatory implications
- Increased anonymity
- Implementation challenges
- Difficult to understand
Benefits
- Increased transparency
- Improved security
- Permanent public ledger
- Reduced costs
- Faster transaction time
Potential applications
Cryptocurrency
Uses encryption techniques to control the creation of digital currency and to verify the transfer of funds. Bitcoin is a good example.
Voting
Using a blockchain code, con-stituents could cast votes via smartphone, tablet or com-puter, resulting in immediately verifi-able results.
Healthcare
Patients' encrypted health in-formation could be shared with multiple providers without the risk of privacy breaches.

Figure 2: Overview of Blockchain technology (Stevens, 2018).

Types of Blockchain. Blockchain is a chain of blocks, a chained ledger that holds the list of transactional data. The specific term for the first block in the chain is Genesis Block. A Genesis block will not have any parent blocks, as shown in Figure 3.

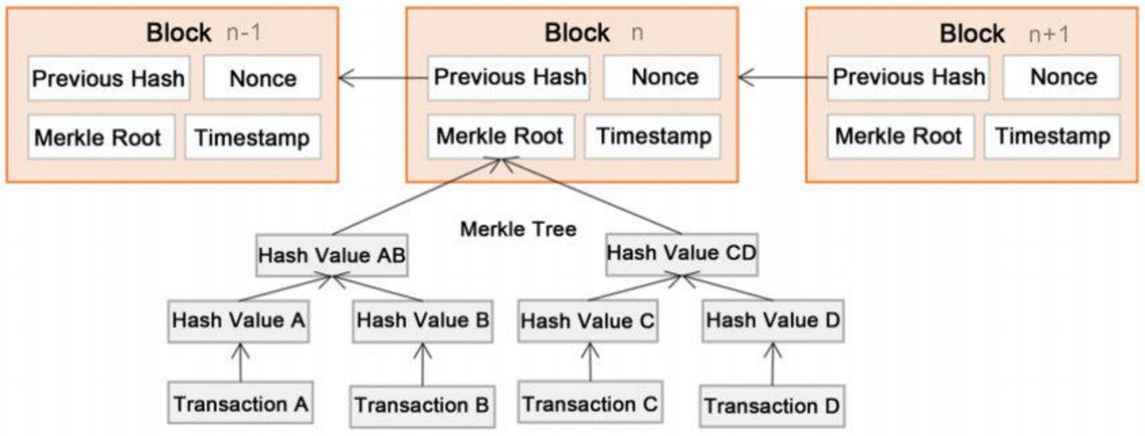


Figure 3: Merkle root hash generation from transactional hashes (Hong, Wang, Cai, & Leung, 2017)

Block, as shown in Figure 4, has Block Header and body. Block header is an 80-byte long string and carries the essential information that helps in maintaining consistency and immutability of the chain.

1. A 4-byte long Block version.
2. A 32-byte long Merkle root has the hash of all the transactions combined in a block
3. A 4-byte timestamp of the block which helps in avoiding double spending
4. 4-byte long difficulty target for the block, this POW difficulty index is calculated by averaging transfer rate. If the blocks are processing with high velocity, the difficulty should increase proportionally. The framework is set to handle the difficulty intelligently.

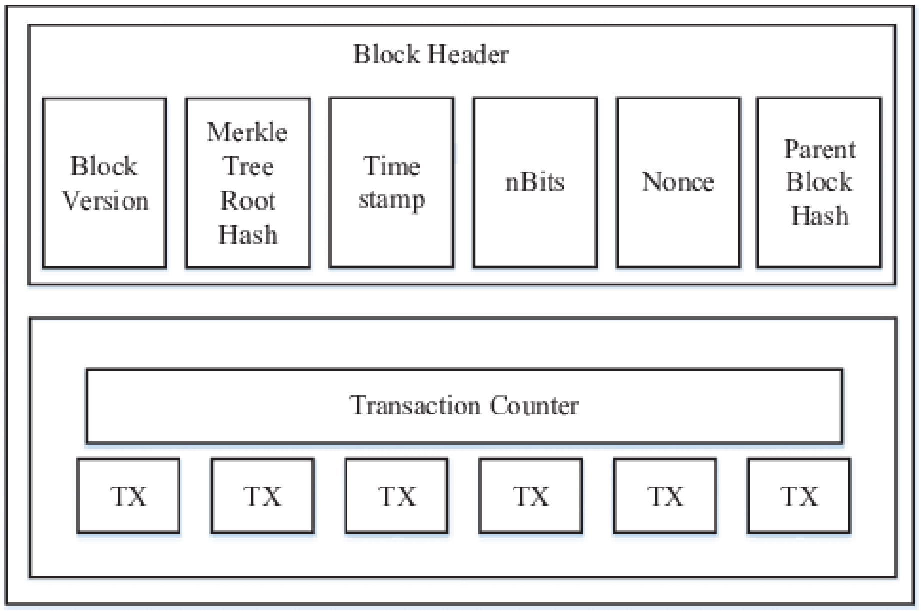


Figure 4: Structure of a block (Hong et al., 2017).

5. Nonce is a 4-Byte field, which is a random string that needs to be generated using trial and error method, which should be appended to the hash of the current header so that the hash meets the difficulty requirements.
6. Parent block hash is a 256-bit and is the hash value calculated for the previous block, Thus forming a chain of transactional blocks.

The body of the block mainly consists of transaction counter and transactions. The size of each constituting block will affect the total number of transactions that can be held on a block.

Cryptography in Blockchain. The concept of Cryptography is the backbone of the blockchain. Blockchain uses public-key cryptography, It has a public key and private key to perform tasks. Public keys, as the name suggests, are distributed, whereas the private key should be personal to a user.

One can encrypt information using a person's public key, which can only be reversed to its original state, in technical terms, decrypted by using the corresponding private key. This public-private key encryption method brings in the concept of Data integrity, which means verifying that the data reached its destination unharmed and uncorrupted. So, using the private key, a digital signature can be generated so that with the respective public key associated with that, anyone can verify the integrity of the data over the network.

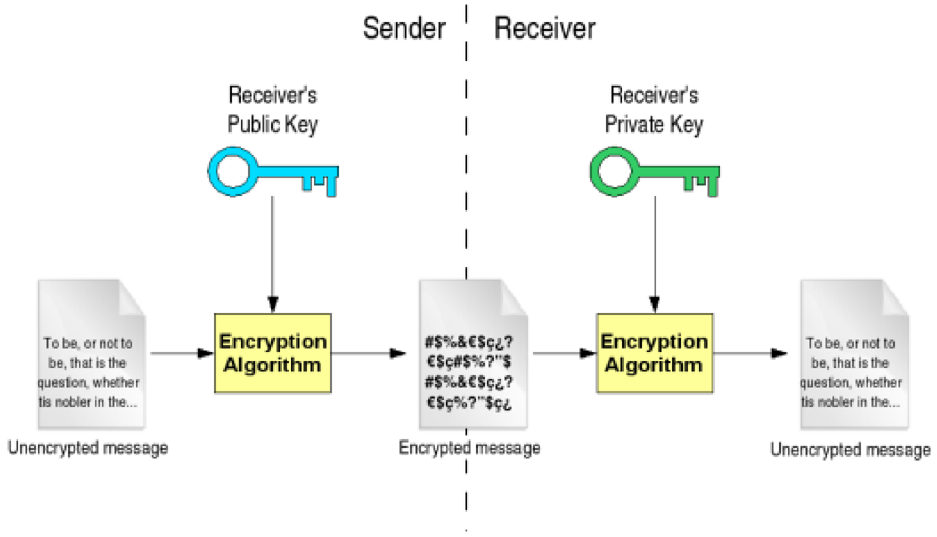

Sender
Receiver
Receiver's Public Key
Receiver's Private Key
Encryption Algorithm
Encryption Algorithm
To be, or not to be, that is the question, whether tis nobler in the...
#$%&€$ç¿? €$ç#$%?"$ #$%&€$ç¿? €$ç%?"$ç¿
To be, or not to be, that is the question, whether tis nobler in the...
Unencrypted message
Encrypted message
Unencrypted message

*Figure 5***:** Key-based asymmetric algorithm.

A Hashing algorithm has a significant place in the cryptography, A hashing algorithm applied to an input of any size results in a fixed-length output, depending on the hashing function used. Any hash function divides the input into blocks of a certain length and process them using several mathematical functions and produces a hash result/ hash digest.

Since the input data is divided up and each block is processed one at a time, the output of one block's hash digest is carried forward into the next one and the algorithm process it, thus the output will be the of the combined value of all the inputs, this way if somehow one bit of data changes the hash digest will be an entirely different value, This phenomenon is known as the Avalanche effect.

	Hash output size (bits)	Cycles per block	Throughput at 100 KHz (Kbps)	Area GE
MD4	128	456	28	7350
MD5	128	612	20.9	8400
SHA-1	160	1274	12.55	8120
SHA-256	256	1128	22.7	10868
NAME	256	96	266.67	8100

*Figure 6***:** Hash algorithm comparison.

An encryption algorithm, when applied to the information, which was supposed to be input for a proposal, and is signed using the private key, and Its respective key which is being shared up over the network, can help in verifying the integrity of the data received. This secure method ensures that every transaction can be easily verifiable and logged. As long as the private key is secure, none of the transactions are tied to anyone. The digitally signed transactions are distributed all over the network.

Every user who participates in the blockchain network with the generated address never has to reveal his identity. Consensus algorithms in blockchain ensure the data consistency is the P2P network.

Hashing function has two unique properties,

1. It is a one-way function,
2. Produces a fixed-length output.

The mathematical functions applied for the hashing function make sure that the input never be will be generated from the hash digest. That is the reason why Hash digest technical term is digital fingerprint of the data processed through the hashing algorithm.

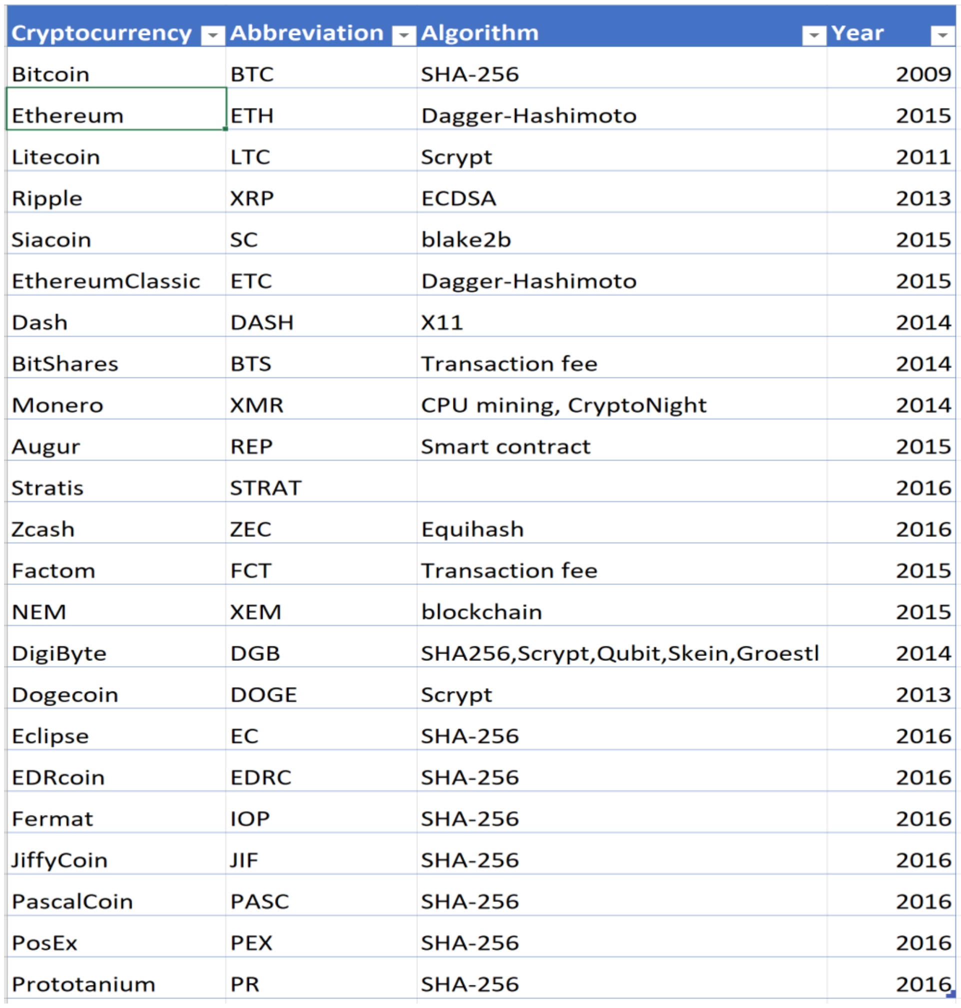
Moreover, the same input must always produce the same result. It should not produce a different hash digest. Restoring input from the hash digest is a lost cause; there should be no way to break the mathematical hashing process to see the original input.

Even a small change in the input should affect an entirely different hash digest, even changing the case of a character in the string should alter the hash digest. Moreover, most importantly, the hash digest should always be of a fixed size. And this entire process of producing the hash digest should use minimal computational power.

Any hash function should be one way and collision-resistant, which makes them an essential application for password protection and digital signatures.

The existing most popular hashing algorithms MD-4 and MD-5 produces hash values of length 128 bits, and SHA-1 results in a 160-bit hash digest, which means in case of collision attacks, these algorithms cannot provide security for more than 64 and 80 bits respectively. So, to improve security, the goal is to use better algorithms to prevent these collision attacks, AES offers in three crypto variable sizes of 256, 384, and 512. SHA- 265 is supposed to provide 128-bits of security against the collision attacks.

Table 2**:** Cryptocurrencies and hashing algorithms.



Cryptocurrency	Abbreviation	Algorithm	Year
Bitcoin	BTC	SHA-256	2009
Ethereum	ETH	Dagger-Hashimoto	2015
Litecoin	LTC	Scrypt	2011
Ripple	XRP	ECDSA	2013
Siacoin	SC	blake2b	2015
EthereumClassic	ETC	Dagger-Hashimoto	2015
Dash	DASH	X11	2014
BitShares	BTS	Transaction fee	2014
Monero	XMR	CPU mining, CryptoNight	2014
Augur	REP	Smart contract	2015
Stratis	STRAT		2016
Zcash	ZEC	Equihash	2016
Factom	FCT	Transaction fee	2015
NEM	XEM	blockchain	2015
DigiByte	DGB	SHA256,Scrypt,Qubit,Skein,Groestl	2014
Dogecoin	DOGE	Scrypt	2013
Eclipse	EC	SHA-256	2016
EDRcoin	EDRC	SHA-256	2016
Fermat	IOP	SHA-256	2016
JiffyCoin	JIF	SHA-256	2016
PascalCoin	PASC	SHA-256	2016
PosEx	PEX	SHA-256	2016
Prototanium	PR	SHA-256	2016

The hashing algorithm we are using for this research will be SHA-256, which is the algorithm used for bitcoin as listed in Table 2.

The steps involved in the SHA-256 hash algorithm are, the message in action here is first padded within its length of 512, which is a message M with right padding and then parsed into blocks $M^{(1)}$; $M^{(2)}$ ……$M^{(N)}$.

Each block gets prepared and used at one time, and the result of one stage gets passed into the next stage,

$$H^{(i)} = H^{(i-1)} + C_{M^{(i)}}(H^{(i-1)}),$$

Where C is the algorithms compression function. The final results will be the hash of the given message.

SHA-256 compression function acts on 512-bit message blocks and 256-bit hash intermediate values; it is a 256-bit cipher algorithm which will encrypt the intermediate hashes by using the 512-bit message strings. The initial hash values are the square roots of the first eight prime numbers

$$\begin{aligned}
H_1^{(0)} &= \texttt{6a09e667} \\
H_2^{(0)} &= \texttt{bb67ae85} \\
H_3^{(0)} &= \texttt{3c6ef372} \\
H_4^{(0)} &= \texttt{a54ff53a} \\
H_5^{(0)} &= \texttt{510e527f} \\
H_6^{(0)} &= \texttt{9b05688c} \\
H_7^{(0)} &= \texttt{1f83d9ab} \\
H_8^{(0)} &= \texttt{5be0cd19}
\end{aligned}$$

The hash function compression function shown in Figure 7.

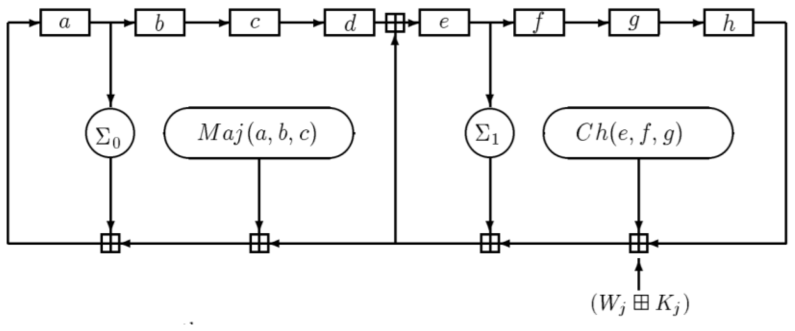


Figure 7**:** jth step of the compression function C.

The input blocks of message schedule W gets passed, one after the other, the function represented below as a graph. The input blocks get shuffled as shown in Figure-8, and the shuffle function takes inputs as Wi(t), and the message schedule input block $w^i(t)$ and outputs a hash $\omega^i(t+1)$.

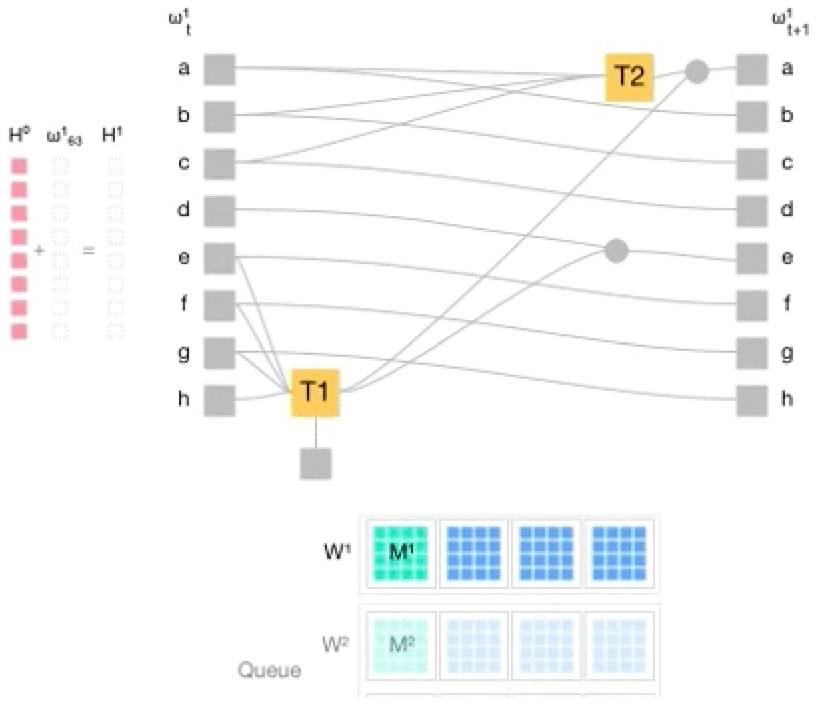


*Figure 8***:** Shuffling the blocks (Madeira, 2019).

POW in blockchain is a consensus algorithm, which requires exceptionally high processing power and is very time consuming to produce a piece of data. This theoretical concept helps systems, ensure that security, integrity, and consensus throughout the blockchain network. However, advantageous in helping quick verification of the solution. Hash cash is a POW algorithm for Bitcoin. For a network to arrive at consensus, it performs proof-of-work on its transactions block.

When peers mine a block, it should satisfy consensus. So, the miners in the network needs to complete the POW to verify all the transactions in a given block. The difficulty set by the consensus will not be the same all the time. It varies every time so that new blocks are in for processing every 10 minutes, as shown in Figures 9.

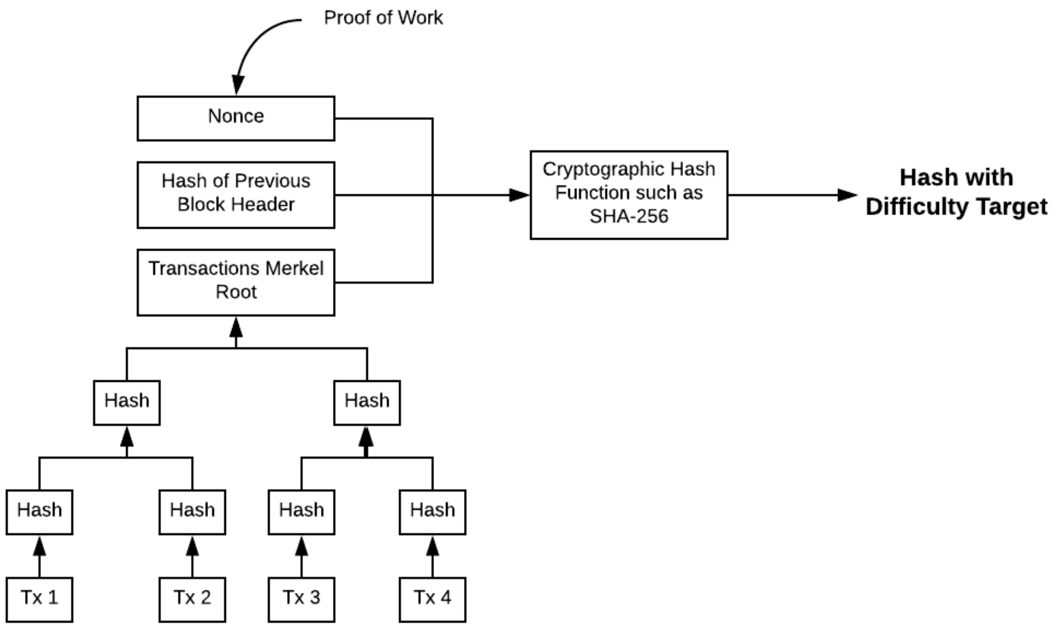

Proof of Work
Nonce
Hash of Previous Block Header
Transactions Merkel Root
Cryptographic Hash Function such as SHA-256
Hash with Difficulty Target
Hash
Hash
Hash
Hash
Hash
Hash
Tx 1
Tx 2
Tx 3
Tx 4

Figure 9: Mining a block of transaction (Kumar, 2018).

Expected is that many of the miners try to perform proof of work on a given set of transactions and add the block to the network. Nevertheless, given the complexity, there is a very low probability of being successful, so it is almost impossible to predict which miner will complete the Proof of work algorithm on the block of transactions and add the computed block to the ledgers.

To Calculate the hash value, a miner needs to start hashing out each of the transactions using the SHA-256 algorithm. The order of fields for transaction hashing is version number, then comes in input counter, list of inputs, output counter, list of outputs, lock time. The order of fields for inputs is a previous transaction hash, output index, input script length, a sequence number. When the miner was done calculating the hash of all these transactions, then hash Merkle root should be calculated.

To calculate Merkle root hash, Start with each pair of adjacent transactions are grouped and computed hash to create an upper-level hash value, every pair of transactions gets processed like explained, and then secondly, two adjacent upper-level hashes are combined and are hash out until It generates a unique Merkle root hash. In case someone tries to modify a single bit in a transaction, due to this algorithm, the Hash Merkle root changes and changes help in identifying the change.

After obtaining the unique hash Merkle root for the transactions, the block header can be hashed to get the final block hash. To calculate the block hash, the order of fields is version, earlier computed block hash, hash Merkle root, time, target, nonce. These required fields need to be put together and are hashed twice to get the final hash of the block of transactions.

A node/miner compiles all the proposals broadcast in the network and verifies all these proposals by digital signatures associated with them. Then the miner puts all the transactional records together and calculates the hash along with the unique hash Merkle root which makes sure its total hash be less than the target hash,

$$H\ (N,\ P\_Hash,\ Merkle\_Root) < Difficulty$$

Where N refers to nonce variable,

P_Hash is the hash value of the block mined earlier to this,

This calculation can be achieved by altering nonce, starting with value 0, and incrementing it every time and recalculate the hash until the total hash is less than that of the target hash. And then, when achieved, the node/miner will broadcast this block onto the network. When a node receives a first block in the network, it will verify all the transactions in the block and will verify the hash value of the block.

Hyperledger Fabric is a permissioned distributed blockchain infrastructure, initially contributed by IBM and hosted by Linux Foundation, It brings elements of confidentiality, privacy, and trust. It is an extensive scalable system that supports smart contracts. To improve cross-industry blockchain technologies, the Hyperledger project hosted by Linux Foundation has developed an HLF framework. HLF possesses some unique characteristics which will help speed up the blockchain adoption. Unlike Ethereum, Developing smart contracts can use programming languages such as Node.js, Java, or Python.

Moreover, the consensus protocol can be swapped depending upon the organizational requirements. Furthermore, It enables TLS communication between the participants of the network.

A Hyperledger Fabric architecture also offers the security of channels, meaning that a different ledger can for different channels, which can store different types of records. This framework can be tuned to business requirements and does not need to establish a cryptocurrency. The HLF ledger has the following core components, Certificate authority (MSP), Chaincode Containers, nodes/peers, Ordering service, Channels, and Shared Ledger, as shown in Figure 10.

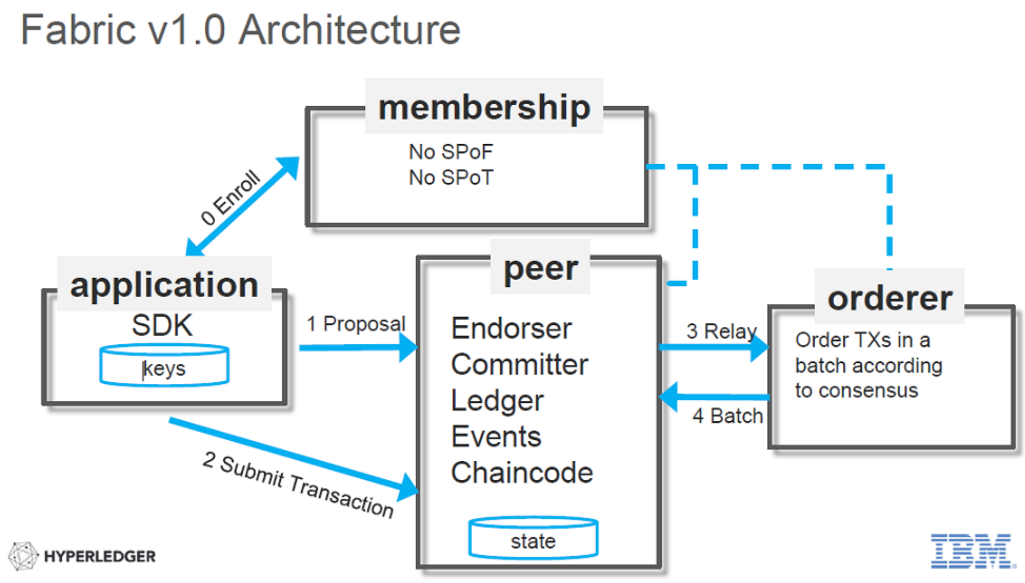


Figure 10: Hyperledger fabric architecture (Sean, 2017).

When a user enrolls into The network through MSP (Membership service provider) and submits a proposed transaction to the endorsing peer, the peer executes the chain code (smart contract), endorses it, and returns the transaction to the user/client. The user then submits the proposal to the orderer. Orderer is the automated service that bundles in all transactions. The orderer service verifies the received transaction and adds this transaction along with other transactions, sorts the transactions, and creates a block of transactions. Then the peers validate the transaction in the returned block and commit the ordered block to the ledger.

Membership service provider (MSP) is a Certificate authority, which provides authentication, registration, and certificate generation services to its users. A Membership service provider also generates a public key/Private key pair for its members. Each member of the network should present a digital certificate to join the HLF network. MSP will create these certificates for all its members in the network.

Chaincode is a smart contract that can be installed on a peer and is used to handle the data on the ledger, This programming language can be written in python, Java or node.js, to handle the consensus agreed upon by users, when a user requested a transaction, a chain code is written to make sure the proposal meets all its constraints, A chain code can invoke to query the ledger in the database or to invoke other chain codes, with appropriate permissions.

The HLF network supports multiple channels/ledgers to avoid data exchange. A channel acts as a messenger as in, all the users belonging to that channel will be able to see the messages in that channel. HLF allows users to be part of multiple channels, but cross-communication between the channels is not possible. Thus Channels add up an extra layer of security to the HLF architecture. Ledger/Blockchain is a place to maintain all the transactions local to a channel. It should be tamper-proof (The records in the ledger are immutable). It should be storing all the transactions as well as the successful and unsuccessful transactional logs. HLF also has a database system that stores the current state of the system, and termed as the World State database; this will store the current or most recent state of all the channels in the network. The World State database will store the most recent state of all the channels, whereas Transactional log stores all the transactions that happened till date. World State database's primary purpose is for query performance optimization, instead of working through all the transactional log, we can get the most recent state of all the channels through the World State database.

Nodes host both the ledgers and chaincode; MSP authenticates any user or peer in the network and can be of either of these endorsing, ordering, or committing peers.
As explained earlier in the transaction flow, each peer has bear responsibility in each phase.

Initially, when a user sends a transactional proposal to the endorser. An endorsing peer will validate the proposal. A transaction that is valid/legitimate should achieve enough endorsements from the network. Each endorsing peer runs the chain code on the proposal and generate a response and endorse it by digitally signing the response.

Therefore, the response from the endorsing peer is a digitally signed response from the chain code, which makes it tamper-proof. Then the endorsing peer sends the response back to the client. The client then will create a transaction message by putting together all the responses from the endorsing peers. The constraint for the transaction message is that the proposal needs endorsements from more than fifty percent of the peers.

An ordering peer is the which collects all the transaction messages from the client. A transaction message is the one which is a bundle of all the digitally signed responses from the endorsing peers. The ordering peers collect these transaction messages from all the channels, chronologically sorts them, and bundles them all together into blocks. Each block is a list of transactions ordered in chronological order grouped by channel. The ordering service then sends the block to the leader peer of each channel.

The committing peer only hosts a leger; it does not host chain code like the endorsing peer, This means the committing peer cannot be an endorsing peer, but an endorsing peer can always be a committing peer. When the committing peer receives the block of transactions from the ordering peer, each peer independently validates all the transactions and check if all the transactions satisfy the endorsement policy. Then they will update all the valid and invalid transactions into the ledger, which is a distributed database among the network. The invalid transaction will help in identifying troubling peers.

A protocol called Gossip protocol used in HLF to share the workload between the participating nodes. An online peer always broadcast alive signal to all other peers making them know that it is available. If a peer stops sending in the alive signal, it will be considered dead and will get its membership revoked from the channel.

Moreover, peers in the network can either elect their Anchor peer to communicate with the ordering service to get the block of a transaction of the channel. An Anchor peer can also communicate with an anchor peer of another organization. This node takes the responsibility of propagating the received blocks to the nodes in the channel.

The ordering service forwards the block of transactions only to the leader of the peer to save the bandwidth. A leader can either be selected by the administrator or by a dynamic voting among the nodes, where voting happens at regular intervals of time.

Literature Related to the Problem

In Mukhopadhyay, Skjellum, and Hambolu’s (2017) paper “A Brief Survey of Cryptocurrency Systems”, the authors have discussed the underlying blockchain architecture and how the initial models faced the privacy and security issues and also explained how the recent developments on this technology are supporting the business models. The paper talks about the basic structure of the bitcoin block, the Genesis block, and how the links between the blocks will happen at a very high level. The paper also discusses different consensus mechanisms and hashing algorithms in various cryptocurrencies.

In Chen, Xu, and Lu’s (2018) paper “Exploring blockchain technology and its potential applications for education”, the authors researched a lot of cryptocurrencies and their potential. This paper also lists out many advantages of adopting the blockchain technologies over traditional methods. This article also introduced the features and advantages of blockchain technology following by exploring some of the current blockchain applications for education. This paper also talks about how blockchain technology can help reduce the degree fraud, and how the blockchain can validate a person’s certificates. This paper also talks about various future innovative ideas that blockchain can help revolutionize the field of education.

In Ark’s (2018) paper “20 Ways Blockchain Will Transform (Okay, May Improve) Education” article, the author had discussed various industrial applications of blockchain and explained about various frameworks under development for industrial applications. This article also listed out various industries which are trying to develop innovative solutions based on blockchain frameworks.

In Cachin’s (2016) paper on “Architecture of the Hyperledger Blockchain Fabric”, the author has discussed about the underlying architecture of Hyperledger Fabric framework. Furthermore, it discusses how a permissioned based model of blockchain can control who participates invalidation, and the protocol helps in building industrial models. This paper also discusses different consensus protocols that can be employed depending on the industrial structure.

Literature Related to the Methodology

In Gräther et al.’s (2018) paper “Blockchain for Education: Lifelong Learning Passport”, the author had discussed the importance of certificates in a person’s personal and professional career. The author also describes the conceptual system overview and then presents in detail the platform implementation, including management of certification authorities and certificates, smart contracts as well as services for certifiers, learners, and third parties such as employer. In this implementation study, the author had proposed to build an application on the Ethereum framework (Gräther et al., 2018), which generates digital certificates and stores the document’s hash in the blockchain database, and any certificate can easily get verified against the hash value in the database. This paper also documented with many user test cases to help testing the application.

In Valenta and Sandner's (2017) paper about "Comparison of Ethereum, Hyperledger Fabric, and Corda", the author has discussed about various frameworks built upon the blockchain database and mentioned advantages of one framework over the other. Also, in this paper, the author had discussed implementation projects being carried out on each framework.

Summary

This chapter gives a detailed overview of a Distributed ledger Blockchain technology, and about cryptocurrencies which are actively in constant development on them and technology involved, such as Consensus, proof of work algorithms, encryption, taxonomy of blockchain systems and most importantly data mining methods in basic blockchain system, bitcoin and in Hyperledger Fabric framework. The next chapter covers the design of the approach to building up the solution.

Chapter III: Methodology

Introduction

In this chapter, we will discuss in-depth about process to implement a HyperLedger fabric framework using a Linux machine and what are the steps to be taken to develop the application. Apart from we will be discussing hardware and software components needed to implement this.

Design of the Study

Designing of this implementation project requires the study of various frameworks and pre-designed models, which includes learning the new frameworks, development plan, and writing user test cases to deal with any issues that come up during the implementation. For this, a development environment with at least four nodes or participants.

Designing the development network involves the following steps

1. Learn and understand all the execution examples of the Hyperledger Fabric Framework.
2. Try to analyze each algorithm and consensus mechanism in this framework.
3. Understand the different components of the framework and their roles.
4. Create and set up virtual machines where the nodes will live.
5. Install and configure the HLF system on a Linux machine with virtualization.
6. Make a test run inserting a few records to make sure that the implementation works.

The design flow chart developed shown in the figure below; The HLF framework should be set up with a minimum of 4 nodes. As shown in Figure 11 use case, Workflow should be as follows,

1. An instructor and student will be peers in the HLF network.
2. The instructor will then send in a request to insert in a student marks/grade request.

3. Endorsing peers will then write a smart contract (Chaincode) on this record and moves this transaction to Ordering peer.

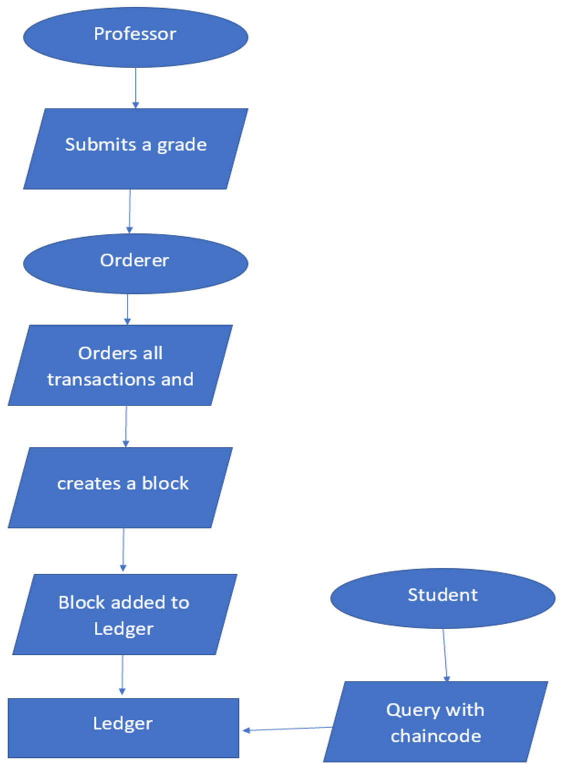


Figure 11: Flowchart for implementing the certificate system.

4. The ordering peer than validates all the proposals and sorts them out alphabetically, and then the block gets generated and forwarded to the committing peer.
5. Committing peer then validates the received block from Orderer, commits the block onto ledger, which is immutable and tamper-free.

Data Collection

The data required for this implementation is available and is accessible through various scientific journals and repositories. Since the tool used for this implementation is HyperLedger fabric, which is an open-source software developed by IBM. The data collection for this project is going through project documentation and looking through various designed project templates in academic and scientific repositories. The research included going through various white papers, conference papers, various technical blog sites, and repositories to get the up to date information.

Tools and Techniques

To implement the project, we need to get the Hyperledger fabric framework, which runs on the docker platform. Since we are developing a multi peer network using the architecture, docker helps us create containers for the peer and the corresponding services. Docker containers are lightweight alternatives for virtual machines.

Docker containers need a Linux or Unix based host machines. Docker containers do not need a preallocated RAM or disc space. The containers dynamically use up RAM and disk space as required. Docker containers created by docker use the Host operating system as shown in Figure 12,

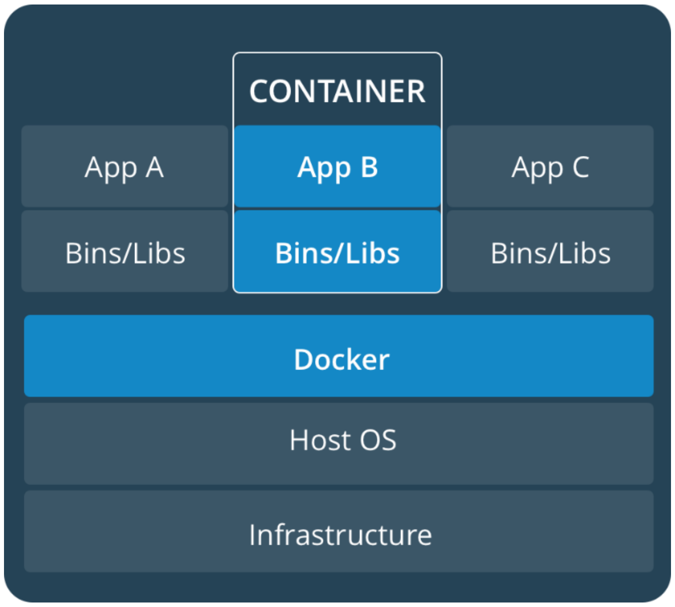


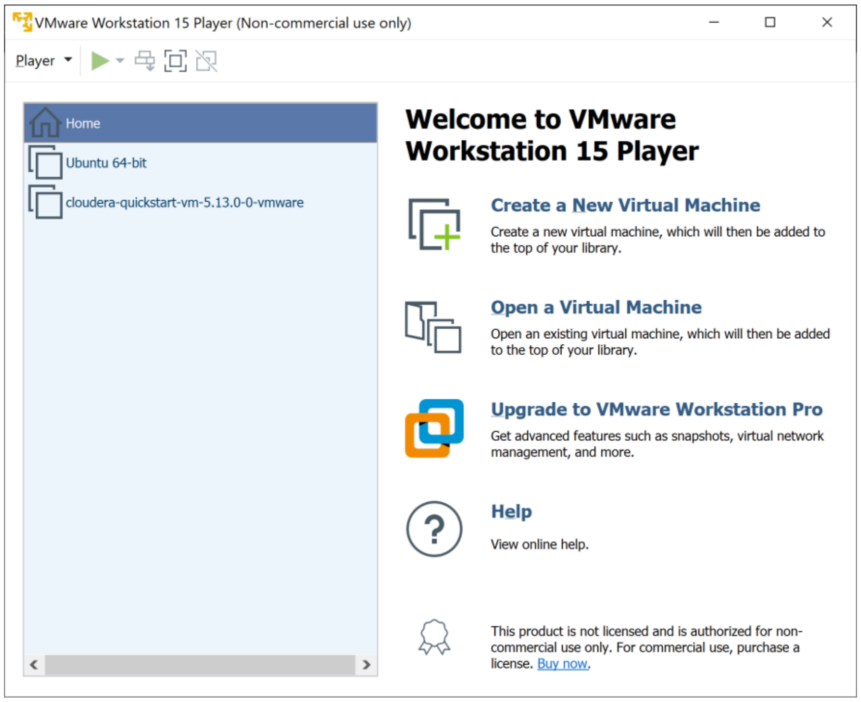
Figure 12: Docker container (Orientation and setup, 2019).

The docker containers have applications running on them, and these containers hold the required information regarding these corresponding applications such as Binaries and libraries. So on top of a Host machine, there will be a docker engine, and there will be multiple containers running on the Docker engine. So the libraries and binary files local to each application will be stored locally in each container.

To work with the Hyperledger fabric architecture provided docker images and containers, the project implementation will be carried out on a Linux machine. In order to install a Ubuntu operating service, It needs a Virtual machine installer.

A Virtual machine Installer such as VMware Workstation Player helps in running multiple operating systems on a single host machine. It enables the host user to set up multiple virtual machines with a guest operating system on a single host physical machine and run them simultaneously utilizing the host physical machine resources. VMware provides a complete virtual environment, i.e., a completely independent virtual hardware to the guest operating system.

A user can work on work on the guest machine, pause and make a copy of the virtual machine and use a copy of the machine on any other host machine, which makes it highly portable. This feature will be useful in development projects such as this.



VMware Workstation 15 Player (Non-commercial use only)

Player

Home

Ubuntu 64-bit

cloudera-quickstart-vm-5.13.0-0-vmware

Welcome to VMware Workstation 15 Player

Create a New Virtual Machine

Create a new virtual machine, which will then be added to the top of your library.

Open a Virtual Machine

Open an existing virtual machine, which will then be added to the top of your library.

Upgrade to VMware Workstation Pro

Get advanced features such as snapshots, virtual network management, and more.

Help

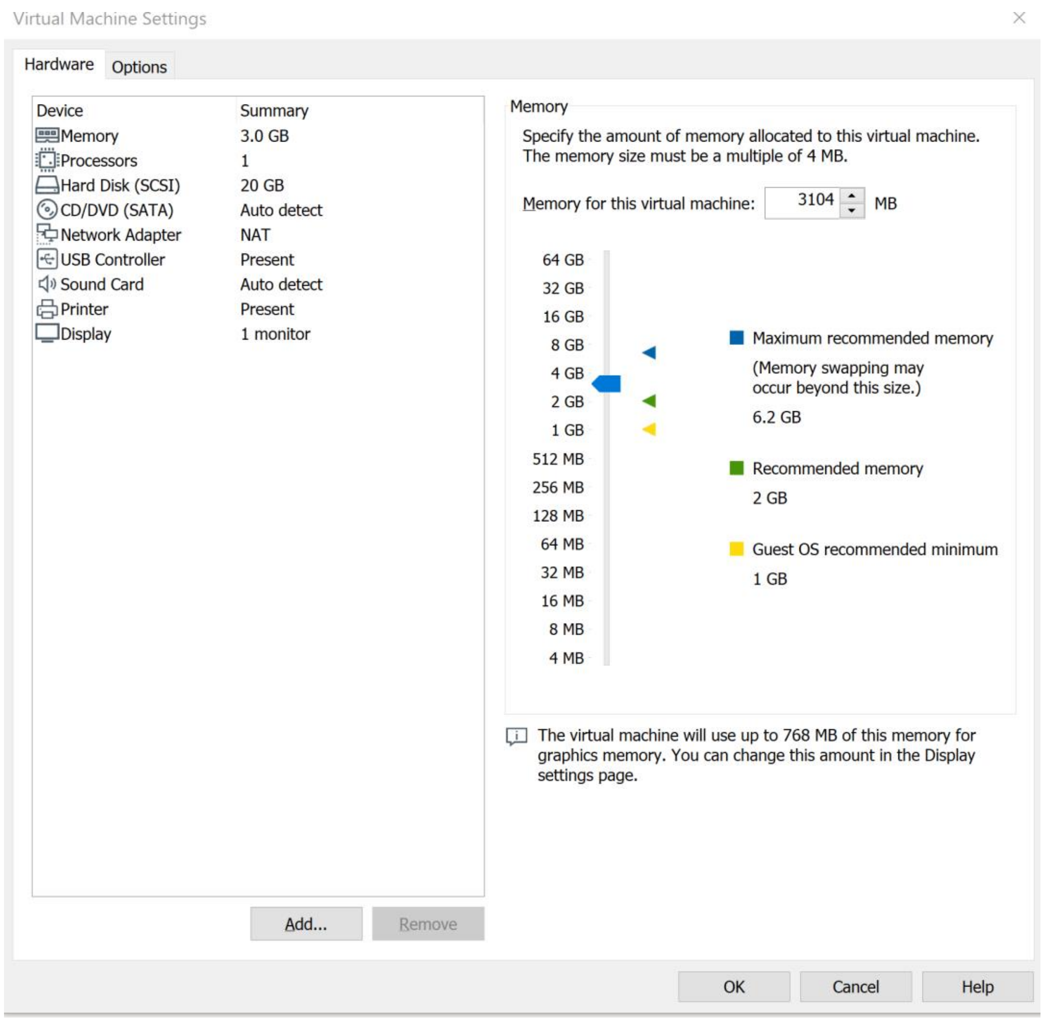
View online help.

This product is not licensed and is authorized for non-commercial use only. For commercial use, purchase a license. Buy now.

Figure 13: VMware workstation homepage.

For this project implementation, an Ubuntu 64 bit Operating system will support the cause, and VMware helps us in mounting up the Operating system.

VMware allows users to set up system resources such as RAM and disk space for guest Operating systems since the project setup is a multi-peer network and requires much computational power, a three GB RAM, and 20 GB disk space, leaving the rest as defaults as shown in Figure 14.



Device	Summary
Memory	3.0 GB
Processors	1
Hard Disk (SCSI)	20 GB
CD/DVD (SATA)	Auto detect
Network Adapter	NAT
USB Controller	Present
Sound Card	Auto detect
Printer	Present
Display	1 monitor

Memory

Specify the amount of memory allocated to this virtual machine. The memory size must be a multiple of 4 MB.

Memory for this virtual machine: 3104 MB

64 GB, 32 GB, 16 GB, 8 GB, 4 GB, 2 GB, 1 GB, 512 MB, 256 MB, 128 MB, 64 MB, 32 MB, 16 MB, 8 MB, 4 MB

🟦 Maximum recommended memory
(Memory swapping may occur beyond this size.)
6.2 GB

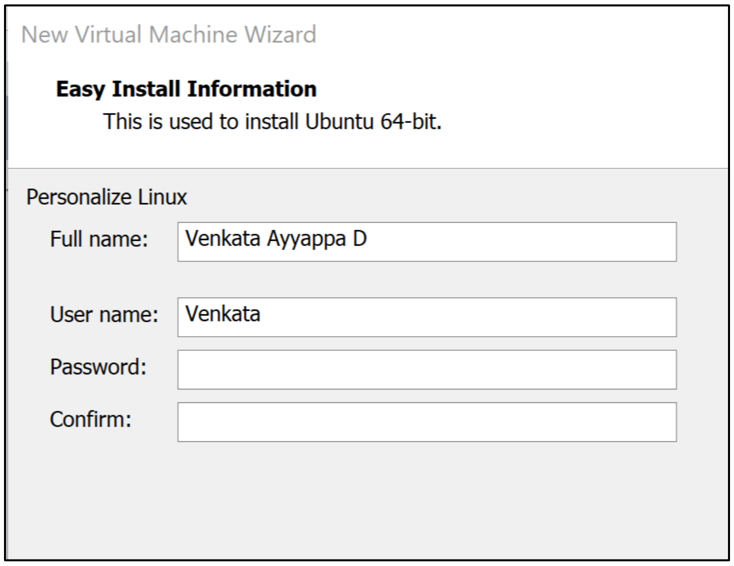
🟩 Recommended memory
2 GB

🟨 Guest OS recommended minimum
1 GB

The virtual machine will use up to 768 MB of this memory for graphics memory. You can change this amount in the Display settings page.

*Figure 14***:** Virtual machine settings.

The next step is installing the operating system on the machine, which involves setting up the user and login details for Ubuntu OS, as explained below in Figure 15. Then the VMware software initializes the user details on the machine and boots up the operating system on the host physical device.



New Virtual Machine Wizard

Easy Install Information

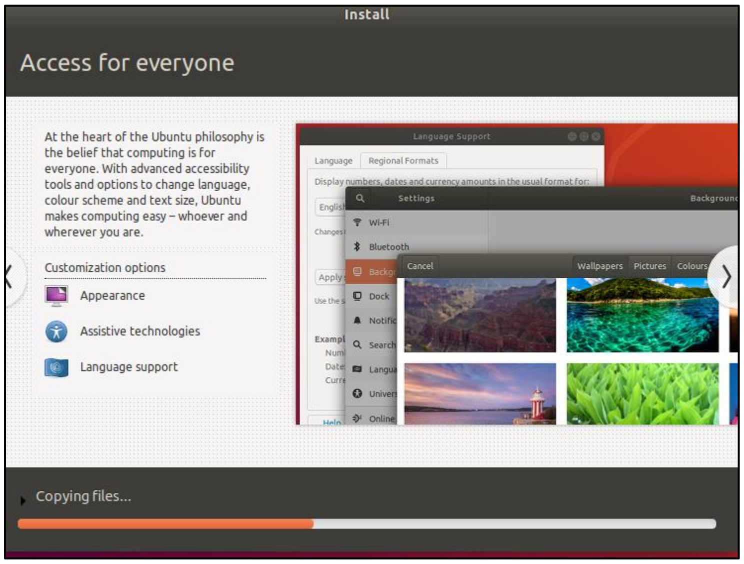
This is used to install Ubuntu 64-bit.

Personalize Linux

Full name:	Venkata Ayyappa D
User name:	Venkata
Password:	
Confirm:	

Figure 15: Virtual machine setup.

Then the VMware software initializes the user details on the machine and boots up the operating system on the host physical device. VMware Player will boot up the machine, configure all the hardware components required, and Initialises the operating system with defaults, as shown below in Figure 16; this may take a while and mostly depends upon the version.



Install

Access for everyone

At the heart of the Ubuntu philosophy is the belief that computing is for everyone. With advanced accessibility tools and options to change language, colour scheme and text size, Ubuntu makes computing easy – whoever and wherever you are.

Customization options

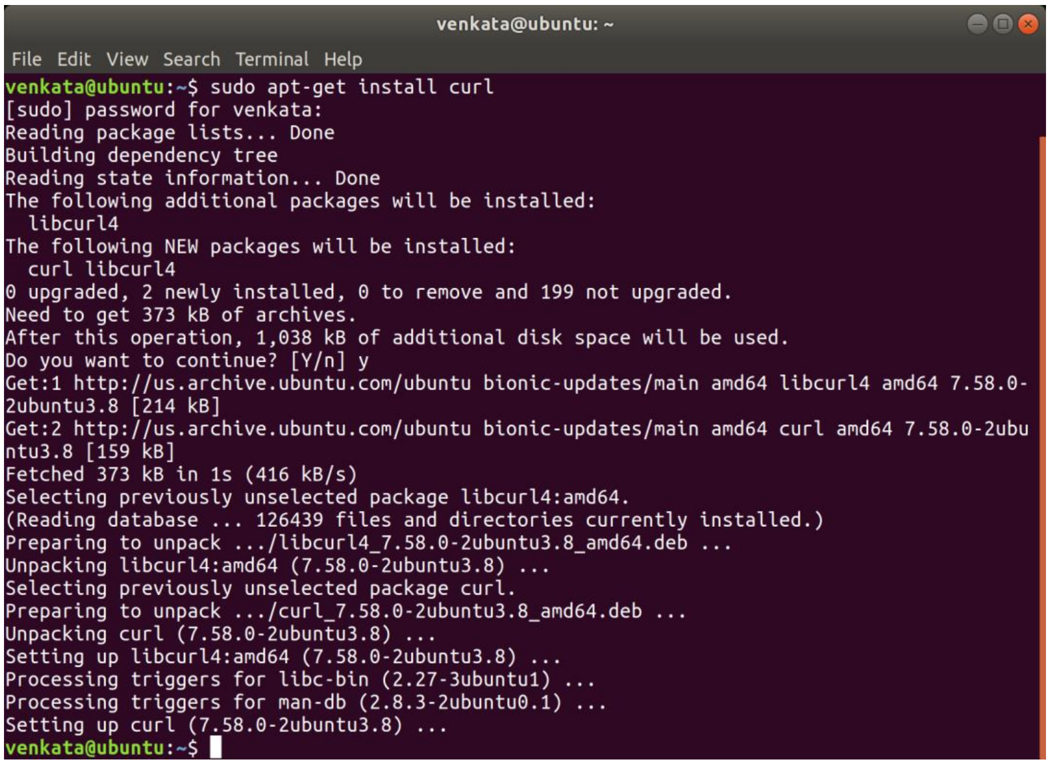
Copying files...

Figure 16: Ubuntu installation screen.

Once the Virtual machine's Operating system is up and running, log-in with the user credentials and Start to set up the Linux virtual machine ready to use HyperLedger fabric architecture. Open the terminal on the VM and Install CURL as a superuser using the following command.

```
sudo apt-get install curl
```

cURL is a handy command-line browser/tool or client URL library, and It is used to interact with servers using the command line. It helps in sending in data or extract data from servers. cURL allows users to upload/extract multiple files with a single command. Multiple URLs can be specified in a single command and are downloaded/uploaded into the server in the given order of the URL using the supported protocols. cURL tool will help in downloading Hyperledger fabric prebuild docker images that help in building up specific containers.



```
venkata@ubuntu:~$ sudo apt-get install curl
[sudo] password for venkata:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libcurl4
The following NEW packages will be installed:
  curl libcurl4
0 upgraded, 2 newly installed, 0 to remove and 199 not upgraded.
Need to get 373 kB of archives.
After this operation, 1,038 kB of additional disk space will be used.
Do you want to continue? [Y/n] y
Get:1 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 libcurl4 amd64 7.58.0-
2ubuntu3.8 [214 kB]
Get:2 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 curl amd64 7.58.0-2ubu
ntu3.8 [159 kB]
Fetched 373 kB in 1s (416 kB/s)
Selecting previously unselected package libcurl4:amd64.
(Reading database ... 126439 files and directories currently installed.)
Preparing to unpack .../libcurl4_7.58.0-2ubuntu3.8_amd64.deb ...
Unpacking libcurl4:amd64 (7.58.0-2ubuntu3.8) ...
Selecting previously unselected package curl.
Preparing to unpack .../curl_7.58.0-2ubuntu3.8_amd64.deb ...
Unpacking curl (7.58.0-2ubuntu3.8) ...
Setting up libcurl4:amd64 (7.58.0-2ubuntu3.8) ...
Processing triggers for libc-bin (2.27-3ubuntu1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Setting up curl (7.58.0-2ubuntu3.8) ...
venkata@ubuntu:~$
```

Figure 17: cURL installation.

Once the cURL tool installation finishes, as shown in Figure 17, we need to set up the Go Programming language in our Linux machine. Go programming language is one of the programming languages that Hyperledger fabric architecture supports. Go Programming language, also called Golang, is also an open-source language launched by Google.

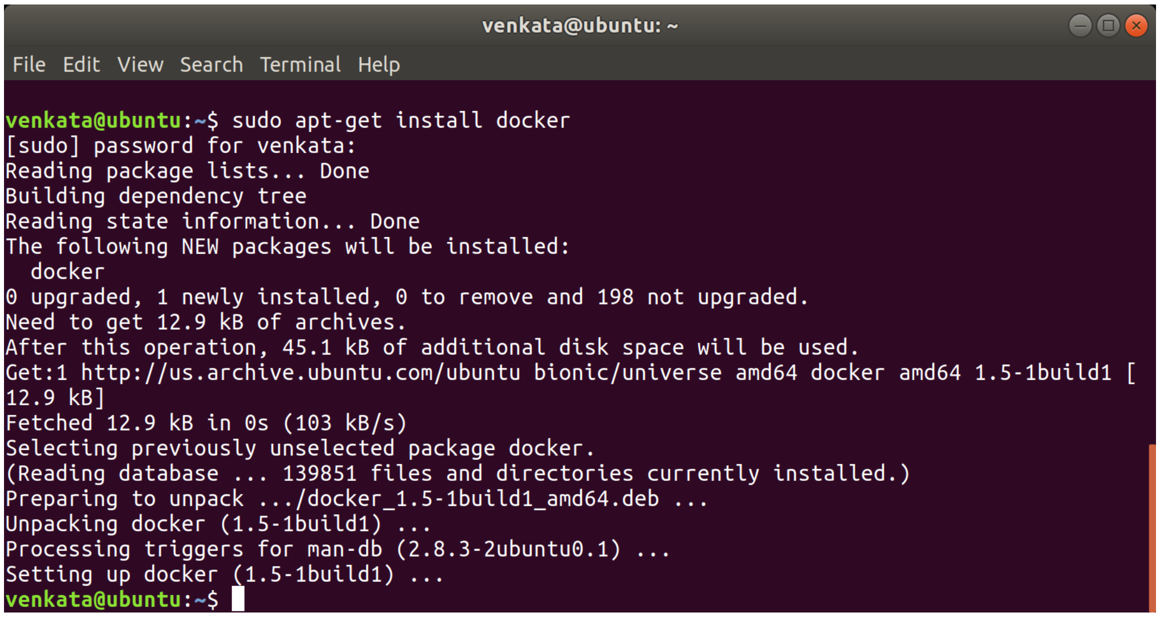
```
sudo apt-get install golang-go
```

Chaincode, which are smart contracts in this architecture, are written using the Go Programming language. Chaincode in Hyperledger fabric runs in a separate container, which makes it isolated and secure. Chaincode manages and updates ledger through requests submitted by peers in the network.

The next step in the process is installing docker in our guest machine. We can run the command below in terminal and pass the password when invoked, to install it as shown in Figure 18.

```
sudo apt-get install docker
```

Command *sudo* command will run the command following it as a superuser and *apt* is a potent tool in ubuntu, which refers to the Application Packaging Tool, which means it handles all installations in Ubuntu machine such as new software package installation, upgrading of existing installed software and delete any software packages.



```
venkata@ubuntu:~$ sudo apt-get install docker
[sudo] password for venkata:
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  docker
0 upgraded, 1 newly installed, 0 to remove and 198 not upgraded.
Need to get 12.9 kB of archives.
After this operation, 45.1 kB of additional disk space will be used.
Get:1 http://us.archive.ubuntu.com/ubuntu bionic/universe amd64 docker amd64 1.5-1build1 [
12.9 kB]
Fetched 12.9 kB in 0s (103 kB/s)
Selecting previously unselected package docker.
(Reading database ... 139851 files and directories currently installed.)
Preparing to unpack .../docker_1.5-1build1_amd64.deb ...
Unpacking docker (1.5-1build1) ...
Processing triggers for man-db (2.8.3-2ubuntu0.1) ...
Setting up docker (1.5-1build1) ...
venkata@ubuntu:~$
```

Figure 18: Docker installation.

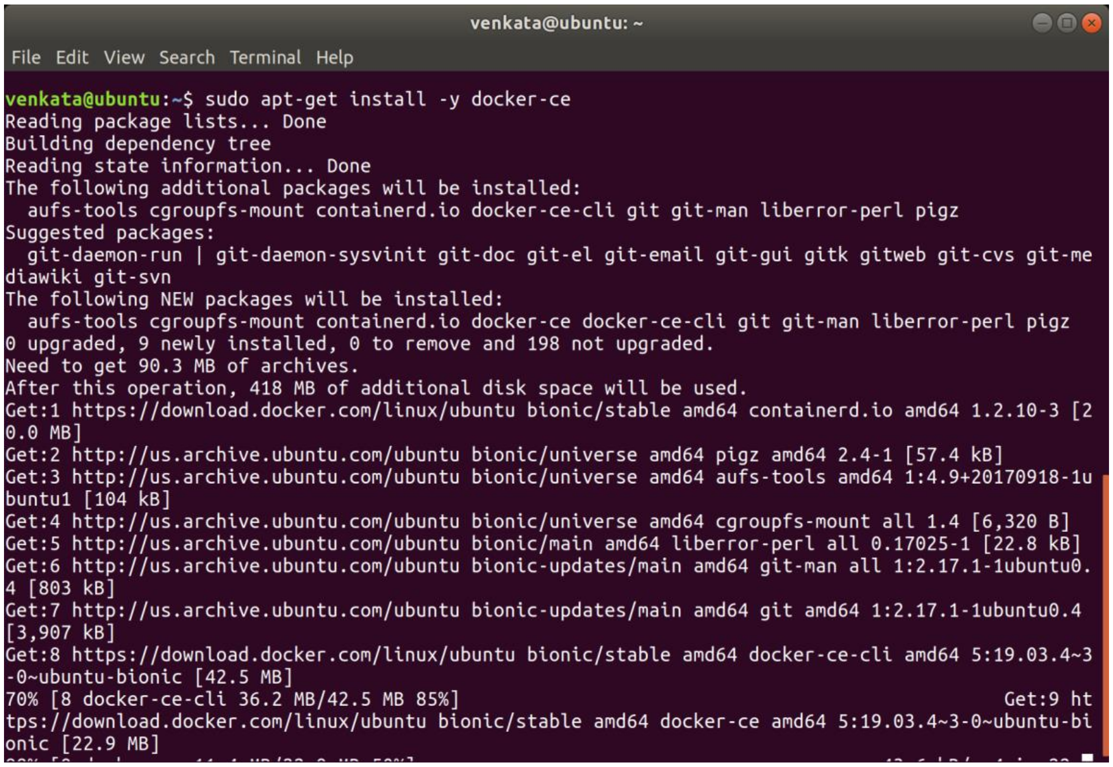
Running this command in the terminal will install Community Edition of docker. After the successful installation of the Docker community edition. We need to install another package of docker, which is docker-compose.

```
sudo apt-get install -y docker-ce
```

Docker-compose is a potent tool in docker, which allows users to create containers using existing docker images or software packages installed in the machine. Figure 19 shown that the community edition of Docker has installed completely.

```
sudo apt-get install docker-compose
```

Docker-compose uses configuration files to generate docker containers for applications and can generate multiple containers. We use a YAML file to compose all the configurations of a container.



```
venkata@ubuntu:~$ sudo apt-get install -y docker-ce
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  aufs-tools cgroupfs-mount containerd.io docker-ce-cli git git-man liberror-perl pigz
Suggested packages:
  git-daemon-run | git-daemon-sysvinit git-doc git-el git-email git-gui gitk gitweb git-cvs git-me
diawiki git-svn
The following NEW packages will be installed:
  aufs-tools cgroupfs-mount containerd.io docker-ce docker-ce-cli git git-man liberror-perl pigz
0 upgraded, 9 newly installed, 0 to remove and 198 not upgraded.
Need to get 90.3 MB of archives.
After this operation, 418 MB of additional disk space will be used.
Get:1 https://download.docker.com/linux/ubuntu bionic/stable amd64 containerd.io amd64 1.2.10-3 [2
0.0 MB]
Get:2 http://us.archive.ubuntu.com/ubuntu bionic/universe amd64 pigz amd64 2.4-1 [57.4 kB]
Get:3 http://us.archive.ubuntu.com/ubuntu bionic/universe amd64 aufs-tools amd64 1:4.9+20170918-1u
buntu1 [104 kB]
Get:4 http://us.archive.ubuntu.com/ubuntu bionic/universe amd64 cgroupfs-mount all 1.4 [6,320 B]
Get:5 http://us.archive.ubuntu.com/ubuntu bionic/main amd64 liberror-perl all 0.17025-1 [22.8 kB]
Get:6 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 git-man all 1:2.17.1-1ubuntu0.
4 [803 kB]
Get:7 http://us.archive.ubuntu.com/ubuntu bionic-updates/main amd64 git amd64 1:2.17.1-1ubuntu0.4
[3,907 kB]
Get:8 https://download.docker.com/linux/ubuntu bionic/stable amd64 docker-ce-cli amd64 5:19.03.4~3
-0~ubuntu-bionic [42.5 MB]
70% [8 docker-ce-cli 36.2 MB/42.5 MB 85%]                                               Get:9 ht
tps://download.docker.com/linux/ubuntu bionic/stable amd64 docker-ce amd64 5:19.03.4~3-0~ubuntu-bi
onic [22.9 MB]
```

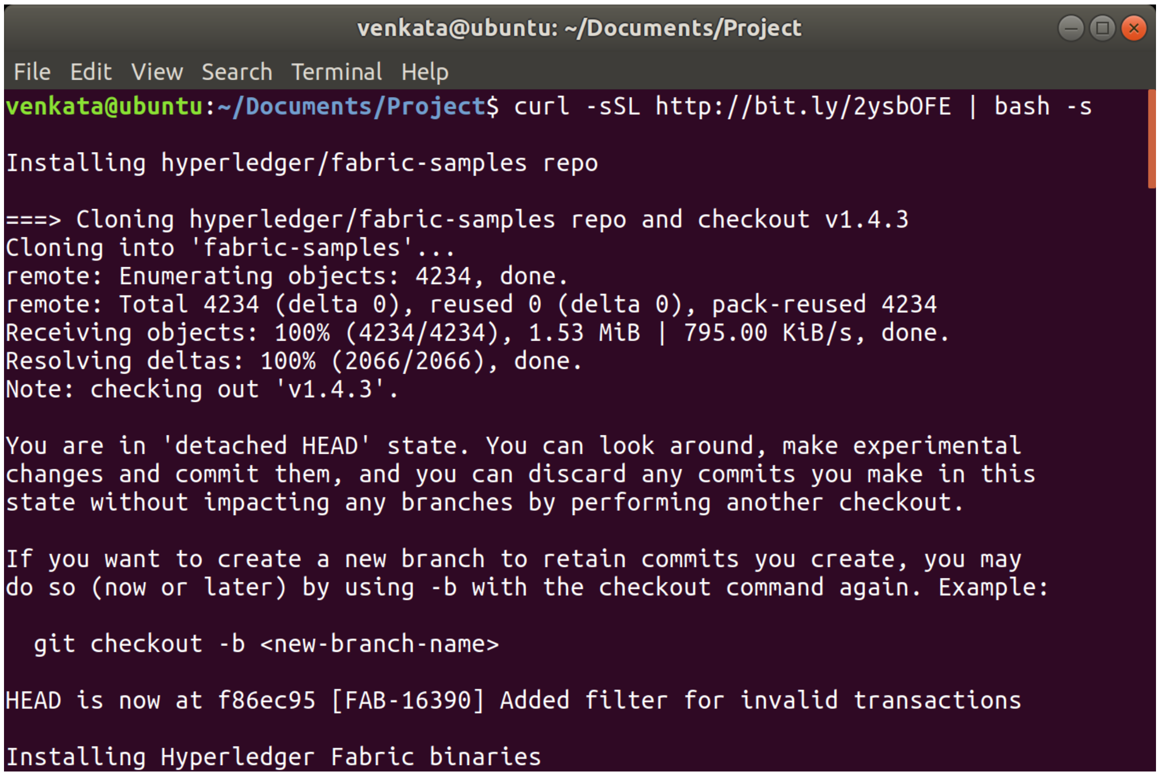
Figure 19: Docker community edition installation on VM.

Once all the docker related packages are downloaded and installed in the Linux machine. The following command upgrades all the existing software packages and gets the system updated for Hyperledger fabric docker images.

```
sudo apt-get upgrade
```

When the system upgrade with the latest software patches finishes, let us use cURL to get the Hyperledger fabric binaries and Docker images. To do this, the user creates a directory and open *terminal* in that folder and execute the following command.

```
curl -sSL http://bit.ly/2ysbOFE | bash -s
```


venkata@ubuntu: ~/Documents/Project
File Edit View Search Terminal Help

```
venkata@ubuntu:~/Documents/Project$ curl -sSL http://bit.ly/2ysbOFE | bash -s

Installing hyperledger/fabric-samples repo

===> Cloning hyperledger/fabric-samples repo and checkout v1.4.3
Cloning into 'fabric-samples'...
remote: Enumerating objects: 4234, done.
remote: Total 4234 (delta 0), reused 0 (delta 0), pack-reused 4234
Receiving objects: 100% (4234/4234), 1.53 MiB | 795.00 KiB/s, done.
Resolving deltas: 100% (2066/2066), done.
Note: checking out 'v1.4.3'.

You are in 'detached HEAD' state. You can look around, make experimental
changes and commit them, and you can discard any commits you make in this
state without impacting any branches by performing another checkout.

If you want to create a new branch to retain commits you create, you may
do so (now or later) by using -b with the checkout command again. Example:

  git checkout -b <new-branch-name>

HEAD is now at f86ec95 [FAB-16390] Added filter for invalid transactions

Installing Hyperledger Fabric binaries
```

Figure 20: Cloning in all Hyperledger fabric docker images.

Figure 20 shows that, when the command executes, the cURL tool copies all the files available on the website and clones a copy in our local folder. The cloned copy also contains come templates, which will be useful for reference.

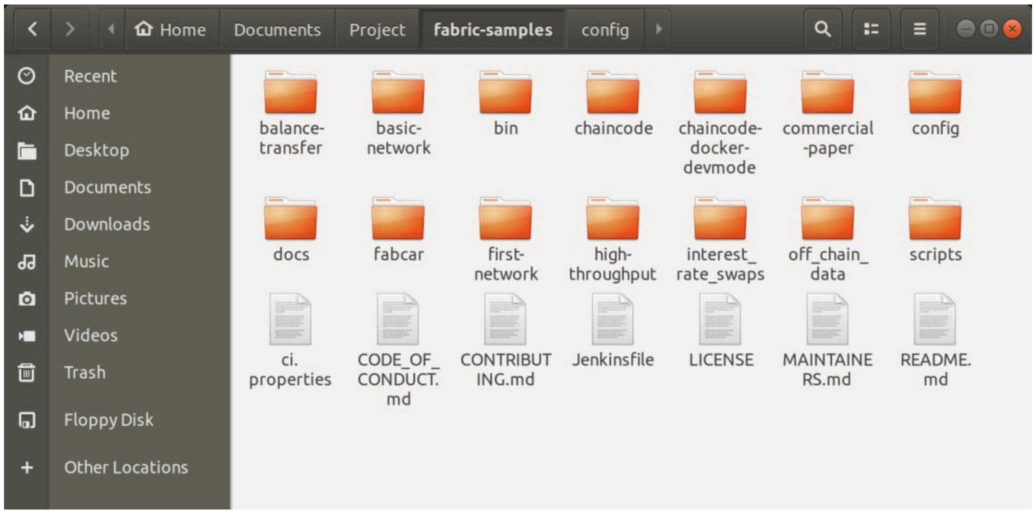


Figure 21: Cloned copies in VM.

Figure 21 shows how the cURL tool managed to clone in all the files and directories from the URL on to the directory specified above.

Moreover, in this folder *chain code,* there are sample chain code templates written in golang and java. Different prebuilt hyper ledger projects such as *byfn, basic network,* and *fabcar* are provided as templates for reference. The binaries that the tool downloaded are in the *bin* folder.

Cryptogen tool is one of the docker image command-line interface tools provided. It is used to pre-configuration of the network in development environments. Cryptogen tool works on a YAML file and generates a list of certificates for the entities listed in the YAML file.

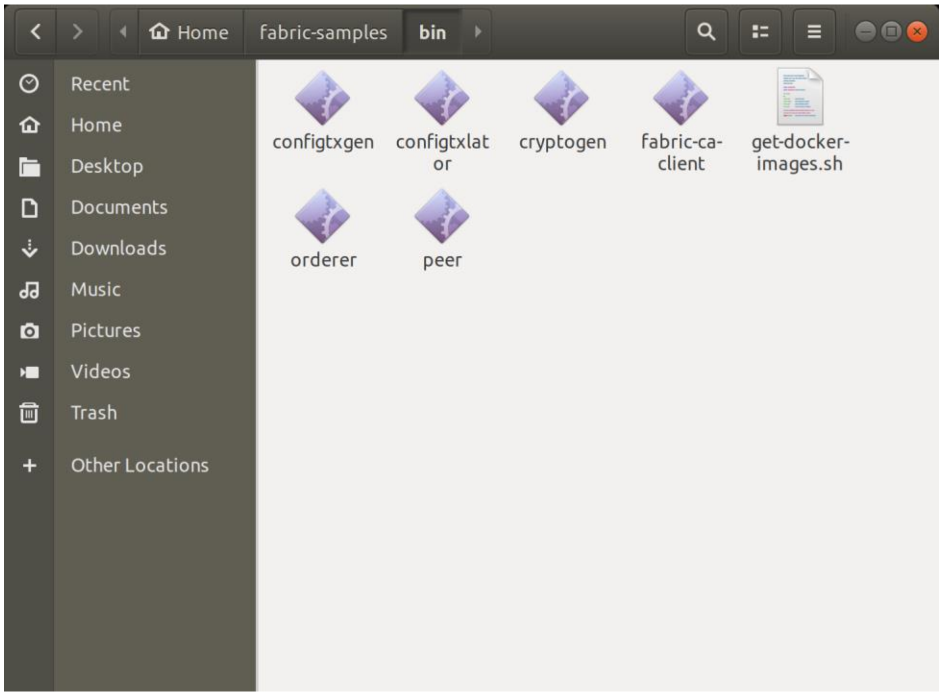


*Figure 22***:** Docker images provided by Hyperledger fabric architecture.

Figure 22 shows the prebuilt docker images provided by Hyperledger fabric, which will be tools helpful for this project. The YAML file with network and peer configurations are available in *crypto-config.yaml*. The cryptogen tool use this YAML file and generate membership certificates. The command to generate certificates is

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

Furthermore, here is a YAML file configuration; it has the following important things listed in the YAML file, Orderer and its domain; peer organizations, and their domains. Below is the YAML configurations for this network.

```
OrdererOrgs:
- Name: orderer
Domain: etranscripts.com
EnableNodeOUs: true
Specs:
- Hostname: orderer
PeerOrgs:
- Name: scsu
Domain: scsu.minnstate.edu
EnableNodeOUs: true
Template:
Count: 1
Users:
Count: 1

- Name: umn
Domain: umn.minnstate.edu
EnableNodeOUs: true
Template:
Count: 1
Users:
Count: 1
- Name: msu
Domain: msu.minnstate.edu
EnableNodeOUs: true
Template:
Count: 1
Users:
Count: 1
- Name: bsu
Domain: bsu.minnstate.edu
EnableNodeOUs: true
Template:
Count: 1
Users:
Count: 1
```

Configtxgen tool is the next docker image provided by Hyperledger fabric; it also takes a YAML file and generates different channel level configuration files, as in genesis block configuration, channel configurations, and anchor peer configurations. Below here is a small snippet of code from YAML file

```
    eTranscriptsGenesis:


Consortiums:
            MinnStateUniv:
                Organizations:
                - *scsu
                - *umn
                - *msu
                - *bsu
    eTranscriptsChannel:
        Consortium: MinnStateUniv
        Application:
            <<: *ApplicationDefaults
            Organizations:
                - *scsu
                - *umn
                - *msu
                - *bsu
            Capabilities:
                <<: *ApplicationCapabilities
```

And the command used to generate configuration files using the tool are,

```
//genesis block configuration
./configtxgen -profile eTranscriptsGenesis -outputBlock ./channel-

artifacts/genesis.block



//channel configuration
./configtxgen -profile eTranscriptsChannel -outputCreateChannelTx ./channel-

artifacts/channel.tx -channelID minnstate



//Anchor peer configuration
./configtxgen -profile eTranscriptsChannel -outputAnchorPeersUpdate

./channel-artifacts/bsuMSPanchors.tx -channelID minnstate -asOrg scsuMSP
```

Now we need to use up other docker images from bin folders such as Orderer, peer, and Fabric-ca client from the bin folder with other docker specific configurations such as port addresses, MSPid, genesis file path, volumes for specifying the critical paths from local machine and others to bring up containers.

For this implementation project, a YAML file with configurations for Orderer, peers, CLI, CouchDB, and Certificate authorities (CA) is passed on to docker-compose command as,

```
docker-compose -f docker-compose-org.yaml -f docker-compose-cli.yaml -f
docker-compose-couch.yaml -f docker-compose-ca.yaml up
```

The up command will create all requested containers listed in the YAML files. The sample configuration written for the Orderer is,

```
orderer-base:
    image: hyperledger/fabric-orderer:latest
    environment:
      - ORDERER_GENERAL_GENESISMETHOD=file
      - FABRIC_LOGGING_SPEC=INFO
      - ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
      - ORDERER_GENERAL_LOCALMSPID=ordererMSP

      -ORDERER_GENERAL_GENESISFILE=/orderer/orderer.genesis.block    //genesis
block path
      - CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=transcriptshlf_TranscriptsHLF
      - ORDERER_GENERAL_TLS_ENABLED=true //TLS information
      -  ORDERER_GENERAL_TLS_PRIVATEKEYorderer/tls/server.key  //TLS  key  for
authentication
      - ORDERER_GENERAL_LOCALMSPDIR=/var/hyperledger/orderer/msp

    working_dir:/opt/gopath/src/hyperledger/fabric//Orderer working directory
    volumes:
    - ../channel-
artifacts/genesis.block:/var/hyperledger/orderer/orderer.genesis.block
    - ../crypto-
config/ordererOrganizations/etranscripts.com/orderers/orderer.etranscripts.co
m/msp:/var/hyperledger/orderer/msp
    - ../crypto-
config/ordererOrganizations/etranscripts.com/orderers/orderer.etranscripts.co
m/tls:/var/hyperledger/orderer/tls
    - orderer.etranscripts.com:/var/hyperledger/production/orderer
    ports:
      - 7050:7050
command: orderer
```

Below is a snippet of code written in Golang as Chaincode program in order to maintain data on ledger,

```
func (c *CourseProcessor) Init(stub shim.ChaincodeStubInterface) pb.Response{
  return shim.Success(nil)
}
```

```
func (c *CourseProcessor) Invoke(stub shim.ChaincodeStubInterface) pb.Response{
  function, args := stub.GetFunctionAndParameters()

  if function == "submit-Scheme" {
    return c.submitGradeScheme(stub, args)
  } else if function == "initLedger" {
    return c.initLedger(stub)
  } else if function == "submit-grade" {
    return c.submitGrades(stub, args)
  } else if function == "query-grade-student" {
    return c.getGrades(stub, args)
  }

  return shim.Error("Invalid Smart Contract function name.")
}

func (c *CourseProcessor) initLedger(stub shim.ChaincodeStubInterface)  pb.Response{
  return shim.Success(nil)
}
```

Summary

This chapter deals with an in-depth discussion about how to implement a Hyperledger fabric application on a Linux machine and also gives detail about different software components and tools used for this implementation. It also gives a brief explanation about different configurations and programming code used to give a high-level view of the implementation.

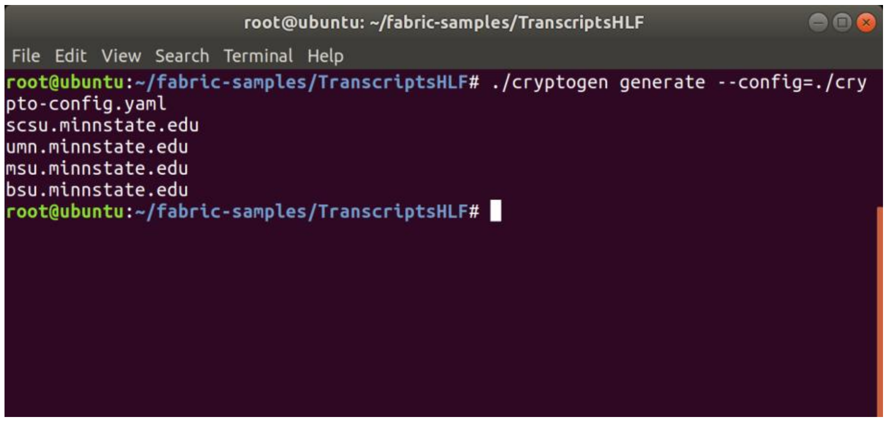
Chapter IV: Data Presentation and Analysis

Introduction

This chapter deals with the analyzation of the results of implementation. In the previous chapter, we discussed various algorithms, tools, and various techniques used in the implementation. This section will discuss an in-depth analysis of data obtained from tools and screenshots related to the implementation and provides detailed information about the implementation and results obtained at various stages.

Data Presentation

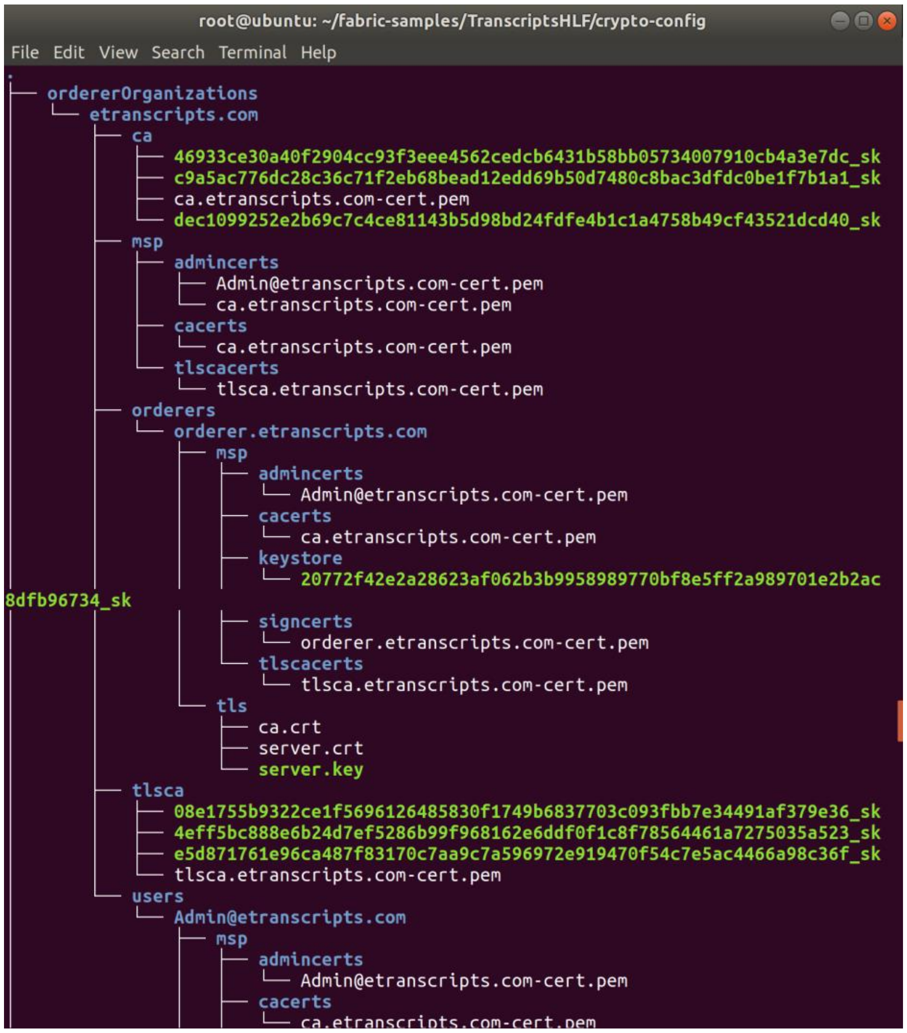
The Hyperledger fabric architecture was installed on a Linux machine (Ubuntu VM). The last chapter provided a walkthrough of the software setup, installations, and upgrades for architecture. Implementation of the blockchain architecture needs at least one orderer and peer.


root@ubuntu: ~/fabric-samples/TranscriptsHLF
File Edit View Search Terminal Help

```
root@ubuntu:~/fabric-samples/TranscriptsHLF# ./cryptogen generate --config=./cry
pto-config.yaml
scsu.minnstate.edu
umn.minnstate.edu
msu.minnstate.edu
bsu.minnstate.edu
root@ubuntu:~/fabric-samples/TranscriptsHLF# 
```

Figure 23: Generating Crypto materials.

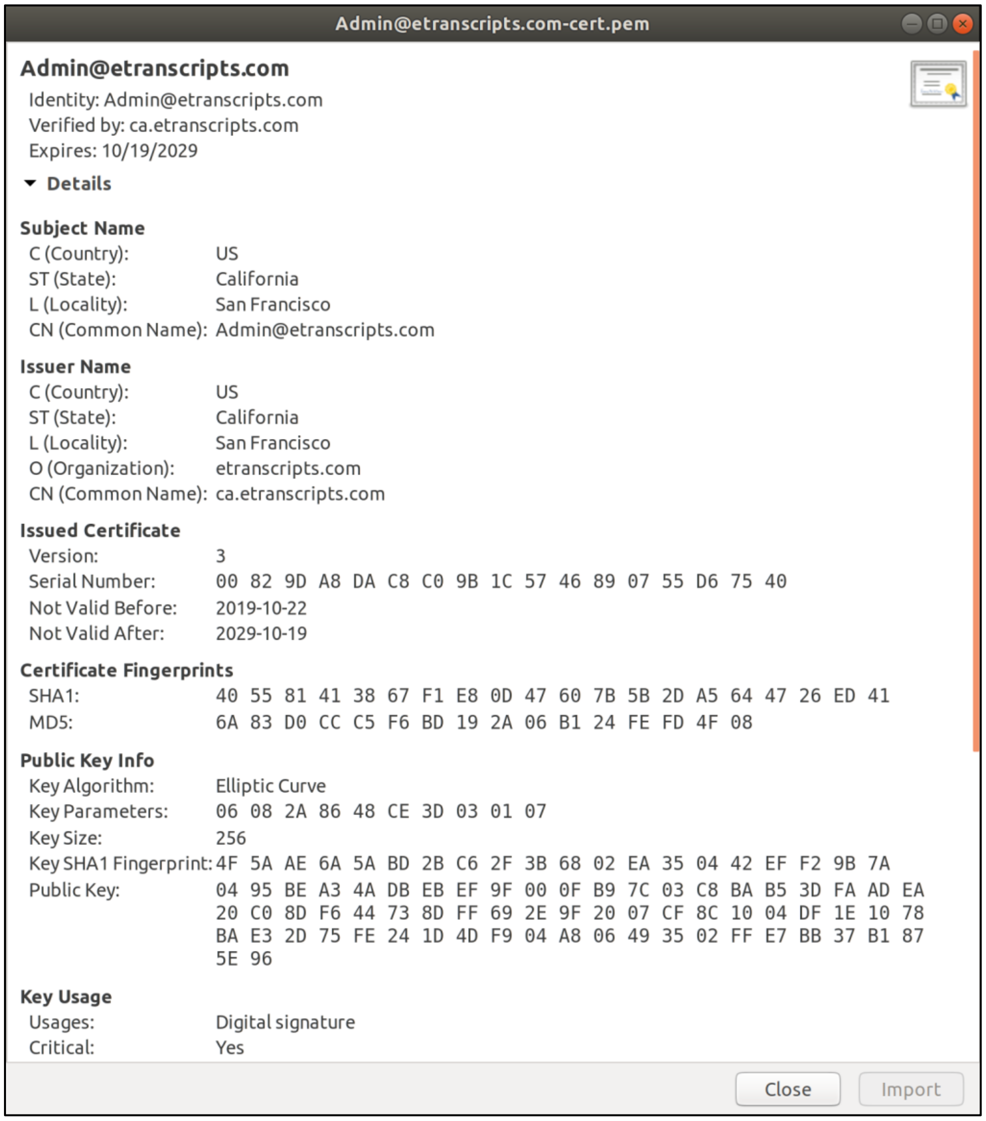
For this project, let us continue with a selection of one orderer and four peers. Figure 23 shows that the cryptogen tool generated certificates to entities listed in the YAML file. This step is to get the certificates for TLS communication. TLS is Transport layer security, which provides an end to end communication security over a network using cryptographic protocols. TLS also supports Pre-shared keys and secure remote passwords. When cryptogen works on the YAML file, which holds organizational information. It produces certificates, keys, and MSP information. The above screenshot command is to use the cryptogen tool and generate required certificates using the configuration file. As explained earlier, the configuration file has details about Orderer and four participating organizations.


root@ubuntu: ~/fabric-samples/TranscriptsHLF/crypto-config
File Edit View Search Terminal Help

```
.
├── ordererOrganizations
│   └── etranscripts.com
│       ├── ca
│       │   ├── 46933ce30a40f2904cc93f3eee4562cedcb6431b58bb05734007910cb4a3e7dc_sk
│       │   ├── c9a5ac776dc28c36c71f2eb68bead12edd69b50d7480c8bac3dfdc0be1f7b1a1_sk
│       │   ├── ca.etranscripts.com-cert.pem
│       │   └── dec1099252e2b69c7c4ce81143b5d98bd24fdfe4b1c1a4758b49cf43521dcd40_sk
│       ├── msp
│       │   ├── admincerts
│       │   │   ├── Admin@etranscripts.com-cert.pem
│       │   │   └── ca.etranscripts.com-cert.pem
│       │   ├── cacerts
│       │   │   └── ca.etranscripts.com-cert.pem
│       │   └── tlscacerts
│       │       └── tlsca.etranscripts.com-cert.pem
│       ├── orderers
│       │   └── orderer.etranscripts.com
│       │       ├── msp
│       │       │   ├── admincerts
│       │       │   │   └── Admin@etranscripts.com-cert.pem
│       │       │   ├── cacerts
│       │       │   │   └── ca.etranscripts.com-cert.pem
│       │       │   ├── keystore
│       │       │   │   └── 20772f42e2a28623af062b3b9958989770bf8e5ff2a989701e2b2ac
8dfb96734_sk
│       │       │   ├── signcerts
│       │       │   │   └── orderer.etranscripts.com-cert.pem
│       │       │   └── tlscacerts
│       │       │       └── tlsca.etranscripts.com-cert.pem
│       │       └── tls
│       │           ├── ca.crt
│       │           ├── server.crt
│       │           └── server.key
│       ├── tlsca
│       │   ├── 08e1755b9322ce1f5696126485830f1749b6837703c093fbb7e34491af379e36_sk
│       │   ├── 4eff5bc888e6b24d7ef5286b99f968162e6ddf0f1c8f78564461a7275035a523_sk
│       │   ├── e5d871761e96ca487f83170c7aa9c7a596972e919470f54c7e5ac4466a98c36f_sk
│       │   └── tlsca.etranscripts.com-cert.pem
│       └── users
│           └── Admin@etranscripts.com
│               ├── msp
│               │   ├── admincerts
│               │   │   └── Admin@etranscripts.com-cert.pem
│               │   ├── cacerts
│               │   │   └── ca.etranscripts.com-cert.pem
```

Figure 24: Folder structure of Crypto documents.

As shown in Figure 24, Cryptogen works on this configuration and generates the folder tree structure. The Cryptogen tool was able to create keys related to a certificate authority and MSP certificates at an organizational level and also at the user level.



Admin@etranscripts.com-cert.pem

Admin@etranscripts.com

Identity: Admin@etranscripts.com
Verified by: ca.etranscripts.com
Expires: 10/19/2029

▾ Details

Subject Name

C (Country):	US
ST (State):	California
L (Locality):	San Francisco
CN (Common Name):	Admin@etranscripts.com

Issuer Name

C (Country):	US
ST (State):	California
L (Locality):	San Francisco
O (Organization):	etranscripts.com
CN (Common Name):	ca.etranscripts.com

Issued Certificate

Version:	3
Serial Number:	00 82 9D A8 DA C8 C0 9B 1C 57 46 89 07 55 D6 75 40
Not Valid Before:	2019-10-22
Not Valid After:	2029-10-19

Certificate Fingerprints

SHA1:	40 55 81 41 38 67 F1 E8 0D 47 60 7B 5B 2D A5 64 47 26 ED 41
MD5:	6A 83 D0 CC C5 F6 BD 19 2A 06 B1 24 FE FD 4F 08

Public Key Info

Key Algorithm:	Elliptic Curve
Key Parameters:	06 08 2A 86 48 CE 3D 03 01 07
Key Size:	256
Key SHA1 Fingerprint:	4F 5A AE 6A 5A BD 2B C6 2F 3B 68 02 EA 35 04 42 EF F2 9B 7A
Public Key:	04 95 BE A3 4A DB EB EF 9F 00 0F B9 7C 03 C8 BA B5 3D FA AD EA 20 C0 8D F6 44 73 8D FF 69 2E 9F 20 07 CF 8C 10 04 DF 1E 10 78 BA E3 2D 75 FE 24 1D 4D F9 04 A8 06 49 35 02 FF E7 BB 37 B1 87 5E 96

Key Usage

Usages:	Digital signature
Critical:	Yes

Figure 25: TLS certificate generated by the docker image.

Cryptogen can produce certificates for a user, Admin, and also for the organizational level. The above figure tells that certificate authority related to the orderer generates the certificate, and it has an expiry date and has digital fingerprints and also a version number. This certificate shown in Figure 25 acts as Digital Signature during TLS communication. It was generated for the Administrator for the organization specified. Similarly, the cryptogen tool generates TLS, MSP, and stored keys for initial implementation or development projects.

Next, using the *configtxgen* tool will generate configuration material for the channel; the tool works on the corresponding YAML file and can generate channel configuration, which is useful in creating the channel using channel ID, and also starting up a genesis block which is the first block in the channel. It is a configuration block that initializes the orderer. *Configtxgen* will also be able to produce a configuration file for anchor peers, and Anchor peers are necessary in Hyperledger fabric, only these peers will be able to communicate with an anchor peer of another organization.

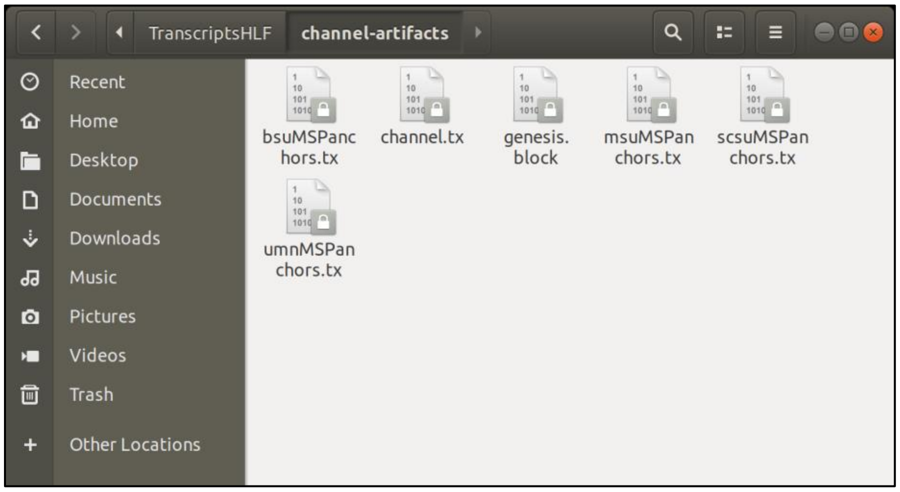
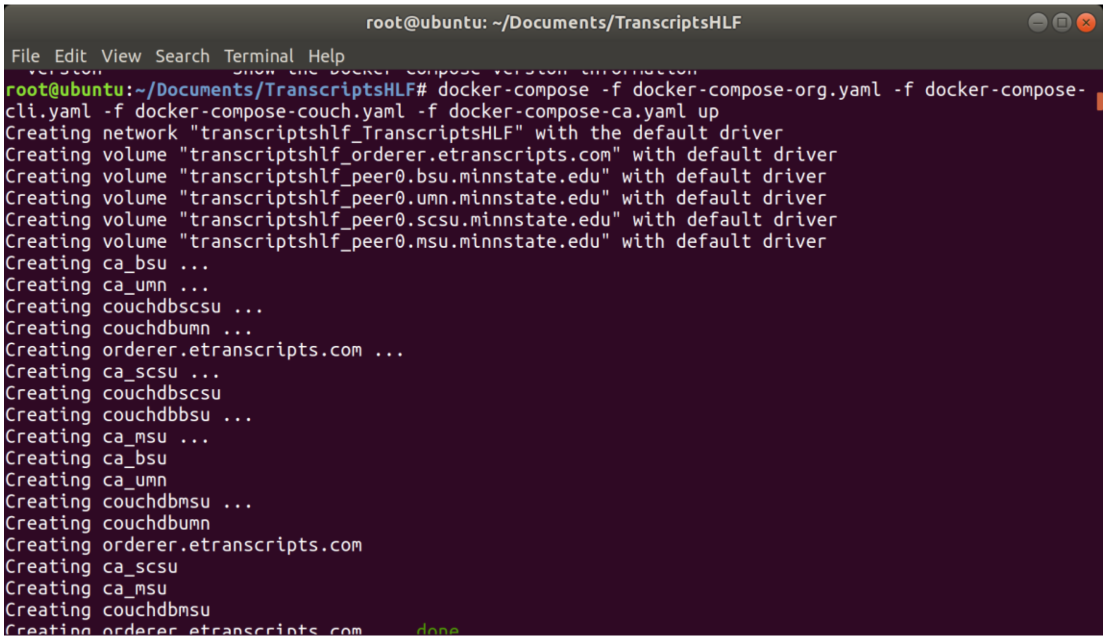


Figure 26: Channel artifacts generated by HLF docker image.

From Figure 26, the *Configtxgen* tool generated a folder with a genesis block configuration file, Channel Configuration file, and Anchor peer configurations. These files are used to update an existing channel configuration.

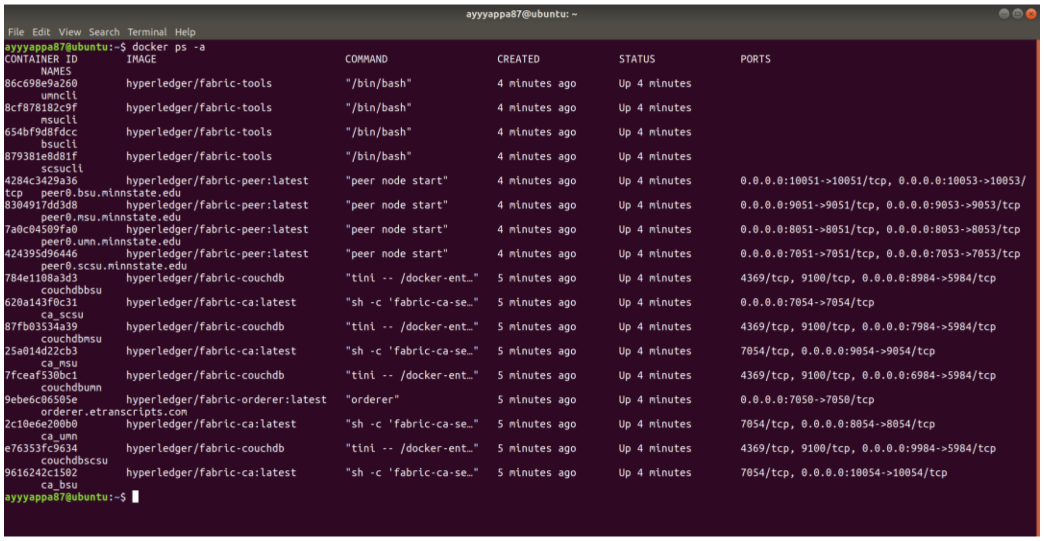
Moreover, we now need to create containers using docker-compose, Container for orderer, four organizational peers, four CLI containers for each organization, four CouchDB database containers, and four certificate authority containers, each representing one organization. We had already installed docker-compose in the previous chapter. The configuration files do have specified ports to each container and when the command executes,



```
root@ubuntu:~/Documents/TranscriptsHLF# docker-compose -f docker-compose-org.yaml -f docker-compose-
cli.yaml -f docker-compose-couch.yaml -f docker-compose-ca.yaml up
Creating network "transcriptshlf_TranscriptsHLF" with the default driver
Creating volume "transcriptshlf_orderer.etranscripts.com" with default driver
Creating volume "transcriptshlf_peer0.bsu.minnstate.edu" with default driver
Creating volume "transcriptshlf_peer0.umn.minnstate.edu" with default driver
Creating volume "transcriptshlf_peer0.scsu.minnstate.edu" with default driver
Creating volume "transcriptshlf_peer0.msu.minnstate.edu" with default driver
Creating ca_bsu ...
Creating ca_umn ...
Creating couchdbscsu ...
Creating couchdbumn ...
Creating orderer.etranscripts.com ...
Creating ca_scsu ...
Creating couchdbscsu
Creating couchdbbsu ...
Creating ca_msu ...
Creating ca_bsu
Creating ca_umn
Creating couchdbmsu ...
Creating couchdbumn
Creating orderer.etranscripts.com
Creating ca_scsu
Creating ca_msu
Creating couchdbmsu
Creating orderer.etranscripts.com ... done
```

Figure 27: Docker compose tool–containers and volumes.

Docker-compose works on the YAML files and create containers, volumes specified, and initiates the network as shown in Figure 27. The next thing is to check on the status of these containers; to do this, let us open a new terminal and run the command "*docker ps -a*" which gives all the docker containers running on top of docker.



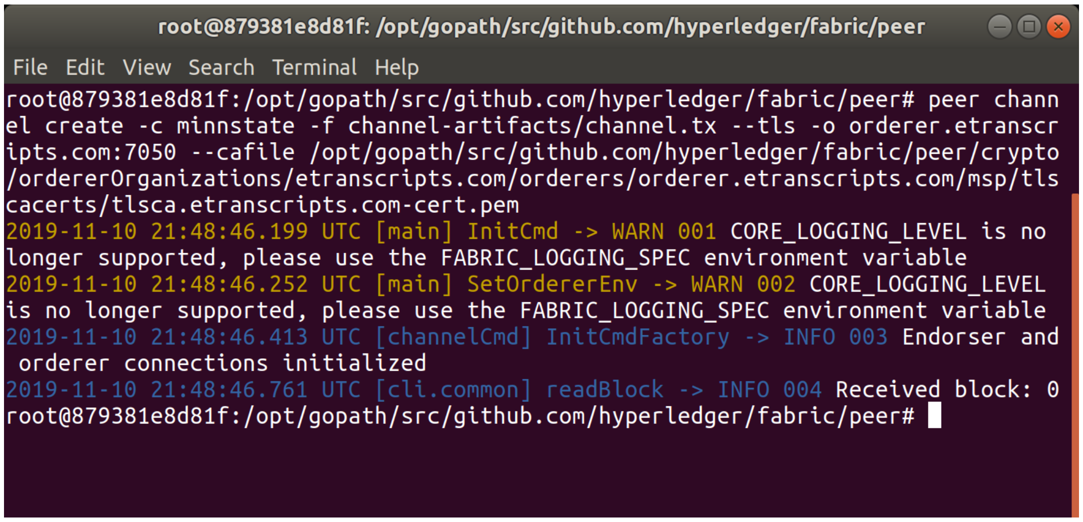
```
ayyyappa87@ubuntu:~$ docker ps -a
CONTAINER ID        IMAGE                               COMMAND                  CREATED             STATUS              PORTS
      NAMES
86c698e9a260        hyperledger/fabric-tools            "/bin/bash"              4 minutes ago       Up 4 minutes
      umncli
8cf878182c9f        hyperledger/fabric-tools            "/bin/bash"              4 minutes ago       Up 4 minutes
      msucli
654bf9d8fdcc        hyperledger/fabric-tools            "/bin/bash"              4 minutes ago       Up 4 minutes
      bsucli
879381e8d81f        hyperledger/fabric-tools            "/bin/bash"              4 minutes ago       Up 4 minutes
      scsucli
4284c3429a36        hyperledger/fabric-peer:latest      "peer node start"        4 minutes ago       Up 4 minutes        0.0.0.0:10051->10051/tcp, 0.0.0.0:10053->10053/
tcp   peer0.bsu.minnstate.edu
8304917dd3d8        hyperledger/fabric-peer:latest      "peer node start"        4 minutes ago       Up 4 minutes        0.0.0.0:9051->9051/tcp, 0.0.0.0:9053->9053/tcp
      peer0.msu.minnstate.edu
7a0c04509fa0        hyperledger/fabric-peer:latest      "peer node start"        4 minutes ago       Up 4 minutes        0.0.0.0:8051->8051/tcp, 0.0.0.0:8053->8053/tcp
      peer0.umn.minnstate.edu
424395d96446        hyperledger/fabric-peer:latest      "peer node start"        4 minutes ago       Up 4 minutes        0.0.0.0:7051->7051/tcp, 0.0.0.0:7053->7053/tcp
      peer0.scsu.minnstate.edu
784e1108a3d3        hyperledger/fabric-couchdb          "tini -- /docker-ent…"   5 minutes ago       Up 4 minutes        4369/tcp, 9100/tcp, 0.0.0.0:8984->5984/tcp
      couchdbbsu
620a143f0c31        hyperledger/fabric-ca:latest        "sh -c 'fabric-ca-se…"   5 minutes ago       Up 4 minutes        0.0.0.0:7054->7054/tcp
      ca_scsu
87fb03534a39        hyperledger/fabric-couchdb          "tini -- /docker-ent…"   5 minutes ago       Up 4 minutes        4369/tcp, 9100/tcp, 0.0.0.0:7984->5984/tcp
      couchdbmsu
25a014d22cb3        hyperledger/fabric-ca:latest        "sh -c 'fabric-ca-se…"   5 minutes ago       Up 4 minutes        7054/tcp, 0.0.0.0:9054->9054/tcp
      ca_msu
7fceaf530bc1        hyperledger/fabric-couchdb          "tini -- /docker-ent…"   5 minutes ago       Up 4 minutes        4369/tcp, 9100/tcp, 0.0.0.0:6984->5984/tcp
      couchdbumn
9ebe6c06505e        hyperledger/fabric-orderer:latest   "orderer"                5 minutes ago       Up 4 minutes        0.0.0.0:7050->7050/tcp
      orderer.etranscripts.com
2c10e6e200b0        hyperledger/fabric-ca:latest        "sh -c 'fabric-ca-se…"   5 minutes ago       Up 4 minutes        7054/tcp, 0.0.0.0:8054->8054/tcp
      ca_umn
e76353fc9634        hyperledger/fabric-couchdb          "tini -- /docker-ent…"   5 minutes ago       Up 4 minutes        4369/tcp, 9100/tcp, 0.0.0.0:9984->5984/tcp
      couchdbscsu
9616242c1502        hyperledger/fabric-ca:latest        "sh -c 'fabric-ca-se…"   5 minutes ago       Up 4 minutes        7054/tcp, 0.0.0.0:10054->10054/tcp
      ca_bsu
ayyyappa87@ubuntu:~$
```

Figure 28: Docker containers.

Figure 28 shows all the details regarding containers, Container ID is a unique ID created for the container, Image–is the docker prebuilt images used to create the specific container, Command–is the command used to get the container created, which is specified in the YAML files. Created and Status gives information about whether the container is running or stopped due to some issues and finally most important information is the port ID's, both internal and external port numbers are shown here, which are used to communicate with the container; for example, Orderer address from the above window is *orderer.etranscripts.com:7050.*

Once the peers are up, and the status of its containers are active. We should start working on creating a channel. A CLI container will work as an Interface for a peer to work on the network. To create a channel in the network, the user must be within the container. To do this, the user should open the terminal and enter command *"docker exec -it scsucli bash",* This command will get the user into the container environment. Once the user is in the container, the following command will create a channel - '*minnstate.'* In the command *-f* tag specifies the path to file, where the channel configuration is stored, *--tls* specifies the communication method, *-o* specifies the orderer information, *--cafile* specifies the TLS certificate path in the container and Figure 29 shows that upon executing the command, we are able to generate genesis block for the ledger.

Command: "peer channel create -c minnstate -f channel-artifacts/channel.tx --tls -o orderer.etranscripts.com:7050 –cafile ./crypto/ordererOrganizations/etranscripts.com/orderers/orderer.etranscripts.com/msp/tlscacerts/tlsca.etranscripts.com-cert.pem"

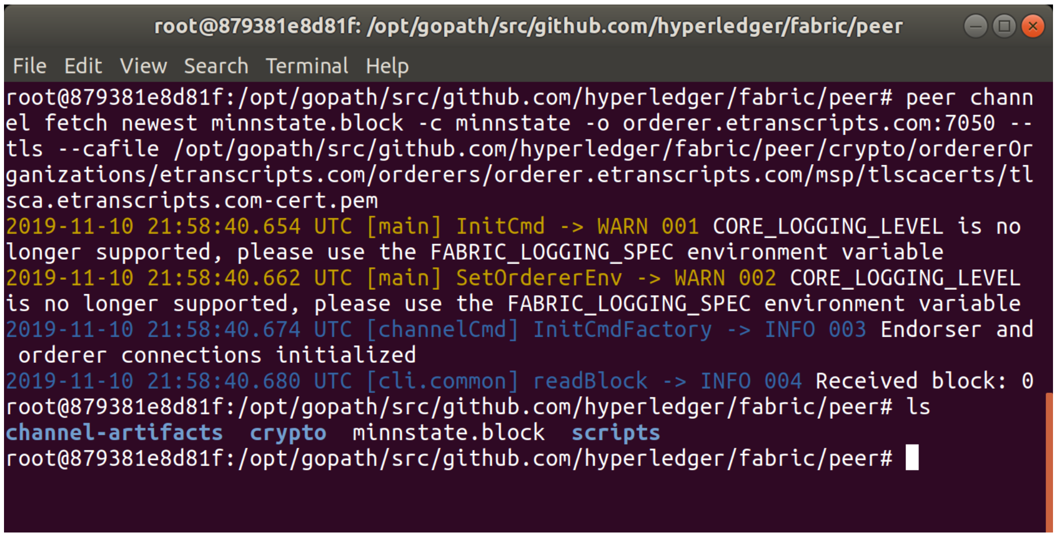

root@879381e8d81f: /opt/gopath/src/github.com/hyperledger/fabric/peer
File Edit View Search Terminal Help

```
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chann
el create -c minnstate -f channel-artifacts/channel.tx --tls -o orderer.etranscr
ipts.com:7050 --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto
/ordererOrganizations/etranscripts.com/orderers/orderer.etranscripts.com/msp/tls
cacerts/tlsca.etranscripts.com-cert.pem
2019-11-10 21:48:46.199 UTC [main] InitCmd -> WARN 001 CORE_LOGGING_LEVEL is no
longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2019-11-10 21:48:46.252 UTC [main] SetOrdererEnv -> WARN 002 CORE_LOGGING_LEVEL
is no longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2019-11-10 21:48:46.413 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and
 orderer connections initialized
2019-11-10 21:48:46.761 UTC [cli.common] readBlock -> INFO 004 Received block: 0
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer#
```

*Figure 29***:** Creating a channel in HLF network using CLI.

The command Create channel requires TLS connection between peer and orderer, In order to do that, we pass the TLS certificate address in order to establish the connection between the peer and orderer, If the certificate looks valid, the connection between the endorser and the orderer will establish and channel creation will be successful and a ledger named *minnstate.block* will appear in the folder when user tries to fetch the latest block from the ledger, This can be performed using the following command. In the command *-f* tag specifies the path to file, where the channel configuration is stored, --*tls* specifies the communication method, -*o* specifies the orderer information, --*cafile* specifies the TLS certificate path in the container.

Command: "peer channel fetch newest minnstate.block -c minnstate -o orderer.etranscripts.com:7050 --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/etranscripts.com/orderers/orderer.etranscripts.com/msp/tlscacerts/tlsca.etranscripts.com-cert.pem"

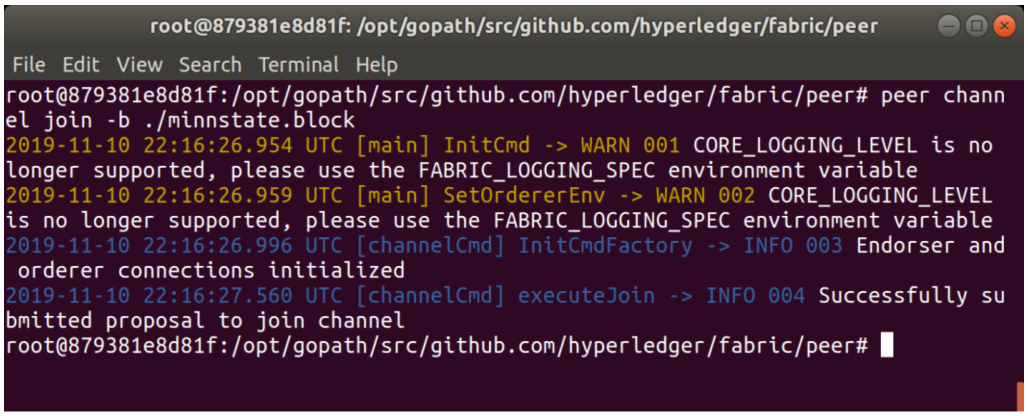

root@879381e8d81f: /opt/gopath/src/github.com/hyperledger/fabric/peer
File Edit View Search Terminal Help

```
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chann
el fetch newest minnstate.block -c minnstate -o orderer.etranscripts.com:7050 --
tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOr
ganizations/etranscripts.com/orderers/orderer.etranscripts.com/msp/tlscacerts/tl
sca.etranscripts.com-cert.pem
2019-11-10 21:58:40.654 UTC [main] InitCmd -> WARN 001 CORE_LOGGING_LEVEL is no
longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2019-11-10 21:58:40.662 UTC [main] SetOrdererEnv -> WARN 002 CORE_LOGGING_LEVEL
is no longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2019-11-10 21:58:40.674 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and
 orderer connections initialized
2019-11-10 21:58:40.680 UTC [cli.common] readBlock -> INFO 004 Received block: 0
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer# ls
channel-artifacts  crypto  minnstate.block  scripts
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer#
```

Figure 30: Genesis block.

From Figure 30, it is to be observed that a block file has is obtained in the container; this action requires a TLS connection between the peer and the orderer; TLS connection is performed using –*tls* command and is initialized using the orderer admin certificate link. Moreover, the orderer address is supplied alongside the name of the channel.

Once the peer fetches the newest block, then the peer can use the block file to join the ledger and channel; This can be performed using a simple join command from the container using the block fetched from ledger, as shown in Figure 31.


root@879381e8d81f: /opt/gopath/src/github.com/hyperledger/fabric/peer
File Edit View Search Terminal Help

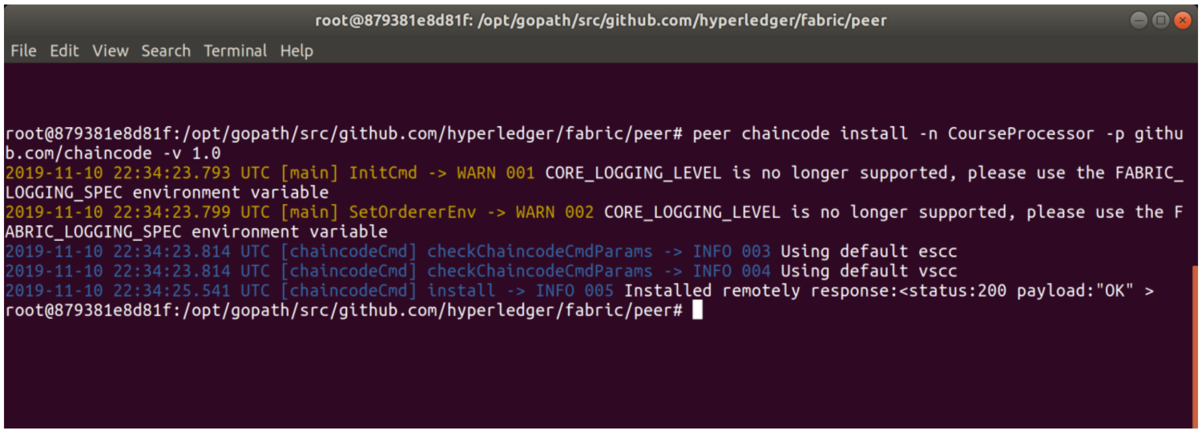
```
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chann
el join -b ./minnstate.block
2019-11-10 22:16:26.954 UTC [main] InitCmd -> WARN 001 CORE_LOGGING_LEVEL is no
longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2019-11-10 22:16:26.959 UTC [main] SetOrdererEnv -> WARN 002 CORE_LOGGING_LEVEL
is no longer supported, please use the FABRIC_LOGGING_SPEC environment variable
2019-11-10 22:16:26.996 UTC [channelCmd] InitCmdFactory -> INFO 003 Endorser and
 orderer connections initialized
2019-11-10 22:16:27.560 UTC [channelCmd] executeJoin -> INFO 004 Successfully su
bmitted proposal to join channel
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer#
```

Figure 31: Peer proposal to join channel.

This action of fetching the newest block from the channel and joining the channel is performed on all participating peers. Once all the peers join the channel, chain-code smart contracts are to be installed on to individual peers and instantiated on to the network.

The chain code for this project is written in Go Programming language. It should contain methods Init, to initialize the smart contract on to network. Then other methods are to be included in invoke, which should manage the data on the ledger. Installing the chain-code on a peer requires the path of the code, name of the smart-contract, and version number, -*v* tag specifies the version the chaincode, -*p* tag specifies the path to the chaincode and -*n* specifies the name of the chaincode.

Command: *"peer chaincode install -n CourseProcessor -p github.com/chaincode -v 1.0"*


root@879381e8d81f: /opt/gopath/src/github.com/hyperledger/fabric/peer
File Edit View Search Terminal Help

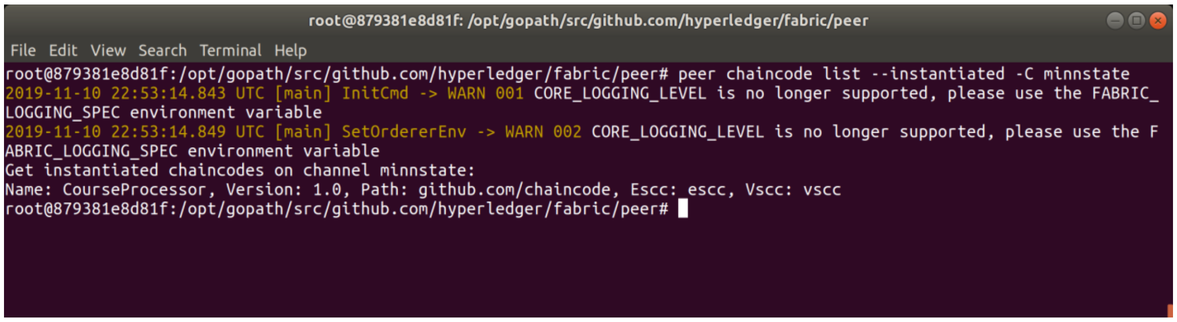
```
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode install -n CourseProcessor -p githu
b.com/chaincode -v 1.0
2019-11-10 22:34:23.793 UTC [main] InitCmd -> WARN 001 CORE_LOGGING_LEVEL is no longer supported, please use the FABRIC_
LOGGING_SPEC environment variable
2019-11-10 22:34:23.799 UTC [main] SetOrdererEnv -> WARN 002 CORE_LOGGING_LEVEL is no longer supported, please use the F
ABRIC_LOGGING_SPEC environment variable
2019-11-10 22:34:23.814 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 003 Using default escc
2019-11-10 22:34:23.814 UTC [chaincodeCmd] checkChaincodeCmdParams -> INFO 004 Using default vscc
2019-11-10 22:34:25.541 UTC [chaincodeCmd] install -> INFO 005 Installed remotely response:<status:200 payload:"OK" >
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer# 
```

*Figure 32***:** Installing chaincode on a peer.

Figure 32 shows that the Peer has installed the chaincode on its ledger. All the participating peers can install the chaincode on top of their ledgers. Once the installation finishes, The chaincode needs to be initiated on the network; Initiating a chaincode can be done only once. Since initiating a chaincode is at the network level, communicating with orderer peers using TLS communication requires -*o* tag. , --*tls* specifies the communication method, -*o* specifies the orderer information, --*cafile* specifies the TLS certificate path in the container and -*v* specifies the version of chaincode.

Command:

"peer chaincode instantiate -o orderer.etranscripts.com:7050 -n CourseProcessor -v 1.0 -C minnstate -c '{"Args":[]}' --tls --cafile /opt/gopath/src/github.com/hyperledger/fabric/peer/crypto/ordererOrganizations/etranscripts.com/orderers/orderer.etranscripts.com/msp/tlscacerts/tlsca.etranscripts.com-cert.pem"


root@879381e8d81f: /opt/gopath/src/github.com/hyperledger/fabric/peer
File Edit View Search Terminal Help

```
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode list --instantiated -C minnstate
2019-11-10 22:53:14.843 UTC [main] InitCmd -> WARN 001 CORE_LOGGING_LEVEL is no longer supported, please use the FABRIC_
LOGGING_SPEC environment variable
2019-11-10 22:53:14.849 UTC [main] SetOrdererEnv -> WARN 002 CORE_LOGGING_LEVEL is no longer supported, please use the F
ABRIC_LOGGING_SPEC environment variable
Get instantiated chaincodes on channel minnstate:
Name: CourseProcessor, Version: 1.0, Path: github.com/chaincode, Escc: escc, Vscc: vscc
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer# 
```

Figure 33: Instantiating chaincode on network.

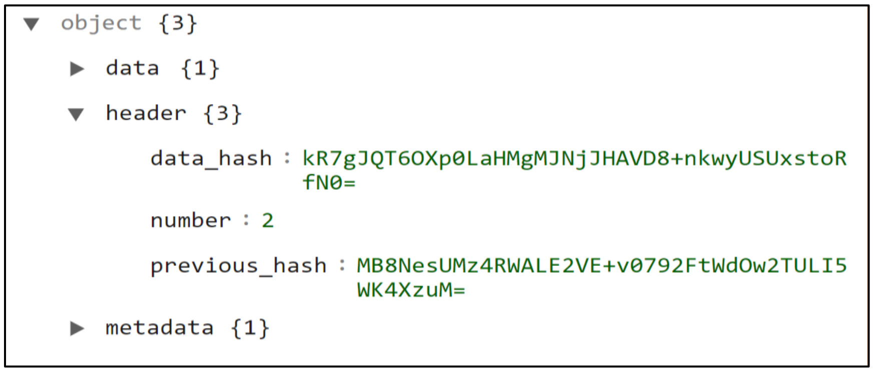
Figure 33 shows that the instantiated chaincode on this channel is *CourseProfessor* and is of 1.0 version.

Data Analysis

Instantiating the ledger on the current channel has generated a block. Using the CLI, we fetched the newest block on the ledger.

Figure 34 shows the basic structure of a block in a Blockchain. The extracted file has three sections; Data, Header, and Metadata.

The header section of the block has data_hash, which is a hash value calculated from all the transactions in the current block.

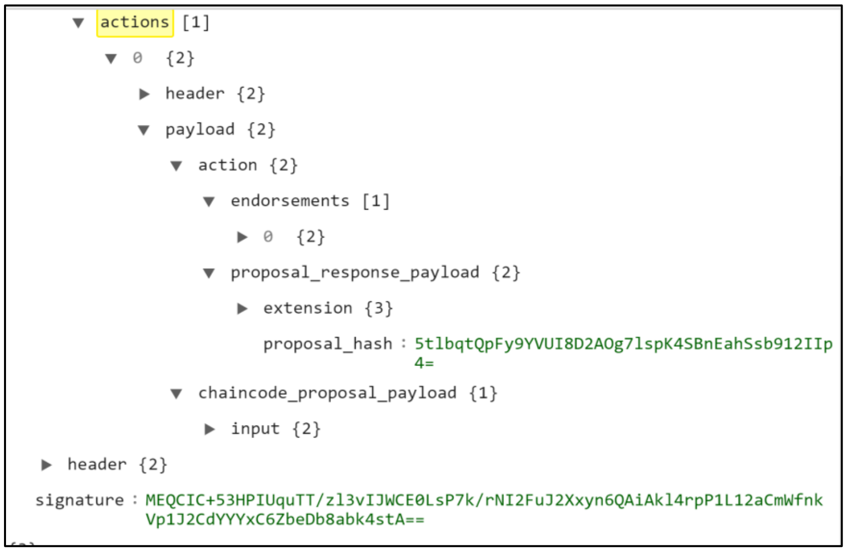


```
▼ object {3}
    ▶ data {1}
    ▼ header {3}
        data_hash : kR7gJQT6OXp0LaHMgMJNjJHAVD8+nkwyUSUxstoR
                    fN0=
        number : 2
        previous_hash : MB8NesUMz4RWALE2VE+v0792FtWdOw2TULI5
                        WK4XzuM=
    ▶ metadata {1}
```

Figure 34: Structure of a block header.

Number refers to the current block number, and It is an integer, which starts at genesis block with value zero, and the blocks on the channels add up; it gets incremented by the value of 1. *Previous_hash* holds thee value of previous block hash value, and This is what brings in the chain structure for all the blocks.

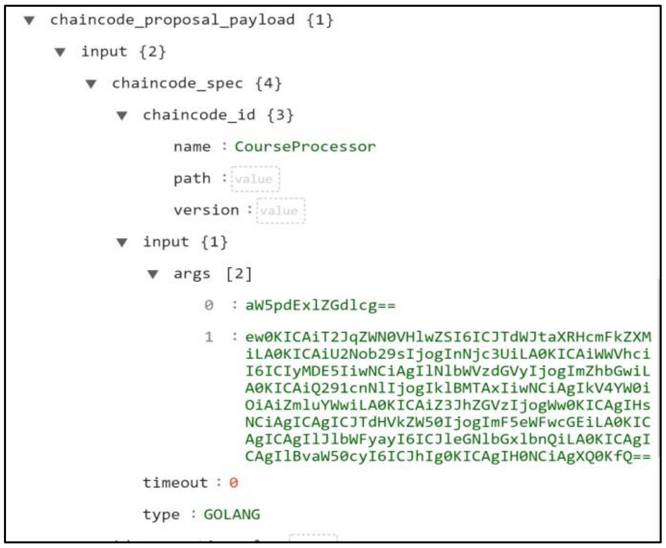
The data section of the block shown in Figure 35 has details about the transactions bundled in order. Each transaction section has three portions, Which are Transaction proposal, Endorsements, and Proposal response.



```
        ▼ actions [1]
            ▼ 0  {2}
                ▶ header {2}
                ▼ payload {2}
                    ▼ action {2}
                        ▼ endorsements [1]
                            ▶ 0  {2}
                        ▼ proposal_response_payload {2}
                            ▶ extension {3}
                              proposal_hash : 5tlbqtQpFy9YVUI8D2AOg7lspK4SBnEahSsb912IIp
                                              4=
                    ▼ chaincode_proposal_payload {1}
                        ▶ input {2}
    ▶ header {2}
    signature : MEQCIC+53HPIUquTT/zl3vIJWCE0LsP7k/rNI2FuJ2Xxyn6QAiAkl4rpP1L12aCmWfnk
                Vp1J2CdYYYxC6ZbeDb8abk4stA==
```

Figure 35: Structure of a block data -1.

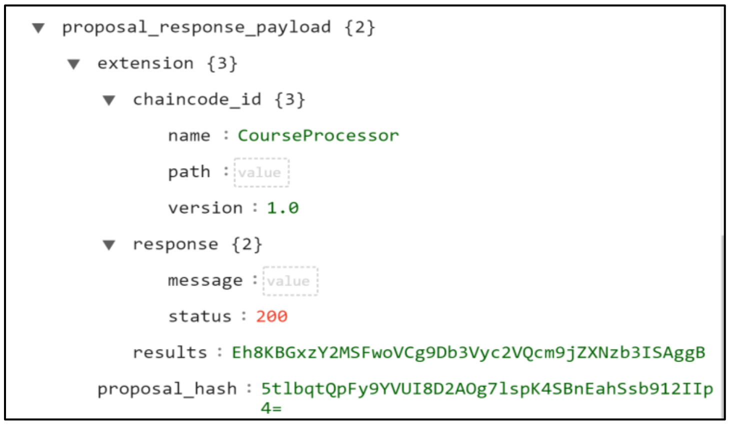
We can observe all the three portions in this block, *chaincode_proposal_payload* portion all the details about the proposal command submitted in the CLI; Such as name, path, version of chaincode, and arguments specified during instantiation in hash format and language of chaincode, In this case, it is two.



```
▼ chaincode_proposal_payload {1}
    ▼ input {2}
        ▼ chaincode_spec {4}
            ▼ chaincode_id {3}
                  name : CourseProcessor
                  path : value
                  version : value
            ▼ input {1}
                ▼ args [2]
                      0  : aW5pdExlZGdlcg==
                      1  : ew0KICAiT2JqZWN0VHlwZSI6ICJTdWJtaXRHcmFkZXM
                           iLA0KICAiU2Nob29sIjogInNjc3UiLA0KICAiWWVhci
                           I6ICIyMDE5IiwNCiAgIlNlbWVzdGVyIjogImZhbGwiL
                           A0KICAiQ291cnNlIjogIklBMTAxIiwNCiAgIkV4YW0i
                           OiAiZmluYWwiLA0KICAiZ3JhZGVzIjogWw0KICAgIHs
                           NCiAgICAgICJTdHVkZW50IjogImF5eWFwcGEiLA0KIC
                           AgICAgIlJlbWFyayI6ICJleGNlbGxlbnQiLA0KICAgI
                           CAgIlBvaW50cyI6ICJhIg0KICAgIH0NCiAgXQ0KfQ==
              timeout : 0
              type : GOLANG
```

Figure 36: Structure of a block data -2.

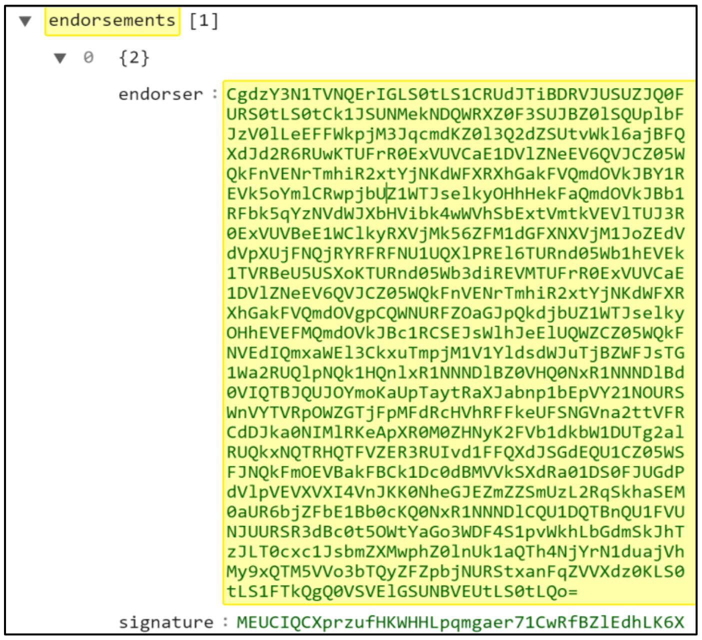
Proposal_response_payload portion of the transaction is shown in Figure 37 and has details about the details of the response of the transaction. It has related chaincode details and the response from the ledger and also the hash value calculated for the proposal submitted.



```
▼ proposal_response_payload {2}
    ▼ extension {3}
        ▼ chaincode_id {3}
              name : CourseProcessor
              path : value
              version : 1.0
        ▼ response {2}
              message : value
              status : 200
          results : Eh8KBGxzY2MSFwoVCg9Db3Vyc2VQcm9jZXNzb3ISAggB
      proposal_hash : 5tlbqtQpFy9YVUI8D2AOg7lspK4SBnEahSsb912IIp
                      4=
```

Figure 37: Structure of a block data -3.

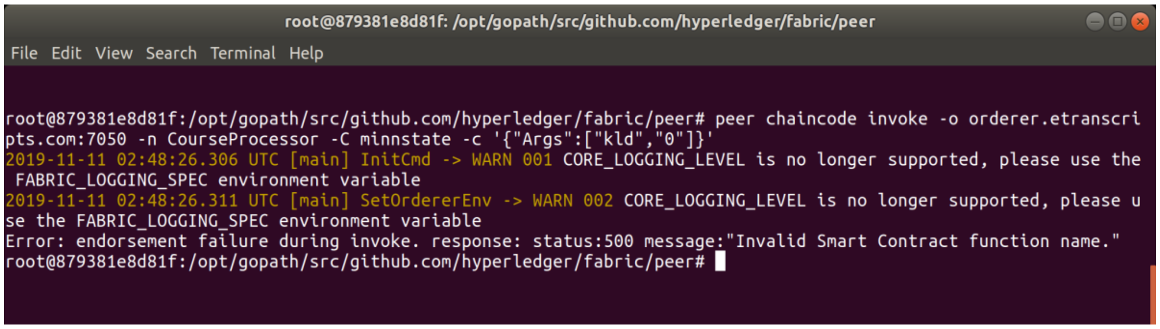
Finally, the *'Endorsement'* portion of the transaction; Which has details of endorser of this transaction, endorser ID, and his signature. In the last section, the block is *metadata*, which holds information related to the entire block, such as the time when the block is written onto ledger, certificate, keys, and signature of the block.



```
▼ endorsements [1]
   ▼ 0  {2}
        endorser : CgdzY3N1TVNQErIGLS0tLS1CRUdJTiBDRVJUSUZJQ0F
                   URS0tLS0tCk1JSUNMekNDQWRXZ0F3SUJBZ01SQUplbF
                   JzV0lLeEFFWkpjM3JqcmdKZ013Q2dZSUtvWkl6ajBFQ
                   XdJd2R6RUwKTUFrR0ExVUVCaE1DVlZNeEV6QVJCZ05W
                   QkFnVENrTmhiR2xtYjNKdWFXRXhGakFVQmdOVkJBY1R
                   EVk5oYmlCRwpjbUZ1WTJselkyOHhHekFaQmdOVkJBb1
                   RFbk5qYzNVdWJXbHVibk4wWVhSbExtVmtkVEVlTUJ3R
                   0ExVUVBeE1WClkyRXVjMk56ZFM1dGFXNXVjM1JoZEdV
                   dVpXUjFNQjRYRFRFNU1UQXlPREl6TURnd05Wb1hEVEk
                   1TVRBeU5USXoKTURnd05Wb3diREVMTUFrR0ExVUVCaE
                   1DVlZNeEV6QVJCZ05WQkFnVENrTmhiR2xtYjNKdWFXR
                   XhGakFVQmdOVgpCQWNURFZOaGJpQkdjbUZ1WTJselky
                   OHhEVEFMQmdOVkJBc1RCSEJsWlhJeE1UQWZCZ05WQkF
                   NVEdIQmxaWEl3CkxuTmpjM1V1YldsdWJuTjBZWFJsTG
                   1Wa2RUQlpNQk1HQnlxR1NNNDlBZ0VHQ0NxR1NNNDlBd
                   0VIQTBJQUJOYmoKaUpTaytRaXJabnp1bEpVY21NOURS
                   WnVYTVRpOWZGTjFpMFdRcHVhRFFkeUFSNGVna2ttVFR
                   CdDJka0NIMlRKeApXR0M0ZHNyK2FVb1dkbW1DUTg2al
                   RUQkxNQTRHQTFVZER3RUIvd1FFQXdJSGdEQU1CZ05WS
                   FJNQkFmOEVBakFBCk1Dc0dBMVVkSXdRa01DS0FJUGdP
                   dVlpVEVXVXI4VnJKK0NheGJEZmZZSmUzL2RqSkhaSEM
                   0aUR6bjZFbE1Bb0cKQ0NxR1NNNDlCQU1DQTBnQU1FVU
                   NJUURSR3dBc0t5OWtYaGo3WDF4S1pvWkhLbGdmSkJhT
                   zJLT0cxc1JsbmZXMwphZ0lnUk1aQTh4NjYrN1duajVh
                   My9xQTM5VVo3bTQyZFZpbjNURStxanFqZVVXdz0KLS0
                   tLS1FTkQgQ0VSVElGSUNBVEUtLS0tLQo=
        signature : MEUCIQCXprzufHKWHHLpqmgaer71CwRfBZlEdhLK6X
```

Figure 38: Structure of a block data -4.

Tested the system- without TLS settings on the network with TLS handshake enabled, and the CLI was not able to invoke the chaincode erroring out as Endorsement failure, as shown in Figure 39.


root@879381e8d81f: /opt/gopath/src/github.com/hyperledger/fabric/peer
File Edit View Search Terminal Help

```
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer# peer chaincode invoke -o orderer.etranscri
pts.com:7050 -n CourseProcessor -C minnstate -c '{"Args":["kld","0"]}'
2019-11-11 02:48:26.306 UTC [main] InitCmd -> WARN 001 CORE_LOGGING_LEVEL is no longer supported, please use the
 FABRIC_LOGGING_SPEC environment variable
2019-11-11 02:48:26.311 UTC [main] SetOrdererEnv -> WARN 002 CORE_LOGGING_LEVEL is no longer supported, please u
se the FABRIC_LOGGING_SPEC environment variable
Error: endorsement failure during invoke. response: status:500 message:"Invalid Smart Contract function name."
root@879381e8d81f:/opt/gopath/src/github.com/hyperledger/fabric/peer# 
```

Figure 39: Testing without TLS.

Summary

In this chapter, we had discussed about results of developing the architecture. We also looked at the results of running the application and in-depth documentation of implementing the architecture. This chapter also talks about troubles faced and how to overcome them during development.

Chapter V: Results, Conclusion, and Recommendations

Introduction

This chapter discusses the overall summary of the results obtained from this implementation. We looked at the results of the application used to connect educational organizations to share sensitive information related to students. The project mainly focusses on developing a network for safe and secure communication without any middleware organizations to take care of data integrity and confidentiality. This chapter mainly focusses on concluding the entire study. This section also discusses about the future work that can be applied to the current system.

Discussion and Results

This research discusses in detail various cryptocurrency frameworks and architectures. Discussions and comparisons of the frameworks, starting with Etherueum, Ethereum manages a public blockchain, and It is just like a social network, any person who is willing to be part of the network can join and start doing transactions using smart contracts. Moreover, the Ethereum framework does not support multi-channel communication. This implementation can use Ethereum architecture, but it not suitable for this project. Whereas, an Hyperledger Fabric manages a permissioned blockchain.

Furthermore, unlike Ethereum, which runs on Mining based on Proof of Work, which is managed by its participants, HLF architecture provides an Ordering service, which is an automated system that helps in maintaining consensus among the peers. Hyperledger Fabric is very scalable; it supports multiple programming languages and the ability to integrate components such as consensus algorithms and membership services, which issues and validates certificates.

Regarding the security provided by this architecture, Hyperledger Fabric bundles in TLS encryption and Membership Service providers for proper certificate handling. The data on the ledger is by default, encrypted by Hyperledger Fabric native encryption. Even communication between the peer requires a TLS, secure connection between the two nodes. Channels in Hyperledger Fabric adds up another layer of security, as even though a node is a member of the network cannot access any data If the node is not part of the channel.

The network handled the upgrades as expected. It supports easy install of chaincode smart contract codes on nodes, and the command *upgrade* instantiates the installed chaincode on the network.

This paper also dives into giving step by step details of developing an entire network of organizations using the architecture on a Linux based Virtual machine. This paper also discusses a structure of block with multiple transactions and how the structure helps in making the entire blockchain immutable. This paper also discusses how to implement a consensus mechanism and to bring in the ability to avoid middle man and how well the architecture is designed that it can take care of the transactions and consensus mechanism. By providing a ledger that nobody administers, these crypto blockchain systems with their strict consensus methods can handle any transactional data with trust and accuracy.

Conclusion

This implementation study of Hyperledger fabric is to develop a network between the participating educational organizations; This is an open-source architecture that aims to develop distributed ledger applications. Since the application manages the ledger without any administering it, It needs to have a consensus algorithm, and to look at the blocks, the tool generated and crypto algorithms it uses in the process makes the ledger immutable, and every transaction needs to be signed, verified and valid.

Future Work

Currently, there are only a few stable releases for this architecture, the version that was used is the most up to dated and stable version of this opensource project hosted by the Linux Foundation. This architecture supports various plug and play services. Currently, able to use the project using the command-line interface. This project can be further improved by adding Nodejs and a UI with authentication services, which makes it easier for the end-user to work on the tool.

References

Ark, T. V. (2018, August 20). 20 ways blockchain will transform (okay, may improve) education. Retrieved January 15, 2019, from https://www.forbes.com/sites/tomvanderark/2018/08/20/26-ways-blockchain-will-transform-ok-may-improve-education/#450f3cf74ac9

Cachin, C. (2016). *Architecture of the hyperledger blockchanin fabric*. Retrieved from https://www.zurich.ibm.com/dccl/papers/cachin_dccl.pdf

Chen, G., Xu, B., Lu, M. (2019). Exploring blockchain technology and its potential applications for education. *Smart Learning Environment*, 5(1). Retrieved February 02, 2019, from https://slejournal.springeropen.com/articles/10.1186/s40561-017-0050-x

Gibson, K. (2017, May 9). *Your MD may have a phony degree*. Retrieved January 25, 2019, from https://www.cbsnews.com/news/your-md-may-have-a-phony-degree/

Gräther, W., Kolvenbach, S., Ruland, R., Schütte, J., Torres, C., & Wendland, F. (2018, May 8). *Blockchain for education: Lifelong learning passport*. Retrieved March 21, 2019, from https://dspace.wineme.fb5.uni-siegen.de/handle/20.500.12015/3163

Hong, Z., Wang, Z., Cai, W., & Leung, V. (2017). Blockchain-empowered fair computational resource sharing system in the D2D network. *Future Internet. 9.85*. 10.3390/fi9040085

Kumar, A. (2018, July 3). *Bitcoin blockchain–what is proof of work?* Retrieved February 25, 2019, from https://vitalflux.com/bitcoin-blockchain-proof-work/

Madeira, A. (2019, March 13). *How does a hashing algorithm work?* Retrieved March 25, 2019, from https://www.cryptocompare.com/coins/guides/how-does-a-hashing-algorithm-work/

Mukhopadhyay, U., Skjellum, A., & Hambolu, O. (2017, April). *A brief survey of cryptocurrency systems*. Retrieved from ieeexplore.ieee.org: https://ieeexplore.ieee.org/document/7906988/

Orientation and setup. (2019, November 11). Retrieved November 11, 2019, from Docker Documentation website: https://docs.docker.com/get-started/

Sean: Understanding Hyperledger in a bit more detail. (2017, December 21). Retrieved March 20, 2019, from https://decentralize.today/understanding-hyperledger-in-a-bit-more-detail-3d40a37c74f2

Stevens, A. (2018, April 23). *Gaining clarity on key terminology: Bitcoin versus blockchain versus distributed ledger technology*. Retrieved February 21, 2019, from https://hackernoon.com/gaining-clarity-on-key-terminology-bitcoin-versus-blockchain-versus-distributed-ledger-technology-7b43978a64f2

Valenta, M., & Sandner, P. (2017, June). Comparison of Ethereum, Hyperledger fabric, and Corda. Retrieved March 31, 2019, from https://pdfs.semanticscholar.org/00c7/5699db7c5f2196ab0ae92be0430be4b291b4.pdf

Appendix A: Additioal Sources

A practical introduction to blockchain with Python // Adil Moujahid // Data Analytics and more. (n.d.). Retrieved November 11, 2019, from http://adilmoujahid.com/posts/2018/03/intro-blockchain-bitcoin-python/

Audhikesavan, L. (2018, June 30). *Hyperledger fabric: How to setup application from scratch using Nodejs series—part 3*. Retrieved November 11, 2019, from Medium website: https://medium.com/coinmonks/hyperledger-fabric-how-to-setup-application-from-scratch-using-nodejs-series-part-3-9d795f2d4a8

Audhikesavan, L. (2019, September 24). *Blockchain Hyperledger fabric—errors & solutions*. Retrieved November 11, 2019, from Medium website: https://medium.com/coinmonks/hyperledger-fabric-composer-errors-solutions-827112a3fce6

Blockcerts. (2016). *Blockchain credentials*. Retrieved from https://www.blockcerts.org/guide/

Health, C. (2019, August 7). *Start your own Hyperledger blockchain, the easy way!* Retrieved November 11, 2019, from Medium website: https://medium.com/@mycoralhealth/start-your-own-hyperledger-blockchain-the-easy-way-5758cb4ed2d1

How the blockchain works. (n.d.). Retrieved January 20, 2019, from .https://rubygarage.org/blog/how-blockchain-works

Hyperledger Fabric—The 20 most important terms made simple. (n.d.). Retrieved November 11, 2019, from https://hackernoon.com/hyperledger-fabric-the-20-most-important-terms-made-simple-2753f925db4

Install Hyperledger fabric on Ubuntu 18.04.1–Step by Step « Data Science Evangelist. (n.d.). Retrieved November 11, 2019, from http://www.ziaahmedshaikh.com/install-hyperledger-fabric-on-ubuntu-18-04-1-step-by-step/

Nakamoto, S. (2008, October 31). Bitcoin: A peer-to-peer electronic cash system. Retrieved January 20, 2019, from https://nakamotoinstitute.org/bitcoin/

Peer channel—Hyperledger-fabricdocs master documentation. (n.d.). Retrieved November 11, 2019, from https://hyperledger-fabric.readthedocs.io/en/release-1.4/commands/peerchannel.html

Private data—Hyperledger-fabricdocs master documentation. (n.d.). Retrieved November 11, 2019, from https://hyperledger-fabric.readthedocs.io/en/release-1.4/private-data-arch.html

Sharples, M., & Domingue, J. (2016, September 13). *The Blockchain and kudos: A distributed system for educational record, reputation, and reward*. retrieved February 15, 2019, from https://link.springer.com/chapter/10.1007/978-3-319-45153-4_48

Tam, K. C. (2019, July 22). *Transactions in Hyperledger fabric*. Retrieved November 11, 2019, from Medium website: https://medium.com/@kctheservant/transactions-in-hyperledger-fabric-50e068dda8a9

Technologies, B. (2018, July 10). *Hyperledger fabric: Time to make the leap—an enterprise note*. Retrieved November 11, 2019, from Medium website: https://medium.com/@BangBitTech/hyperledger-fabric-time-to-make-the-leap-an-enterprise-note-78b062d1c6bf

Wearetheledger/awesome-hyperledger-fabric. (2019). Retrieved from https://github.com/wearetheledger/awesome-hyperledger-fabric (Original work published 2018).

What is hashing? Under the hood of blockchain. (n.d.). Retrieved February 21, 2019, from https://blockgeeks.com/guides/what-is-hashing/

What is Ethereum? The Most comprehensive beginners guide. (n.d.). Retrieved January 21, 2019, from https://blockgeeks.com/guides/ethereum/

Your first Hyperledger fabric network. (n.d.). Retrieved November 11, 2019, from The DEV Community website: https://dev.to/damcosset/your-first-hyperledger-fabric-network-2n67

Yousef, H. (n.d.). *Install HyperLedger fabric at Win 10 | Codementor*. Retrieved November 11, 2019, from https://www.codementor.io/hajsf/install-hyperledger-fabric-at-win-10-tb85r9dqg

Appendix B: File crypto-config.yaml

File crypto-config.yaml is created for the tool Cryptogen. It holds the data related to all participants of the network. *Cryptogen* tool uses the following YAML code and generates encryption keys and certificates.

```
OrdererOrgs:
  - Name: orderer
    Domain: etranscripts.com
    EnableNodeOUs: true
    Specs:
      - Hostname: orderer
PeerOrgs:
  # ---------------------------------------------------------------------------
  # Scsu
  # ---------------------------------------------------------------------------
  - Name: scsu
    Domain: scsu.minnstate.edu
    EnableNodeOUs: true
    # ---------------------------------------------------------------------------
    Template:
      Count: 1
    Users:
      Count: 1
  - Name: umn
    Domain: umn.minnstate.edu

    EnableNodeOUs: true

    Template:

      Count: 1

    Users:

      Count: 1


  # Minnesota State University, Mankato:

  - Name: msu

    Domain: msu.minnstate.edu

    EnableNodeOUs: true

    Template:

      Count: 1

    Users:

      Count: 1

  # Bemidji State University:

  - Name: bsu

    Domain: bsu.minnstate.edu

    EnableNodeOUs: true

    Template:

      Count: 1

    Users:

      Count: 1
```

Configtxgen tool uses file configtx.YAML, This file contain all the network related configurations such as Anchor peer configurations for Organizational peers, Channel information, ledger details and Orderer configurations.

```
########################################################################
Section: Organizations
#########################################################################
Organizations:
  - &OrdererOrg
    Name: ordererOrg
    ID: ordererMSP
    MSPDir: crypto-config/ordererOrganizations/etranscripts.com/msp
  - &scsu
    Name: scsuMSP
    ID: scsuMSP
    MSPDir: crypto-config/peerOrganizations/scsu.minnstate.edu/msp
    AnchorPeers:
      - Host: peer0.scsu.minnstate.edu
        Port: 7051
  - &umn
    Name: umnMSP
    ID: umnMSP
    MSPDir: crypto-config/peerOrganizations/umn.minnstate.edu/msp
```

```
        AnchorPeers:
            - Host: peer0.umn.minnstate.edu
              Port: 8051
    - &msu
        Name: msuMSP
        ID: msuMSP
        MSPDir: crypto-config/peerOrganizations/msu.minnstate.edu/msp
        AnchorPeers:
            - Host: peer0.msu.minnstate.edu
              Port: 9051
    - &bsu
        Name: bsuMSP
        ID: bsuMSP
        MSPDir: crypto-config/peerOrganizations/bsu.minnstate.edu/msp
        AnchorPeers:
            - Host: peer0.bsu.minnstate.edu
              Port: 10051
Orderer: &OrdererDefaults
    OrdererType: solo
    Addresses:
        - orderer.etranscripts.com:7050
    BatchTimeout: 122s
    BatchSize:
        MaxMessageCount: 9
        AbsoluteMaxBytes: 9 MB
        PreferredMaxBytes: 256 KB
Profiles:
    eTranscriptsGenesis:
        Orderer:
            <<: *OrdererDefaults
            Organizations:
                - *OrdererOrg
            Capabilities:
                <<: *OrdererCapabilities
        Consortiums:
            MinnStateUniv:
                Organizations:
                - *scsu
                - *umn
                - *msu
                - *bsu
    eTranscriptsChannel:
        Consortium: MinnStateUniv
        Application:
            <<: *ApplicationDefaults
            Organizations:
```

```
    - *scsu
    - *umn
    - *msu
    - *bsu
  Capabilities:
    <<: *ApplicationCapabilities
```

//chaincode for project

This chaincode file is to work with a ledger. This chaincode, after installed on a peer container, acts as a separate container. So when a request to submit a transaction on the ledger. The chaincode needs to be invoked, and the chaincode will query the ledger. The chaincode should have init, initiate methods, and other custom query methods.

```
package main

import (
	"encoding/json"
	"fmt"
	"github.com/hyperledger/fabric/core/chaincode/shim"
	pb "github.com/hyperledger/fabric/protos/peer"
	"strings"
)
```

```
type OrgGrades struct {
}


type submitgrade struct {
        ObjectType string `json:"docType"`
        School     string `json:"school"`
        Semester   string `json:"semester"`
        Year       string    `json:"year"`
        Course     string `json:"course"`
        Grade      string `json:"grade"`
        Name       string `json:"name"`
}
//
==================================================================
==============
// Main
//
==================================================================
==============
func main() {
        err := shim.Start(new(OrgGrades))
        if err != nil {
                fmt.Printf("Error starting Simple chaincode: %s", err)
```

```
	}
}

// Init initializes chaincode
// ===========================
func (t *OrgGrades) Init(stub shim.ChaincodeStubInterface) pb.Response {
	return shim.Success(nil)
}

// Invoke -
// ========================================
func (t *OrgGrades) Invoke(stub shim.ChaincodeStubInterface) pb.Response {
	function, args := stub.GetFunctionAndParameters()
	fmt.Println("invoke is running " + function)
	if function == "initGrade" {
		return t.initGrade(stub, args)
	}else if function == "readGrade" {
		return t.readGrade(stub, args)
	}
	fmt.Println("Function not found for Invoke method: " + function)
	return shim.Error("Received wrong function")
}
func (t *OrgGrades) initGrade(stub shim.ChaincodeStubInterface, args []string) pb.Response {
```

```
var err error


if len(args) != 6 {
	return shim.Error("Incorrect number of arguments. Expecting 6")
}
fmt.Println("- start init grade")
if len(args[0]) <= 0 {
	return shim.Error("1st argument not supplied")
}
if len(args[1]) <= 0 {
	return shim.Error("2nd argument not supplied ")
}
if len(args[2]) <= 0 {
	return shim.Error("3rd argument not supplied ")
}
if len(args[3]) <= 0 {
	return shim.Error("4th argument not supplied ")
}
if len(args[4]) <= 0 {
	return shim.Error("5th argument not supplied ")
}
if len(args[5]) <= 0 {
	return shim.Error("6th argument not supplied ")
```

```
}
school := args[0]
semester := strings.ToLower(args[1])
year := strings.ToLower(args[2])
course := strings.ToLower(args[3])
grade := strings.ToLower(args[4])
name := strings.ToLower(args[5])
nameAsBytes, err := stub.GetState(name)
if err != nil {
	return shim.Error("Failed to get name: " + err.Error())
} else if nameAsBytes != nil {
	fmt.Println("This name already exists: " + name)
	return shim.Error("This name already exists: " + name)
}
objectType := "submitgrade"
submitgrade := &submitgrade{objectType, school, semester, year, course,grade,name}
gradeJSONasBytes, err := json.Marshal(submitgrade)
if err != nil {
	return shim.Error(err.Error())
}

err = stub.PutState(name, gradeJSONasBytes)
if err != nil {
		return shim.Error(err.Error())
	}


	indexName := "school~semester~year~course"
	ssycIndexKey, err := stub.CreateCompositeKey(indexName,
[]string{submitgrade.School, submitgrade.Semester,submitgrade.Year,submitgrade.Course})
	if err != nil {
		return shim.Error(err.Error())
	}
	value := []byte{0x00}
	stub.PutState(ssycIndexKey, value)
	fmt.Println("- end init name")
	return shim.Success(nil)
}



func (t *OrgGrades) readGrade(stub shim.ChaincodeStubInterface, args []string) pb.Response {
	var name, jsonResp string
	var err error


	if len(args) != 1 {
		return shim.Error("Please check arguments. Expecting name of the student to
query")
```

```
        }

        name = args[0]
        valAsbytes, err := stub.GetState(name)
        if err != nil {
                jsonResp = "{\"Error\":\"not able to check " + name + "\"}"
                return shim.Error(jsonResp)
        } else if valAsbytes == nil {
                jsonResp = "{\"Error\":\"name does not exist: " + name + "\"}"
                return shim.Error(jsonResp)
        }

        return shim.Success(valAsbytes)
}
```