Blockchain Application for Verification of Passports

BY

Muhammad Ahmed Saeedi

Fall-2015-MS-SYS&E00000118721

Supervised by

Dr. Jamil Ahmad

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTERS OF SCIENCE
IN
SYSTEMS ENGINEERING

Research Centre for Modeling and Simulation(RCMS)

National University of Sciences and Technology (NUST)
Islamabad, Pakistan

September, 2018

I would like to dedicate this thesis to my loving mother and sisters.

Certificate

It is hereby declared that except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other University. The work presented in this thesis is the result of my own work.

Muhammad Ahmed Saeedi

September, 2018

Acknowledgement

All praise for **Almighty ALLAH** Who is the ultimate source of all knowledge. Almighty Allah has made me reach this present pedestal of knowledge with quality of doing something novel, stimulating and path bearing. All respects are for Holy Prophet Hazrat Muhammad (PBUH) who is the symbol of guidance and fountain of knowledge.

I earnestly thank to my supervisor **Dr. Jamil Ahmad**, for his keen interest, invaluable guidance, encouragement and continuous support during my research work. I am grateful for his thought provoking and illuminating discussions, sound advices, encouragement, and valuable suggestions. He enabled me not only to tackle the problems more meaningfully on the subject but also provided an easy access to work seriously & sincerely to quest after my objectives. I want to thank him for providing me such a scientific knowledge which will help all of the humanity in a long run.

I am thankful to my GEC committee members and other faculty members of RCMS who have been very kind enough to extend their help at various phases of this research, whenever I approached them, and I do hereby acknowledge all of them. I thank **Engr. Sikandar Hayat Mirza**, **Dr. Mian Ilyas Ahmad** and **AP Muhammad Tariq Saeed** for their valuable suggestions and concise comments on some of the research papers of the thesis. I am very thankful to Engr. Usman and Sir hassan for providing us the facilities and research conducive environment at RCMS.

No words can express and no deeds can return the support and inspiration that my family especially my sister **Rahila Batool** permeated in me during the course of my research work. Deepest thanks to my parents, brother and sisters whose prayers, patience, guidance & positive criticism helped me throughout my academic life and particularly during this phase of my research. I am also thankful to **my all friends** for their continuous moral support without whom it may be impossible to reach this stage.

Contents

Abbreviations

Acronyms / Abbreviations

ACL	Access Control Language
BAC	Basic Access Control
BNA	business network archive
CAN	Card Authentication Number
CFT	Crash Fault Tolerant
CLI	Command Line Interface
CRUD	Create, Read, Update or Delete
DAO	Decentralized Autonomous Organization
EAC	Extended Access Control
EACv1	Extended Access Control Version 1
ECC	Elliptic Curve Cryptography
EIP	Ethereum Improvement Proposals
ICAO	International Civil Aviation Organization
MRP	Mahine Readable Passport
MRZ	Machine Readable Zone
OCR	Optical Character Recognition
PACE v2	Password Authenticated Key Agreement Protocol
PIN	Personal Identification Number
PKI	Private Key Infrastructure
PoET	Proof of Elapsed Time
Regex	regular expression
RFID	Radio-frequency Identification
RSA	Rivest Shamir Adleman
SAC	Supplemental Access Control
UML	Unified Modeling Language
UTXO	Unspent Transaction Output

List of Figures

ABSTRACT

Passports are essential traveling documents and are used to verify the credentials of passport holder and verify the authenticity of visas. Pakistan and many other countries have started the use of Machine Readable Passports (MRP). The reason behind moving from traditional passports to MRP was to not only enhance the security of passports, but also to speed up the process of passport verification. There are security mechanisms implemented by International Civil Aviation Organization (ICAO) for passport verification and these mechanisms are standardized and implemented in all over the world. But despite of these security measurements in place, hackers can still retrieve sensitive credentials from passports by exploiting vulnerabilities in security mechanisms. The blockchain is considered as a futuristic infrastructure, as no other technology provides functionalities and decentralization parallel to it. Furthermore, it is designed to be distributed and synchronized across networks. In this research, a blockchain application, Cryptopassports, has been designed and implemented to verify the legitimacy of passports. Furthermore, it will enable the participants on the network to verify the passport instantly. Due to the immutable aspect of data stored on the blockchain, it will be almost impossible for hackers to hack or alter data. Sensitive information about the passport holder will only be accessible to the participant on the network. In addition, the application has been tested on a physical network by creating three different participants and granting them access to the blockchain according to their roles in the network. Participants can only access information and carry out operations that they are given access to. For example, a network administrator will not be able to carry out any operations on passports because his/her responsibility will only be to maintain network and its participants. Furthermore, cryptopassport has been tested by using data set of twenty passports and now it is ready for real world deployment.

Chapter 1

INTRODUCTION

1.1 Background

Passports have become an essential travelling document. They are used to identify the passport holder and verify his/her credentials. Before the technological advancement passports were easy to fake and modify as there was no foolproof way or method for verification. With the technological advancement security of passports increased but along with that security risks and vulnerabilities related to those security measurements were exposed.

In Pakistan and all over the world there are many cases in which people used fake passport to cross borders and enter into a country therefore making it a liability on that country of which passport that person holds. Currently there is not a single foolproof way available by which we can trace and verify the authenticity of any passport. All records are saved in traditional centralized database systems which are vulnerable and data can be altered by using influential means. There are certain amount of limitations in centralized database systems. These systems are highly dependent on network connectivity. If the same set of data is to be accessed by multiple people, it can lead to major decrease in general efficiency of the system. A single copy is maintained in a single location, which makes it a single point of failure Figure 1.1.

Pakistan and many other countries have started the use of Machine Readable Passports (MRP) [1]. The reason behind moving from traditional passports to MRP was not only to enhance the security of passports, but also to speed up the process of passport verification. These methods are standardized and implemented by the

[1] http://www.dgip.gov.pk/Files/Passport.aspx

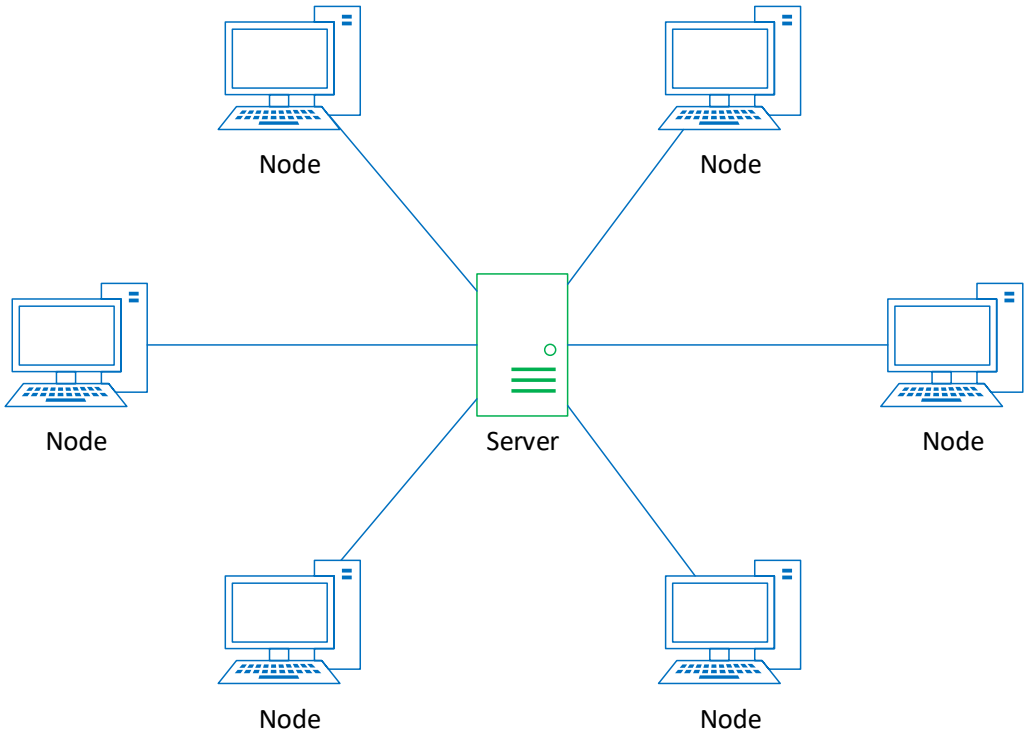

Figure 1.1. Centralized Network

International Civil Aviation Organization (ICAO) [17]. It was ensured that passports were machine readable and human friendly as they must be read by border control officials.

Smartcards are used in passports and they have contactless interface using RFID [7]. To make the passport verification process more secure and faster these features were introduced. However, the problem with these passports was skimming of data and the most critical issue with this system is that it uses centralized database. Central Database is a single point of failure for the whole system if it is corrupted or if it becomes inaccessible.

Last year, an identity confirmation service carried out research on detecting fake passports. The purpose of the research was to gauge how good public is at identifying the fake passports and the results were worrying. Research was carried out on a sample of 1013 people and 68% of those people were unable to identify a

fake passport [2]. Interpol also released the data about lost and stolen travel documents in the European Union and worldwide. There has been a sharp uptake in the number of missing passports [3].

Blockchain is a 'peer to peer' network Figure 1.2. This network is distributed among all the participant or nodes on a network. In traditional methods, it is not possible to transfer digital assets without any intermediaries. In the case of blockchain network, due to its unique principle it is possible to transfer digital assets without any intermediaries. This technology was originally created to support the famous cryptocurrency known as Bitcoin. Bitcoin was the first application of blockchain whose network is distributed across thousands of peers or nodes. Blockchain uses 'peer to peer' communication mode for digital assets transfer. Furthermore, this technology is based on decentralized transaction and data management aspects. All peers in the network verify each transaction by running consensus [12].

The blockchain is poised to innovate and transform a wide range of applications, including goods transfer, (e.g. Supply chain), digital media transfer, (e.g. Sale of art) and remote services delivery, (e.g. Travel and tourism). It can be utilized as a platform for decentralized business logic, for example, moving computing to data sources and distributed intelligence, for example, education credentialing.

1.2 Problem Statement

In past few years, in Pakistan only, there has been a significant rise in scandals related to fake passports. The implementation of biometric system is still not able to stop identity theft [15]. Earlier this year, a Pakistani man was arrested in Malaysia with 62 fake passports [4]. In 2012, a statement was made by British High Commissioner Adam Thomson in which he said that Pakistan is a world leader in the visa fraud

[2]https://www.hooyu.com/resources/blog/70/Majority-of-people-are-unable-to-spot-fake-passports-new-research-reveals

[3]https://www.politico.eu/article/europes-fake-forged-stolen-passport-epidemic-visa-free-travel-rights

[4]https://tribune.com.pk/story/1477053/pakistani-man-arrested-malaysia-62-fake-passports/

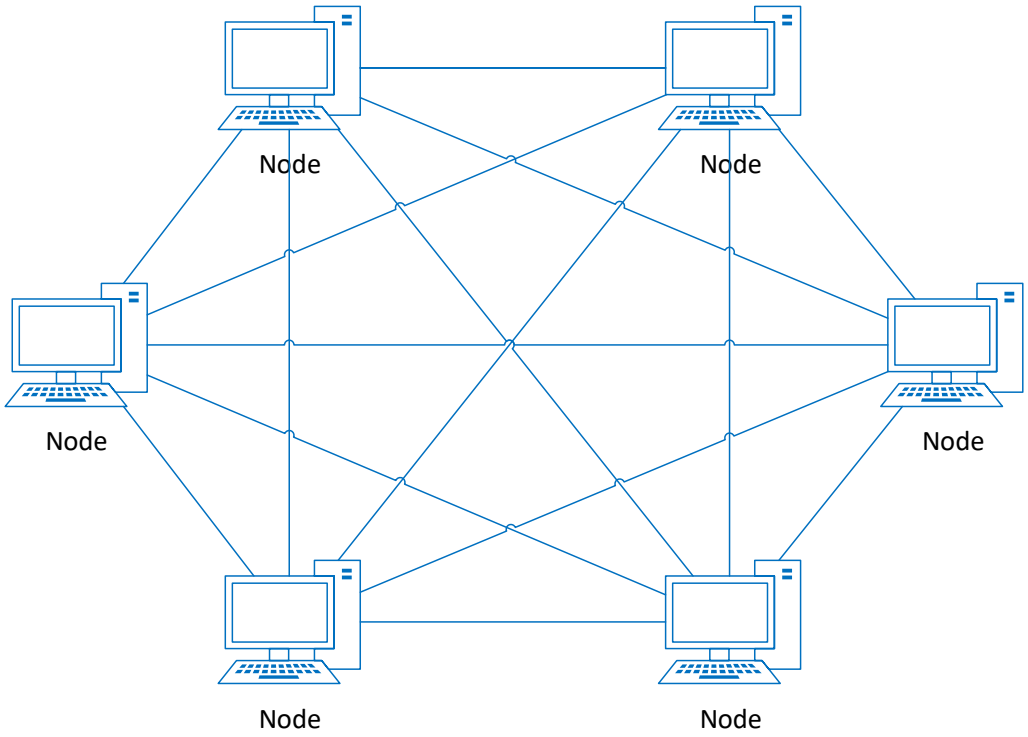

Figure 1.2. Peer to Peer Network

business. Therefore, we must meticulously check every single application, every single passport and every single document. And the high commissioner was also quoted as saying that the UK visa officials had spotted as many as 4000 fake documents submitted by Pakistanis seeking travel documents last year [5].

1.3 Aims and Objectives

Blockchain application, cryptopassport, is to be used by all entities that are identified initially which are embassies, passport offices and airports in Pakistan and any other country. Due to the immutable characteristic of records in Blockchain application, this approach will eliminate the use of fake passports, record forgery or manipulation. Cryptopassport will help in verifying every passport by verifying and backtracking history of that passport which will help in fighting counterfeit and stolen

[5] https://tribune.com.pk/story/413843/ pakistan-is-world-leader-in-visa-fraud-business-british-high-commissioner

passports Figure 1.3. The main objectives of the project are as follows:

- To set up Blockchain network
- To design, model and verify the smart contract for passport verification
- To test and debug the smart contract
- To deploy smart contract to the Blockchain network
- To test Blockchain Application for Passport Verification
- To develop web application for online verification services

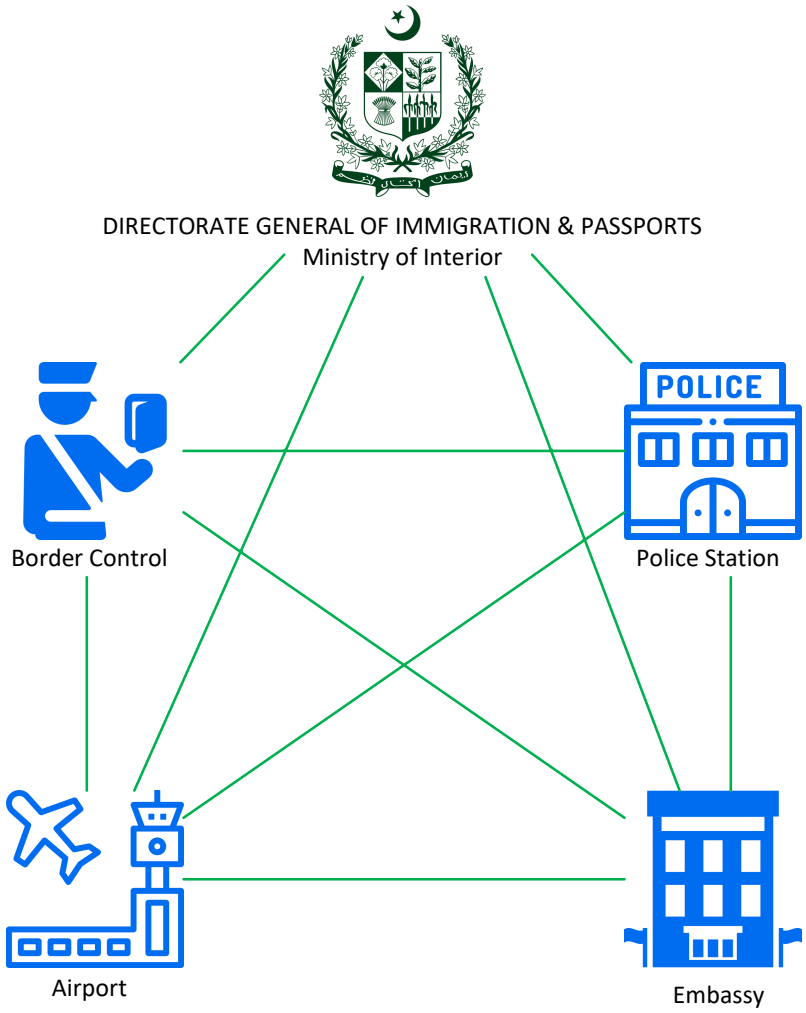

Figure 1.3. Participants of a blockchain network

Chapter 2

THE BLOCKCHAIN TECHNOLOGY

2.1 Evolution

The advent of the internet in the world wide web has transformed every aspect of our lives, from stock markets to street corner food trucks. It has enabled a technology explosion with Web 2.0 and the world of e-commerce applications. Around 2008, 2009, when the trusted institutions and markets went crumbling down, and everybody was running away from the Wall Street, a mysterious person called Satoshi Nakamoto, introduced a new digital cryptocurrency called Bitcoin. Bitcoin enabled an innovative platform for peer to peer transfer of value without any central authority by implementing software programs for validation, verification, consensus in a novel infrastructure called the blockchain. Later, in 2012, 2013, computation elements were added to the blockchain infrastructure that has opened a whole world of possibilities beyond simple currency transfer. These innovations are significantly shaping the direction of Web 3.0 [1].

The concept of the smart contract was there well before the advent of the Bitcoin. Computer scientist, Nick Szabo, detailed his idea of cryptocurrency Bit gold as a sort of a precursor for Bitcoin. He also outlined the concept of smart contract in his 1996 publication. In fact, Szabo coined the term smart contract more than 20 years ago [22]. Smart contract is a centerpiece and powerful feature of a blockchain. Bitcoin has a script feature that includes rules and policies. Linux Foundation's Hyperledger blockchain has a smart contract feature called Chaincode. The Chaincode is written in Go language and executed in a docker environment. Docker is a lightweight container technology for executing programs.

[1] https://www.pcmag.com/article/351486/blockchain-the-invisible-technology-thats-changing-the-wor

2.2 Blockchain

Blockchain technology was initially developed for Bitcoin cryptocurrency [13] which is based on decentralized transaction and data management aspects. Each transaction is verified by all peers in the network by running consensus algorithms. Ledgers on Blockchain networks are all most immune to hacking attacks. A block is a record that could be anything like a bank statement. These records are saved in the form of transactions Figure 2.1. Number of blocks join to form a blockchain Figure 2.2.

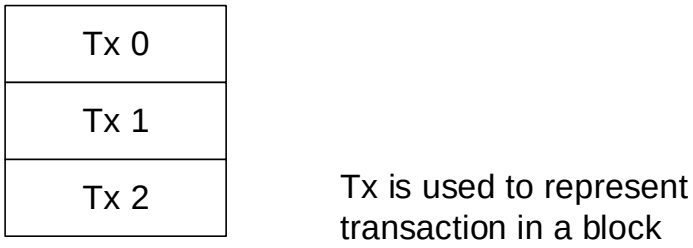

Figure 2.1. Block

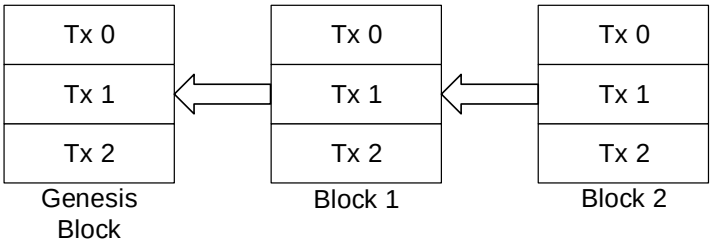

Figure 2.2. Blocks form a blockchain

There are two types of ledgers which are used in Blockchain technology: permission-less and private ledgers [21]. Ledgers in case of Bitcoin are public permission-less shared ledger that is ideal for cryptocurrencies transactions but not for private business transactions since business transaction are needed to be transparent as well as secure. Blockchain are mainly used for Bitcoin applications but now researchers are more focused on non-Bitcoin applications such as document forgery verification,

supply chain and others due to process integrity, transparency, immutability, reliability and efficient transactions.

2.2.1 Decentralization

To understand the difference between centralized and decentralized network, consider a scenario where customer wants to buy an item using his/her credit card. Let's enumerate the intermediaries involved in accomplishing this task. There is a credit card agency, a customer bank, a credit cards bank, an exchange, merchant's bank, and finally, the merchant. This is an example of a centralized system. Now compare this with a decentralized system where peers can transact directly with each other irrespective of where they are located. Functions of the intermediaries are shifted to the periphery to the peer participant in the blockchain infrastructure. Peers are not necessarily known to each other.

2.2.2 Trust in Decentralization

Trust in decentralized system is established by validating and verifying transactions. Furthermore, such transactions are recorded in temper-proof distributed ledger thus creating chain of blocks. In addition, consensus protocol is implemented for blocks to be added to the chain. So, validation, verification, consensus, and immutable recording lead to the trust and security of the blockchain [19]. Consider a scenario in which Ali is lending Amy $10,000. This is one single peer to peer transaction.Both of them make a note of it on a ledger. What if Ali changes his entry from 10,000 to 11,000? Alternatively, Amy changes hers from 10,000 to 1,000. To prevent this trust violation, they need to seek the help of people around them and provide them all a valid copy of this ledger. This is the basic concept of an immutable distributed ledger defined in a blockchain process. In this scenario, all were physically present in one location. In online transaction there are unknown peers and transactions scale up to 10,000 or about a million transactions. A person should be able to transact with equal ease to

any unknown peer in Pakistan, Albany, or Albania. This is the tenet of a decentralized system supported by blockchain.

Trust is maintained with our unknown peers through verification and validation. In above example, Amy requests Kevin to verify the amount Ali transacted with her. Kevin checks it,and finds the amount of the transaction is not 10,000, but 300, not valid then he rejects and nullifies the transaction. Like this case, validation, then verification methods devised by the blockchain and implemented by the peers, provide the collector trust needed in a decentralized system. Summarizing, blockchain technology supports methods for a decentralized peer-to-peer system, a collective trust model and a distributed immutable ledger of records of transactions.

2.2.3 Operations

Operations in the decentralized network are the responsibility of the peer participants and their respective computational nodes. For example, laptop, desktop, and server racks. These operations include validation transactions, gathering the transactions for a block, broadcasting the ballot transactions in the block, and consensus on the next block creation and chaining the blocks to form an immutable record.

There are two major roles for the participants. Participants that initiate transfer of value by creating a transaction, additional participants called miners, who pick on added work or computation to verify transactions, broadcast transaction, compete to claim the right to create a block, work on reaching consensus by validating the block, broadcasting the newly created block, and confirming transactions [16].

The participant would take on additional work because the miners are incentivized with bitcoins for the efforts in managing the blockchain. Transaction validation is carried out independently by all miners. The process involves validation of more than 20 criteria, including size and syntax. Some of these criteria are: Referenced Input Unspent Transaction Output, UTXOs are valid, reference output UTXOs are correct, reference input amount and output amount matched sufficiently, invalid transactions

are rejected and will not be broadcast. All the valid transactions are added to a pool of transactions. Miners select a set of transaction from this pool to create a block. This creates a challenge. If every miner adds the block to the chain, there will be many branches to the chain, resulting in inconsistent state. A system is implemented to overcome this challenge. Miners compete to solving a puzzle to determine who earn the right to create the next block. In the case of bitcoin blockchain, this parcel is a computation of parcel and the central processing unit or CPU intensive. Once a miner solves the puzzle, the announcement is broadcast to the network and the block is also broadcast to the network. Then, another participant verifies the new block.

Participants reach a consensus to add a new block to the chain. This new block is added to their local copy of the blockchain. Thus, a new set of transactions are recorded and confirmed. The algorithm for consensus is called Proof of Work protocol, since it involves work a computational power to solve the puzzle and to claim the right to form the next block. Transaction zero, index zero of the confirmed block is created by the miner of the block. It has a special UTXO (Unspent Transaction Output) and does not have any input UTXO. It is called the coinbase transaction that generates a minor's fees for the block creation. This is how new coin is maintained in bitcoin. To summarize, the main operations in a blockchain are transaction validation and block creation with the consensus of the participants. There are many underlying implicit operations as well in the bitcoin blockchain.

2.2.4 Transactions

Transactions are the basic element of the Bitcoin Blockchain. Transactions are validated and broadcasted. Many transactions form a block and many blocks form a chain through a digital data link. Blocks go through a consensus process to select the next block that will be added to the chain. Chosen block is verified and added to the current chain Figure 2.3. Validation and consensus process are carried out by special peer nodes called miners. These are powerful computers executing software

defined by the blockchain protocol [16].

A fundamental concept of a bitcoin network is an Unspent Transaction Output, also known as UTXO. The set of all UTXOs in a bitcoin network collectively defined the state of the Bitcoin Blockchain. UTXO's are referenced as inputs in a transaction. UTXO's those are also outputs generated by a transaction. All that UTXO's is in a system, are stored by the participant nodes in a database. Transactions uses the amount specified by one or more UTXOs and transmits it to one or more newly created output UTXOs, according to the request finitiated by the sender. The structure of a given UTXO is very simple. It includes a unique identifier of the transaction that created this UTXO, an index or the position of the UTXO in the transaction output list, a value or the amount it is good for. And an optional script, the condition under which the output can be spent. The transaction itself includes a reference number of the current transaction, references to one no more input UTXOs, references to one or more output UTXOs newly generated by the current transaction, and the total input amount and output amount. Participants can validate the transaction contents. The UTXO's reference input exist in the network state. This is the only one of the many validation criteria. To summarize, transaction bring about transfer of value in the Bitcoin Blockchain. The concept of UTXO defines the inputs and outputs of such a transaction. Once a block is verified an algorithmic-ally agreed upon by the miners, it is added to the chain of blocks, namely the Blockchain.

2.2.5 Public-Key Cryptography

Two techniques are predominantly used for securing the chain and for efficient validation and verification. Hashing and asymmetric key encryption. These techniques depend on several complex proven algorithms [24].

Blockchains decentralized network participants are not necessarily known to each other. Credentials cannot be checked by the conventional means such as verifying who you are with your driver's license. Participants can join and leave the

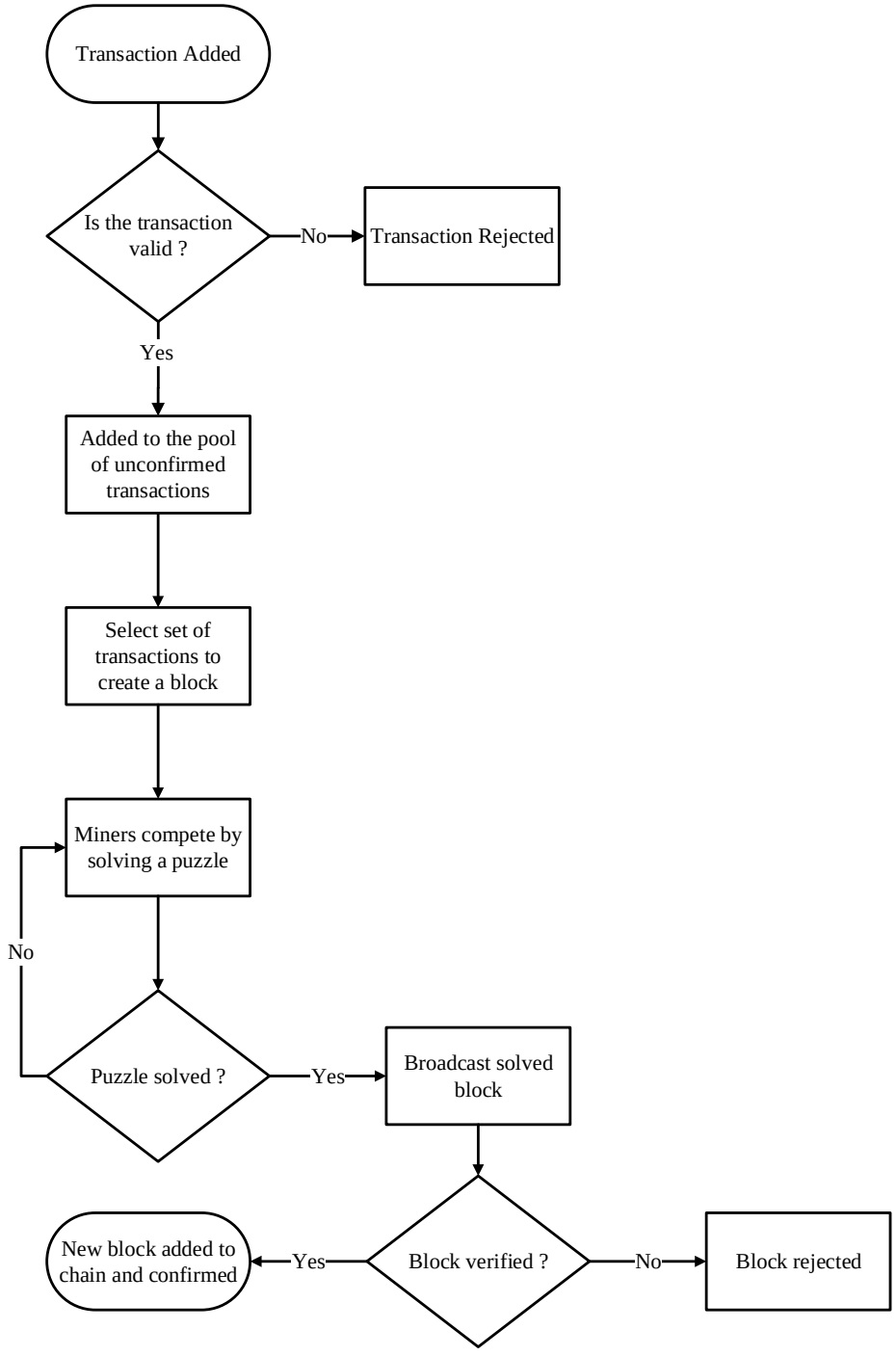

Figure 2.3. Block is added into blockchain after miners verify both transaction and block.

chain as they wish. They operate beyond the boundaries of trust. Given this context. Public-key cryptography is used to not only identify the peer participants but also to authorize, authenticate and to detect forged or faulty transactions. In simple symmetric

key encryption the same key is used for encryption and decryption, so it is called symmetric key. Example, Caesar encryption is the simplest one with alphabets of a message are shifted by a fixed number, and this number is called the Key. Consider "Meeting at noon" the key value of every letter is sifted by three to encrypt it, and receiver decrypts it, using the same three as the key. Shift the other way every character to view the original message. Three is the key in this trivial example. Since the same key is used for encryption and decryption, it is a symmetric key. Note that the key and the encryption and decryption functions are typically much more complex in a real application. In this case, it is easy to derive the secret key from the encrypted data . These issues are further exasperated in a block chain decentralized network where participants are unknown to each other. Public-key cryptography addresses these issues. Instead of a single secret key, it employs two different keys that take care of both the issues of symmetric key encryption [9]. Let, 'nN' be the private public-key pair for a participant in NUST Islamabad Pakistan. Let 'iI' be the pair of keys for the participant in USA. Public-key is published, private key is kept safe and locked. Typically using a passphrase and the pair works as follows; encrypting function holds two properties with a key pair. The public-key private key pair has the unique quality that even though a data is encrypted with the private key, it can be decrypted with the corresponding public-key and vice versa. If a participant in NUST wants to transact with the participant in USA. Instead of sending just a simple message, a participant in NUST will send a transaction data encrypted by NUST's private key, and then encrypted by USA's public key. USA will first decrypt the data using its own private key, then use NUST's public key to decrypt assigned transaction data. This ensures that only USA can decrypt and receive the data and that only NUST could have sent the data. A popular implementation of public key, private key is the Rivest Shamir Adleman (RSA) algorithm. Common application of RSA is the password-less user authentication, for example for accessing a virtual machine on Amazon cloud. Though RSA is very commonly used in many applications, blockchains need a more efficient

and stronger algorithm. Efficiency is a critical requirement since public key pair is frequently used in many different operations in block chain protocol. Elliptic Curve Cryptography, ECC family of algorithms is used in the bitcoin as well as an Ethereum block chain for generating the key pair.

2.2.6 Hashing

The private public key pair is a metaphorical passport to participating in transacting on the blockchain. Like it is necessary to not only learn use of credit card but also to secure it and protect it. The private key should be protected for the security of your assets on the blockchain.

A hash function or hashing transforms and maps an arbitrary length of input data value that could be a document, any data structure or a block, to a unique fixed length value Figure 2.4. Even a slight difference in the input data would produce a totally different hash output value [20].

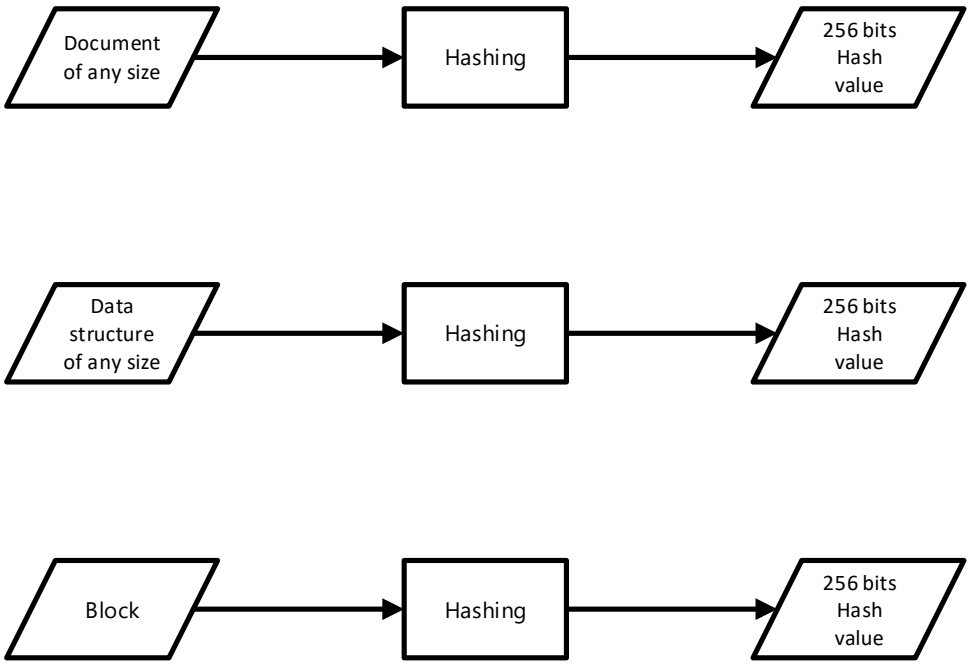

Figure 2.4. Hashing

There are two basic requirements of a hash function. The algorithm chosen for the hash function should be a one-way function and it should be collision free or exhibit extremely low probability of collision. The first requirement is to make certain that no one can derive the original items hashed from the hash value. The second requirement is to make sure that the hash value uniquely represents the original items hashed. There should be extremely low probability that two different datasets map onto the same hash value.

These requirements are achieved by choosing a strong algorithm such as secure hash, and by using appropriately large number of bits in the hash value. Most common hash size now is 256 bits and the common functions are SHA-3, SHA-256 and Keccak.

A 256-bit hash value space is indeed very large. 2 to the power of 256 possible combinations of values. That is approximately 10 to the power of 77. That is 10 followed by 77 zeros. Odds of a meteor strike is higher than generating two of the same hash values of 256 bits when applying this algorithm.

There are two different approaches for hashing based on how the constituent elements are organized. A simple hash and a Merkle tree hash.

In the simple hash approach, all the data items are linearly arranged and hashed. In a tree-structured approach, the data is at the leaf nodes of the tree, leaves are pairwise hashing to arrive at the same hash value as a simple hash. Simple hash is used when a fixed number of items to be hashed, such as the items in a block header, and verifying the composite block integrity and not the individual item integrity. The merkle tree hash is used when the number of items differ from block to block, for example, number of transactions, number of states, number of receipts, with the tree structure for computing the hash. Note that the state is a variable that may be modified by a smart contract execution, and the result of the execution may be returned in a receipt. Tree structure helps the efficiency of repeated operations, such as transaction modification and the state changes from one block to the next.

Summarizing, in blockchain, hashing functions are used for generating account addresses, digital signatures, transaction hash, state hash, receipt hash and block header hash. SHA-3, SHA-256, Keccak-256 are some of the algorithms commonly used by hash generation in blockchains.

2.2.7 Consensus Protocol

A secure chain is a single main chain with a consistent state. Every valid block added to this chain, adds to the trust level of the chain. The miners compete to add their block to the chain. There is a method or protocol for choosing the next block to the main chain called Proof of Work. Proof of Work uses hashing. [18].

Proof of Work is used in both bitcoin and ethereum Figure 2.5. In the process of mining, a hash of the block header elements that is a fixed value and a nonce that is a variable is computed. If hash value is less than 2 par 128 for bitcoin, and less than function of difficulty for Ethereum, the puzzle has been solved. If it has not been solved, repeat the process after changing the nonce value. If the puzzle has been solved, broadcast the winning block that will be verified by other miners. Non-winning miner nodes add the new block to the local copy of the chain and move on to working on the next block. The winner gets an incentive for creating the block. Proof of Work is a consensus protocol used by bitcoin block chain and by the current version of Ethereum. The protocol may be the same, the implementations in these two block chains are different. Many other approaches such as Proof of Stake, Proof of Elapsed Time have been proposed.

2.2.8 Robustness

Trust is not only about executing regular operations correctly but also about managing exception satisfactory. Robustness is the ability to satisfactorily manage exceptional situations. It's more important in a decentralized autonomous network such as a blockchain where there are no intermediaries minding the store [11]. Double

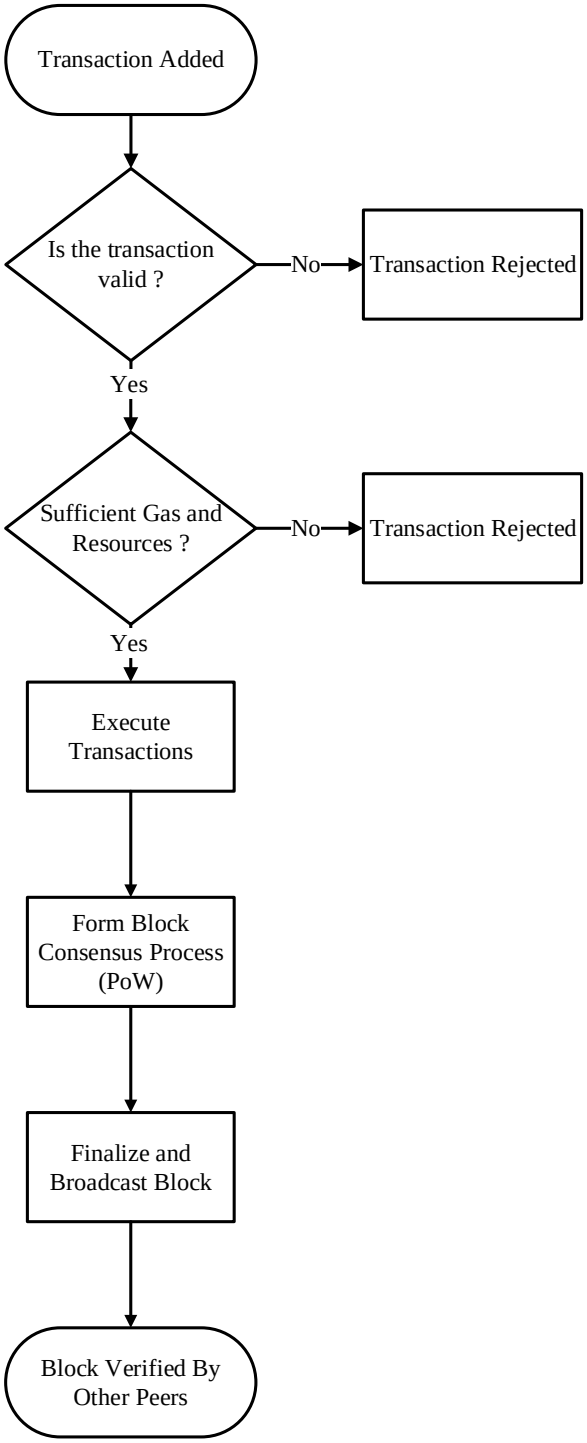

Figure 2.5. Proof of Work

spending is a situation in which there might be more than one miner who solves the consensus puzzle or there might be a case in which one transaction references as input the same digital asset. Handling such exception satisfactorily is critical for ensuring

the security of the blockchain. If there are two miners who have solved the consensus puzzle very close to each other. Bitcoin protocol allows this chain split or two chains for the next cycle Figure 2.6. One led by each of the competing blocks. The probability that the next block will happen at the same time in both these chains is extremely low. So, the winner of the next cycle for block creation consolidates one of the chains and that chain becomes the accepted chain. In this case, the newest block is added to the main chain. Now this chain is the longest and the valid main chain Figure 2.7. The transaction in the other blocks are returned to the unconfirmed pool. Summarizing with a very low probability, the main chain may split but if it does, the bitcoin protocol has methods to consolidate it to a single chain within a cycle. Ethereum handles more than one person by allowing Runner-Up blocks and allocating a small incentive for these Runner-Up blocks. This incentive model helps in keeping the chains secure. New blocks are added only to the mainchain and not to the Runner-Up chains. That are Runner-up blocks are maintained for six more blocks after they were added. If there is a blockchain in which there are two blocks one at the height, 4567, another one at a height, 4557. The one deeper inside the chain is more trustworthy than the one newly added. There's a possibility that digital currency and other consumables are single used digital assets, can be intentionally or inadvertently reused in transactions. This is like an airline double booking a seat on a flight. In this case, a gate crew solved this problem by using Ad hoc methods such as asking for volunteers to relinquish their seats for money, etc. In a decentralized network, like a blockchain, there is no intermediary. A policy and an automatic deterministic way is needed to handle this situation. For handling transaction and double spending in Bitcoin it is allowed to the first transaction that reference the digital asset and reject the rest of the transaction that reference the same digital asset. In Ethereum, a combination of a call number and a global nons is used to address the doublet spending issue. Every time a transaction is initiated by an account, a global nonce is included in the transaction. After that, the nonce is incremented. Time stamp on the nonce in the transaction should be unique

and verified to prevent any double use of digital asset. Summarizing, well-defined processes for handling exception improve trust in the blockchain.

2.2.9 Forks

Trust in Forks, a fork in the packed. Fork, hard fork and soft fork, are most common phrases uttered in the context of a blockchain. Forks are just normal processes in an evolutionary path of the nascent technology enabling a blockchain [3]. If robustness and trust is about managing exceptional situations, hard forks and soft forks are indeed at the front and center. In previous section change split was discussed that is a minor perturbation in the chain. Such situation is handled as a naturally expected occurrence within the block chain. On the other hand, occasionally, a minor process adjustment must be carried out typically by bootstrapping a new software to the already running processes. This is soft fork. For example, the script concept in Bitcoin was introduced using this method. It can be considered as a software patch or a bug fix to address an issue. Hard fork implies a major change in the protocol. For example, the recent change from Ethereum Homestead to Metropolis Byzantium version was a planned hard fork and important note after a hard fork the emerging two chains are incompatible. There was an unplanned hard fork in Ethereum protocol, Ethereum Core and Ethereum Classic split, that was enacted to address a critical software issue in a decentralized autonomous organization (DAO).

Summarizing, soft fork and hard fork in the blockchain world are similar to the release of software patches, and new versions of operating systems respectively. Forks are mechanisms that add to the robustness of the blockchain framework. Well-managed forks help build credibility in the blockchain by providing approaches to manage unexpected faults and planned improvements [2].

[2]https://www.trustnodes.com/2017/12/02/blockchain-forks-shown-hayek-right-wrong

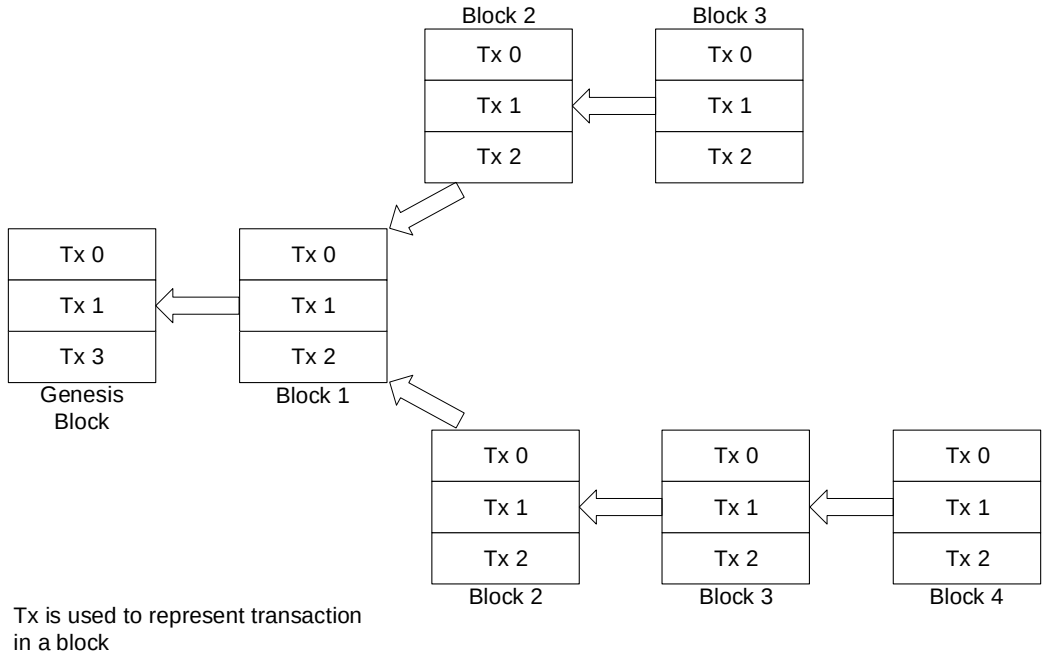

Figure 2.6. Chain Split

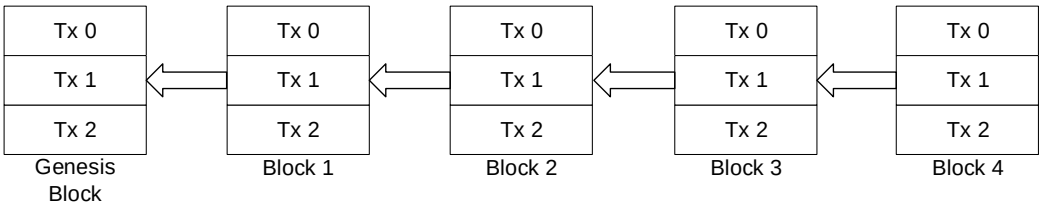

Figure 2.7. Chain Merge

2.2.10 Security

The main components of the block are the header, the transactions, including the transaction hash or the transaction root, and the state radio booth, the state hash, or the state root. Integrity of the block is managed by assuring that the block header contents along with the transactions are not modified, state transitions are efficiently computed, hashed, and verified [10]. The block chain is supposed to be an immutable record. In Ethereum, the block hash is the block of all the elements in the block header which includes the transaction root and state root hashes. It is computed by applying a variant of SHA-3 algorithm called Keccak and all the items of the block header.

A typical block has about 2,000 transactions in bitcoin and about 100 transaction

Ethereum. An efficient way is needed to detect tampering and validate the transaction efficiently. Hashes of transaction in a block are processed in a tree structure called Merkle tree hash. Merkle tree hash is also used for computing the state root hash, since only the hash of the chained states from block to block must be re-computed. It is also used for receipt hash root. If any transaction is to be verified, only one path to the tree must be checked and it is not needed to go through the entire set of transactions. Smart contract execution in Ethereum results in state transitions. Every state change requires state root hash re-computation. Instead of computing hash for the entire set of states, only the affected path in the Merkle tree needs to be re-computed. When the state 19 is changed to 20, that results in the path including 31, 41, and the state root hash 64 to be re-computed. Only that path is re-computed, not the entire tree. Now, let's move on to block hash computation.

Block hash in Ethereum is computed by first computing the state root hash, transaction root hash and then receipt root hash, shown at the bottom of the block header. Hash is calculated from these roots and all the other items in the header with the variable nodes to solve the proof of work puzzle. Block hash serves two important purposes which are verification of the integrity of the block and the transactions, formation of the chain link by embedding the previous block hash in the current block header. If any participant node tampers with the block, its hash value changes resulting in the mismatch of the hash values and rendering the local chain of the node in an invalid state. Any future blocks initiated by the node would be rejected by other miners due to hash mismatch. This enforces the immutability of the chain.

Summarizing, a combination of hashing and encryption are used for securing the various elements of the block chain. Private public key pair and hashing are important foundational concepts in decentralized networks that operate beyond trust boundaries.

2.3 Types of Blockchain

Bitcoin blockchain is open-source and the entire code is available on the GitHub. During the initial years beginning roughly in 2009, this open-source code was extended to release different cryptocurrencies. About 300 plus cryptocurrencies were introduced. Bitcoin supports an optional and special feature called scripts for conditional transfer of values [14]. Ethereum Blockchain extended the scripting feature into a full-blown code execution framework called smart contract. A smart contract provides the very powerful capability of code execution for embedding business logic on the blockchain. Based on such capabilities, three major types of blockchains emerge from Bitcoin foundation. Type one deals with the coins in cryptocurrency currency chain. Example, Bitcoin. Type two supports cryptocurrency and a business logic layer supported by code execution. Example, Ethereum. Type three involves no currency but supports software execution for business logic. Example, The Linux Foundation's Hyperledger. With the addition of code execution, comes the serious consideration about public access to the blockchain hence, the classification of public, private, and permissioned blockchains based on access limits [21].

2.3.1 Cryptocurrency Blockchain

Continuous operation of Bitcoin blockchain has been observed since its inception. All supported by its public participants. Thus, Bitcoin is a fantastic example of a public blockchain class. Anybody can join and leave as they wish. Transaction blocks and the blockchain are publicly observable even though participants are anonymous. It is open-source. It is also possible to create new coin digital currency by modifying the Bitcoin code. Wallet applications provide the basic interface to transfer value through the Bitcoin blockchain [14].

2.3.2 Cryptocurrency Based Smart Contract Blockchain

Bitcoin blockchain is the mother of all blockchains. It was intended for peer to peer transfer of value and it does that well. Around 2013, a framework for code execution was introduced by Ethereum Founders. The centerpiece and thrust of this Ethereum blockchain is a smart contract. Consider figure 2.8 and figure 2.9, comparing Bitcoin and Ethereum blockchain. There is the Bitcoin blockchain and a wallet application for initiating transactions. On the other hand is Ethereum that took a significant step towards transforming the blockchain into a computational framework that opened a whole world of opportunities in the decentralized realm. Ethereum supports smart contracts and of which will machine on which smart contracts execute. Smart contracts in turn enable decentralized application that accomplish more than a transfer of value. Efficient automation of decentralized application such as supply chain [5].

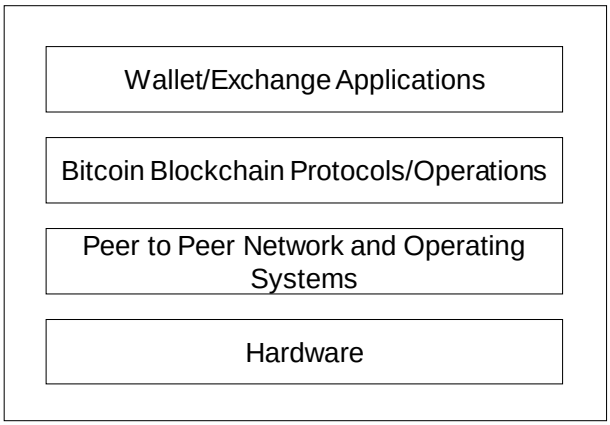

Figure 2.8. Bitcoin Stack

2.3.3 Non-Cryptocurrency Based Smart Contract Blockchain

Non-cryptocurrency based smart contract blockchain uses permissioned blockchain. Permissioned blockchain also called consortium blockchain. It is meant for a consortium of collaborating parties to transact on a blockchain for ease of governance, provenance, and accountability for example, a consortium of all automobile companies or healthcare

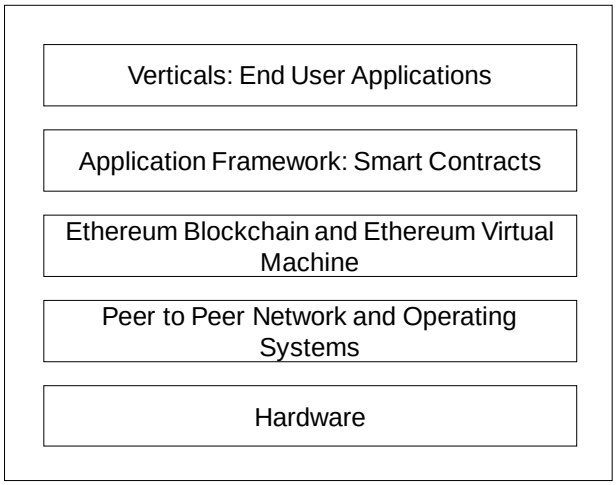

Figure 2.9. Ethereum Stack

organizations [5]. Permissioned blockchain has the benefits of a public blockchain with allowing only users with permission to collaborate and transact. It involves no currency but supports software execution for business logic. Example, The Linux Foundation's Hyperledger Figure 2.10. The architecture of this blockchain is different as it doesn't rely on cryptocurrencies.

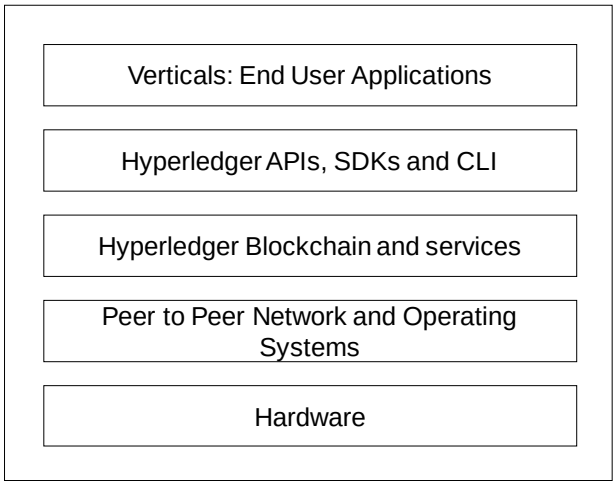

Figure 2.10. Hyperledger Stack

In summary, significant innovations such as smart contracts have opened broader applications for blockchain technology. Private and permissioned blockchain allow for controlled access to the blockchain enabling many diverse business models.

2.4 Development Framework

There are different types of ledgers which are used in Blockchain technology, which includes public permission-less and private ledgers. Ledgers in case of Bitcoin are public permission-less shared ledger which is ideal for cryptocurrencies transactions but not for private business transactions.

Linux foundation introduced a project named "Hyperledger", which aims to identify and articulate goals for a cross industry standard for Blockchain [4]. Hyperledger has divided its development area into two parts, one focuses on developing tools and other one focuses on developing framework. There are following frameworks in Hyperledger:

1. Iroha
2. Sawtooth
3. Indy
4. Fabric

We have chosen Hyperledger Fabric as our development framework but lets take a brief look at the other sister projects of Fabric to understand why we are using Fabric and not any other framework [4].

2.4.1 Iroha

Iroha is a Hyperledger framework that focuses on financial and identity management on a blockchain network. It uses algorithm known as Sumeragi. The limitation of this algorithm is that with the increase in nodes n the network, the more time it would take to reach consensus.

2.4.2 Sawtooth

Hyperledger Sawtooth uses Proof of Elapsed Time (PoET) algorithm and in the algorithm the participant whom finishes his waiting time first, becomes the leader of the new block. The limitation of this algorithm is that finality can be delayed due to forks that must be resolved first and therefore it becomes not a suitable framework for blockchain solution development for our use case.

2.4.3 Indy

Indy is another sister project of fabric and it uses Redundant Byzantine Fault Tolerance algorithm and in this blockchain framework, all the instances on the network order the request but only those which are ordered by master instance are executed. The limitation with this algorithm is same as the one with Hyperledger Iroha and that was, the more nodes there are on the network, the more time it would take to reach the consensus.

2.4.4 Fabric

Hyperledger Fabric is framework built over Hyperledger project to offer shared, permissioned and distributed ledger with support for smart contracts. Smart contracts are business terms embedded in the ledger and are executed with each transaction. It introduces the concept of special nodes called validating leader nodes and validating nodes to offer complete security, transparency and accessibility to records. For data security, it records and encrypts transaction on public ledger so that only those nodes which were involved in transaction can make sense of data on ledger.

2.5 Hyperledger Fabric

A platform which is open source. It uses distributed ledger technology which is permissioned. This platform is designed for contexts of enterprises. Architecture

in this platform is not only modular but also highly configurable. The modularity and configurability enable this platform to be used for many of the use cases including insurance, supply chain etc [1]. The typical flow of execution in this platform comprises of execution, ordering, validation and finally updating the state of an asset Figure 2.11.

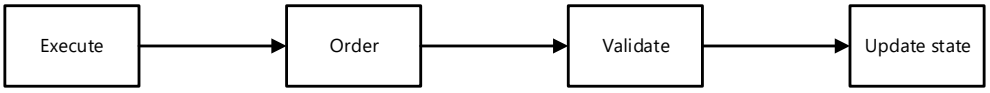

Figure 2.11. Flow of execution in Hyperledger Fabric

Typical blockchain solution uses domain specific languages but fabric is first of its kind which supports many programming languages like java, go etc. This allows developers to quickly familiarize themselves with programming architecture of this platform. It omits the need for learning a new language.

The Hyperledger Fabric platform is highly customizable and can be used to fit use cases. It also supports protocols consensus which is pluggable, which is also a great factor in its modularity and customizability.

Pluggable consensus allows organization to use specific consensus according to their needs. For example, crash fault tolerant (CFT) might be sufficient for single organization and it might prove a drag to use complex consensus like fully byzantine fault tolerant.

Fabric is a platform which provides bases for a blockchain solution that doesn't depends on the cryptocurrency or mining. This helps in reducing significant risks involving cryptocurrencies.

2.5.1 Structure of Block

Block in hyperledger Fabric comprised of three parts Figure 2.13 that are [23]

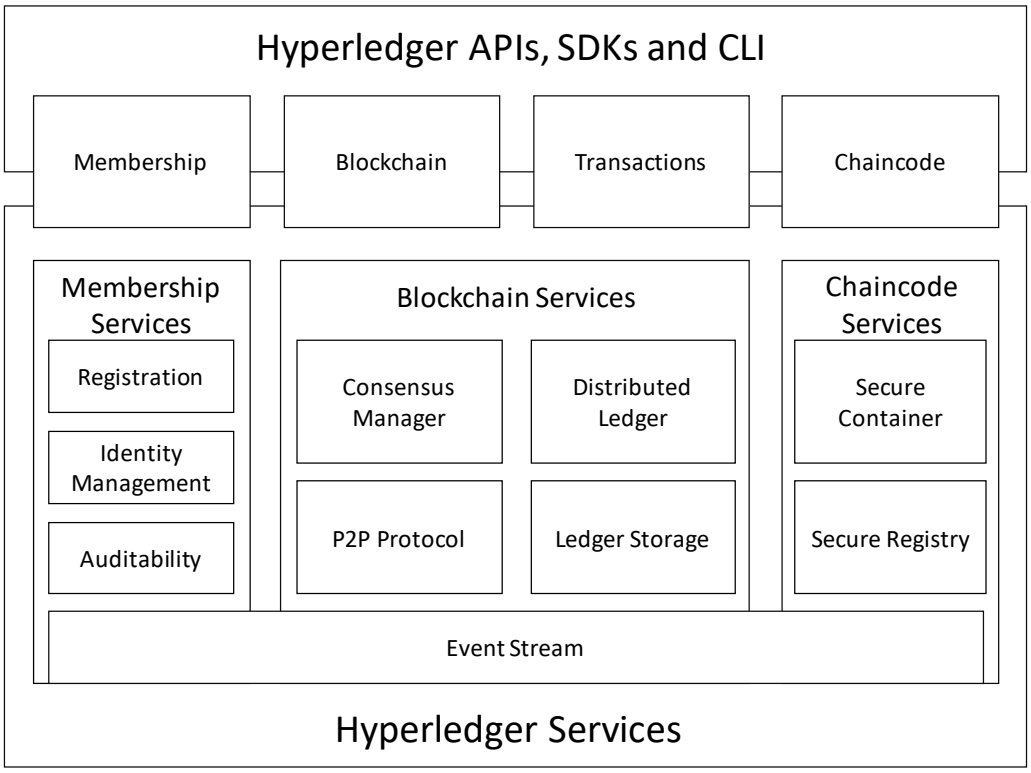

Figure 2.12. Hyperledger Fabric Architecture

(A) Block Header

Block header is the part of a block in Hyperledger Fabric and it consists of a number that is a unique identity of that specific block, a hash of the previous block and finally a hash of the data which this block carries in it Figure 2.14.

(B) Block Data

Transaction which can be carried out, are stored in block data and it also contains all the actions that can be carried out for a transaction on a blockchain network Figure 2.15. Transactions are further divided into two parts, payload and signature. Payload contains details of channel and signatures on it. And signature part holds all the information about signatures that are used for executing of a transactions and entities involved in creation of this block.

(C) Block Metadata

Metadata is a kind of data that gives information about the other data and here it provides the information about signatures, last configuration block, all the transactions and finally the metadata of ordering service Figure 2.16.

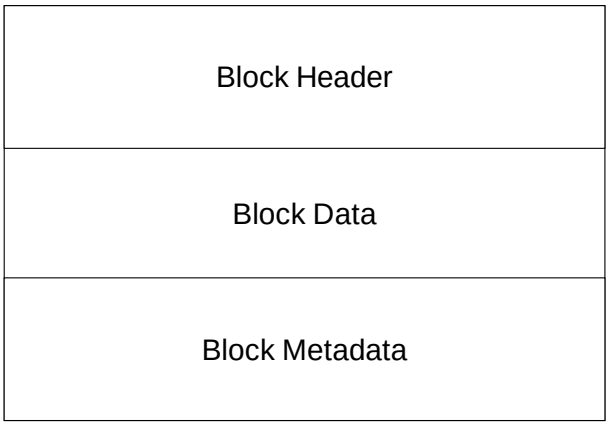

Figure 2.13. Block

Figure 2.14. Block Header

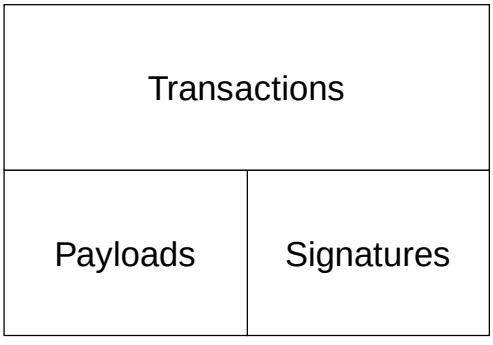

Figure 2.15. Block Data

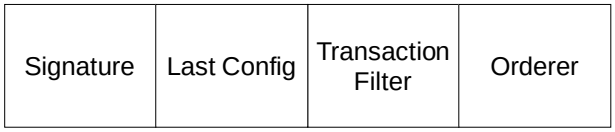

Figure 2.16. Block Metadata

2.5.2 Functionalities

There are number of functionalities provided in Hyperledger Fabric. In this section we have provided a brief introduction about those functionalities [3].

(A) Identity Management

Membership identity service is provided in Hyperledger Fabric which enables permissioned networks and this service is used for managing IDs of users and it is used for authentication of all the participant on the network. We can further manage users by defining access controls for each participant depending on their role in a network. For example, we can permit a user to invoke a chaincode application but prohibit that same user from deploying a new chaincode.

(B) Privacy and confidentiality

Permissioned network allows coexistence of different competing business groups, provided with private and confidential transactions on the same network. We can divide fabric network into subset of networks using private channels. They behave like a messaging path which is restricted to a specific member of a network. Thus, providing confidentiality and privacy along with it.

(C) Efficient Processing

Hyperledger Fabric provides a functionality of assigning network roles according to the type of node. In general, action on transactions are separated from each other

[3]https://hyperledger-fabric.readthedocs.io/en/release-1.1/functionalities.html

to provide parallelism in the network. Transactions are executed prior to ordering, this enables each peer in the network to process number of transactions in any given time. This doesn't only dramatically increase transactions execution but also helps with accelerating the delivery to the ordering service of the transactions.

(D) Chaincode Functionality

Logic is encoded in chaincode applications and they are invoked depending on the type of transaction that is executed on the channel. We can use chaincode to define the parameters that can be used to change ownership of the asset. Other than that, there is also system chaincode which defined the parameters that are used for operating an entire channel. To define the rules of the channel, lifcycle and configuration chaincodes are used and we can also use endorsement and validation chaincode which defines all the requirement which must be satisfied for the endorsement and validation of transaction.

2.5.3 Architecture and Hierarchy

There are many components in Hyperledger Fabric Figure 2.17 that work together to make up the whole system [2]. Those components and their hierarchy of those components is discussed below:

(A) Domain

The domain is the namespace for the project. It is usually the name of project.

(B) Orderers

The orderers provides the functionality of creating a block from the new transaction and then forwarding and committing it to all the adjacent peers in the blockchain network. There can be multiple orderers. Orderer is informed about a

transaction which is proposed and committed by a peer. They are not organization dependent.

(C) Organizations

Every organization has its own peers and certificate authority. These organizations act as a container of these network elements. They are used so that blockchain network can be easily separated physically.

(D) Certificate Authorities

Like peers, every organization has its own certificate authority and is tied to it. It is used to verify the ownership in the network and for creating users with their respective certificates.

(E) Peers

These nodes are responsible for committing transactions and clients are connected to these nodes. A CouchDB database is maintained in each peer where it has its own copy of ledger. Number of peers can change from organization to organization depending on the requirement and model of network.

2.6 Hyperledger Composer

Hyperledger fabric serves as a platform for development of business models which do not use cryptocurrency within their transactions. It is not only pluggable but also highly modular and customizable. Developing and deploying a blockchain solution might take months in Hyperledger Fabric. Hyperledger composer makes it not only easier but much faster also to develop and deploy blockchain solutions. It is a framework that makes developing blockchain application easier. It also makes it much easier to integrate with existing business systems [6].

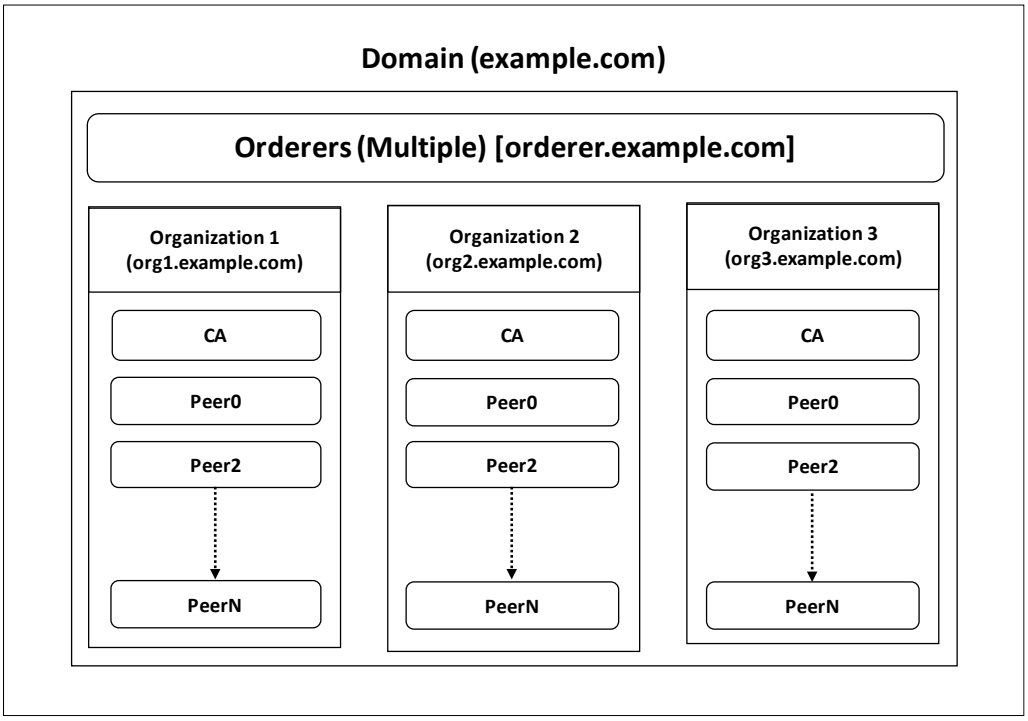

Figure 2.17. Components of Hyperledger Fabric Network

Pluggable blockchain consensus is supported by hyperledger fabric blockchain, which makes sure that every transaction is validated according to the policy that is according to the participants in the network. Access points are provided which can be used by applications to consume data from business network.

We can use Hyperledger Composer for the quick modeling of our business network Figure 2.18. We can map assets and transactions relating to our business network into it. Assets can be tangible or intangible. Transactions are defined as an interaction with the assets. We can also define which participants can interact with them. Unique identity is associated with every participant.

2.6.1 Key Concepts

It is necessary for a developer to understand the basic and important key concepts of Hyperledger Composer as they hold utmost importance in developing

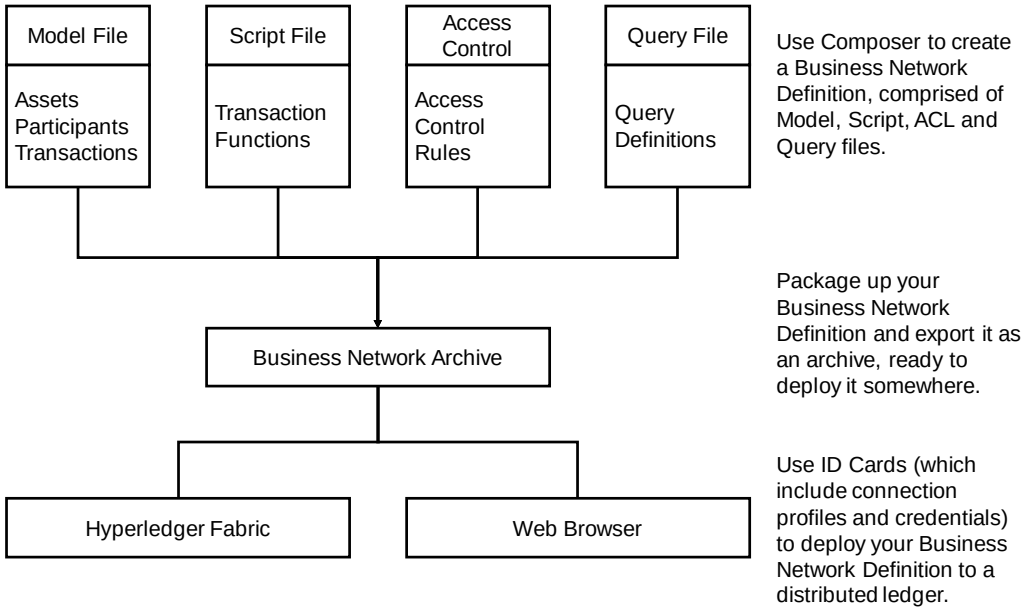

Figure 2.18. Hyperledger Composer

a better blockchain solution and to implement it successfully [4].

(A) Blockchain State Storage

Blockchain ledger is used to store all transactions which are submitted through business network. Along with it, there is also a database which is used for storing the current state of assets and participants called blockchain state. Blockchain state and ledger is distributed uniformly across all peers and blockchain ensures that they are consistent using a algorithm used for consensus.

(B) Connection Profiles

A JSON document which is a part of business network card is used by Hyperledger Composer to define the system to which it connects to. They are referred and are used to create cards for business network, which are used to connect to that system.

[4]https://hyperledger.github.io/composer/latest/introduction/key-concepts

(C) Assets

These are tangible or intangible goods or services which are stored in registries. Asset are used to represent anything in the business network. Each asset has a unique identifier and they contain properties that defined at the time of business modeling.

(D) Participants

Members in a network whom may own assets and are able to submit transactions. Just like modeling and defining assets, participants are also modeled. They also have a unique identifier and related properties.

(E) Identities

Identity is comprised of private key and a digital certificate. For submitting transactions, identities must be mapped. It is mapped to a participant. Typically, a single identity is stored in a business network card and it allows that user or a member of that business network to commit transactions on a business network.

(F) Business Network cards

Business network cards consists of an identity, metadata and a connection profile. It is used to simplify the process of connecting to a business network.

(G) Transactions

Participants on a business network interact with assets through transactions.

(H) Queries

A participant can extract data in blockchain world-state using queries. They are defined in the process of business modeling. Hyperledger Composer provides us with an API, which is used to send queries.

(I) Events

Events are used to give indication to external systems that something has happened on blockchain business network. They are omitted when a transaction process functions execute. Applications can be codded to subscribe to these events emitted by transaction function.

(J) Access Control

A set of access control is defined in a business network which defines the operations that are allowed by the participants on a business network under certain conditions.

(K) Historian registry

It is a registry which records all the successful transactions. It also records which participant submitted those transactions.

(L) Hyperledger Composer REST Server

Hyperledger Composer provides us a way to interact with the deployed business network using REST API. This contains all the operations or transactions that could be carried out on assets or participant etc. in a blockchain network.

(M) Business Network Archive

A typical blockchain solution in Hyperledger Composer comprises four types of files or components that make up a whole solution. Each file has its own unique characteristics and properties Figure 2.19.

1. Model File (.cto)

   We use modeling language to model our business models. Modeling language

has the following elements.

- Namespace

  It should be only one in a single model file. It is used to uniquely identify each model.

- Resource Definitions

  It is a set of assets, transactions, participants, concepts, and events.

- Import

  It is optional declarations, which are used to import foreigner resources.

2. Script File

   Script file or logic.js file contains functions for transaction processor. These functions are responsible for implementing the transactions that are defined in the previous file i.e. Model file.

   We use Business-Network-Connection Application Programming Interface (API) to submit transactions, which in return invoke these functions. These functions are coded using JavaScript Language.

3. Access Control (.acl)

   To define or to permit different operations depending on the rule of the user in the network, we use Access Control Language (ACL).

   We define rules so that we can only allow a certain user or users to execute Create, Read, Update or Delete (CRUD) operations on the assets or elements defined in the business model.

4. Query File (.qry)

   Queries are an important part or element of a business model as it returns the required result by filtering hundred or maybe thousands of records. The bespoke query language is used in the hyperledger composer to code queries for a business model and all these queries are stored in this file.

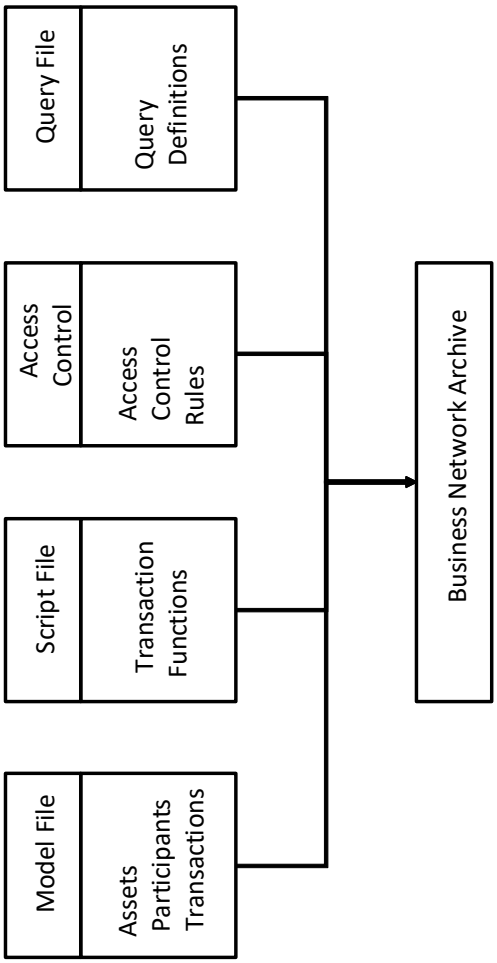

Figure 2.19. Business Network Archive

Chapter 3

PASSPORT VERIFICATION

Pakistan and many other countries have started the use of Machine Readable Passports (MRP). The reason behind moving from traditional passports to MRP was to enhance the security of passports. In addition, it also speed up the process of passport verification. These methods are standardized and implemented by the International Civil Aviation Organization (ICAO). They made sure that passports are not only machine readable but also human friendly as they must be read by border control officials.

3.1 Existing Security Mechanisms

Prevention of data skimming and sniffing became the priority to put stop to data leakage as it resulted in identity theft causing numerous crimes.

Using Personal Identification Number (PIN) was not ideal as not everyone would remember their PIN and it would make the whole process of verification much more slower and may cause many problems [8], [7].

3.1.1 Basic Access Control

To make process much secure and faster, Basic Access Control (BAC) was implemented. It stores a pair of keys that are cryptographic keys, in the passport chip. Passport number, birth date and issue date are accessed by reading Machine Readable Zone (MRZ) with Optical Character Recognition (OCR). In the process of scanning the passport it would engage in a protocol that comprises of challenge response control. During this protocol upon successful authentication, contents are released and if it fails, the holder i.e. passport holder is considered unauthorized.

Problem with this security implementation was the limited entropy of the secret keys. ICAO acknowledged that in few circumstances it might be possible for someone to guess those bits, as they are comprised of data like birthdate or expiration date etc. Another problem associated with security measure in place was the use of single key for its lifetime that would mean that it would not revoke the readers access for the lifetime which may result in compromise of sensitive data. It was a security risk that just could not be ignored.

Security measure in place were not sufficient as the security issues were not solved as efficient as they thought it would.

3.1.2 Extended Access Control

Extended Access Control (EAC) was implemented to overcome the shortcomings in the previous passport security implementations. The whole process of verification starts in this mechanism with verifying the certificates on the gates by the chip inside of passports. Those certificates inside of gates (or e-gates) were short lived and on verification by chip on passport the gates could read the data. But this whole process required a larger sophisticated infrastructure. Infrastructure would be heavily depended on the supportability of Private Key Infrastructure (PKI) in every country and then every country would have to sign their own certificates. Furthermore, those certificates should be stored in a centralized database or a repository from where all the gates would retrieve certificates. Such large and complex infrastructure would expose that much of vulnerabilities and security concerns in the system.

EACv1 was the first version of EAC. And it was tested and found much more likely to be attacked and accessed despite its security measures in place. To overcome those vulnerabilities, second version of EAC was introduced codenamed as EACv2. It could prevent attacks that were induced on previous version of EAC but there were few compatibility issues regarding implementation of this solution because not all gates supported EAC, there were still large numbers of gates that were still using

BAC. Another issue with this version was the possibility of expiration and leakage of short life certificates. Expired certificates could still be used to retrieve the data from the chip.

3.1.3 Supplemental Access Control

Despite of all the above implementations of security measures, it was still possible to access and retrieve data and use it for identity theft and other crimes.

Password Authenticated Key Agreement Protocol (PACE v2) was implemented to create Supplemental Access Control (SAC). It used Diffe-Hellman and mutual authentication. It works in such a way that, it uses a small password which generates a secret. But unlike in BAC, it would be of high entropy. PIN or Card Authentication Number (CAN) are used as the password and it is printed on the passport. There are two implementations of SAC. General Mapping and Integrated Mapping. Map2Point algorithm is used in the latter one which would be a hub of vulnerabilities if it becomes widely adopted.

Despite of much improvements in ICAO protocols, they still have doubtful security. There are security concerns regarding brute-forcing.

3.2 Verification on Blockchain

The problem with current system in place is that despite of having countermeasures in place, the system fails to detect every fake passport. Few might pass through the verification process. That makes these security measures and mechanism insufficient. Current systems are still using centralized databases which makes it a single point of failure thus making it vulnerable to hacking attacks.

Blockchain addresses all the issues highlighted above. Furthermore, blockchain provides immutability, provenance, finality and transparency.

In this research, the aim is to build blockchain application, Cryptopassports, and model business network logic for the verification of passports. In this model, a

passport is considered an asset as our whole business model revolves around it. And to extend the verification of passports during the process of visa creation, a visa is considered as a transaction for a passport Figure 3.1.

This model can be implemented and extended to the needs of the user or stakeholders. Three types of users or peer nodes are modeled using ACL Figure 3.2.

1. Network Administrator, who is responsible for managing all the nodes on the network, deploying or upgrading a business network and creating users.

2. Passport Office Personnel, a user who is responsible for creating or adding and updating a passport record.

3. Visa office Personnel, this user is using the record created by the Passport Office Personnel to verify the passport before creating a visa, different checks are created in passport verification that would be discussed in the model development part.

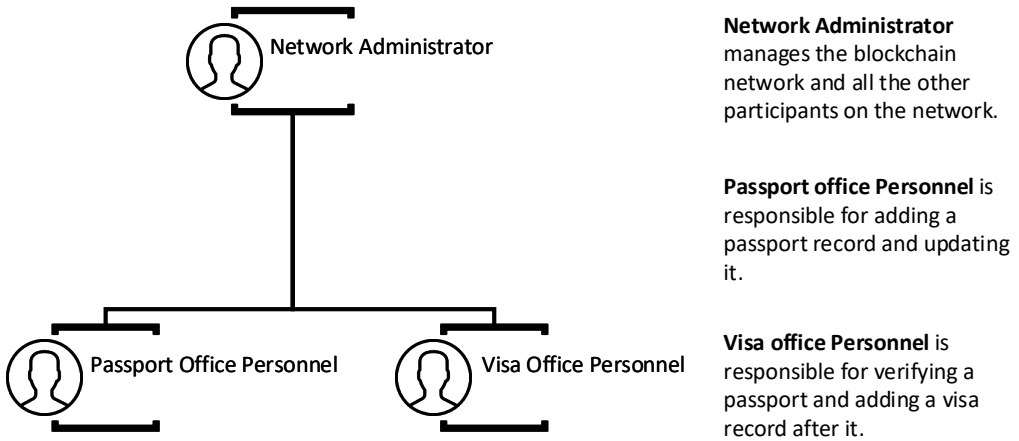

Figure 3.1. Network Model

3.3 Requirements

This blockchain solution is modeled on the basic requirements on the stakeholders and model can be evolved with iterations to extend it to the requirements of the

stakeholders.

1. Storing records of passport in an immutable database. It must be immune to hacking attacks.

2. Verification of passport must be confidential and should only be accessible by stakeholders or verified users on the network.

3. Before visa creation existence of passport should be verified in the record.

4. Business network should be flexible and manageable in such a way that new users can be added and access of existing users could be revoked if required.

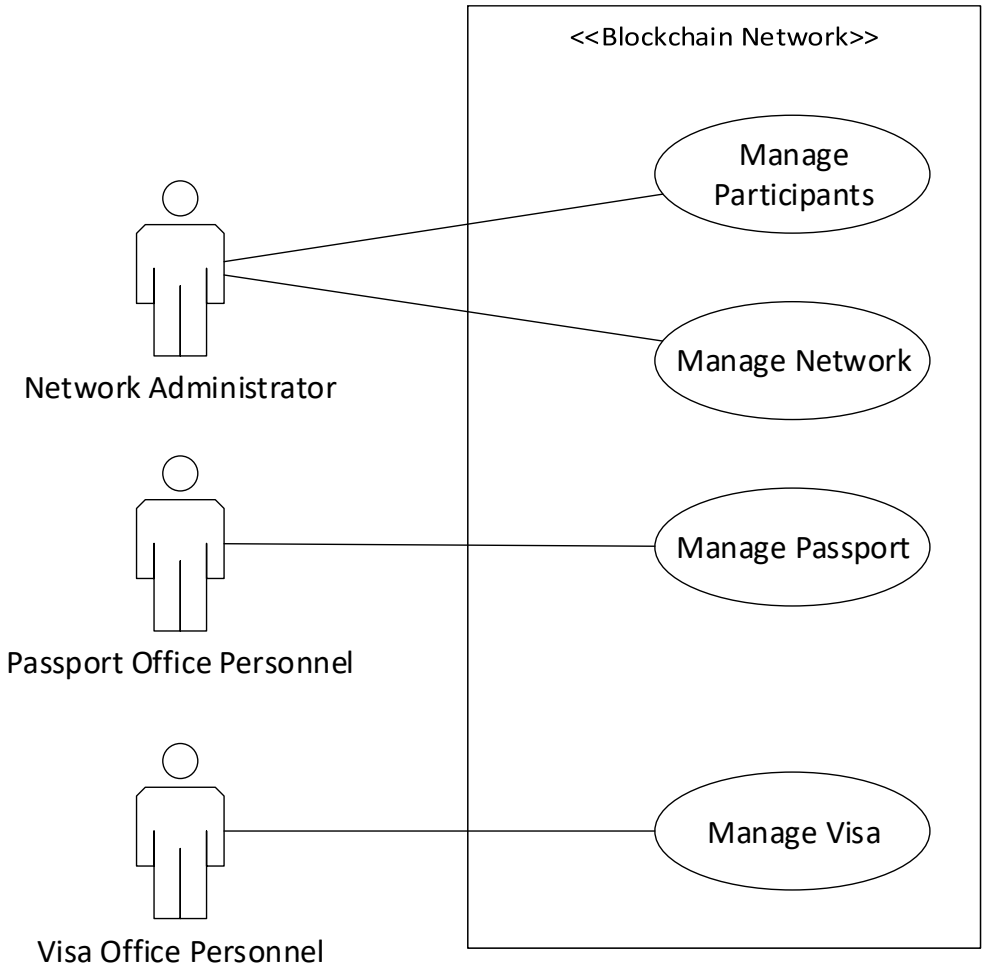

Figure 3.2. Use Case Of Network

There are three actors on a blockchain network that are Network Administrator, Passport Office Personnel and Visa Office Personnel.

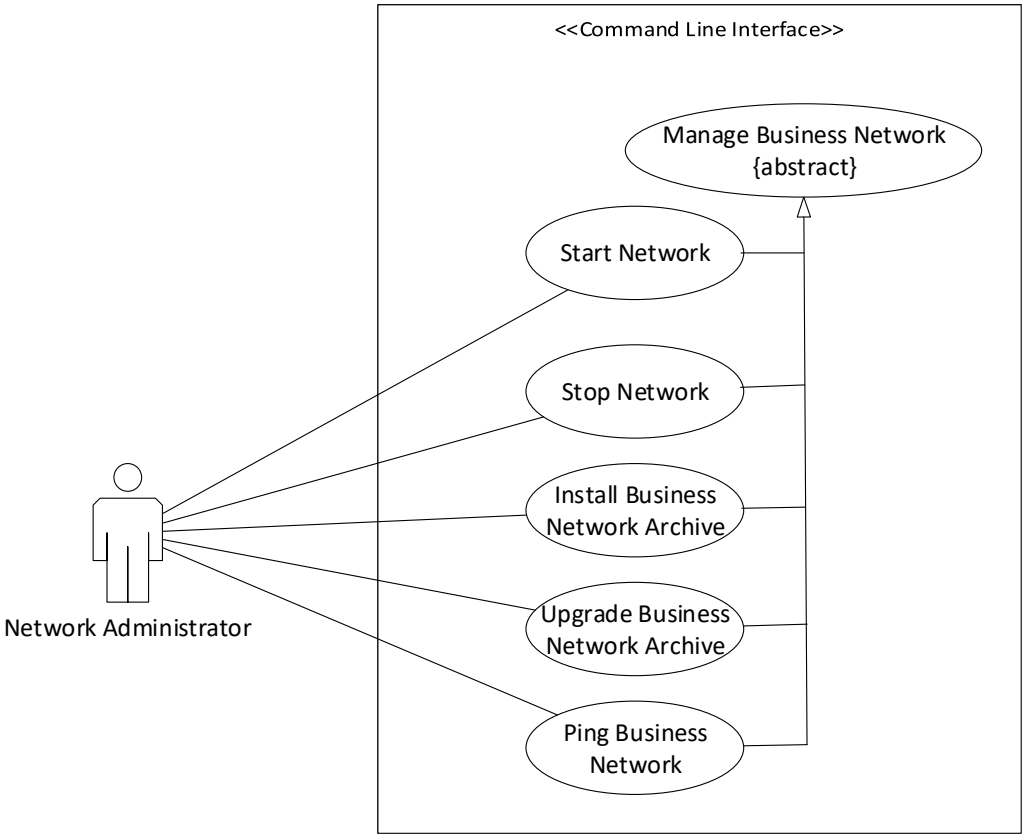

Figure 3.3. Use Case Of Managing Network

Managing network Figure 3.3 is an abstract use case which is specialized by other use cases. The idea is that a network administrator could start or stop a network, install business archive over a deployed network, upgrade existing network and it is possible to test the network connection by pinging the network.

User or participants are managed by a network administrator Figure 3.4. A network administrator can perform CRUD (Create, Read, Update, Delete) operations on participants. Other operations included are binding of participant with some identity and revoking access of current user if required in some scenario e.g. participant leaving the network etc.

Passport office personnel is responsible for creating, searching and updating

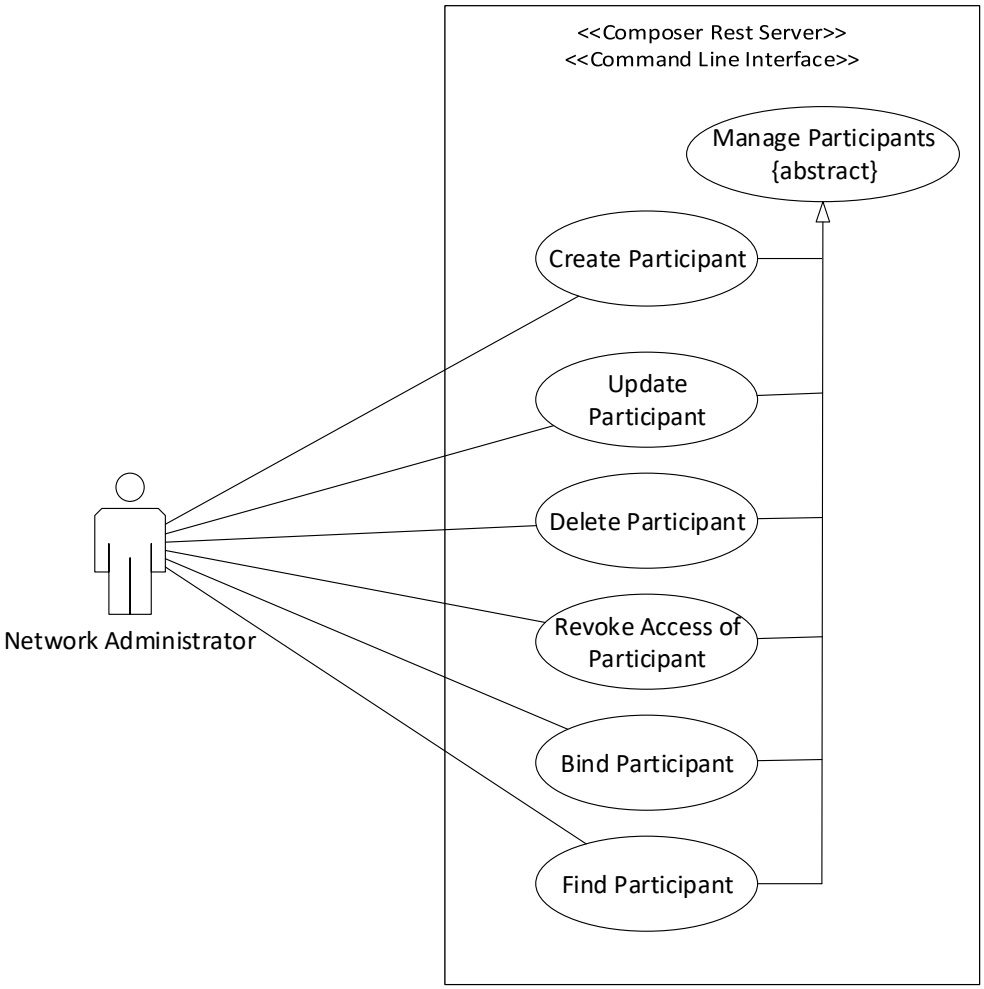

Figure 3.4. Use Case Of Managing Participants

passports Figure 3.5. The participant is not allowed to delete the passport to add a layer of security from accidental deletion.

Visa office personnel can create and search a specific visa Figure 3.6. For visa creation it is necessary to check if the passport that is being used for visa acquirement, does exists or not and if it does exists then it is checked whether it is permitted and not expired.

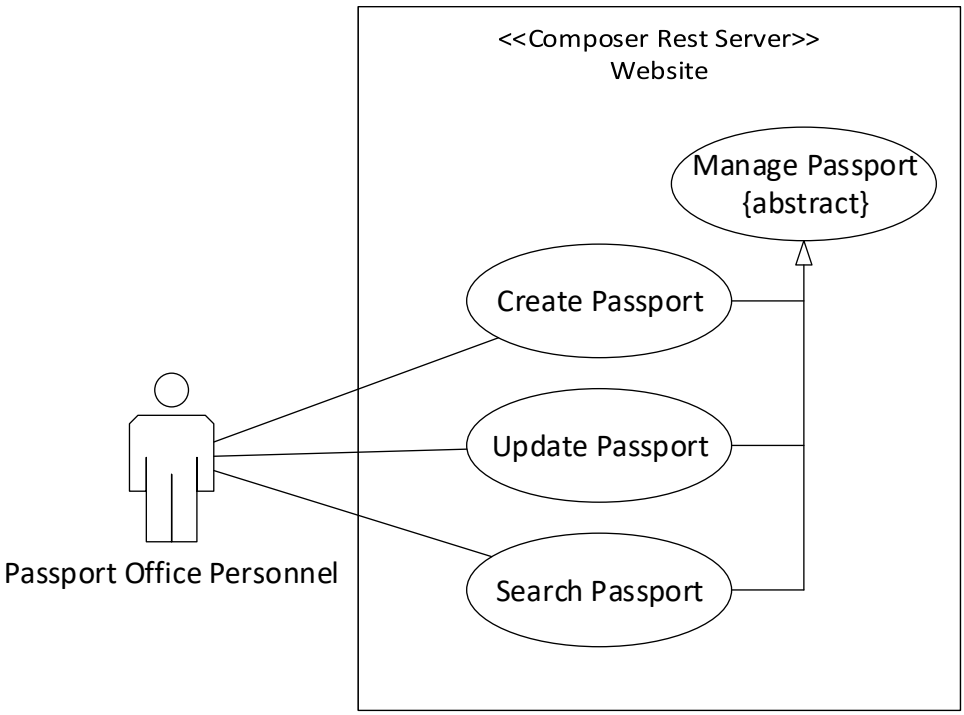

Figure 3.5. Use Case Of Managing Passports

3.4 Modeling of Requirements

Immutable record is provided by the blockchain called ledger. This section will focus on depicting the use cases about creating or updating a passport record and creating a visa.

Unified Modeling Language (UML) is used for modeling of user requirements because it could be much easier to dynamically view the interactions between the number of objects that are arranged in sequence. For that purpose, sequence diagram of UML is used.

Passport office personnel sends a command to Hyperledger of creating a passport record using web application. This invokes the function of creating passport in the smart contract. Furthermore, before adding the passport it validates the data, id and role of the user. On successful validation, it creates the record on a blockchain Figure 3.7.

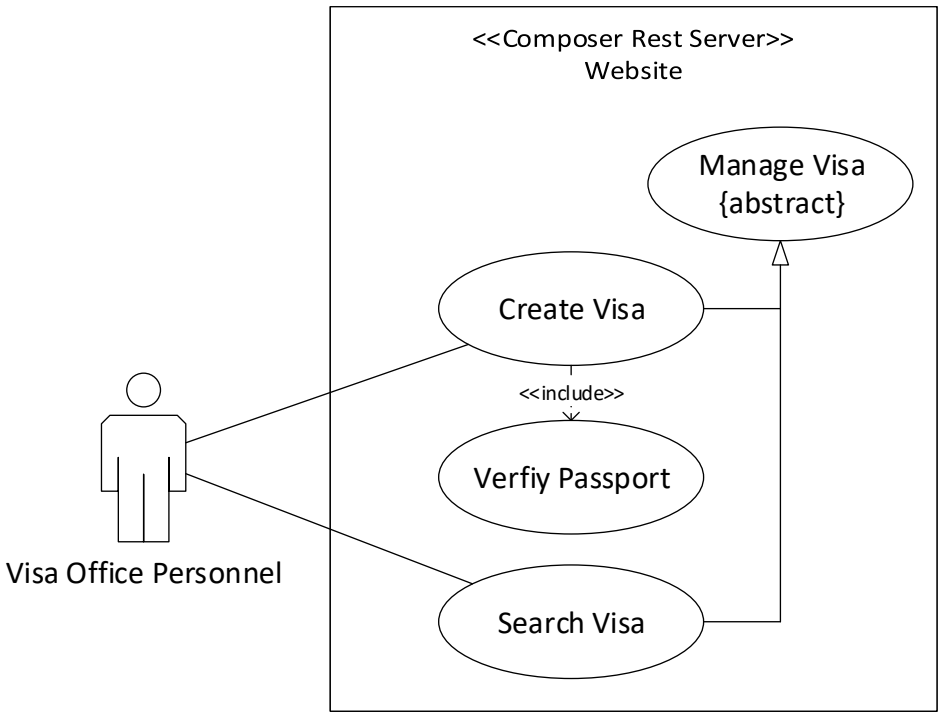

Figure 3.6. Use Case Of Managing Visa

Passport office personnel sends a command to Hyperledger of updating a passport record using web application. This invokes the function of updating passport in the smart contract. It validates the data, id and role of the user. On successful validation, it updates the record on a blockchain Figure 3.8.

Visa office personnel sends a command to Hyperledger of creating a visa record using web application. This invokes the function of creating visa in the smart contract. Furthermore, before adding the record it validates the data, id and role of the user. In addition, it verifies whether passport number which is referred is valid and does the record exists. On successful validation and verification, it creates the record on a blockchain Figure 3.9.

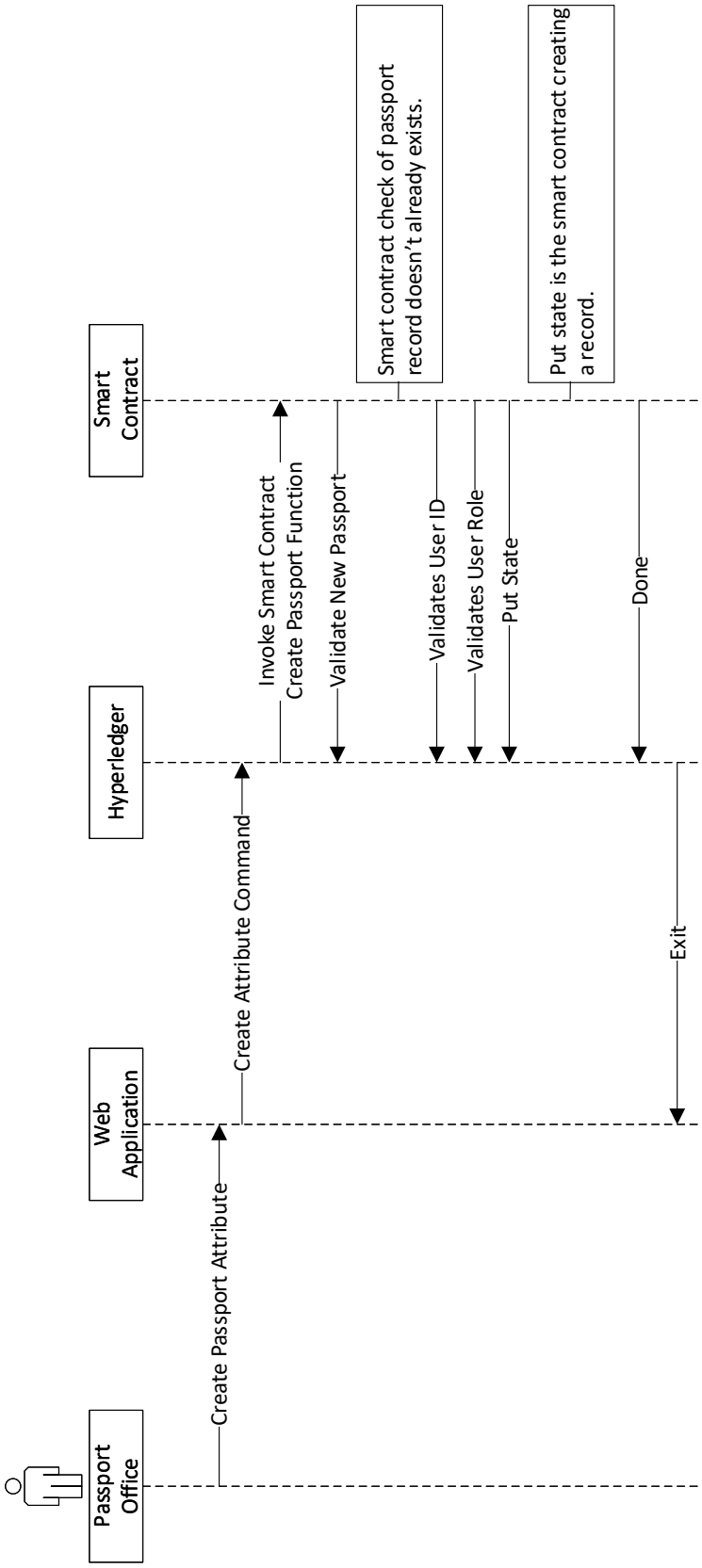

Figure 3.7. Creating A Passport

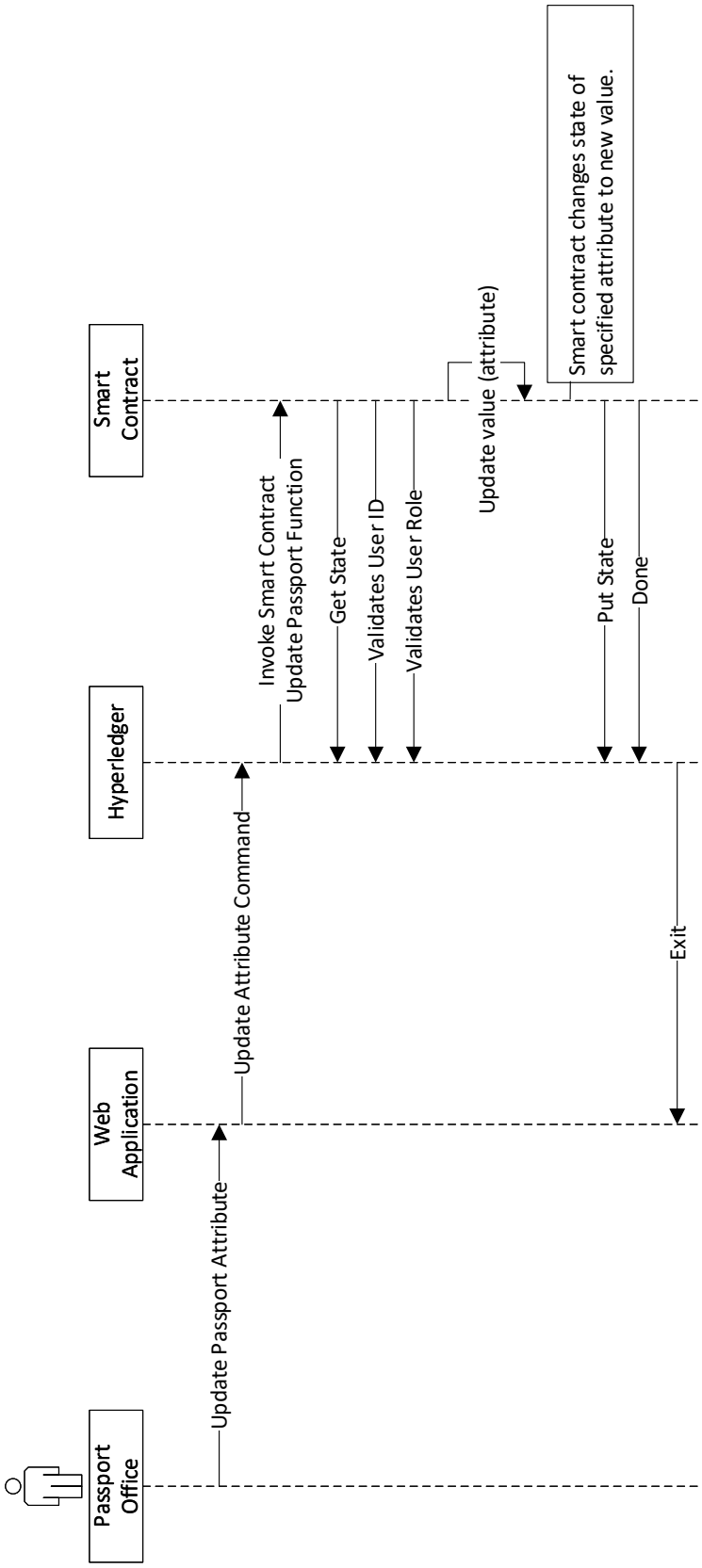

Figure 3.8. Updating A Passport

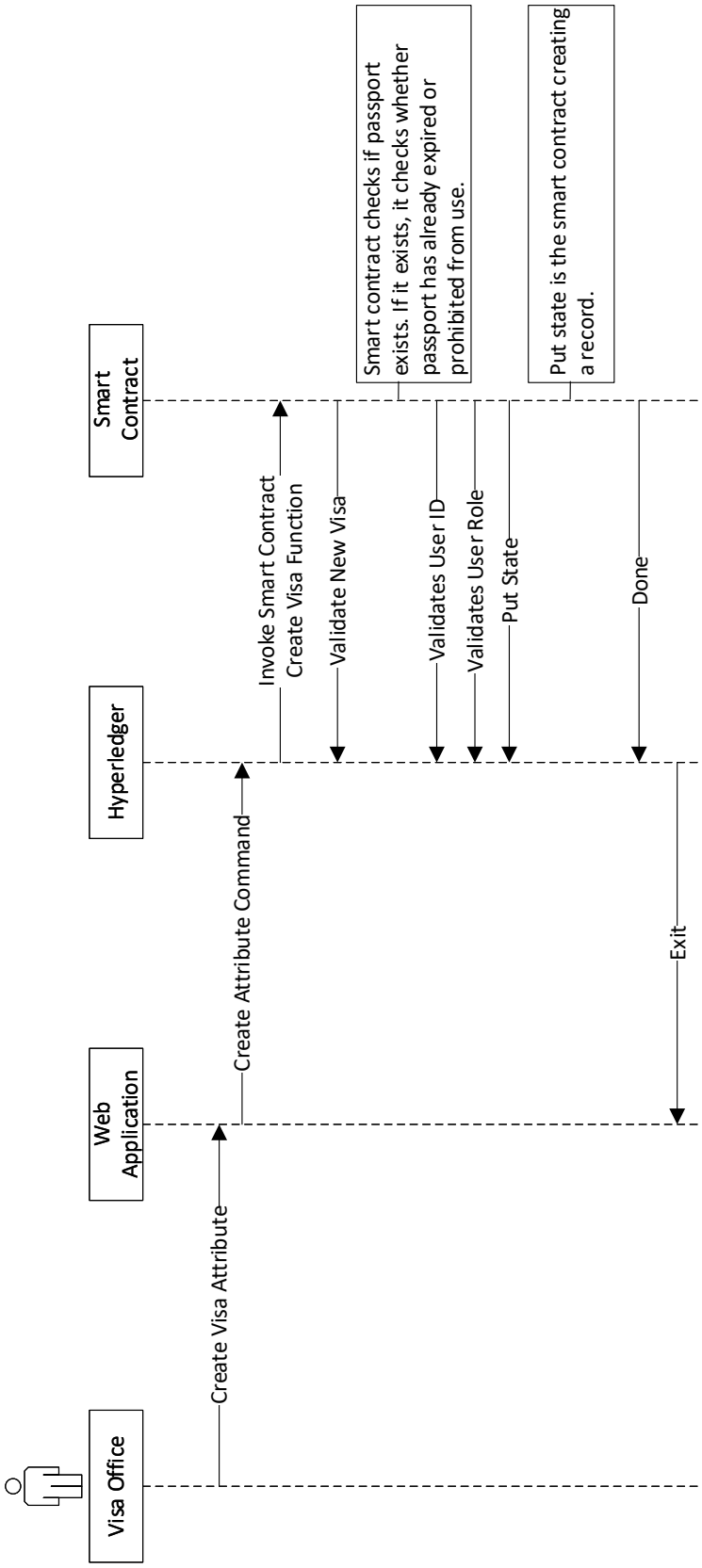

Figure 3.9. Creating A Visa

Chapter 4

DEVELOPMENT OF BLOCKCHAIN SOLUTION

4.1 Business Model Development

For the development of the business model it is necessary to set up a development environment on our machines.

We have used four machines, one for development and remaining for deployment and as well as for testing. Each machine comprises of following specs. We have used a light version of Ubuntu also called Lubuntu for setup of the development environment.

4.1.1 Machine Specifications

1. Operating System: Lubuntu 16.04 x64
2. RAM: 8 GB
3. Hard Drive: 400 GB

In this chapter, we will discuss setup on all those four machines into two parts. We must make sure that we are using the fresh/clean machines to prevent us from redundancy and other errors.

First part will emphasize on discussing the setup of a development environment on one machine and second part will emphasize on implementation, network deployment and environment setup.

4.1.2 Development Environment

Official installation guide is available on the website of the Hyperledger Composer [1].

The development environment is crucial and for that we followed the instructions provided on the official website of Hyperledger Composer.

Prerequisites comprise of following packages.

- Node
- Docker Engine
- npm
- Docker-Compose
- git
- Python

To simplify the installation process, official documents on Hyperledger composer provides us with the simple commands and an automated script to install the above-mentioned prerequisites [6]. The process comprises opening a new terminal and executing the following commands.

```
curl -O https://hyperledger.github.io/composer/latest/prereqs-ubuntu.sh
chmod u+x prereqs-ubuntu.sh
./prereqs-ubuntu.sh
```

Now we must install packages that provides us with Command Line Interface (CLI) tools to interact with business model deployed locally or over the network.

[1] https://hyperledger.github.io/composer/latest/installing/installing-index

1. Step

   Installation of the essential CLI tools.

   ```
   npm install -g composer-cli
   npm install -g composer-rest-server
   npm install -g generator-hyperledger-composer
   npm install -g yo
   ```

2. Step

   Installation of Composer Playground (optional)

   ```
   npm install -g composer-playground
   ```

   *user has a choice of either using the online browser-based application or install locally.

3. Step

   Installation of Interactive Development Environment (optional)
   Download VSCode, navigate to extensions, search for Hyperledger Composer and download that extension.

4. Step

   Installation of Hyperledger Fabric

   (a)
   ```
   mkdir  /fabric-dev-servers && cd  /fabric-dev-servers
   curl -O https://raw.githubusercontent.com/hyperledger/composer-tools/
   master/packages/fabric-dev-servers/fabric-dev-servers.tar.gz
   tar -xvf fabric-dev-servers .tar.gz
   ```

(b)

```
cd /fabric-dev-servers
./downloadFabric.sh
```

5. Step

   Finalizing the environment.

```
cd /fabric-dev-servers
./startFabric.sh
./createPeerAdminCard.sh
```

   By completing all the above steps, we have a working environment and all the necessary images of the Hyperledger Fabric.

4.2 Development and Testing

The typical flow Figure 4.1 of developing a complete business model is as follows.

1. Create or define a business model. The business model contains a definition of not only assets or elements in a business but also the definition of transactions. Therefore, we cannot move forward without completing it beforehand.

2. If there are any transactions defined in a business model, the second step in the development would be to define processors for those functions in Script or Logic file.

3. Permissions are defined for certain user regarding operations and access to elements in a business model.

4. Finally, queries are developed to return the required data from records.

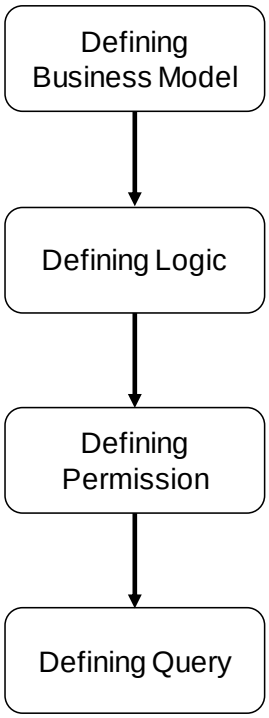

Figure 4.1. Flow of Development

For the development of this business model, the vscode is used.

4.2.1 Defining Business Model

The development of the business model is divided into two parts, the first one focuses on creating a business model while the second one focuses on creating a model file for defining participants.

(A) Business Definition

In this section we will be developing a business model for the blockchain solution.

To give this file a unique identification, we have named it as org.pakistan. cryptopassport. cto A.1.

A namespace is defined for unique identification. namespace org.pakistan. cryptopassport.

The passport is defined as an asset in this file. It is defined by using asset keyword, followed by asset name and property name by which it is identified. To make sure that user only enters only valid input, we have used regular expression with each field or data property to make sure only certain type of input is accepted as an input.

The asset defined in a business model should have all the properties that are defined in real world. Therefore, in this model we have stored all the details that are available on a passport. There are certain properties to be stored in this asset and we must make sure that user only enter valid inputs. So, we have used regular expressions with those data fields and regular expressions are easily identifiable as they are defined with regex keyword.

```
asset Passport identified by passportNumber {
o String         passportNumber     regex = /^[0-9]+$/


//Details about passport


o PassportType  passportType         default="ORDINARY"
o PassportStatus passportstatus      default="PERMITTED"
o String         bookletNumber       regex = /^[0-9]+$/
o String         trackingNumber      regex = /^[0-9]+$/
o String         issuingAuthority
o String         placeOfIssue
o DateTime       dateOfIssue
o DateTime       dateOfExpiry
o String         previousPassportNumber optional
o String         Exclusion           default="ISREAL"


//Personal Details
o String         givenName
```

```
o String        surName
o Gender        gender
o String        religion
o String        fatherName
o String        placeOfBirth
o DateTime      dateOfBirth
o String        citizenshipNumber    regex = /^[0-9]+$/
o String        nationality
}
```

Data types in modeling language consist of but not limited to String, DateTime. Other than these data types, enumerated data types are also used, which are passportStatus, passportType and Gender. With the use of these variables we make sure that user is only able to choose input from options as an input that are available in these data types and these are defined as follows.

```
enum PassportType {
o ORDINARY
o DIPLOMATIC
o OFFICIAL
}


enum PassportStatus{
o PERMITTED
o PROHIBITED
}


enum Gender {
o MALE
o FEMALE
o OTHER
```

```
}
```

The above definition of passport asset follows the structure of a typical passport and all the data or information that a passport contains are stored in this asset. It saves not only the details of a passport holder but also details about passport itself.

To make sure that no invalid characters are stored in a field, regular expressions are used to check the validity of each input.

Visa is used as a transaction in this business model but first, it must be defined in the form of asset and make sure that user accessing that can only create or add this asset's record using transactions only.

```
asset Visa identified by visaNumber{


o String         visaNumber          regex = /^[0-9]+$/
o String         passportNumber       regex = /^[0-9]+$/


//Details about visa


o String         country
o DateTime       issueDate
o DateTime       expiryDate
o String         visaType
o String         duration
o NumberOfEntries numberOfEntries
o String         placeOfIssue
o VisaStatus     visaStatus
o String         applicationNumber    regex = /^[0-9]+$/
}
```

It contains all the data a visa stamp contains on a passport. The asset definition is the same as before in case of passport. Enumerated data types used in this asset

definition are NumberOfEnteries and VisaStatus.

```
enum NumberOfEntries {
o Multiple
o Single
}

enum VisaStatus {
o Accepted
o Rejected
}
```

As described before, a visa is created using transaction, now transaction function is defined in this same file.

The difference between defining an asset and transaction is that we use keyword transaction instead of an asset. There is also no identifier in the case of the transaction.

```
transaction CreateVisa{
-->Passport     passportResource
o String        visanumber
o DateTime      issueDate
o DateTime      expiryDate
o String        country
o String        visaType
o String        duration
o NumberOfEntries numberOfEntries
o String        placeOfIssue
o VisaStatus    visaStatus
o String        applicationNumber
}
```

In our business model, we have defined visa as a part of the passport asset and we

have defined that relationship by using following code of line.

```
-->Passport      passportResource
```

It takes passport number as an input for it to be used as a reference to the passport asset in the records. When a transaction executes two more data fields are attached to it with those that are defined in this model automatically. Those data fields are timestamp and transactionId. Timestamp holds the info about time at which transaction was executed while transactionId is a unique identifier that is also automatically generated.

(B) Participant Definition

In this model A.2, we define users that interact with the business model that is to be deployed. The namespace of this model is defined as follows:

```
namespace org.pakistan.cryptopassport.participant
```

There are different resources definitions in the business model. Before defining each participant separately, we must define an abstract type which holds participant key and contact. Participant key is the unique identifier of that participant and contact is a concept that holds the details of participant like first name, last name, and email.

```
abstract participant CryptoPassportParticipant identified by
    participantKey {
o String        participantKey
o Contact       contact
}


concept Contact {
o String        firstName
o String        lastname
o String        email
}
```

Here we are defining three types of participants according to the roles that are to be assigned.

1. Network Admin

```
participant CryptoPassportNetworkAdmin extends
    CryptoPassportParticipant {
/** This is a concrete resource definition */
}
```

2. Passport Office Personnel

```
  participant CryptoPassportOfficePersonnel extends
      CryptoPassportParticipant {
  o String        department
  o String        City
  }
```

3. Visa Office Personnel

```
  participant CryptoPassportVisaOfficePersonnel extends
      CryptoPassportParticipant {
  o String        department
  o String        City
  }
```

4.2.2 Defining Logic

Visa is used as a transaction and for that, we must create a script file that contains the logic for executing the transaction A.3.

Parameter tag is used to represent the definition of the parameter. It is followed by the complete name of the specific transaction that is defined in a specific model.

Following this, is the name of the parameter, which in this case is AddVisa.

```
/**
* @param {org.pakistan.cryptopassport.CreateVisa} AddVisa
* @transaction
*/
```

A new async function named as CreateVisa that takes AddVisa as an input parameter is created.

```
async function CreateVisa(AddVisa) {
}
```

This function contains all the actions that are to be executed upon creating a transaction to which it is related.

```
const assetRegistry    = await getAssetRegistry('org.pakistan.
    cryptopassport.Passport');
const validPassport    = await assetRegistry.exists(AddVisa.
    passportResource.passportNumber);
var passportStatus     = AddVisa.passportResource.passportstatus;
var passportDateOfExpiry = new Date(AddVisa.passportResource.
    dateOfExpiry).getTime();
```

There are four variables defined, assetRegistry, validPassport, passportStatus, and passportDateOfExpiry.

The assetRegistry is storing the registry of asset we have defined as a passport. It is to check whether the passport number that is being referred in the creation of visa really exists or not. Therefore, first, we must access the passport registry using this variable.

For the successful creation of visa, passport number is entered as an input. The validPassport check for the existence of that passport and stores the result.

The variable passportStatus is storing the status of passport that is being

referred.

The pdoe variable is accessing the date of expiry of that passport and storing it.

We have implemented three checks for the creation of visa, which are then being implemented in if statements.

1. The existence of a valid passport.
2. Status of that passport.
3. The expiry date of that passport.

If a passport exists, then it checks its status that whether it has been permitted for use or not and if that checks out, we finally check whether the passport has expired or not. If any of it fails, this function will throw an error and function will exist without creating a visa.

```
if (validno) {
if(passportStatus=="PERMITTED") {
//statements
}}
else{throw new Error("Passport has been prohibited !!!");}}
else {throw new Error('this transaction failed, no such Passport');}
```

If the passport has been permitted for use then the following statements will execute.

```
let factory            = getFactory();
let registry           = await getAssetRegistry("org.pakistan.
    cryptopassport.Visa");
let resource           = factory.newResource("org.pakistan.
    cryptopassport","Visa", AddVisa.visanumber);
```

Three variables have been defined in the above statements.

1. The factory variable is used to create an instance of the resource.

2. The registry variable is accessing the registry of visa asset.

3. The resource variable is storing a template based on the definition in a model file and assigning it identifier as it is in the model definition.

```
resource.passportNumber = AddVisa.passportResource.passportNumber;
resource.country        = AddVisa.country
resource.expiryDate     = AddVisa.expiryDate
resource.issueDate      = AddVisa.issueDate
resource.visaType       = AddVisa.visaType
resource.duration       = AddVisa.duration
resource.numberOfEntries = AddVisa.numberOfEntries
resource.placeOfIssue   = AddVisa.placeOfIssue
resource.visaStatus     = AddVisa.visaStatus
resource.applicationNumber = AddVisa.applicationNumber
var dateOfIssue         = new Date(AddVisa.issueDate).getTime();
var dateOfExpiry        = new Date(AddVisa.expiryDate).getTime();
```

Now we are accessing the data input given at the time of executing this transaction and storing it in the template created before.

```
if(dateOfExpiry < dateOfIssue){throw new Error("Date of expiry cannot
    be in the past!!!");}
else if(passportDateOfExpiry<dateOfIssue){throw new Error("Passport has
     already expired!!!");}
else if(passportDateOfExpiry<dateOfExpiry){throw new Error("Passport is
     expiring in visa duration!!!");}
else {await registry.add(resource);}
```

Here one more check has been placed before creating visa, which checks whether the dates of a visa issue and visa expiry are valid or not. If it satisfies that

condition, a visa is created successfully.

4.2.3 Defining Permissions

In this section, we have defined the access rules for each participant that has been previously created in model definition A.4.

Rules definition follows a certain template. First, we write rule keyword followed by name of the rule. Then we define description, participant, operation, resource and action.

Description contains the short description about what this rule does. Participant define which participant can access this rule or for what participant it is created. Operation contains information about what actions can be performed by that participant. Resource contains information about which resource. And finally, action defines whether participant can perform above mentioned operations on defined resources or not.

To allow network admin to create participants and to maintain the business network, we have defined following access.

```
rule NetworkControlPermission {
description:    "give admin ALL access to system resources"
participant:    "org.hyperledger.composer.system.NetworkAdmin"
operation:      ALL
resource:       "org.hyperledger.composer.system.*"
action:         ALLOW
}

rule ParticipantRegistryControlPermission {
description:    "give admin ALL access to CryptoPassport participant
    types"
participant:    "org.hyperledger.composer.system.NetworkAdmin"
operation:      ALL
```

```
resource:      "org.pakistan.cryptopassport.participant.*"
action:        ALLOW
}
```

This rule is defined so that every participant could read the system registries.

```
rule CryptoPassportPermissionSystem {
description:  "can READ the system registries"
participant:  "ANY"
operation:    READ
resource:     "org.hyperledger.composer.system.**"
action:       ALLOW
}
```

Following rules are defined so that passport office personnel could create, read or update the records related to asset which is passport. And, a rule to allow this user to read history of the system.

```
rule CryptoPassportOfficePersonnelPermissionSystem {
description:  "can CREATE the system registries"
participant:  "org.pakistan.cryptopassport.participant.
    CryptoPassportOfficePersonnel"
operation:    CREATE
resource:     "org.hyperledger.composer.system.**"
action:       ALLOW
}

rule CryptoPassportOfficePersonnelPermission {
description:  "only passport Office personnel can CREATE, READ and
    UPDATE a passport"
participant:  "org.pakistan.cryptopassport.participant.
```

```
    CryptoPassportOfficePersonnel"
operation:    CREATE,READ,UPDATE
resource:     "org.pakistan.cryptopassport.Passport"
action:       ALLOW
}

rule CryptoPassportParticipantPermissionHistorian {
description:  "can CREATE HistorianRecord to the Historian"
participant:  "org.pakistan.cryptopassport.participant.
    CryptoPassportOfficePersonnel"
operation:    CREATE
resource:     "org.hyperledger.composer.system.HistorianRecord"
action: ALLOW
}
```

To make sure that user responsible for creation of visa, following rules are written for him. These rules allow him to create visa and as well read the records of passport.

```

rule CryptoPassportVisaOfficeParticipantPermissionHistorian {
description:  "can write HistorianRecord to the Historian"
participant:  "org.pakistan.cryptopassport.participant.
    CryptoPassportVisaOfficePersonnel"
operation:    CREATE
resource:     "org.hyperledger.composer.system.HistorianRecord"
action:       ALLOW
}

rule CryptoPassportVisaOfficePersonnelPermission {
description:  "only Visa Office personnel can CREATE and READ a visa"
participant:  "org.pakistan.cryptopassport.participant.
```

```
    CryptoPassportVisaOfficePersonnel"
operation:    CREATE,READ
resource:     "org.pakistan.cryptopassport.**"
transaction:  "org.pakistan.cryptopassport.CreateVisa"
action:       ALLOW
}



rule CryptoPassportVisaOfficePersonnelReadPassportPermission {
description:  "Reading a passport before creating a visa"
participant:  "org.pakistan.cryptopassport.participant.
    CryptoPassportVisaOfficePersonnel"
operation:    READ
resource:     "org.pakistan.cryptopassport.Passport"
action:       ALLOW
}

rule CryptoPassportVisaOfficePersonnelReadVisaPermission {
description:  "Reading a visa"
participant:  "org.pakistan.cryptopassport.participant.
    CryptoPassportVisaOfficePersonnel"
operation:    READ
resource:     "org.pakistan.cryptopassport.Visa"
action:       ALLOW
}
```

4.2.4 Defining Query

Queries are used to return data from records of an asset. We define queries by using query keyword, followed by short description and finally statement. Statement

tells the about the records of an asset that are to be accessed A.5.

By completing all the steps, we have created a complete business model for the passport verification and extended it by verifying passport on visa creation. And with that we have also defined certain users depending on their roles. And have created access rules for them. And finally defined all the queries that can be executed on assets on this model.

4.2.5 Generating Business Network Archive

A typical solution comprises four components, model, script, access control and queries. These components are then merged together in an archive file called, business network archive.

To create an archive file from files that have been previously created, we open a new terminal from the directory of our working and execute the following command.

```
composer archive create --sourceType dir --sourceName ../
```

This commands generate a single file with extension '.bna'. In this it has created a cryptopassports.bna file.

Now we need to install our business solution on the development environment that we have deployed previously.

First, we install archive onto the environment and then we use another command to start the business network on the environment. These commands are as follows:

```
composer network install-a.\cryptopassports@0.0.1.bna-c PeerAdmin@hlfv1
composer network start-c PeerAdmin@hlfv1-n cryptopassports-V 0.0.1-A
    admin-S adminpw
```

Here PeerAdmin@hlfv1 is a peer administrator that is created by default at the time of setting up development environment. And admin is default username and adminpw is default secret or password.

Now we have a working environment with business network deployed over it.

We test the rules defined in the access control by creating participants and issuing them identity. To check this, we will be creating three users:

1. Network Administrator
2. Passport Office Personnel
3. Visa Office Personnel

Access rules and how these participants will make use of this network, has already been described in previous section therefore, here we will focus on creating these participants and verify their access control.

1. Network Administrator

   Network administrator is responsible for creating a participant, here by default a network administrator is created admin@cryptopassports. But here we will be creating a new participant who is a network administrator. Following command is executed for creating a participant.

```
composer participant add -d '{"$class":"org.pakistan.
    cryptopassport.participant.cryptopassportnetworkadmin","
    participantKey":"mahmedsaeedi","contact":{"$class":"org.
    pakistan.cryptopassport.participant.Contact","firstName":"
    Muhammad Ahmed","lastname":"Saeedi","email":"m.ahmed.
    saeedi@outlook.com"}}' -c admin@cryptopassports
```

   Above command creates a participant with details entered in this command. It is as exactly discussed in the section of creating access control. From same terminal now we need to issue identity to this participant by executing following command

```
composer identity issue -u mahmedsaeedi -a org.pakistan.
    cryptopassport.participant.CryptoPassportNetworkAdmin#
    mahmedsaeedi -c admin@cryptopassports
```

Finally, we export identity card that has been created as a successful execution of previous commands. Following command is executed in terminal

```
Command import -f mahmedsaeedi@cryptopassports.
```

This card that has been created contains all the information regrading this newly created participant. And now same process will be repeated for creating other participants and exporting their cards.

2. Passport Office Personnel

   In this section we are creating personnel for passport office, Shahzad. And issuing him identity and exporting card.

```
composer participant add -d '{"$class":"org.pakistan.
    cryptopassport.participant.CryptoPassportOfficePersonnel","
    participantKey":"shahzad","contact":{"$class":"org.pakistan.
    cryptopassport.participant.Contact","firstName":"Shahzad","
    lastname":"Shoukat","email":"shahzad@outlook.com"},"
    department":"Passport office","City":"Islamabad"}' -c
    admin@cryptopassports
```

```
composer identity issue -u shahzadshoukat -a org.pakistan.
    cryptopassport.participant.CryptoPassportOfficePersonnel#
    shahzadshoukat -c admin@cryptopassports
```

```
composer import -f shahzadshoukat@cryptopassports.card
```

3. Visa Office Personnel

   In this section we are creating personnel for passport office, Asad. And issuing him identity and exporting card.

```
composer participant add -d '{"$class":"org.pakistan.
    cryptopassport.participant.CryptoPassportVisaOfficePersonnel
```

```
    ","participantKey":"asadhayat","contact":{"$class":"org.
    pakistan.cryptopassport.participant.Contact","firstName":"
    Asad","lastname":"Hayat","email":"asad@outlook.com"},"
    department":"visa office","City":"Islamabad"}' -c
    admin@cryptopassports
```

```
composer identity issue -u asadhayat -a org.pakistan.
    cryptopassport.participant.CryptoPassportVisaOfficePersonnel
    #asadhayat -c admin@cryptopassports
```

```
composer import -f asadhayat@cryptopassports.card
```

We can start a Composer Rest Server with each card created to verify the access and operations which are executable by each participant on the business network.

4.2.6 Participant Access Verification

(A) Network Administrator

Firstly, we will start Composer Rest Server with network administrator participant. Following command is executed to start server with card of this participant.

```
composer-rest-server -c masaeedi@cryptopassports -n never -w true.
```

This participant can only create participant Figures 4.3, 4.4 and if he tries to access or create passport or any other asset it will produce error Figures 4.5, 4.6.

(B) Passport Office Personnel

First we will exit the current terminal and will open a new terminal and execute following command to start rest server with the passport office personnel participant that we have already created.

```
composer-rest-server -c shahzad@cryptopassports -n never -w true.
```

This participant can only create and update passport record on the ledger. We can verify this by opening a browser and entering the local address of rest server that has deployed because of the execution of above command. The interface in figure 4.2 is provided by default when we start a rest server. Names of assets might change according to the business model. All the transactions, assets, participants and queries are displayed as they were defined at the time of business modeling.

According to the access control that has been defined at the time of business modeling, this participant should only be able to create or update a passport record and should not be able to create a visa or participant.

1. **List passport**

   We can see all the list of operations that is available by default. And we can further expand each operation by clicking on it. Here we will be expanding the first operation that is *GET*, this operation will list all the passports in the record.

   If we click on *Try it out!* Figure 4.7, it will return empty because at this point we haven't entered a record. By the end of this section we have not only created a business model and deployed it over development environment but also, we have created participants. Next section deals with network deployment over three physical machines that we have set up earlier.

2. **Create a Passport**

   Next step is to check if participant can successfully create a passport or not. For this, we click on *POST* operation and enter following data into value section of parameters. Make sure data that is being entered doesn't violates any regular expression that has been defined otherwise it will produce an error Figure 4.8. Now, if we click on *Try it out!*, it can be observed that operation has been successful Figure 4.9. In the same way we can navigate to other operations and

try if we have access to only available operations or all the operations. User will fail to delete passport as we have not provided such access to the participant Figures 4.10, 4.11.

(C) Visa Office Personnel

We will exit the previously opened terminal and will open a new terminal and execute following command to start rest server with the visa office personnel participant that we have already created.

```
composer-rest-server -c asad@cryptopassports -n never -w true
```

This participant can only create a visa record on the ledger. We can verify this by opening a browser and entering the local address of rest server that has deployed because of the execution of above command.

1. **Create a Visa**

   Here we will create a visa by navigating to *CreateVisa:* A transaction named *CreateVisa*, here recall that we have defined visa as an asset, but it is created by using a transaction as we want our visa record to not be tempered with.

   For this, we click on *POST* operation and enter following data into value section of parameters.

   Make sure data that is being entered doesn't violates any regular expression that has been defined otherwise it will produce an error. Transaction id should be left empty as it is created automatically on successful execution Figure 4.12.

   Now, if we click on Try it out!, it can be observed that operation has been successful Figure 4.13.

   In the same way we can navigate to other operations and try if we have access to only available operations or all the operations.

By the end of this section we have not only created a business model and deployed it over development environment but also, we have created participants.

Next section deals with network deployment over three physical machines that we have set up earlier.

4.3 Implementation

In this research we have built a blockchain solution based on the development framework that can be implemented in accordance to the requirement on the country. Network structure and business model might differ from country to country but the asset around which that business model would be built, would remain same. For implementation of our solution, we have built a small network consisting of three physical machines. We have assigned each machine a role in accordance to the network model discussed in earlier sections. We have also built a frontend application based on web languages html, css and java. User have the choice of choosing any other language according to the requirement.

In the next section we will discuss how we have deployed our business model on our network.

Prerequisites should be installed on each machine like we did during the development of business model and with that we will also download platform specific binaries on each machine that would provide us with necessary images for business network and solution deployment.

```
curl -sSL http://bit.ly/2ysbOFE | bash -s 1.1.0
```

This will download fabric samples and binaries along with it. We will be using these samples in upcoming steps. Before moving on, we must make sure that each peer can discover each other. We can achieve this by adding ip addresses of each peer on every other peer or machine that we are using for network deployment. We can ping every other machine from each machine to check if it is able to discover those machines or not. We will be using these machines to setup a simple fabric network. Each machine will be used for different participant in the network.

4.3.1 Structure of network

We will refer machine 1, machine 2 and machine 3, to network admin, passport office personnel and visa office personnel respectively.

Orderer, certificate authority will be deployed on the same machine that is network admin but in production deployment they should be on separate machine to prevent it from becoming a single point of failure.

Each machine has its own peer and CouchDB image or binaries deployed to be used by itself.

There are many files in the fabric-dev-servers that we have downloaded previously. We will be focusing on the ones that we will be using for network deployment. For the configuration we will navigate to the directory where fabric-dev-servers, we will further navigate into a folder called fabric-scripts. There are two folders hlfv1 and hlfv11. We will be working in hlfv11 as we are using fabric 1.1 version.

There is a folder within this folder called composer and in composer there will be many files and folder. If there is a folder named crypto-config, we must first delete it before continuing. There is a file named as crypto-config.yaml. In this file we define the domain of our network and numbers of peers other than admin peer of this network.

```
OrdererOrgs:
-Name:Orderer
Domain:nust.com
Specs:
-Hostname:orderer
PeerOrgs:
-Name:Org1
Domain:org1.nust.com
Template:
Count:3
```

```
Users:
Count: 2
```

It contains all the necessary information to establish a network between peers in a blockchain network. Template is used to define the number total users on the blockchain network and Users is used to define the number of users other than admin user. In this model we have one network admin and two other participants therefore we have used this configuration.

Now we need to run a command to generate these templates, for that we will be executing following command in the new terminal.

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

This will generate artifacts in the same directory under the folder, crypto-config. Now we must create a genesis block and channel for our business network to communicate on. For that we will be executing following commands in the terminal.

```
configtxgen -profile ComposerOrdererGenesis -outputBlock ./composer-
    genesis.block
configtxgen -profile ComposerChannel -outputCreateChannelTx ./composer-
    channel.tx -channelID composerchannel
```

This will produce composer-genesis.block and composer-channel.tx files.

Before continuing, we must modify the contents of other files. We will start modification from docker-compose.yml file. Replace all the contents with the following contents. This contains all the necessary modification for successful deployment of a blockchain network.

```
version: '2'


services:
ca.org1.nust.com:
image: hyperledger/fabric-ca:$ARCH-1.1.0
```

```
environment:
- FABRIC_CA_HOME=/etc/hyperledger/fabric-ca-server
- FABRIC_CA_SERVER_CA_NAME=ca.org1.nust.com
ports:
- 7054:7054
command: sh -c 'fabric-ca-server start --ca.certfile /etc/hyperledger/
    fabric-ca-server-config/ca.org1.nust.com-cert.pem --ca.keyfile /etc/
    hyperledger/fabric-ca-server-config/3
    bea6f6f32ec415fe7b3bff9b39044a4cfa789a484b0014b88f4580377715800_sk -
    b admin:adminpw -d'
volumes:
- ./crypto-config/peerOrganizations/org1.nust.com/ca/:/etc/hyperledger/
    fabric-ca-server-config
container_name: ca.org1.nust.com


orderer.nust.com:
container_name: orderer.nust.com
image: hyperledger/fabric-orderer:$ARCH-1.1.0
environment:
- ORDERER_GENERAL_LOGLEVEL=debug
- ORDERER_GENERAL_LISTENADDRESS=0.0.0.0
- ORDERER_GENERAL_GENESISMETHOD=file
- ORDERER_GENERAL_GENESISFILE=/etc/hyperledger/configtx/composer-
    genesis.block
- ORDERER_GENERAL_LOCALMSPID=OrdererMSP
- ORDERER_GENERAL_LOCALMSPDIR=/etc/hyperledger/msp/orderer/msp
working_dir: /opt/gopath/src/github.com/hyperledger/fabric
command: orderer
ports:
- 7050:7050
```

```
volumes:
- ./:/etc/hyperledger/configtx
- ./crypto-config/ordererOrganizations/nust.com/orderers/orderer.nust.
    com/msp:/etc/hyperledger/msp/orderer/msp


peer0.org1.nust.com:
container_name: peer0.org1.nust.com
image: hyperledger/fabric-peer:$ARCH-1.1.0
dns_search: .
extra_hosts:
- "orderer.nust.com:xx.x.xx.xxx"
environment:
- CORE_LOGGING_LEVEL=debug
- CORE_CHAINCODE_LOGGING_LEVEL=DEBUG
- CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
- CORE_PEER_ID=peer0.org1.nust.com
- CORE_PEER_ADDRESS=peer0.org1.nust.com:7051
- CORE_CHAINCODE_STARTUPTIMEOUT=1200s
- CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=composer_default
- CORE_PEER_LOCALMSPID=Org1MSP
- CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/peer/msp
- CORE_LEDGER_STATE_STATEDATABASE=CouchDB
- CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb:5984
working_dir: /opt/gopath/src/github.com/hyperledger/fabric
command: peer node start
ports:
- 7051:7051
- 7053:7053
volumes:
- /var/run/:/host/var/run/
```

```
- ./:/etc/hyperledger/configtx
- ./crypto-config/peerOrganizations/org1.nust.com/peers/peer0.org1.nust.
    com/msp:/etc/hyperledger/peer/msp
- ./crypto-config/peerOrganizations/org1.nust.com/users:/etc/
    hyperledger/msp/users
depends_on:
- orderer.nust.com
- couchdb


couchdb:
container_name: couchdb
image: hyperledger/fabric-couchdb:$ARCH-0.4.6
ports:
- 5984:5984
environment:
DB_URL: http://localhost:5984/member_db
```

Replace the ip address of the orderer with the ip address of a local machine. Also replace certificate in the following line with the one in the directory.

```
composer/crypto-config/peerOrganizations/org1.example.com/ca/.
```

```
command: sh -c 'fabric-ca-server start --ca.certfile /etc/hyperledger/
    fabric-ca-server-config/ca.org1.nust.com-cert.pem --ca.keyfile /etc/
    hyperledger/fabric-ca-server-config/3
    bea6f6f32ec415fe7b3bff9b39044a4cfa789a484b0014b88f4580377715800_sk -
    b admin:adminpw -d'
```

This file is responsible for deploying images on the local machines. We will be modifying this file for other machines accordingly.

For second machine we will create a new file docker-compose-peer2.yml.

```
version: '2'
```

```
services:
peer1.org1.nust.com:
container_name: peer1.org1.nust.com
image: hyperledger/fabric-peer:$ARCH-1.1.0
extra_hosts:
- "orderer.nust.com:10.9.21.250"
environment:
- CORE_LOGGING_LEVEL=debug
- CORE_CHAINCODE_LOGGING_LEVEL=DEBUG
- CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
- CORE_PEER_ID=peer1.org1.nust.com
- CORE_PEER_ADDRESS=peer1.org1.nust.com:7051
- CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=composer_default
- CORE_PEER_LOCALMSPID=Org1MSP
- CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/peer/msp
- CORE_LEDGER_STATE_STATEDATABASE=CouchDB
- CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb2:5984
working_dir: /opt/gopath/src/github.com/hyperledger/fabric
command: peer node start
ports:
- 8051:7051
- 8053:7053
volumes:
- /var/run/:/host/var/run/
- ./:/etc/hyperledger/configtx
- ./crypto-config/peerOrganizations/org1.nust.com/peers/peer1.org1.nust.
    com/msp:/etc/hyperledger/peer/msp
- ./crypto-config/peerOrganizations/org1.nust.com/users:/etc/
    hyperledger/msp/users
depends_on:
```

```
- couchdb2


couchdb2:
container_name: couchdb2
image: hyperledger/fabric-couchdb:$ARCH-0.4.6
ports:
- 6984:5984
environment:
DB_URL: http://localhost:6984/member_db
```

Similarly, we will create a file for last machine docker-composer-peer3.yml

```
version: '2'
services:
peer2.org1.nust.com:
container_name: peer2.org1.nust.com
image: hyperledger/fabric-peer:$ARCH-1.1.0
extra_hosts:
- "orderer.nust.com:10.9.21.250"
environment:
- CORE_LOGGING_LEVEL=debug
- CORE_CHAINCODE_LOGGING_LEVEL=DEBUG
- CORE_VM_ENDPOINT=unix:///host/var/run/docker.sock
- CORE_PEER_ID=peer2.org1.nust.com
- CORE_PEER_ADDRESS=peer2.org1.nust.com:7051
- CORE_VM_DOCKER_HOSTCONFIG_NETWORKMODE=composer_default
- CORE_PEER_LOCALMSPID=Org1MSP
- CORE_PEER_MSPCONFIGPATH=/etc/hyperledger/peer/msp
- CORE_LEDGER_STATE_STATEDATABASE=CouchDB
- CORE_LEDGER_STATE_COUCHDBCONFIG_COUCHDBADDRESS=couchdb2:5984
working_dir: /opt/gopath/src/github.com/hyperledger/fabric
```

```
command: peer node start
ports:
- 9051:7051
- 9053:7053
volumes:
- /var/run/:/host/var/run/
- ./:/etc/hyperledger/configtx
- ./crypto-config/peerOrganizations/org1.nust.com/peers/peer2.org1.nust.
    com/msp:/etc/hyperledger/peer/msp
- ./crypto-config/peerOrganizations/org1.nust.com/users:/etc/
    hyperledger/msp/users
depends_on:
- couchdb2


couchdb2:
container_name: couchdb2
image: hyperledger/fabric-couchdb:$ARCH-0.4.6
ports:
- 7984:5984
environment:
DB_URL: http://localhost:7984/member_db
```

Now go to directory hlfv11, there is another file that we will modifying. That file is createPeerAdminCard.sh.

Replace the contents of this machine with the following

```
#!/bin/bash


Usage() {
echo ""
echo "Usage: ./createPeerAdminCard.sh [-h host] [-n]"
```

```
echo ""
echo "Options:"
echo -e "\t-h or --host:\t\t(Optional) name of the host to specify in
    the connection profile"
echo -e "\t-n or --noimport:\t(Optional) don't import into card store"
echo ""
echo "mat: ./createPeerAdminCard.sh"
echo ""
exit 1
}

Parse_Arguments() {
while [ $# -gt 0 ]; do
case $1 in
--help)
HELPINFO=true
;;
--host | -h)
shift
HOST="$1"
;;
--noimport | -n)
NOIMPORT=true
;;
esac
shift
done
}

HOST=localhost
```

```
Parse_Arguments $@


if [ "${HELPINFO}" == "true" ]; then
Usage
fi


# Grab the current directory
DIR="$( cd "$( dirname "${BASH_SOURCE[0]}" )" && pwd )"


if [ -z "${HL_COMPOSER_CLI}" ]; then
HL_COMPOSER_CLI=$(which composer)
fi


echo
# check that the composer command exists at a version >v0.16
COMPOSER_VERSION=$("${HL_COMPOSER_CLI}" --version 2>/dev/null)
COMPOSER_RC=$?


if [ $COMPOSER_RC -eq 0 ]; then
AWKRET=$(echo $COMPOSER_VERSION | awk -F. '{if ($2<19) print "1"; else
    print "0";}')
if [ $AWKRET -eq 1 ]; then
echo Cannot use $COMPOSER_VERSION version of composer with fabric 1.1,
    v0.19 or higher is required
exit 1
else
echo Using composer-cli at $COMPOSER_VERSION
fi
else
echo 'No version of composer-cli has been detected, you need to install
```

```
    composer-cli at v0.19 or higher'
exit 1
fi

cat << EOF > connection.json
{
"name": "hlfv1",
"x-type": "hlfv1",
"x-commitTimeout": 300,
"version": "1.0.0",
"client": {
"organization": "Org1",
"connection": {
"timeout": {
"peer": {
"endorser": "300",
"eventHub": "300",
"eventReg": "300"
},
"orderer": "300"
}
}
},
"channels": {
"composerchannel": {
"orderers": [
"orderer.nust.com"
],
"peers": {
"peer0.org1.nust.com": {},
```

```
"peer1.org1.nust.com": {},
"peer2.org1.nust.com": {}
}
}
},
"organizations": {
"Org1": {
"mspid": "Org1MSP",
"peers": [
"peer0.org1.nust.com",
"peer1.org1.nust.com",
"peer2.org1.nust.com"
],
"certificateAuthorities": [
"ca.org1.nust.com"
]
}
},
"orderers": {
"orderer.nust.com": {
"url": "grpc://xx.x.xx.xxx:7050"
}
},
"peers": {
"peer0.org1.nust.com": {
"url": "grpc://xx.x.xx.xxx:7051",
"eventUrl": "grpc://xx.x.xx.xxx:7053"
},
"peer1.org1.nust.com": {
"url": "grpc://xx.x.xx.xxx:8051",
```

```
"eventUrl": "grpc://xx.x.xx.xxx:8053"
},
"peer2.org1.nust.com": {
"url": "grpc://xx.x.xx.xxx:9051",
"eventUrl": "grpc://xx.x.xx.xxx:9053"
}
},
"certificateAuthorities": {
"ca.org1.nust.com": {
"url": "http://xx.x.xx.xxx:7054",
"caName": "ca.org1.nust.com"
}
}
}
EOF

PRIVATE_KEY="${DIR}"/composer/crypto-config/peerOrganizations/org1.nust.
    com/users/Admin@org1.nust.com/msp/keystore/
    c8c9bbdfa447c296cc15e76b4161ec6319487cbe46f622530203d7f6fbea07ca_sk
CERT="${DIR}"/composer/crypto-config/peerOrganizations/org1.nust.com/
    users/Admin@org1.nust.com/msp/signcerts/Admin@org1.nust.com-cert.
    pem

if [ "${NOIMPORT}" != "true" ]; then
CARDOUTPUT=/tmp/PeerAdmin@hlfv1.card
else
CARDOUTPUT=PeerAdmin@hlfv1.card
fi

"${HL_COMPOSER_CLI}" card create -p connection.json -u PeerAdmin -c "${
```

```
    CERT}" -k "${PRIVATE_KEY}" -r PeerAdmin -r ChannelAdmin --file
    $CARDOUTPUT

if [ "${NOIMPORT}" != "true" ]; then
if "${HL_COMPOSER_CLI}" card list -c PeerAdmin@hlfv1 > /dev/null; then
"${HL_COMPOSER_CLI}" card delete -c PeerAdmin@hlfv1
fi

"${HL_COMPOSER_CLI}" card import --file /tmp/PeerAdmin@hlfv1.card
"${HL_COMPOSER_CLI}" card list
echo "Hyperledger Composer PeerAdmin card has been imported, host of
    fabric specified as '${HOST}'"
rm /tmp/PeerAdmin@hlfv1.card
else
echo "Hyperledger Composer PeerAdmin card has been created, host of
    fabric specified as '${HOST}'"
fi
```

Replace the ip addresses of peers with ip addresses of your own machines.

Save all files and copy whole folder of fabric-dev-servers on other machines. After copying whole folder, you can simplify the structure of files by removing irrelevant docker-compose file and renaming the relevant one to docker-compose.yml for example, remove docker-compose.yml file from second machine or peer and rename docker-compose-peer2.yml to docker-composer.yml

Open a new terminal on first machine from within the directory and execute following command.

```
./startFabric.sh && ./createPeerAdminCard.sh
```

This will set up the network and deploy relevant images locally. In second machine, execute

```
./startFabric.sh
```

Also, in third machine, execute

```
./startFabric.sh
```

We don't need to execute ./createPeerAdminCard.sh other than first machine.

Now we need to install and start the business network, we can achieve this by changing our directory to where our business network archive and card is located. We will doing this step on first machine. And execute following commands

```
composer network install --card PeerAdmin@hlfv1 --archiveFile
    cryptopassports.bna
```

```
composer network start -n cryptopassports -V 0.0.1 -A admin -S adminpw
    --card PeerAdmin@hlfv1 -f admin@cryptopassports.card
```

```
composer import -f admin@cryptopassports.card
```

On this machine we will be creating other users or participants cards for use in the network just like we did in the development environment.

After creating cards, we can transfer or copy that file to respective machine. We can check if business network has been deployed successfully by executing following command

```
composer network ping --card admin@cryptopassports
```

On each machine we will deploy rest server. Execute following commands respective to each machine.

Machine 1:

```
composer-rest-server -c masaeedi@cryptopassports -n never -w true.
```

Machine 2:

```
composer-rest-server -c shahzad@cryptopassports -n never -w true.
```

Machine 3:

```
composer-rest-server -c asad@cryptopassports -n never -w true.
```

In above commands it can be noted that we have passed few other arguments. In this command we must mention card that we are using for deployment of local rest server and we supply the name of card with -c. In this case we have not enabled authentication '-n', tls '-w' and use of namespaces 'never'. But we have allowed our system to listen to events published. We have specified it in this command by using 'true'. This can be changed according to the requirement of the business model.

We can again verify access rules of each participant and it can be observed that each participant is only able to carry out operations that he has been allowed to. In this chapter we have learned to set up four machines one for development and other three for deployment of business network. And by the end of this chapter we have a working business network.

Now we can use our frontend application and use it to interact with the business model in a friendly manner.

Figure 4.2. Composer Rest Server

```
aeedi@Research:~$ composer participant add -d '{"$class":"org.pakistan.cryptopassport.participant.CryptoPassport
VisaOfficePersonnel","participantKey":"asadhayat","contact":{"$class":"org.pakistan.cryptopassport.participant.
ontact","firstName":"Asad","lastname":"Hayat","email":"asad@outlook.com"},"department":"visa office","City":"Is
amabad"}' -c admin@cryptopassports
articipant was added to participant registry.

ommand succeeded

aeedi@Research:~$ composer identity issue -u asadhayat -a org.pakistan.cryptopassport.participant.CryptoPassport
VisaOfficePersonnel#asadhayat -c admin@cryptopassports
ssue identity and create Network Card for: asadhayat

 Issuing identity. This may take a few seconds...

uccessfully created business network card file to
        Output file: asadhayat@cryptopassports.card

ommand succeeded
```

Figure 4.3. Creating Passport Office Personnel

```
saeedi@Research:~$ composer participant add -d '{"$class":"org.pakistan.cryptopassport.participant.CryptoPassport
:OfficePersonnel","participantKey":"shahzadshoukat","contact":{"$class":"org.pakistan.cryptopassport.participant
.Contact","firstName":"Shahzad","lastname":"Shoukat","email":"shahzad@outlook.com"},"department":"Passport offic
","City":"Islamabad"}' -c admin@cryptopassports
Participant was added to participant registry.

Command succeeded

saeedi@Research:~$ composer identity issue -u shahzadshoukat -a org.pakistan.cryptopassport.participant.CryptoPa
sportOfficePersonnel#shahzadshoukat -c admin@cryptopassports
Issue identity and create Network Card for: shahzadshoukat

Issuing identity. This may take a few seconds...

Successfully created business network card file to
        Output file: shahzadshoukat@cryptopassports.card

Command succeeded
```

Figure 4.4. Creating Visa Office Personnel

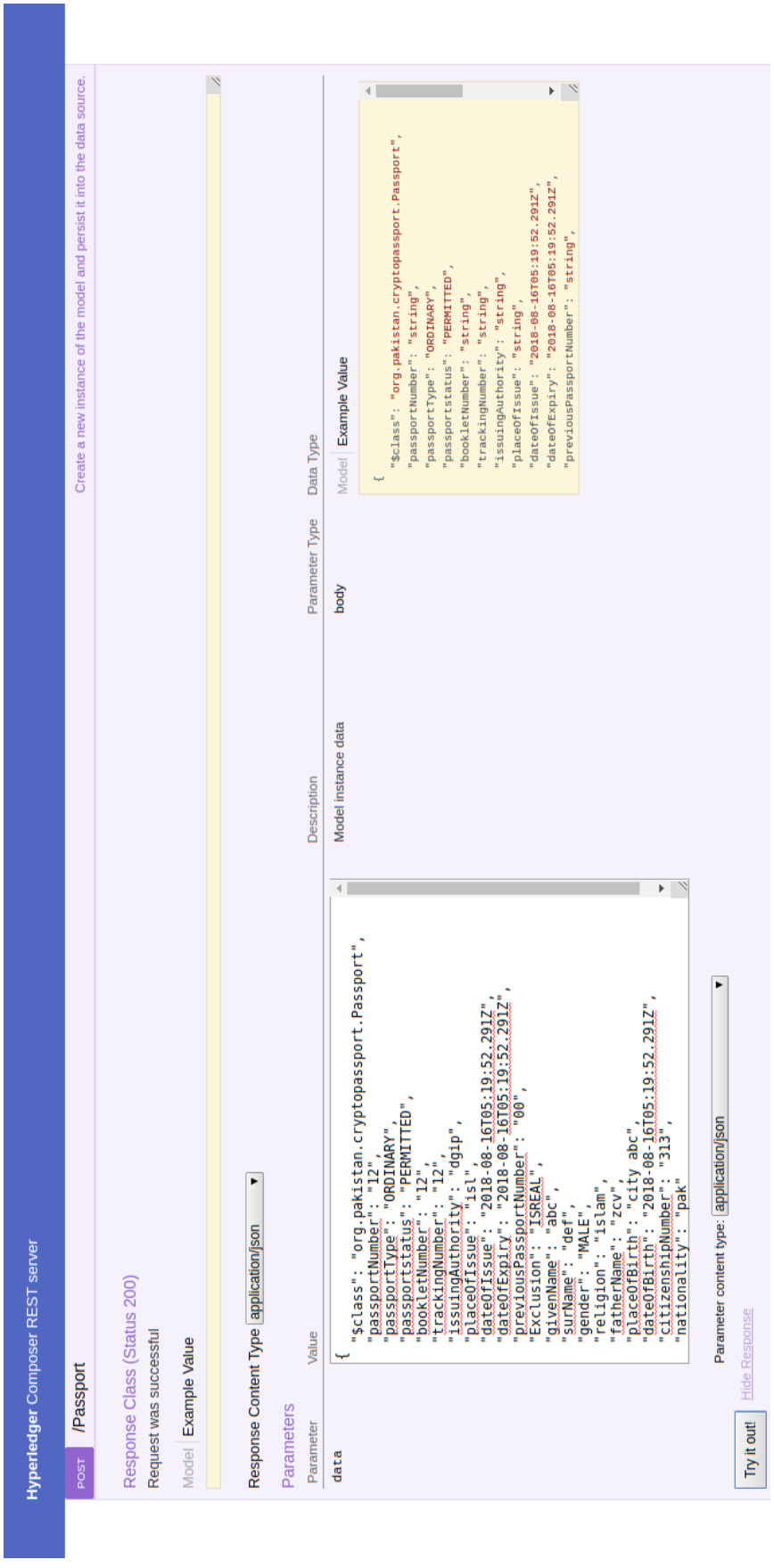

Figure 4.5. Network Admin Trying To Create Passport

Figure 4.6. Access Error While Creating Passport

Figure 4.7. List Passport

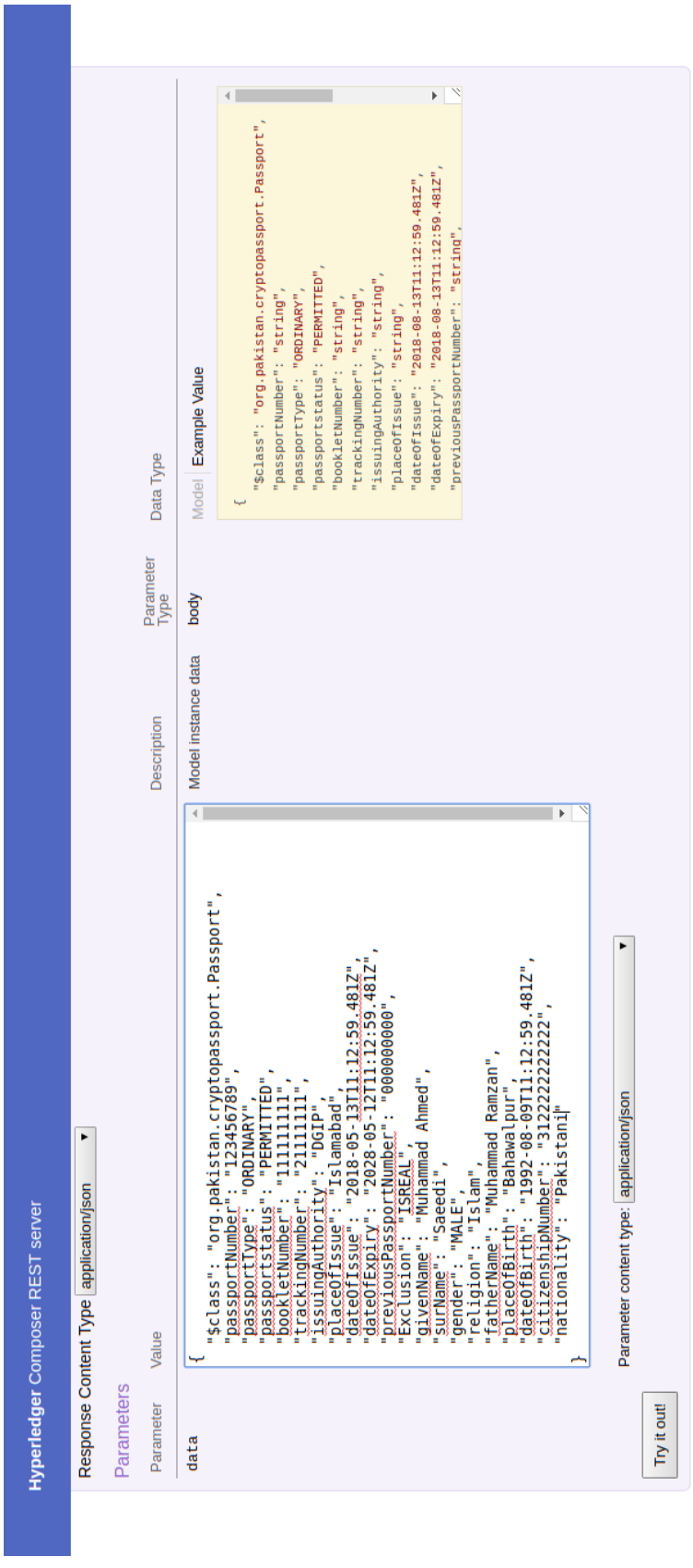

Figure 4.8. Creating Passport

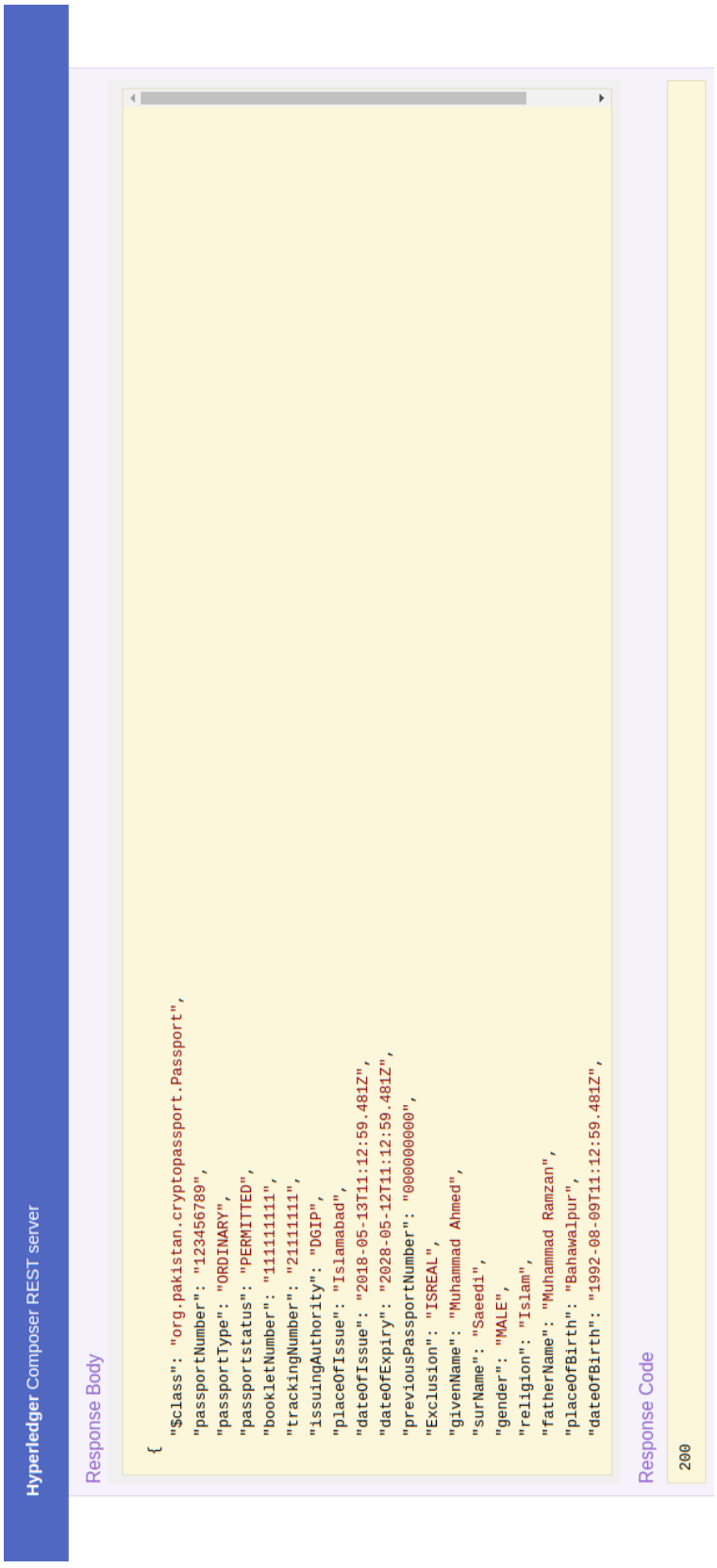

Figure 4.9. Creating Passport Success Message

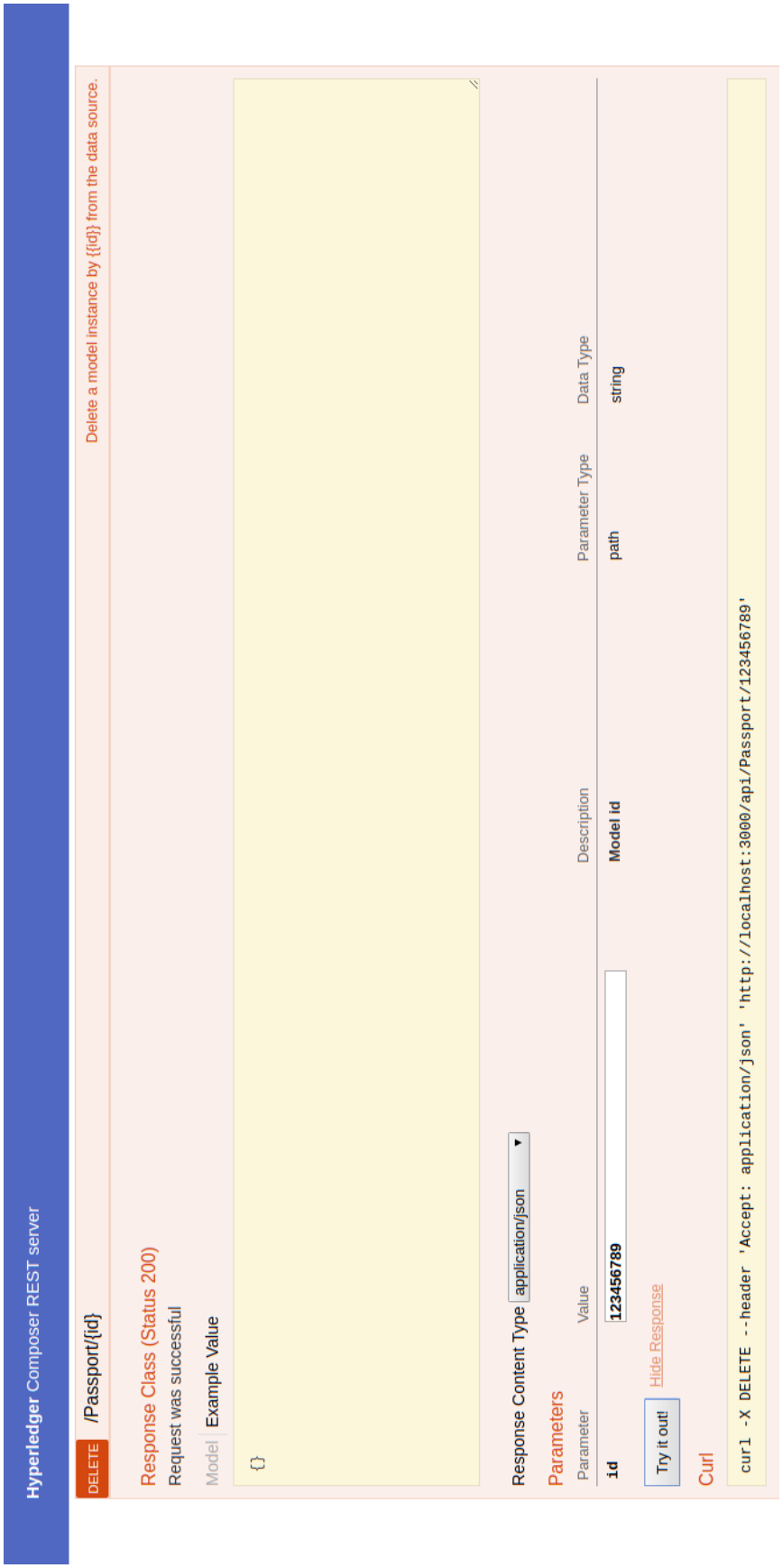

Figure 4.10. Entering Details To Delete Passport

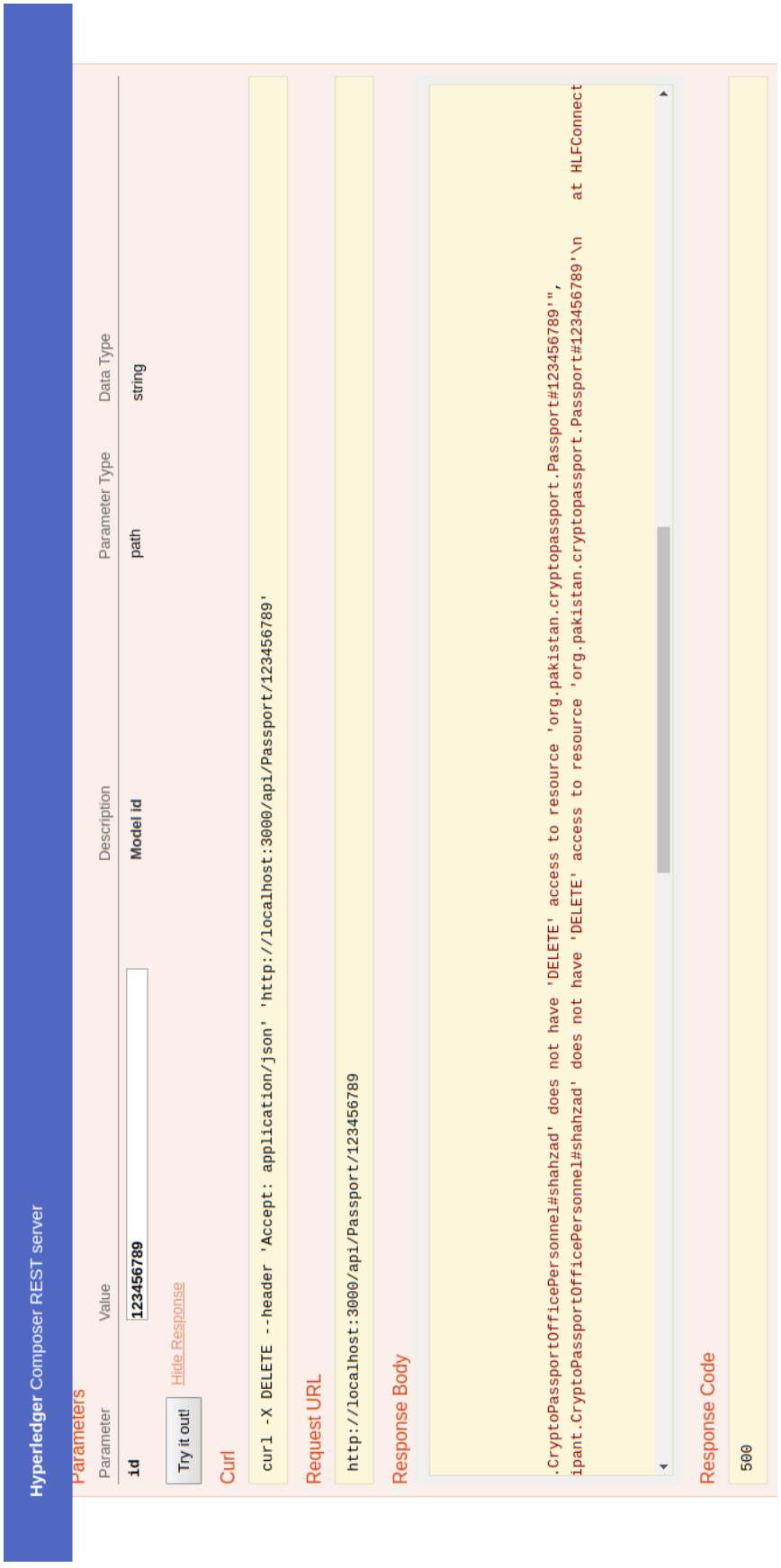

Figure 4.11. Error Produced While Deleting Passport

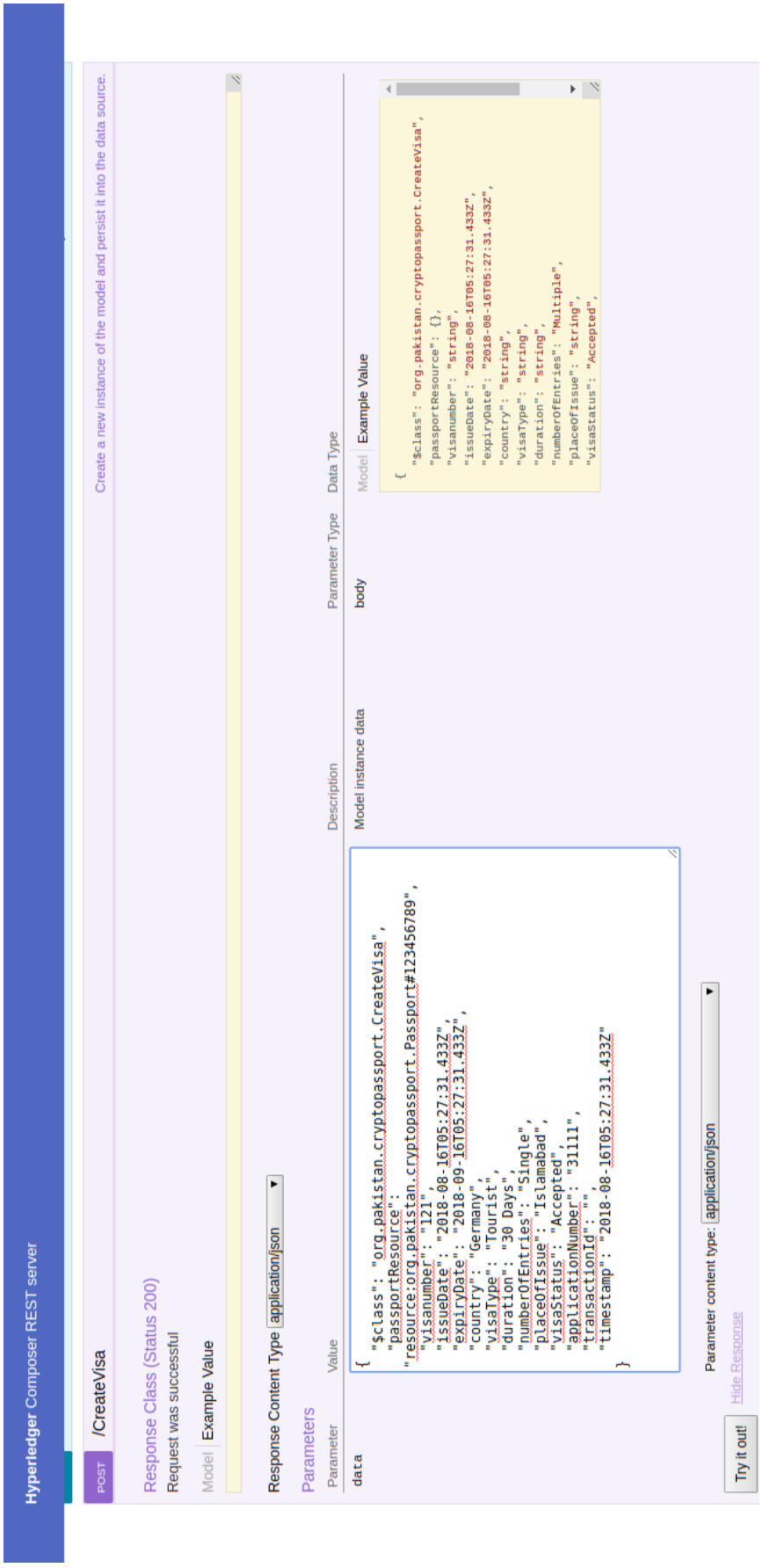

Figure 4.12. Creating Visa

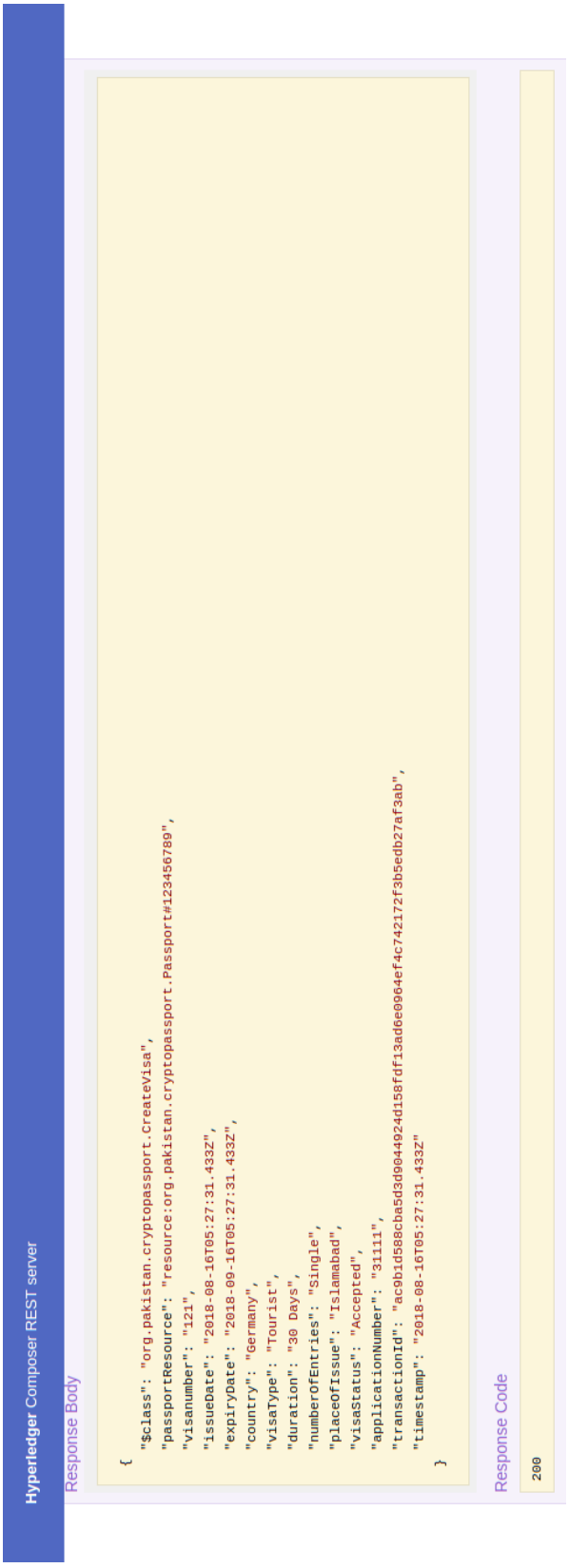

Figure 4.13. Visa Creation Successful

Chapter 5

DISCUSSION AND CONCLUSION

5.1 Discussion

Blockchain is an emerging technology in web. In this research, a blockchain application, Cryptopassports, for passport verification has been developed. In comparison to current implementations for passport security by ICAO, cryptopassports provides the participant on the network with secure access to sensitive information about a passport and its holder. Data is recorded in the form of ledger which is maintained by all peers on the network. In addition, data is almost impossible to hack and hackers will not be able to access sensitive data or to make any alterations to data records. The participants on the network will only be able to access the records. Each participant has been given certain access to record or carry out CRUD (Create, Read, Update, Delete) operations on data records. Cryptopassports has been tested using data set of twenty passports. And for the proof of concept, it has been deployed on a physical network. For testing purposes, three participants were created for carrying out different operations on a network. Those participants comprise a network administrator, a passport office personnel and a visa office personnel. These participants were given access according to their roles in the network. A network administrator was only allowed to carry out operations regarding network and its participants. In comparison to that, passport office personnel was allowed to carry out operations on passport. For example, passport office personnel was only able to create, update or verify the passport. And similarly, visa office personnel was only allowed to create or verify visa. Furthermore, creation of visa also verifies the passport. The record was maintained on all peers of the network but different access was granted to each participant according to the roles given to these participants.

5.2 Conclusion

In this research, a blockchain application, Cryptopassports, for verification of passport has been created. Cryptopassports has been tested and implemented on a physical network. Furthermore, cryptopassports provides a way to verify a passport instantly. With the implementation of crypotpassports, it will be almost impossible for hackers to access or alter data record in any way. Due to its unique technological aspect, the data will be maintained by all peers. The application, cryptopassport, has been found to be more efficient and secure then the current systems that are currently implemented for passport verification in many countries including Pakistan. With the implementation of the proposed system, it will be nearly impossible for anyone to hack or penetrate the database. This will result in efficient and instant verification of passports. And it would be made sure that only permissioned and verified participants on the network, are able to access the record.

A future prospect may be to integrate the cryptopassports with other tools of the Hyperledger. It is also possible to deploy Hyperledger network on other tools e.g. docker swarm or kubernetes. We can add more features to make access to passport verification more secure by using biometric or retina scan. It might be possible to add functionality of storing picture of passport holder off the chain and then accessing it at the time of passport verification. Passport verification is one use case of blockchain, there are many other use cases that can be used for implementation of blockchain based applications e.g. document verification, supplychain and manymore.

REFERENCES

[1] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the Thirteenth EuroSys Conference*, page 30. ACM, 2018.

[2] Christian Cachin. Architecture of the hyperledger blockchain fabric. In *Workshop on Distributed Cryptocurrencies and Consensus Ledgers*, volume 310, 2016.

[3] Christian Decker and Roger Wattenhofer. Information propagation in the bitcoin network. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–10. IEEE, 2013.

[4] Vikram Dhillon, David Metcalf, and Max Hooper. *The Hyperledger Project*, pages 139–149. Apress, Berkeley, CA, 2017.

[5] Garry Gabison. Policy considerations for the blockchain technology public and private applications. *SMU Sci. & Tech. L. Rev.*, 19:327, 2016.

[6] Richard Hull. Blockchain: Distributed event-based processing in a data-centric world. In *Proceedings of the 11th ACM International Conference on Distributed and Event-based Systems*, pages 2–4. ACM, 2017.

[7] Ari Juels, David Molnar, and David Wagner. Security and privacy issues in e-passports. In *Security and Privacy for Emerging Areas in Communications Networks, 2005. SecureComm 2005. First International Conference on*, pages 74–88. IEEE, 2005.

[8] Eleni Kosta, Martin Meints, Marit Hansen, and Mark Gasson. An analysis of security and privacy issues relating to rfid enabled epassports. In *IFIP International Information Security Conference*, pages 467–472. Springer, 2007.

[9] Yogesh Kumar, Rajiv Munjal, and Harsh Bardhan Sharma. Comparison of symmetric and asymmetric cryptography with existing vulnerabilities and countermeasures. 2011.

[10] Iuon-Chang Lin and Tzu-Chun Liao. A survey of blockchain security issues and challenges. *IJ Network Security*, 19(5):653–659, 2017.

[11] Mahdi H Miraz. Blockchain: Technology fundamentals of the trust machine. *Machine Lawyering, Chinese University of Hong Kong, 23rd December*, 2017.

[12] Axel Moinet, Benoît Darties, and Jean-Luc Baril. Blockchain based trust & authentication for decentralized sensor networks. *arXiv preprint arXiv:1706.01730*, 2017.

[13] Satoshi Nakamoto. Whitepaper: Bitcoin: A peer-to-peer electronic cash system. *Email posted to listserv*, 9:04, 2008.

[14] Arvind Narayanan, Joseph Bonneau, Edward Felten, Andrew Miller, and Steven Goldfeder. *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[15] Wayne Penny. Biometrics: A double edged sword-security and privacy. *SANS Institute*, 2002.

[16] Fergal Reid and Martin Harrigan. An analysis of anonymity in the bitcoin system. In *Security and privacy in social networks*, pages 197–223. Springer, 2013.

[17] Zdenek Riha and Vashek Matyas. Privacy issues of electronic passport. 01 2011.

[18] Lakshmi Siva Sankar, M Sindhu, and M Sethumadhavan. Survey of consensus protocols on blockchain applications. In *Advanced Computing and Communication Systems (ICACCS), 2017 4th International Conference on*, pages 1–5. IEEE, 2017.

[19] Bikramaditya Singhal, Gautam Dhameja, and Priyansu Sekhar Panda. How blockchain works. In *Beginning Blockchain*, pages 31–148. Springer, 2018.

[20] Melanie Swan. *Blockchain: Blueprint for a new economy*. " O'Reilly Media, Inc.", 2015.

[21] Tim Swanson. Consensus-as-a-service: a brief report on the emergence of permissioned, distributed ledger systems. *Report, available online, Apr*, 2015.

[22] Nick Szabo. Smart contracts: building blocks for digital markets. *EXTROPY: The Journal of Transhumanist Thought,(16)*, 1996.

[23] Parth Thakkar, Senthil Nathan, and Balaji Vishwanathan. Performance benchmarking and optimizing hyperledger fabric blockchain platform. *arXiv preprint arXiv:1805.11390*, 2018.

[24] Moti Yung, Yevgeniy Dodis, Aggelos Kiayias, and Tal Malkin. *Public Key Cryptography-PKC 2006: 9th International Conference on Theory and Practice in Public-Key Cryptography, New York, NY, USA, April 24-26, 2006. Proceedings*, volume 3958. Springer, 2006.

Appendix A

Appendix

A.1 Business Model

```
namespace org.pakistan.cryptopassport
//Passport Definition

asset Passport identified by passportNumber {
o String        passportNumber     regex = /^[0-9]+$/

//Details about passport

o PassportType  passportType         default="ORDINARY"
o PassportStatus passportstatus      default="PERMITTED"
o String        bookletNumber        regex = /^[0-9]+$/
o String        trackingNumber       regex = /^[0-9]+$/
o String        issuingAuthority
o String        placeOfIssue
o DateTime      dateOfIssue
o DateTime      dateOfExpiry
o String        previousPassportNumber optional
o String        Exclusion            default="ISREAL"

//Personal Details
o String        givenName
o String        surName
o Gender        gender
o String        religion
o String        fatherName
o String        placeOfBirth
o DateTime      dateOfBirth
o String        citizenshipNumber    regex = /^[0-9]+$/
o String        nationality
}

// Visa Definition

asset Visa identified by visaNumber{

o String        visaNumber         regex = /^[0-9]+$/
o String        passportNumber       regex = /^[0-9]+$/

//Details about visa

o String        country
o DateTime      issueDate
```

```
o DateTime      expiryDate
o String        visaType
o String        duration
o NumberOfEntries numberOfEntries
o String        placeOfIssue
o VisaStatus    visaStatus
o String        applicationNumber    regex = /^[0-9]+$/
}

enum PassportType {
o ORDINARY
o DIPLOMATIC
o OFFICIAL
}

enum PassportStatus{
o PERMITTED
o PROHIBITED
}

enum Gender {
o MALE
o FEMALE
o OTHER
}

enum NumberOfEntries {
o Multiple
o Single
}

enum VisaStatus {
o Accepted
o Rejected
}

// Transaction For Creating Visa

transaction CreateVisa{
-->Passport     passportResource
o String        visanumber
o DateTime      issueDate
o DateTime      expiryDate
o String        country
o String        visaType
o String        duration
o NumberOfEntries numberOfEntries
o String        placeOfIssue
o VisaStatus    visaStatus
```

```
o String        applicationNumber
}
```

A.2 Modelling of Participant

```
namespace org.pakistan.cryptopassport.participant

abstract participant CryptoPassportParticipant identified by
    participantKey {
o String        participantKey
o Contact       contact
}

concept Contact {
o String        firstName
o String        lastname
o String        email
}

participant CryptoPassportNetworkAdmin extends
    CryptoPassportParticipant {
/** This is a concrete resource definition */
}

participant CryptoPassportOfficePersonnel extends
    CryptoPassportParticipant {
o String        department
o String        city
}

participant CryptoPassportVisaOfficePersonnel extends
    CryptoPassportParticipant {
o String        department
o String        city
}
```

A.3 Transaction logic

```
/**
* @param {org.pakistan.cryptopassport.CreateVisa} AddVisa
* @transaction
*/

async function CreateVisa(AddVisa) {
```

```
const assetRegistry    = await getAssetRegistry('org.pakistan.
    cryptopassport.Passport');
const validPassport    = await assetRegistry.exists(AddVisa.
    passportResource.passportNumber);
var passportStatus     = AddVisa.passportResource.passportstatus;
var passportDateOfExpiry = new Date(AddVisa.passportResource.
    dateOfExpiry).getTime();
if (validPassport) {
if(passportStatus=="PERMITTED")
{
let factory             = getFactory();
let registry            = await getAssetRegistry("org.pakistan.
    cryptopassport.Visa");
let resource            = factory.newResource("org.pakistan.
    cryptopassport","Visa", AddVisa.visanumber);


resource.passportNumber  = AddVisa.passportResource.passportNumber;
resource.country         = AddVisa.country
resource.expiryDate      = AddVisa.expiryDate
resource.issueDate       = AddVisa.issueDate
resource.visaType        = AddVisa.visaType
resource.duration        = AddVisa.duration
resource.numberOfEntries = AddVisa.numberOfEntries
resource.placeOfIssue    = AddVisa.placeOfIssue
resource.visaStatus      = AddVisa.visaStatus
resource.applicationNumber = AddVisa.applicationNumber
var dateOfIssue          = new Date(AddVisa.issueDate).getTime();
var dateOfExpiry         = new Date(AddVisa.expiryDate).getTime();

if(dateOfExpiry < dateOfIssue){throw new Error("Date of expiry cannot
    be in the past!!!");}

else if(passportDateOfExpiry<dateOfIssue){throw new Error("Passport has
     already expired!!!");}

else if(passportDateOfExpiry<dateOfExpiry){throw new Error("Passport is
     expiring in visa duration!!!");}

else {await registry.add(resource);}
}else{throw new Error("Passport has been prohibited !!!");}

}else {throw new Error('this transaction failed, no such Passport');}

}
```

A.4 Access Rules

```
rule CryptoPassportPermissionSystem {
description:  "can READ the system registries"
participant:  "ANY"
operation:    READ
resource:     "org.hyperledger.composer.system.**"
action:       ALLOW
}

rule CryptoPassportOfficePersonnelPermissionSystem {
description:  "can CREATE the system registries"
participant:  "org.pakistan.cryptopassport.participant.
    CryptoPassportOfficePersonnel"
operation:    CREATE
resource:     "org.hyperledger.composer.system.**"
action:       ALLOW
}

rule NetworkControlPermission {
description:  "give admin ALL access to system resources"
participant:  "org.hyperledger.composer.system.NetworkAdmin"
operation:    ALL
resource:     "org.hyperledger.composer.system.*"
action:       ALLOW
}

rule ParticipantRegistryControlPermission {
description:  "give admin ALL access to CryptoPassport participant
    types"
participant:  "org.hyperledger.composer.system.NetworkAdmin"
operation:    ALL
resource:     "org.pakistan.cryptopassport.participant.*"
action:       ALLOW
}

rule CryptoPassportParticipantPermissionHistorian {
description:  "can CREATE HistorianRecord in the Historian"
participant:  "org.pakistan.cryptopassport.participant.
    CryptoPassportOfficePersonnel"
operation:    CREATE
resource:     "org.hyperledger.composer.system.HistorianRecord"
action: ALLOW
}


rule CryptoPassportVisaOfficeParticipantPermissionHistorian {
description:  "can write HistorianRecord to the Historian"
participant:  "org.pakistan.cryptopassport.participant.
    CryptoPassportVisaOfficePersonnel"
operation:    CREATE
```

```
resource:     "org.hyperledger.composer.system.HistorianRecord"
action:       ALLOW
}

rule CryptoPassportVisaOfficePersonnelPermission {
description:  "only Visa Office personnel can CREATE and READ a visa"
participant:  "org.pakistan.cryptopassport.participant.
    CryptoPassportVisaOfficePersonnel"
operation:    CREATE,READ
resource:     "org.pakistan.cryptopassport.**"
transaction:  "org.pakistan.cryptopassport.CreateVisa"
action:       ALLOW
}


rule CryptoPassportVisaOfficePersonnelReadPassportPermission {
description:  "Reading a passport before creating a visa"
participant:  "org.pakistan.cryptopassport.participant.
    CryptoPassportVisaOfficePersonnel"
operation:    READ
resource:     "org.pakistan.cryptopassport.Passport"
action:       ALLOW
}

rule CryptoPassportVisaOfficePersonnelReadVisaPermission {
description:  "Reading a visa"
participant:  "org.pakistan.cryptopassport.participant.
    CryptoPassportVisaOfficePersonnel"
operation:    READ
resource:     "org.pakistan.cryptopassport.Visa"
action:       ALLOW
}


rule CryptoPassportOfficePersonnelPermission {
description:  "only passport Office personnel can CREATE, READ and
    UPDATE a passport"
participant:  "org.pakistan.cryptopassport.participant.
    CryptoPassportOfficePersonnel"
operation:    CREATE,READ,UPDATE
resource:     "org.pakistan.cryptopassport.Passport"
action:       ALLOW
}
```

A.5 Queries

```
query AllPassports {
description: "Returns all passports in the registry"
```

```
statement:  SELECT org.pakistan.cryptopassport.Passport
}

query AllVisas {
description: "Returns all passports in the registry"
statement:  SELECT org.pakistan.cryptopassport.Visa
}

query PassportByNumber {
description: "Returns a passport in the registry"
statement:  SELECT org.pakistan.cryptopassport.Passport
WHERE  (passportNumber == _$Passport_Number)
}

query VisaByNumber {
description: "Returns a Visa in the registry"
statement:  SELECT org.pakistan.cryptopassport.Visa
WHERE  (visaNumber == _$Visa_Number)
}

query HistoryOfVisasForAPassport {
description: "List all records of Visa for a specific Passport"
statement:  SELECT org.pakistan.cryptopassport.Visa
WHERE  (passportNumber == _$Passport_Number)
}
```