M.Sc. (IT)

SEMESTER - IV (CBCS)

BLOCK CHAIN

SUBJECT CODE : PSIT401

Prof. Suhas Pednekar
Vice-Chancellor,
University of Mumbai,

Prof. Ravindra D. Kulkarni
Pro Vice-Chancellor,
University of Mumbai,

Prof. Prakash Mahanwar
Director,
IDOL, University of Mumbai,

Programme Co-ordinator : **Prof. Mandar Bhanushe**
Head, Faculty of Science and Technology,
IDOL, University of Mumbai, Mumbai

Course Co-ordinator : **Ms. Preeti Bharanuke**
Asst. Professor, M.Sc. I.T.
IDOL, University of Mumbai, Mumbai

Editor : **Dr. Rajendra Patil**
Associate Professor
Vidyalankar school of Information technology,
Wadala (East), Mumbai

Course Writers : **Ms. Arati Bansode**
Assistant Professor,
SIES College of Arts, Science and Commerce,
Nerul, Navi Mumbai

: **Ms. Arati Sahitya**
Assistant Professor,
Somaiya college, Mumbai

: **Ms. Mannat Amit Doultani**
Assistant Professor,
Vivekanand Education Society's Institute of
Technology, Mumbai

: **Ms. Disha Roshan Bhakta**
Assistant Professor,
S M Shetty College of Science Commerce
and Management Studies, Powai, Mumbai

: **Ms. Rani Narayan Podechetty**
Assistant Professor,
K.B college of Arts and commerce for Women

December 2021, Print - I

Published by : Director
Institute of Distance and Open Learning ,
University of Mumbai,
Vidyanagari, Mumbai - 400 098.

DTP Composed : Mumbai University Press
Printed by Vidyanagari, Santacruz (E), Mumbai

CONTENTS

Unit No.	Title	Page No.
	Unit- I	
1.	Introduction to Blockchain	01
2.	Blockchain Foundation	18
	Unit- II	
3.	Ethereum	74
4.	Solidity programming	104
	Unit - III	
5.	Hyperledger	124
6.	Smart Contracts and tokens	134
	Unit- IV	
7.	Mining Ether	144
8.	cryptoeconomics	161
	Unit - V	
9.	Blockchain Applications Development	165
10.	Building an Ethereum DApp:	174

M. Sc (Information Technology)		Semester – IV	
Course Name: Blockchain		**Course Code: PSIT401**	
Periods per week (1 Period is 60 minutes)		4	
Credits		4	
		Hours	**Marks**
Evaluation System	**Theory Examination**	2½	60
	Internal	--	40

Course Objectives:

- To provide conceptual understanding of the function of Blockchain as a method of securing distributed ledgers, how consensus on their contents is achieved, and the new applications that they enable.
- To cover the technological underpinnings of blockchain operations as distributed data structures and decision-making systems, their functionality and different architecture types.
- To provide a critical evaluation of existing "smart contract" capabilities and platforms, and examine their future directions, opportunities, risks and challenges.

Unit	Details	Lectures	Outcome
I	**Blockchain:** Introduction, History, Centralised versus Decentralised systems, Layers of blockchain, Importance of blockchain, Blockchain uses and use cases. **Working of Blockchain:** Blockchain foundation, Cryptography, Game Theory, Computer Science Engineering, Properties of blockchain solutions, blockchain transactions, distributed consensus mechanisms, Blockchain mechanisms, Scaling blockchain **Working of Bitcoin:** Money, Bitcoin, Bitcoin blockchain, bitcoin network, bitcoin scripts, Full Nodes and SVPs, Bitcoin wallets.	**12**	**CO1**
II	**Ethereum:** three parts of blockchain, Ether as currency and commodity, Building trustless systems, Smart contracts, Ethereum Virtual Machine, The Mist browser, Wallets as a Computing Metaphor, The Bank Teller Metaphor, Breaking with Banking History, How Encryption Leads to Trust, System Requirements, Using Parity with Geth, Anonymity in Cryptocurrency, Central Bank Network, Virtual Machines, EVM Applications, State Machines, Guts of the EVM, Blocks, Mining's Place in the State Transition Function, Renting Time on the EVM, Gas, Working with Gas, Accounts, Transactions, and Messages, Transactions and Messages, Estimating Gas Fees for Operations, Opcodes in the EVM. **Solidity Programming:** Introduction, Global Banking Made Real, Complementary Currency, Programming the EVM, Design Rationale, Importance of Formal Proofs, Automated Proofs, Testing, Formatting Solidity Files, Reading Code, Statements and Expressions in Solidity, Value Types, Global Special Variables, Units, and Functions,	**12**	**CO2**

III	**Hyperledger:** Overview, Fabric, composer, installing hyperledger fabric and composer, deploying, running the network, error troubleshooting. **Smart Contracts and Tokens:** EVM as Back End, Assets Backed by Anything, Cryptocurrency Is a Measure of Time, Function of Collectibles in Human Systems, Platforms for High-Value Digital Collectibles, Tokens as Category of Smart Contract, Creating a Token, Deploying the Contract, Playing with Contracts.	**12**	**CO3**
IV	**Mining Ether:** Why? Ether's Source, Defining Mining, Difficulty, Self-Regulation, and the Race for Profit, How Proof of Work Helps Regulate Block Time, DAG and Nonce, Faster Blocks, Stale Blocks, Difficulties, Ancestry of Blocks and Transactions, Ethereum and Bitcoin, Forking, Mining, Geth on Windows, Executing Commands in the EVM via the Geth Console, Launching Geth with Flags, Mining on the Testnet, GPU Mining Rigs, Mining on a Pool with Multiple GPUs. **Cryptoecnomics:** Introduction, Usefulness of cryptoeconomics, Speed of blocks, Ether Issuance scheme, Common Attack Scenarios.	**12**	**CO4**
V	**Blockchain Application Development:** Decentralized Applications, Blockchain Application Development, Interacting with the Bitcoin Blockchain, Interacting Programmatically with Ethereum—Sending Transactions, Creating a Smart Contract, Executing Smart Contract Functions, Public vs. Private Blockchains, Decentralized Application Architecture, **Building an Ethereum DApp:** The DApp, Setting Up a Private Ethereum Network, Creating the Smart Contract, Deploying the Smart Contract, Client Application, **DApp deployment:** Seven Ways to Think About Smart Contracts, Dapp Contract Data Models, EVM back-end and front-end communication, JSON-RPC, Web 3, JavaScript API, Using Meteor with the EVM, Executing Contracts in the Console, Recommendations for Prototyping, Third-Party Deployment Libraries, **Creating Private Chains.**	**12**	**CO5**

Books and References:

Sr. No.	Title	Author/s	Publisher	Edition	Year
1.	Beginning Blockchain A Beginner's Guide to Building Blockchain Solutions	Bikramaditya Singhal, Gautam Dhameja, Priyansu Sekhar Panda	Apress		2018
2.	Introducing Ethereum and Solidity	Chris Dannen	Apress		2017
3.	The Blockchain Developer	Elad Elrom	Apress		2019
4.	Mastering Ethereum	Andreas M. Antonopoulos Dr. Gavin Wood	O'Reilly	First	2018
5.	Blockchain Enabled Applications	Vikram Dhillon David Metcalf Max Hooper	Apress		2017

Unit I

1

INTRODUCTION TO BLOCKCHAIN

Unit Structure

1.0 Objectives
1.1 Introduction to Blockchain
1.2 Evolution of Blockchain
    1.2.1 TCP/IP
    1.2.2. World Wide Web
    1.2.3 Banking
1.3 What is Blockchain?
    1.3.1 Blockchain
    1.3.2 Centralized vs. Decentralized Systems
        1.3.2.1Centralized Systems
        1.3.2.2 Decentralized Systems
1.4 Layers of Blockchain
1.5 Why is Blockchain Important?
1.6 Blockchain Uses and Use Cases
1.7 Summary
1.8 Questions
1.9 References

1.0 OBJECTIVES:

- Introduction to Blockchain
- Evolution of Blockchain
- What is Blockchain?
- Layers of a Blockchain
- Importance of Blockchain
- Blockchain Uses
- Blockchain Use cases

1.1 INTRODUCTION TO BLOCKCHAIN

Blockchain is the new phenomenon to sweep the world of technology and moreover financial transactions. Many of us are drawn to it because of cryptocurrency. However, there is more to blockchain than cryptocurrency. Blockchain is a technology that can be used for carrying

out financial transactions, creating applications as well as a database for your application. Blockchain has introduced a lot many new concepts as well as reused the existing ones in such an exemplary way that we are stunned by the sheer simplicity of the technology and awed by the possible scope of its applications.

Blockchain technology has a lot of advantages like lowering the costs related to financial transactions, elimination of third parties, speed of transactions, reducing the risk of fraud and the flexibility to involve and be a part of the process.

1.2 EVOLUTION OF BLOCKCHAIN

1.2.1 TCP/IP

TCP/IP was the biggest invention in the field of Technology. It changed the way people perceived and used technology. The circuit switching technology was used prior to TCP/IP for the exchange of data. Circuit switching technology was based on the concept of resource allocation for transmission.

Circuit switching technology characteristics:

- Circuit switching involves resource allocation prior to transmission.
- It consists of three phases: connection setup, data transfer, and connection teardown.
- In circuit switching, the resources are reserved during the setup phase; these resources are reserved for the entire duration of data transfer until the teardown phase.
- Data is transferred as a continuous flow.

TCP/IP introduced the concept of Packet switching. In packet switching, there is no need to allocate resources prior to transmission. Data is sent across the network without dedicated resources yet it is delivered in order and without any errors. The TCP/IP protocol suite ensured that data was delivered to the correct destination in a correct orderly format.

Packet switching technology characteristics:

- In a packet-switching, there is no resource reservation at the start of transmission; resources are allocated on demand.
- Data is not a continuous flow but it is divided into packets of fixed or variable size.
- Each packet is treated independently of all others.

As can be seen that the devices work in tandem with each other for delivery without any central controller, we can say that TCP/IP works in a peer-to-peer or decentralized manner.

1.2.2. World Wide Web

The Invention of the World Wide Web (WWW) in the early 1990s was a game-changer. It was decentralized and used the TCP/IP protocol

for communication. It was a humongous success and the current times are proof as to how it hasn't faded into oblivion. WWW or the Internet has changed the way we work or live our lives. Many applications are built on top of the Internet. However, it has somehow turned into a centralized system.

The usability of the Internet and the various applications that it supports has led it to become a norm. The Internet is currently being used for entertainment, education, e-commerce, banking, to carry out financial transactions, and various other purposes. People have started trusting the internet for carrying out various transactions online.

1.2.3 Banking

The most ancient form of banking is called the Barter System where goods were exchanged for other goods or services. For example, a farmer producing rice would exchange it with another farmer for wheat or other staples. However, there were a lot of loopholes in this system due to lack of trust or the occurrences of fraud and exploitation.

The introduction of fiat currencies solved this problem. Fiat currency is a standard currency introduced by a trusted authority and accepted by all. For example, Rupee is the currency in India and dollars in the USA. Traditionally gold and silver coins were used as fiat currency.

A Transaction can be defined as the currency which is being transferred from one entity to another. For example, you are paying for a book you purchased. The book costs □500. So, you are supposed to give a □ 500 note to the bookseller. Once you handover the note to him, the transaction is settled. In this scenario, there are two parties carrying out a transaction without a need for an intermediary. Imagine the same transaction is being done when you cannot be physically present at the book shop. In this case, you will need a trusted third party for settling your transaction. The banking system is the most widely used third party for settling financial transactions. In the banking scenario, □ 500 will be deducted from your account and deposited in the Booksellers account.

Banking evolved as an intermediary for settling transactions and various other services related to it. The combination of the Internet and the banking system allows us to settle transactions all over the world. So, making a monetary transaction to any part of the world is as easy as doing it locally. However, offshore transactions take more time to settle and are a bit expensive because of the additional fee involved as compared to local transactions.

As can be seen, the role of a trusted third party is extremely important in settling a transaction. Hence, there are many types of intermediaries that exist like banks, escrow services, clearinghouses, and others for settling a transaction.

Blockchain is a solution that eliminates this need for a trusted third party. It provides a way to transact without an intermediary in a secure way by using various cryptographic techniques.

A research paper by Satoshi Nakomoto, which is a pseudonymous name, introduced the concept of "Bitcoin - A peer-to-peer electronic cash system". In this paper, he describes a solution to replace the traditional banking system with a decentralized, peer-to-peer, trustless system for exchanging currency.

Banks are currently the centralized authority for carrying out financial transactions, settling them, maintaining records of transactions, enforcing the security of transactions, and safeguarding the assets of an entity. The entire commerce platform is dependent on the banks for settling their transactions. The bank is the trusted third party for carrying out a financial transaction so that it can intervene in case of a dispute. However, the need for an intermediary leads to an increase in the cost as well as time taken to settle a transaction.

However, with the enormous change in how financial transactions are being done online, Satoshi Nakomoto invented the Bitcoin currency for settling transactions online without using the bank as an intermediary. Bitcoin is a very widely known example of Blockchain Technology implementation.

However, there is more to Blockchain than being a cryptocurrency. Bitcoin is an implementation of Blockchain used for financial purposes.

Barter System => Gold/Silver Coins => Paper Money=> Credit Card => Digital Wallets =>CryptoCurrency

1.3 WHAT IS BLOCKCHAIN?

The Internet has revolutionized many aspects of life, society, and business. However, not much has changed in the past couple of decades with respect to transactions in an untrusted environment. Blockchain is the component that makes the internet more open, more accessible, and more reliable.

Blockchain is a collection of transactions. It is a system that records transactions between two communicating nodes. It works in a peer-to-peer fashion. There is no need for a trusted intermediary in blockchain. The system is designed in such a manner that all transactions are verified by everyone, eliminating the need for a trusted intermediary. Transactions can be monetary or non-monetary.

Let's take a situation where A needs to pay 50$ to B. There are two ways to carry out this transaction.
1. A can pay it in person to B
A=>50$ =>B
2. A has to make a transfer of 50$ via a trusted intermediary such as a bank or escrow or any other trusted third party.
A=>50$ => B

The first transaction is an example of a peer to peer transaction without the need of a trusted third party. However, the challenge is to perform such transactions when both the parties are geographically far placed.

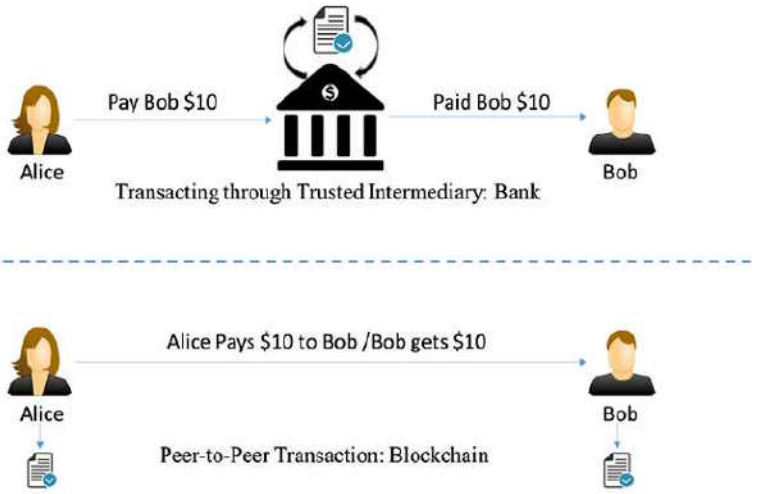


Figure 1-1. Transaction through an intermediary vs. peer-to-peer transaction
#imgref: Beginning Blockchain- Singhal, Dhameja, Panda

Some transactions take weeks to settle. A typical example is a stock transaction which happens in a second but takes weeks to settle. This delay is not desirable in today's fast moving world.

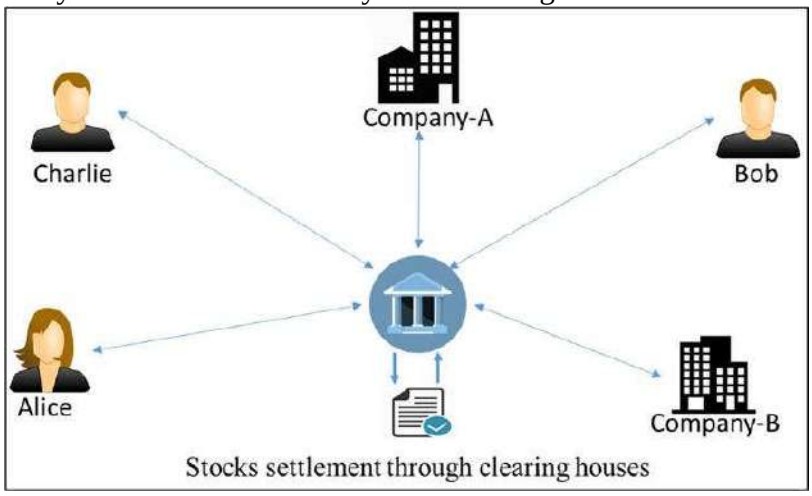


Figure 1-2. Stocks trading through an intermediary clearing house
#imgref: Beginning Blockchain-Singhal, Dhameja, Panda

Stock trading needs to be carried out by eliminating the need for a broker or a clearing house. It needs to be peer to peer and instantaneous.

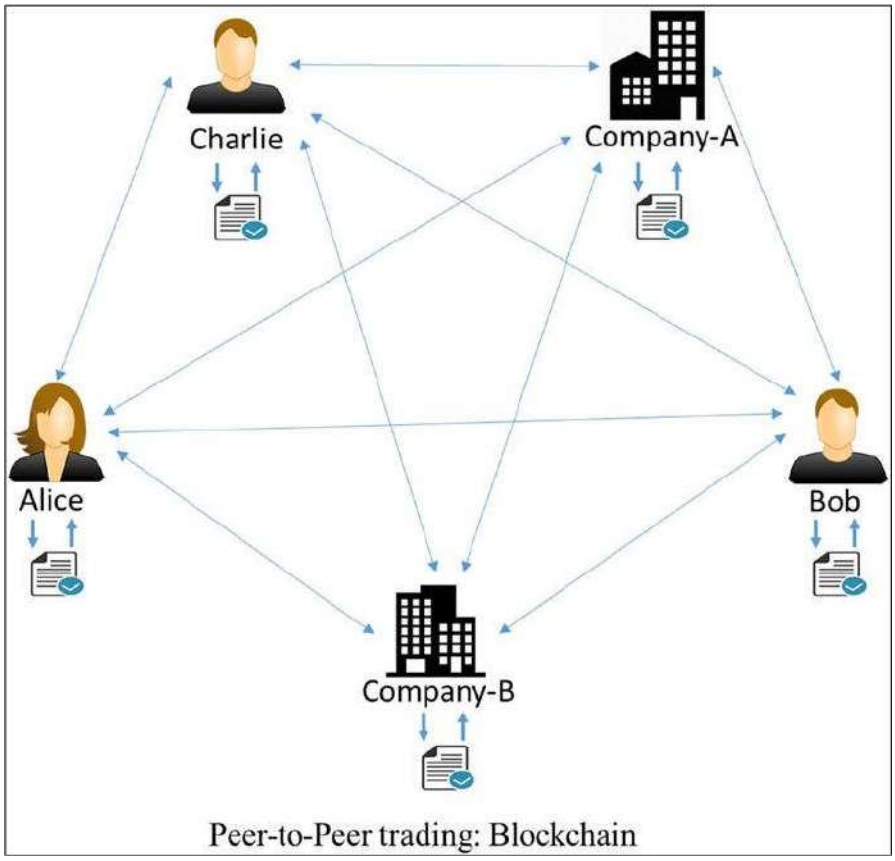


Figure 1-3. Peer-to-peer stock trading
#imgref:Beginning Blockchain-Singhal, Dhameja, Panda

Transactions in Blockchain are similar to the financial transaction carried out with fiat money. A currency note handed over to someone does not belong to the payer. It belongs to the payee. Blockchain transactions are somewhat similar to this.

1.3.1 Blockchain

Blockchain is a peer-to-peer system of transacting values with no trusted third parties in between.The sender and receiver can transact without the need of a third party.

It is a shared, decentralized, and open ledger of transactions. Ledger is a record of transactions. It can be considered as a transaction log, This ledger database or transaction log is replicated across a large number of nodes.

New entries to this ledger database can only be appended. It cannot be changed or altered. Every entry is permanent.

All databases are synchronized across nodes i.e. every new entry is reflected on all copies of the databases hosted on different nodes.

There is no need for any intermediary or trusted third parties to verify, secure, and settle the transactions.

Blockchain is not disruptive and operates above the Internet. It does not affect the other applications on the Internet.

The most important characteristic of Blockchain being that it is open source and decentralized.

A typical blockchain may look as shown in Figure 1-4.

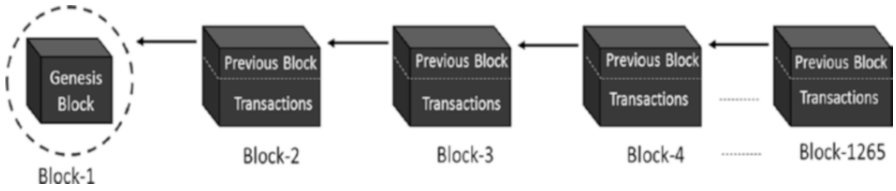


Figure 1-4. The blockchain data structure
#imgref:Beginning Blockchain-Singhal, Dhameja, Panda

A Blockchain consists of multiple blocks of transactions interlinked together. The first block in a blockchain is called the "Genesis" block. Every block contains a block header and body. The body contains the list of transactions and details related to that transaction like the amount, address, etc.

The header contains information about the previous block hence creating a link between the current block and previous block.

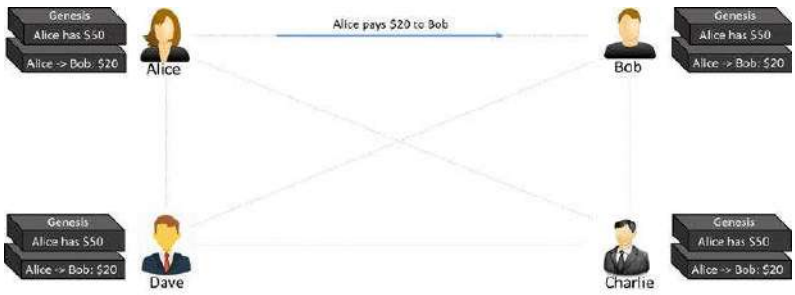
Every node on the blockchain network has an identical copy of the blockchain.

To understand blockchain better, let's look at an example where three candidates are transacting with each other.

Step-1:
Alice has $50 with her. This is the first transaction called genesis. All nodes are updated with the genesis.

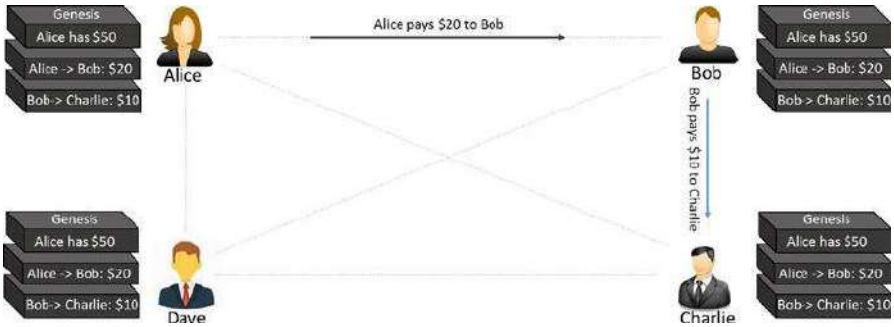
Figure 1-5. The genesis block
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Step-2:
Alice pays $20 to Bob thus making a new transaction. This transaction is updated on all the nodes of the blockchain.



#imgref:Beginning Blockchain-Singhal, Dhameja, Panda

Step-3:
Bob pays $10 to Charlie which is further reflected on all nodes of the Blockchain.



#imgref:Beginning Blockchain-Singhal, Dhameja, Panda

As can be seen, blockchain is a list of transactions. All new transactions are appended to the list. A transaction entry cannot be modified. It is immutable. All transactions are validated by the remaining nodes in a blockchain.

1.3.2 Centralized vs. Decentralized Systems

Blockchain is Decentralized. A distributed system is a system in which the computation is performed on multiple nodes. A distributed system can be centralized or decentralized.

A centralized distributed system has a central controller which breaks down a task and distributes it amongst the other nodes for processing.Hadoop is an example of centralized distributed architecture.

A decentralized distributed system does not have any master node yet the processing is done by multiple nodes.

Blockchain is an example of decentralized distributed architecture.

Figure 1-8 is a pictorial representation of how a centralized distributed system may look.

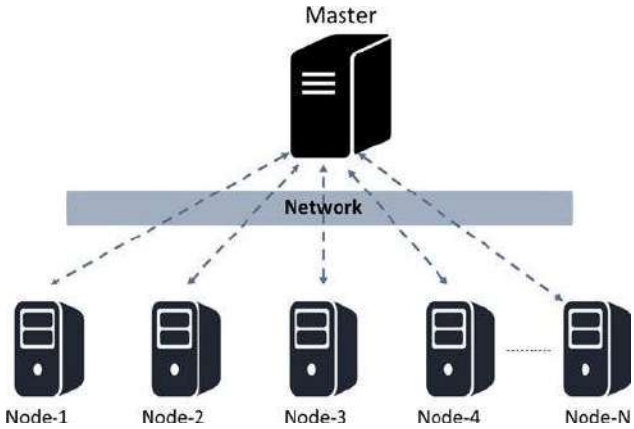


Figure 1-8. A distributed system with centralized control
#imgref: Beginning Blockchain-Singhal, Dhameja, Panda

Centralization or Decentralization can be perceived as being Technical, Political or Logical.

Technical Architecture: This perspective deals with the physical placement of devices in a system. It deals with the system as a whole, the number of devices, the robustness of the system based on the participating nodes, etc.

Political perspective: This perspective is about the control/decision making of the system. When a single node or multiple nodes are controlling the system, it is centralized. However, if all nodes are equally responsible for decision making then it is decentralized.

Logical perspective: The logical perspective deals with how the system performs if it is split horizontally or vertically. If a system is performing similarly before and after a split it is decentralized. It is independent of the other nodes in the system.

An organization has one head office(architecturally centralized), governed by a CEO(politically centralized) and the organization cannot be split in 2 equal halves(logically centralized).

Our language of communication is decentralized from every perspective—architecturally, politically, as well as logically. For two people to communicate with each other, in general, their language is neither politically influenced nor logically dependent on the language of communication of other people.

The torrent systems such as Bit Torrent are also decentralized from every perspective. Any node can be a provider or a consumer, so even if you cut the system into halves, it still sustains.

The Content Delivery Network on the other hand is architecturally decentralized, logically also decentralized, but is politically centralized because it is owned by corporations.

Blockchain is made up of multiple nodes(architectural decentralization) and each node is equal(political decentralization). However, there is a commonly agreed mechanism to operate (logical centralization).

1.3.2.1Centralized Systems

A centralized system has a centralized control with all administrative authority. Such systems are easy to design, maintain, impose trust, and govern. However, they suffer from many inherent limitations:

- The system is not robust as there is a single point of failure.
- Security of system based on the security of the controller so it is vulnerable to attack
- Centralization of power leads to monopoly

A typical centralized system may appear as shown in Figure 1-9.

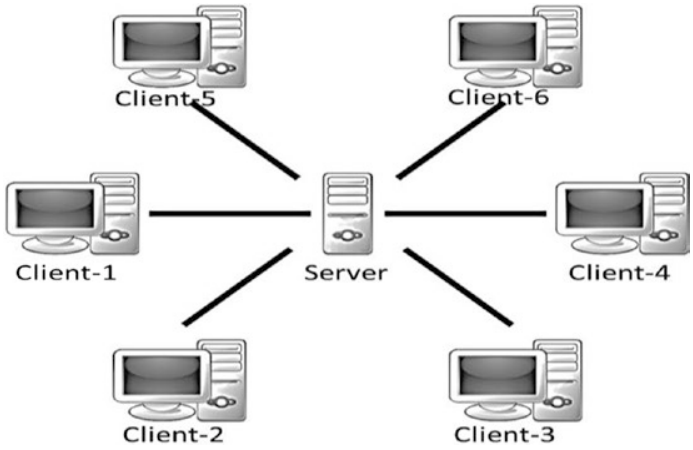


Figure 1-9. A centralized system
#imgref:Beginning Blockchain-Singhal, Dhameja, Panda

1.3.2.2 Decentralized Systems

All systems have the same authority in a decentralized system. Every node is independent of the other node and works in a commonly agreed upon protocol.

Such systems are difficult to design, maintain, govern, or impose trust. However, they offer the following advantages:

- They are extremely robust because they do not have a central point of failure, so more stable and fault tolerant
- They are more secure because there is no central point to easily attack.

- All systems are treated equally so no monopoly. They are democratic in nature.

A typical decentralized system may appear as shown in Figure 1-10.

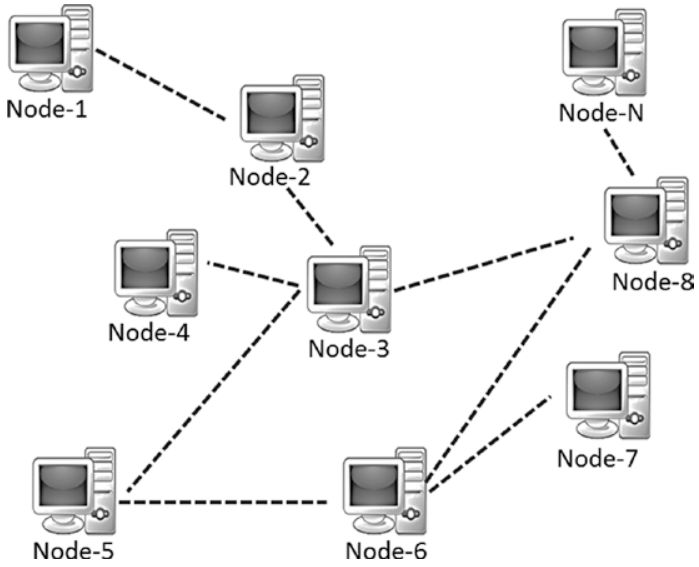


Figure 1-10. A decentralized system
#imgref:Beginning Blockchain-Singhal, Dhameja, Panda

Blockchain is an example of a decentralized distributed system. All nodes participate in the system and are responsible for the entire blockchain.

A typical decentralized and distributed system, which is effectively a peer-to-peer system, may appear as shown in Figure 1-11.

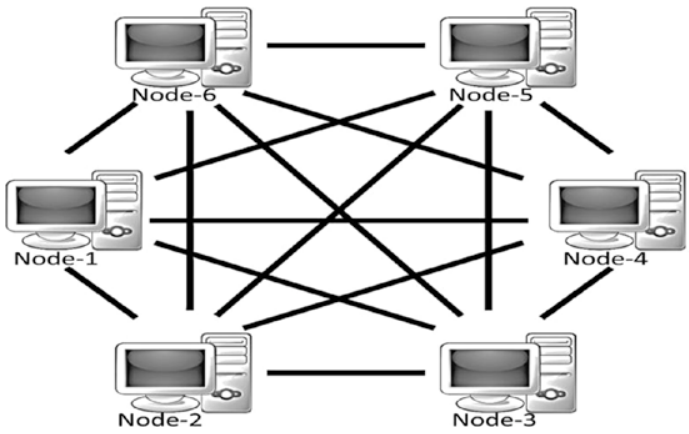


Figure 1-11. A decentralized and peer-to-peer system
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

1.4 LAYERS OF BLOCKCHAIN

The blockchain technology can be visualized as layers similar to the TCP/IP protocol suite. However, it is not standardized yet and provides a basic understanding of the working of a blockchain. These layers are present on all the nodes in a blockchain.

A high-level, layered representation of blockchain consists of 5 layers as shown in Figure 1-12.

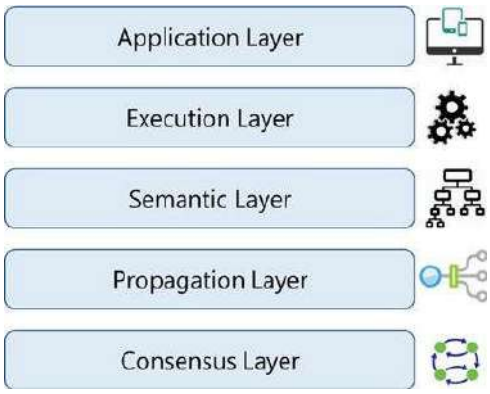


Figure 1-12. Various layers of blockchain
#imgref: Beginning Blockchain-Singhal, Dhameja, Panda

Application Layer

This layer deals with applications which are built on the Blockchain technology as well as applications which enhance the blockchain technology and are part of the various flavors of blockchain. There are various flavors of blockchain in the market and each has its own implementation of blockchain. Some of the flavors have developed applications which are integrated into the technology and become a part of it.

Some applications in blockchain follow the traditional development model and include programming constructs, scripting, APIs, development frameworks, etc.

Other types of applications use blockchain as a backend. These applications might be hosted on some web servers and that might require web application development, server-side programming, and APIs, etc.

Since blockchain is decentralized, an ideal blockchain application will not have a client–server model.

Some applications use off-chain networks so that the resource intensive network operations do not affect the blockchain network.

Execution Layer

The executions of instructions as ordered by the Application Layer take place in the Execution Layer. The instructions could be a simple instruction or a set of multiple instructions in the form of a smart contract.

In either case, a program or a script is executed for the correct execution of the transaction. All the nodes in a block chain network execute the programs/scripts independently. The output on the same set of inputs and conditions always produces the same output on all the nodes.

Bitcoins scripts are simple and have only a few sets of instructions. Ethereum and Hyperledger, use smart contracts that are made up of multiple instructions and can be complex executions.

Ethereum's code is written in solidity which gets compiled to Bytecode or Machine Code that gets executed on its own Ethereum Virtual Machine.

Hyperledger code, called as chaincode smart contracts, supports running of compiled machine codes inside docker images, and supports multiple high-level languages such as Java and Go.

Semantic Layer

The instructions executed in the previous layer are validated in the Semantic layer. It is a logical layer because there is an orderliness in the transactions and blocks. Semantic layer is responsible for checking if a transaction is legitimate or not. It checks for the authorization of a transaction. In case of bitcoin, an amount is paid by the amount received in a previous transaction. This layer checks if the previous transaction is legitimate and has received the amount they are spending. Ethereum has the system of Accounts like banks. It means that the account of the one making the transaction and that of the one receiving it both get updated. The rules of the system, the data models and structures are defined in this layer.

A block usually contains a bunch of transactions and some smart contracts. The data structures such as the Merkle tree are defined in this layer with the Merkle root in the block header to maintain a relation between the block headers and the set of transactions in a block. Also, the data models, storage modes, in- memory/disk based processing, etc. can be defined in this logical layer.

The semantic layer also defines how the blocks are linked with each other. Every block in a blockchain contains the hash of the previous block, all the way to the genesis block. Though the final state of the blockchain is achieved by the contributions from all the layers, the linking of blocks with each other needs to be defined in this layer.

Propagation Layer

This layer is responsible for information dissemination. It deals with the communication between nodes in the Blockchain. The previous layers are all about transaction processing at independent block level. The Propagation Layer is the peer-to-peer communication layer that allows the nodes to discover each other, and talk and sync with each other with respect to the current state of the network.

When a transaction is made it gets broadcast to the entire network. Similarly, when a node wants to propose a valid block, it gets immediately propagated to the entire network so that other nodes could build on it, considering it as the latest block.

So, transaction/block propagation in the network is defined in this layer, which ensures stability of the whole network. By design, most of the blockchains are designed such that they forward a transaction/block immediately to all the nodes they are directly connected to, when they get to know of a new transaction/block.

In the asynchronous Internet network, there are often latency issues for transaction or block propagation. The time taken for propagation is dependent on the capacity of the nodes, network bandwidth, and a few more factors.

Consensus Layer

The Consensus Layer is usually the base layer for most of the blockchain systems. The primary purpose of this layer is to get all the nodes to agree on one consistent state of the ledger. There are different ways of achieving consensus among the nodes. This layer is responsible for the Safety and security of the blockchain.

For a public blockchain to be self-sustainable, an incentives mechanism helps in keeping the network alive as well enforces consensus.In Bitcoin or Ethereum, the incentive technique is called “mining”. Bitcoin and Ethereum use the Proof of Work (PoW) consensus mechanism.

A node is randomly selected that can propose a block. Once a block is proposed, it is propagated to all the nodes. All nodes check to see if it is a valid block with all legitimate transactions and if the PoW puzzle was solved properly. If true, then the block is added to their own copy of the blockchain.There are different variants of consensus protocols such as Proof of Stake (PoS), delegated PoS (dPoS), Practical Byzantine Fault Tolerance (PBFT), etc.

1.5 WHY IS BLOCKCHAIN IMPORTANT?

Blockchain, being a decentralized peer-to-peer system, has some inherent benefits and complexities. There are also scenarios where implementing blockchain for the existing system makes it more robust, transparent, and secured. However, blockchain should be used sparingly with respect to the system. It is not feasible for all kinds of use cases and works efficiently in particular scenarios only.

Limitations of Centralized Systems

The software development trend currently has an inclination towards client server systems. It uses a centralized architecture. Also, all of our systems involve a trusted third party for carrying out transactions.

The current system involves working with a trusted environment or in case of an untrusted environment involves a trusted third party.

E-commerce platforms are the best example of involving a trusted third party in an untrusted environment. The buyers and sellers do not know each other. However, the e-commerce platform is the trusted third party involved. Hence, people can transact with each other via this platform. The buyer and seller both trust the platform and hence can transact in a safe and secure way.

Blockchain eliminates the need for a trusted third party while carrying out transactions in an untrusted environment. The building block of blockchain is trust amongst the transacting nodes.

Disadvantages of a conventional centralized system:

- Trust issues
- Security issue
- Privacy issue—data sale privacy is being undermined
- Cost and time factor for transactions

Advantages of decentralized systems:

- Elimination of intermediaries
- Easier and genuine verification of transactions
- Increased security with lower cost
- Greater transparency
- Decentralized and immutable

Blockchain Adoption

Blockchain became popular with the introduction of Bitcoin in 2009. However, companies have developed different implementations of Blockchain like Ethereum, Hyperledger, etc. Microsoft and IBM came up with SaaS (Software as a Service) offerings on their Azure and Bluemix cloud platforms, respectively.

Many companies started integrating blockchain in their problem solving solutions. Blockchain has a tremendous impact on the financial services market. A lot of industries are moving towards using blockchain as their technological solution. However, the biggest challenge is designing the right kind of Blockchain. Blockchain can be designed as public or private. Using blockchain for building non-financial transactions without disrupting the existing system can be a challenge. Some of the design challenges are whether to convert the entire existing system to Blockchain or to use blockchain as a backend or to use partial implementation of blockchain.

1.6 BLOCKCHAIN USES AND USE CASES

In this section, we will look at some of the implementation of blockchains.

True Sharing Economy

Blockchain can be used to create a true sharing economy where transactions could be carried out in a peer to peer manner like Uber, AirBNB,etc.

Self-Sovereign Digital Identity

Self-Sovereign Digital Identityis an implementation where the citizens can own their identity and monetize their own data.

Asset Registration

Any type of property or asset, whether physical or digital, such as laptops, mobile phones, diamonds, automobiles, real estate, e-registrations, digital files, etc. can be registered on blockchain. Blockchain can keep a track of the transfer of these assets from one entity to another, maintain the transaction log, and check validity or ownerships. It can also be used for notary services, proof of existence, tailored insurance schemes, etc.

Financial Use Cases

Cross-border payments, share trading, loyalty and rewards system, Know YourCustomer (KYC) among banks, ICO,etc., can be implemented using Blockchain.

The Wisdom of Crowds

The Wisdom of Crowds can be used to take the lead and shape businesses, economies, and various other national phenomena by using collective wisdom. Financial and economic forecasts based on the wisdom of crowds, decentralized prediction markets, decentralized voting, as well as stocks trading can be possible on blockchain.

Music Royalties

Blockchain can be used to maintain a public ledger of music rights ownership information as well as authorised distribution of media content. This will help us in determining music royalties in internet enabled music streaming services.

IoT

There are a plethora of IoT devices and from different vendors. Hence, it becomes difficult for all of them to have a centralized system to control devices.

Blockchain can be used to build a decentralized peer- to-peer system for the IoT devices to communicate with each other. ADEPT (Autonomous Decentralized Peer-To-Peer Telemetry) is a joint initiative from IBM and Samsung that has developed a platform that uses elements of the Bitcoin’s underlying design to build a distributed network of devices—a decentralized

IOT.

ADEPT uses three protocols: Bit Torrent for file sharing, Ethereum for smart contracts, and Tele Hash for peer-to-peer messaging on the

platform. The IOTA foundation is another such initiative.

Government Agencies

Land registration, vehicle registration, and management, e-Voting, Supply chains, etc. are some of the use cases where blockchain can be used.

1.7 SUMMARY

This chapter is a brief overview of the evolution of blockchain, the history and the benefits of blockchain. It concentrated on the difference between the centralized and decentralized systems. Also, it listed out the various use cases of Blockchain.

Blockchain is a solution to peer to peer transactions eliminating the need for a trusted third party.

Blockchain deals with three key areas: Control, Trust, and Value.

Control: Blockchain enabled distribution of the control by making the system decentralized.

Trust: Blockchain is an immutable, tamper-resistant ledger. It gives a single, shared source of truth to all nodes, making the system trustless. What it means is that trust is no longer needed to transact with any unknown person or entity and is inherent by design.

Value: Blockchain enables exchange of value in any form. One can issue and transfer assets without central entities or intermediaries.

1.8 QUESTIONS:

1. What is a blockchain?
2. What is the difference between a centralized and decentralized system?
3. Explain the different layers of Blockchain.

1.9 REFERENCES:

© Bikramaditya Singhal, Gautam Dhameja, Priyansu Sekhar Panda 201831

B. Singhal et al., Beginning Blockchain, https://doi.org/10.1007/978-1-4842-3444-0_2

2

BLOCKCHAIN FOUNDATION

Unit Structure:

2.0 Objective

2.1 Laying the Blockchain Foundation

2.2 Cryptography

- 2.2.1. Symmetric Key Cryptography
  - 2.2.1.1 Kerckhoff's Principle and XOR Function
  - 2.2.1.2Stream Ciphers vs. Block Cipher
  - 2.2.1.3 One-Time Pad
  - 2.2.1.4 Data Encryption Standard
  - 2.2.1.5 Advanced Encryption Standard
  - 2.2.1.6 Challenges in Symmetric Key Cryptography

2.3 Cryptographic Hash Functions

- 2.3.1 A Heads-up on Different Hash Functions
  - 2.3.1.1 SHA-2
  - 2.3.1.2 SHA-256 and SHA-512
  - 2.3.1.3 RIPEMD
  - 2.3.1.4 SHA-3
  - 2.3.1.5 Applications of Hash Functions
  - 2.3.1.6 Code Examples of Hash Functions
- 2.3.2 MAC and HMAC
- 2.3.2.1 MAC Strategies

2.4 Asymmetric Key Cryptography

- 2.4.1 RSA
- 2.4.2 Digital Signature Algorithm
- 2.4.3 Elliptic Curve Cryptography
- 2.4.4 Elliptic Curve Digital Signature Algorithm
- 2.4.5 Code Examples of Asymmetric Key Cryptography
- 2.4.6 The ECDSA Algorithm Code

2.5 Diffie-Hellman Key Exchange

- 2.5.1 Code for DH:

2.6 Symmetric vs. Asymmetric Key Cryptography

2.7 Game Theory

- 2.7.1 Nash Equilibrium
- 2.7.2 Prisoner's Dilemma
- 2.7.3 Byzantine Generals' Problem
- 2.7.4 Zero-Sum Games

2.7.5 Why to Study Game Theory
2.8 Computer Science Engineering
2.8.1 The Blockchain
2.8.2 Merkle Trees
2.8.3 Code Snippet for Merkletree
2.9 Putting It All Together
2.10 Properties of Blockchain Solutions
2.11 Blockchain Transactions
2.12 Distributed Consensus Mechanisms
2.13 Proof of Work
2.14 Proof of Stake
2.15 Practical Byzantine Fault Tolerance algorithm (PBFT)
2.16 Blockchain Applications
2.17 Scaling Blockchain
2.17.1 Off-Chain Computation
2.17.2 Sharding Blockchain State
2.18 Summary
2.19 Questions
2.20 References

2.0 OBJECTIVES:

- Understanding the Components of Blockchain
- Cryptography
- Symmetric key Vs Asymmetric key
- Game Theory
- Computer Science
- Blockchain Scaling

2.1 LAYING THE BLOCKCHAIN FOUNDATION

Blockchain is a combination of the concepts from cryptography, game theory, and computer science engineering, as shown below:

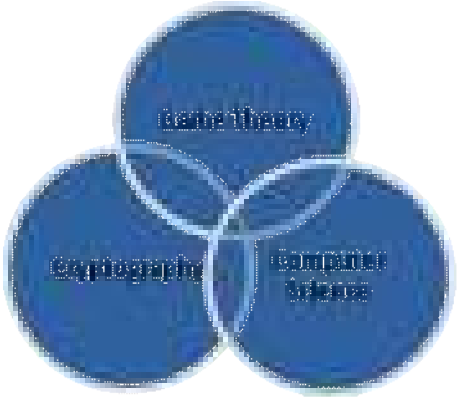


Figure 2-1. Blockchain at its core

Traditional centralized systems allowed only one entity to maintain the history of transactions or modifications to ensure concurrency in a system. However, the issue with a single point of control was the complete trust on that single entity. If all the systems were given full control to modify the transactions then the problems would definitely increase because we never know what those systems might do. So, the trust issue now multiplies to the number of nodes with full control.

Blockchain uses cryptography, game theory and computer science concepts to solve the trust issues plaguing a trustless system with multiple entities.

Cryptography can be used to check the validity of the user performing a particular transaction. However, it does not prevent a user from spending the same amount twice called as the double spend attack. The only way to thwart a double spend is to make all the nodes aware of all the transactions.

This leads to another problem as to how do everyone agree on a common database state? How does the system prevent anyone from injecting a fraudulent database? Such problems can be solved using game theory. There is a popular case study in Game theory called the ByzantineGenerals' Problem. It is synonymous to the problems we face while making several nodes trust each other and to ensure the system is fault tolerant. Game theory provides different approaches to determine the behavior of a system and solutions to problems in a trustless system. Game theory believes that participants do not care about moral values but as per the advantage that they get while playing a game. It does not matter if a node is honest or cheat, malicious or ethical or has any other traits. Its all about winning a game. Blockchain uses the concepts of game theory for ensuring stability in a trustless system. Cryptography and game theory help in validating transactions and carrying out transactions in a trustless system. However, the concepts in computer science tie these two together by providing data structures and network communication techniques.

Computer science techniques incorporate cryptography and game theory concepts into an application and enables decentralized as well as distributed computing amongst the nodes using data structure and network communication components.

2.2 CRYPTOGRAPHY

Cryptography is one of the most important components of blockchain. It plays an important part in validating users and securing transactions. Cryptography consists of many mathematical techniques that can be used to solve various issues in Blockchain.

Cryptography has been used from a long time ago for hiding information i.e. keeping the messages confidential. Cryptography has various applications likemaintaining the integrity of the message, verifying and validating a user,etc.

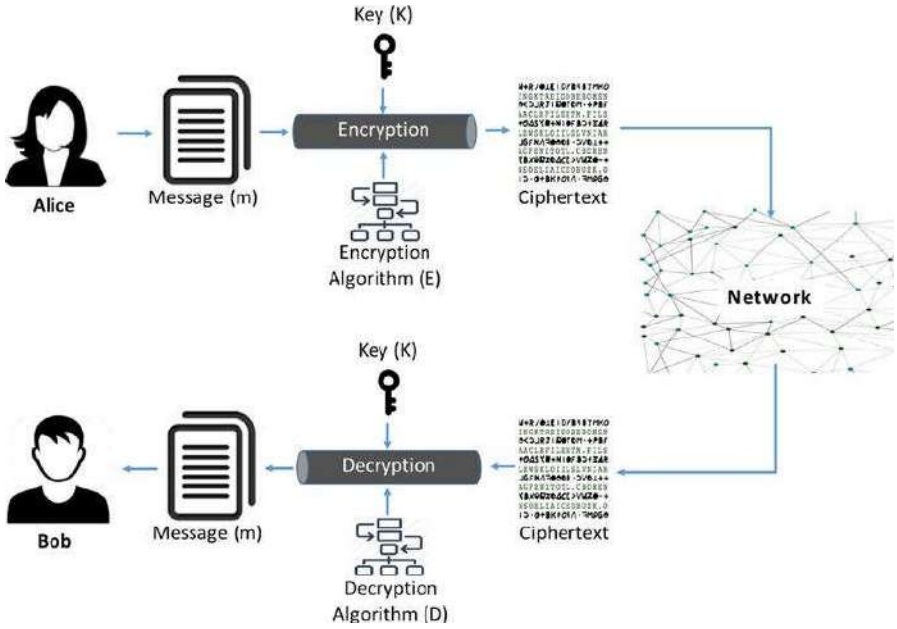


Figure 2-2. How Cryptography works in general
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Confidentiality: Confidentiality means that only the intended recipient can access the message.

Data Integrity: Data Integrity allows us to check if the data has been altered or modified during transmission.

Authentication: The sender of a message is verified and validated using authentication. It checks whether a user is who he/she claims to be.

Non-repudiation: It ensures that the sender cannot deny sending a message.

Plaintext: The actual information that needs to be sent is called plaintext.

Ciphertext: The information to be sent(plaintext) is converted to another form called ciphertext.

Encrypt: The process of converting plaintext to ciphertext is called Encryption or encoding. It usually consists of an algorithm and a key.

Decrypt: The process of recovering plaintext from ciphertext is called Decryption or Decoding. It usually consists of an algorithm and a key.

Alice wants to send a message (m) to Bob. If the message is sent as it is, then anyone can read the message. So, Alice encrypts the message using an encryption algorithm (E) and a secret key (k) to produce the ciphertext. The receiver, Bob, on receiving the cipher text, decrypts the message using an algorithm and the secret key(k) to read the message.

Anyone who wants to decrypt the message needs to know the algorithm and the secret key being used. The strength of the algorithm and the secrecy of the key are factors that influence the efficacy of a cryptosystem.

There are two types of cryptography systems: symmetric key and asymmetric key cryptography.

2.2.1 Symmetric Key Cryptography

Symmetric key cryptography is named so because it uses the same key for encryption and decryption. The sender and receiver agree on a Secret Key(k) before they start sharing confidential information. Since, the success of this system is based on the secrecy of the key, the key is kept extremely secure and private. Hence, this system is also called as private key encryption system. The secret key agreement is performed using an extremely secure channel whereas the information is exchanged using any insecure channel.

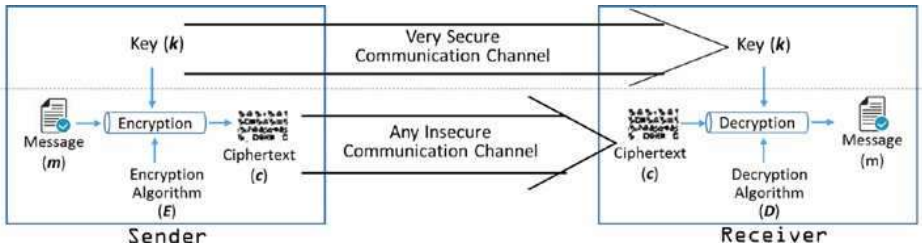


Figure 2-3. Symmetric Key Information Exchange
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

The process of Symmetric Key cryptography(Figure 2-2.) is:

Sender(Alice):

Encrypt the plaintext message m using encryption algorithm E and the previously agreed upon secret key k to generate the ciphertext c.

c = E(k, m)

The ciphertext c is sent to Bob.

Receiver(Bob):

Decrypt the ciphertext c using decryption algorithm D and the previously agreed upon secret key k to retrieve the plaintext m

m = D(k, c)

Symmetric key cryptography is used widely because it is fast and simple to implement. It is used in secure file transfer protocols such as HTTPS, SFTP, and WebDAVS.

There are two variants of symmetric key cryptography: stream ciphers and block ciphers.

2.2.1.1 Kerckhoff's Principle and XOR Function

Kerckhoff's principle states that a cryptosystem should be secured even if everything about the system is publicly known, except the key.

In general, the encryption algorithm E and decryption algorithm Dare public. However, the message cannot be intercepted because the key is known only by the communicating parties. Hence, the security of the keys is paramount in symmetric key cryptography.

The XOR, otherwise known as "Exclusive OR" and denoted by the symbol $\oplus$ is the basic building block for many encryption and decryption algorithms.

The truth table for XOR is:

A	**B**	A $\oplus$ B
0	0	0
1	0	1
0	1	1
1	1	0

Table 2-1. XOR Truth Table

Properties of XOR Function:
Associative: $A \oplus (B \oplus C) = (A \oplus B) \oplus C$
Commutative: $A \oplus B = B \oplus A$
Negation: $A \oplus 1 = \bar{A}$
Identity: $A \oplus A = 0$

The same XOR function is used for both encryption and decryption.
$m \oplus k = c$ and $c \oplus k = m$

XOR is the most basic form of encryption algorithm. It is very simple to get the original plaintext message just by XORing with the key, which is a shared secret and only known by the intended parties.

2.2.1.2Stream Ciphers vs. Block Cipher

Stream Ciphers

In a Stream cipher, encryption and decryption are performed on one symbol at a time.There is a stream of symbols (characters or bits) that is encrypted using a stream of keys.
P=(p1,p2,p3…)
K=(k1,k2,k3..)
C=E(p1,k1),E(p2,k2)…..

Usually a XOR operation is performed in a bit by bit manner to encrypt every bit of the plaintext to generate ciphertext. The key is generated by a pseudorandom keystream generator from a random seed value using digital shift registers. The key generation can be online or offline. However, synchronization of keys can be challenging. The security of the entire system is based on the unpredictability and security of the pseudo random generator which stands to be its biggest disadvantage. The pseudorandom number generator has been attacked many times in the past, which led to deprecation of stream ciphers.

Another disadvantage is that of low diffusion. All information in one bit of input text is contained in its corresponding one bit of ciphertext. It could have been more secure if the information of one bit was distributed across many bits in the ciphertext.

The most widely used stream cipher is RC4 (Rivest Cipher 4) for various protocols such as SSL, TLS, and Wi-Fi WEP/WPA etc. However, it suffers a lot of vulnerabilities and hence is not recommended. Examples of stream ciphers are one-time pad, RC4, FISH, SNOW, SEAL, A5/1, etc.

Block Ciphers

In a block cipher, the plaintext is divided into groups of symbols of fixed size called blocks. The encryption is performed on an entire block of plaintext at a time. The same key is used for encryption of all the blocks. The ciphertext is repeated if the blocks of plaintext are also repeated.

The block size is usually64 bits, 128 bits, and 256 bits called block length, and their resulting ciphertext blocks are also of the same block length.

Block cipher operation is performed in various modes. The simplest mode is the basic mode of encryption and decryption. However, the ciphertext is the same for the same blocks of plaintext. Hence, the other modes can be used which try to introduce randomness in the output. The various modes of operation are: Electronic Codebook (ECB), Cipher Block Chaining (CBC), Cipher Feedback (CFB), Output Feedback (OFB), and Counter (CTR).

Block ciphers are slower compared to stream ciphers. Error propagation is higher in block ciphers, error in one bit could corrupt the whole block. Block ciphers have the advantage of high diffusion, which means that every input plaintext bit is diffused across several ciphertext symbols. Examples of block ciphers are DES, 3DES, AES, etc.

2.2.1.3 One-Time Pad

This is the most popular symmetric stream cipher. The name one-time pad comes from an encryption method in which a large, non repeating set of keys is written on a pad. The sender encrypts the message one bit at a time using keys from the pad. Since, it is a stream cipher; the number of keys is equal to the number of plaintext symbols. For example,

if a sender must transmit a message of 300 characters, the sender would use300 key characters from the pad. The keys are destroyed and never reused. The receiver has the same key pad. On receiving the ciphertext , he decrypts it using the keys from the pad.

The one-time pad method has two problems: the need for absolute synchronization between sender and receiver, and the need for an unlimited number of random keys.

Table 2-2. Example Encryption Using XOR Function

Plain Text	1	0	0	1	1	1	0	0	1	0	1	0	1	1	0	1	1	0
Key	0	1	0	0	1	1	0	1	1	1	0	0	1	0	1	0	1	1
Ciphertext	1	1	0	1	0	0	0	1	0	1	1	0	0	1	1	1	0	1

Another problem with this scheme is how do the sender and receiver agree on a secret key that they can use? If the sender and the receiver already have a secure channel, why do they even need a key? If they do not have a secure channel, then how can they share the key securely? This is called the “key distribution problem.”

2.2.1.4 Data Encryption Standard

The Data Encryption Standard (DES) algorithm is a symmetric block cipher technique having a block size of 64-bits and key size of 64 bits. However, out of the 64-bit key, 8 bits are reserved for parity checks and technically only 56 bits of the key is used for encryption and decryption.

It was widely used in banking applications, ATMs, and other commercial applications, and more so in hardware implementations than software.However, it has been found to be vulnerable to brute force attack. Hence, it is deprecated or no longer recommended.

DES is a Feistel Cipher with 16 rounds. A Feistel cipher consists of multiple rounds for processing the plaintext with the key. Every round consists of a substitution step followed by a permutation step. Increasing the number of rounds makes it more secure however the encryption/ decryption can get slower.

A general sequence of steps in the DES algorithm is shown in Figure 2-4.

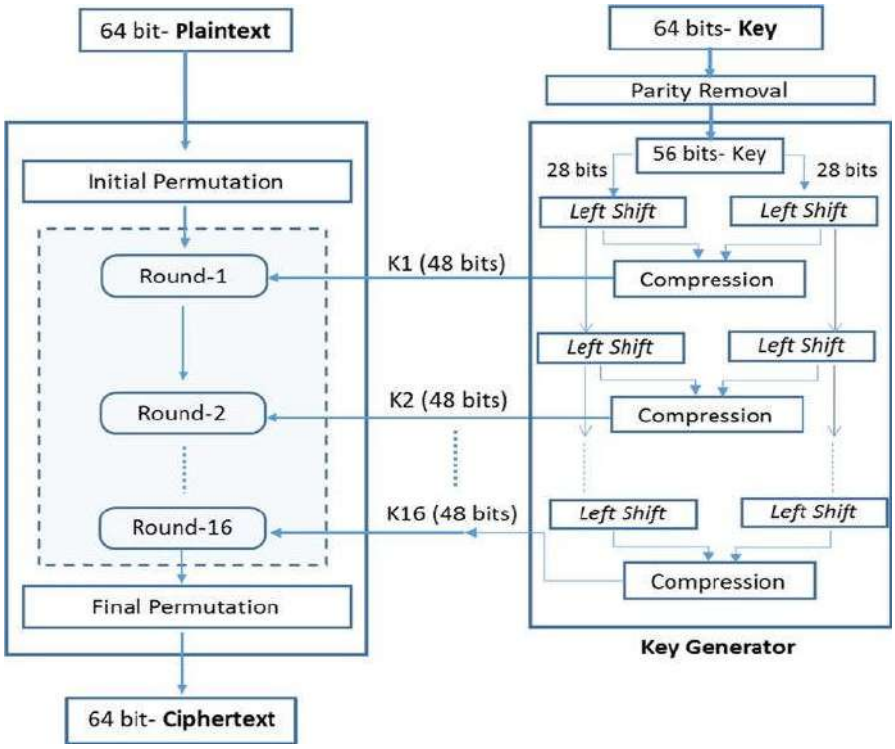


Figure 2-4. DES cryptography
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Key Generation

The key is 64 bits. However, every 8th bit(bit number 8, 16, 24, 32, 40, 48, 56, and 64) is used as a parity bit. So, only 56 bits are used for encryption and decryption.

1	2	3	4	5	6	7	8
9	10	11	12	13	14	15	16
17	18	19	20	21	22	23	24
25	26	27	28	29	30	31	32
33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48
49	50	51	52	53	54	55	56
57	58	59	60	61	62	63	64

The 56-bit key is further divided into two blocks of 28 bits each. Each block is circularly shifted left by one or two positions in every round. Both of these shifted blocks are combined and undergo a compression

mechanism that emits a 48-bit key called subkey which is used for encryption.

This process is repeated in every round.

Encryption

The plaintext block is divided into 64 bits blocks. If the number of bits in the message is not evenly divisible by 64, then the last block is padded to make it a 64-bit block.

Initial Permutation

Each plaintext block goes through an initial permutation (IP) round. The IP round simply permutes, i.e., rearranges the bits within the block in a specific pattern. It has no cryptographic significance as such, and its objective is to make it easier to load plaintext/ciphertext into DES chips in byte-sized format.

Division of Blocks

The 64bit output from the IP round is divided into two 32-bit blocks denoted as L (left block) and R(right block). The blocks are represented as L_i and R_i, where i denotes the number of rounds. The output of the IP round is denoted as L_0 and R_0.

In the first round, the blocks are swapped i.e. the right side 32-bit block (R) becomes the left side (L) and the left side becomes R. The 32-bit block (L) goes through an operation with the key k of that round and the right side 32-bit block (R) as shown following:

$L_i = R_i - 1$

$Ri = Li - 1 \oplus F(Ri - 1, Ki)$

F() is the "Cipher Function" which consists of multiple steps or operations. All the operations are performed on the R block.

1. The 32-bit R-block undergoes an Expansion Permutation and produces a 48-bit block.
2. This 48-bit block is then XORed with the 48-bit subkey supplied by the key generator of the same round.
3. The 48-bit XORed output is fed to the substitution box which reduces the block to 32 bits.
4. The 32-bit output of the S-box is fed to the permutation box (P-box)that outputs a 32-bit block. This is the final output of F() cipher function.

The 32-bit output of F() is then XORed with the 32-bit L-block. This XORed output then becomes the final R-block output of this round.

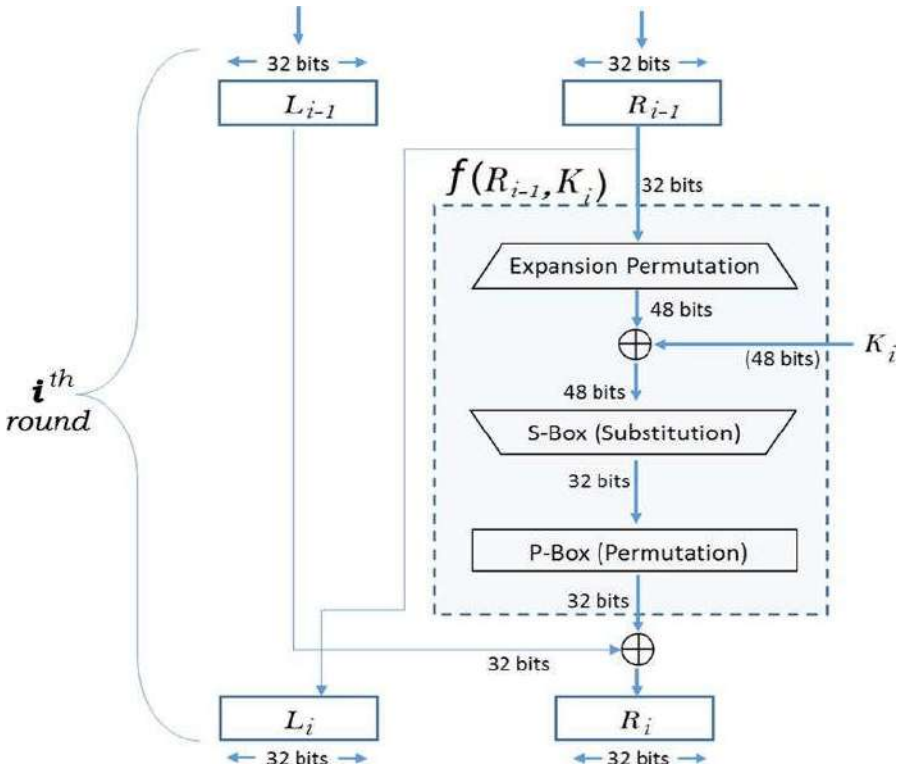


Figure 2-5. Round function of DES
#imgref:Beginning Blockchain-Singhal,Dhameja,Panda

The steps in Feistel round get repeated 16 times, where the output of one round is fed as the input to the following round.Once all the 16 rounds are over, the output of the 16th round is again swapped such that the left becomes the right block and vice versa.
Then the two blocks are clubbed to make a 64-bit block and passed through a permutation operation, which is the inverse of the initial permutation function and that results in the 64-bit ciphertext output.

Since, DES is a symmetric algorithm the decryption process is similar.

The 56-bit key was susceptible to brute force attack and the S-boxes used for substitution in each round were also prone to cryptanalysis attack. Hence, the AES is used over the DES Algorithm.

2.2.1.5 Advanced Encryption Standard

AES algorithm is a symmetric block cipher.The block size is fixed at 128 bits and allows a choice of three keys: 128 bits, 192 bits, and 256 bits.AES is named as AES-128, AES-192, and AES-256 depending on the size of the key.

The number of encryption rounds is dependent on the length of the key. There are ten rounds in AES-128, 12 rounds in AES-192 and 14 rounds in AES-256.

AES- 128

The encryption rounds in AES are iterative and operate on an entire data block of 128 bits in every round.

The 128- bit block can be considered as 16 bytes where individual bytes are arranged in a 4 × 4 matrix called a State Array.

Byte 0	Byte 4	Byte 8	Byte 12
Byte 1	Byte 5	Byte 9	Byte 13
Byte 2	Byte 6	Byte 10	Byte 14
Byte 3	Byte 7	Byte 11	Byte 15

AES uses the concept of words where a word consists of four bytes, that is, 32 bits. Hence, the bytes in each column of the state array together form a 32-bit word called state words.

$word_0$	$word_1$	$word_2$	$word_3$
Byte 0	Byte 4	Byte 8	Byte 12
Byte 1	Byte 5	Byte 9	Byte 13
Byte 2	Byte 6	Byte 10	Byte 14
Byte 3	Byte 7	Byte 11	Byte 15

Also, every byte can be represented with two hexadecimal numbers. Example: if the 8-bit byte is {00111010}, it could be represented as “3A” in Hex notation. “3” represents the left four bits “0011” and “A” represents the right four bits “1010.”

The processing in each round consists of byte-level substitution followed by word-level permutation. The overall encryption and decryption process of AES can be represented in Figure 2-6.

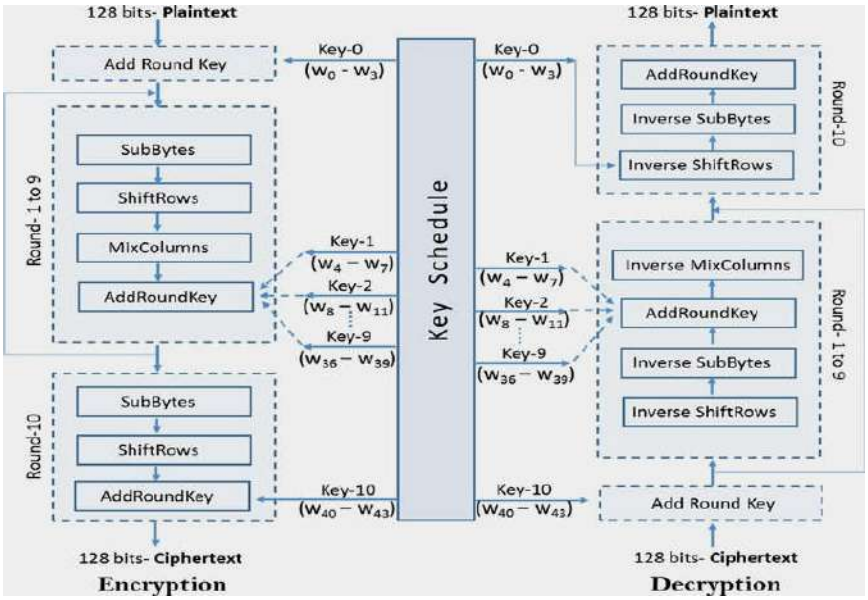


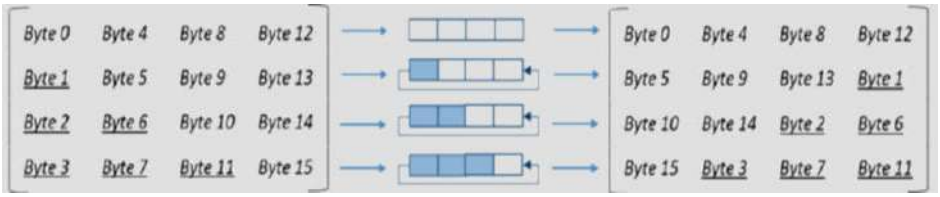
Figure 2-6. AES cryptography
#imgref:Beginning Blockchain-Singhal,Dhameja,Panda

The AES decryption process is not the reverse of the encryption process. The operations in the rounds are executed in a different order. All steps of the round function—Sub Bytes, Shift Rows, Mix Columns, Add Round Key—are invertible. The rounds are iterative in nature. All rounds perform all the four operations but the last round excludes the "Mix Columns" operation.

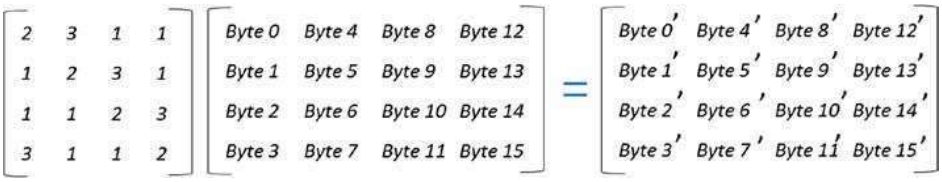
Sub Bytes: This step is the substitution step. Each byte is represented as two hexadecimal digits. For example, the byte {00111010} is represented as {3A}. A S-box lookup table (16 × 16 table) is used to find the corresponding value for row number 3 and column number A.

Shift Rows: This is the transformation step. It is based upon the matrix representation of the state array. It consists of the following shift operations:

- No circular shifting of the first row
- Left circular shifting of one byte from the second row
- Left circular shifting of two bytes from the third row
- Left circular shifting of three bytes from the fourth row



#imgref:BeginningBlockchain-Singhal,Dhameja,Panda



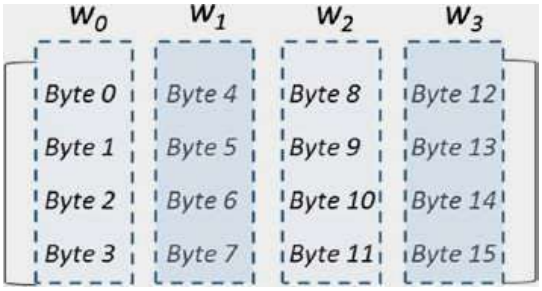
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Mix Columns: This is also a transformation step. All the four columns of the state are multiplied with a fixed polynomial (Cx) and get transformed to new columns. Each byte of a column is mapped to a new value that is a function of all four bytes in the column. This is achieved by the matrix multiplication of state as shown:
Byte 0₹ is calculated as shown:
Byte 0₹ = (2 . Byte0) ⊕ (3 . Byte1) ⊕ Byte3 ⊕ Byte4

The Mix Columns step, along with the Shift Rows step, provide the necessary diffusion property to the cipher.

Add Round Key: This is a transformation step. The 128- bit round key is bitwise XORed columnwise with the 128 bits of state i.e. four bytes of a word state column with one word of the round key.

The 128-bit key can be represented in the same 4 × 4 matrix as shown here:



#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

There are ten rounds in AES-128 and each round has its own round key.

In one round, all 128-bits of the subkey are XORed with the 128-bit input data block.

The key word [w0, w1, w2, w3] gets XORed with the initial input block before the round-based processing begins. The remaining 40 word-keys, w4 through w43, get used four words at a time in each of the ten rounds.

AES Key Expansion

The AES key expansion takes a 128-bit cipher key (four-word key) as input and produces a schedule of 44 key words from it.The key expansion operation is designed such that each grouping of a four-word key produces the next grouping of a four-word key.

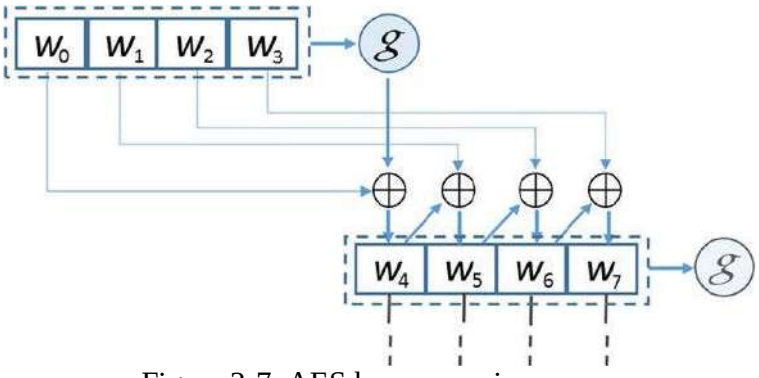


Figure 2-7. AES key expansion
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

The initial 128-bit key is [w0, w1, w2, w3] arranged in four words. The expanded word [w4, w5, w6, w7] is derived from the previous and the corresponding position word in the previous block i.e.4 positions back. For example,w5 is derived from w4 and w1. In the case of the first word, a three step function denoted as “g” is used.

1. The input four-word block goes through a circular left shift by one byte. For example [w0, w1, w2, w3] becomes [w1, w2, w3, w0].

2. The four bytes input word (e.g., [w1, w2, w3, w0]) is taken as input and byte substitution is applied on each byte using S-box.
3. The result of step 2 is XORed with something called round constant denoted as Rcon[]. The round constant is a word in which the right- most three bytes are always zero. For example, [x, 0, 0, 0]. The purpose of Rcon[ ] is to perform XOR on the left-most byte of the step 2 output keyword. The Rcon[ ] is different for each round.

The final output of the complex function "g" is generated, which is then XORed with $w_i - 4$ to get w_i.

The output state array of the last round is rearranged to form the 128-bit ciphertext block.

The AES algorithm is standardized by the NIST (National Institute of Standards and Technology). AES needs a longer processing time making it infeasible for larger data. AES is more secure than DES.

2.2.1.6 Challenges in Symmetric Key Cryptography

Key exchange is the biggest challenge in symmetric key encryption. Trust is another factor because of the shared secret key. If the key is shared by one of the communicating parties, the security of the system is compromised.

The number of keys needed is another biggest drawback of symmetric key cryptography. The number of keys needed for n nodes to communicate securely is n(n–1)/2 key pairs. Also, the keys need to be changed for each communication session. A trusted third party is needed for effective key management.

2.3 CRYPTOGRAPHIC HASH FUNCTIONS

Hash functions are the mathematical functions that are used for checking the integrity of the data. They play a very important role in blockchain as well as many cryptographic protocols and information security applications like Digital Signatures and message authentication codes (MACs). Cryptographic hash functions are a special class of hash functions that are apt for cryptography.

A cryptographic hash function is a one-way function that converts input data of arbitrary length and produces a fixed-length output. The output is usually termed "hash value" or "message digest." The hash is calculated at the receiver to check if the integrity of the message was compromised during transmission. It can be represented as shown in Figure 2-8.

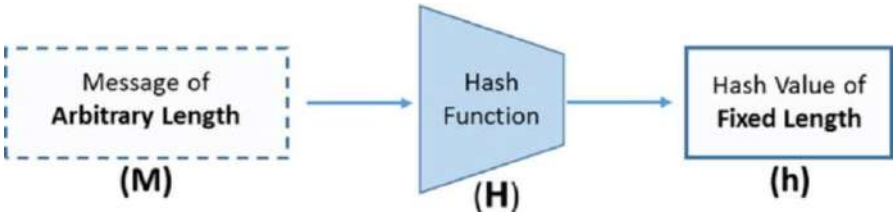
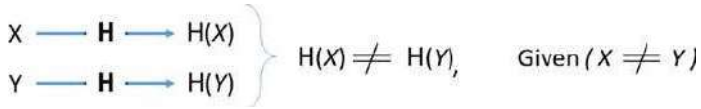


Figure 2-8. Hash function in its basic form
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Properties of hash function:

- The input size is not fixed but the size of the output is fixed.
- The hash value can be computed for any given message.
- Hash function is deterministic i.e. it produces the same hash value for the same input.
- A hash function cannot be inverted i.e. the input cannot be predicted from the hash value.
- The hash value changes drastically even for a small change in the input.

Properties related to cryptography:
Collision resistance: Two different inputs, say, X and Y, never give the same hash value.



#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Hash function is collision resistant. However, the fact that the output function is of fixed size might lead to collision. For example, if the output size is a 256-bit hash value, then the output space can have a maximum of 2256 values which implies that a collision must exist. However, it is extremely difficult to find that collision. This scenario is similar to the birthday paradox.

Collision resistance property is used to check for the integrity of a file uploaded on cloud storage. The hash value of the file is calculated and stored along with the file. It can be checked for tampering by calculating the hash function and comparing it with the previously stored hash. Since, it is extremely rare for a file to have the same hash value, it is collision resistant.

Preimage resistance: This property means that it is computationally impossible to find the input X from the output H(X).It is also called the "hiding" property. This property is effective if the number of input values is infinite or unknown. In case of a limited number of known inputs, an adversary can calculate the hash for all values and compare it with the received hash to predict the input.

Example: If the outcome of an experiment has only 3 values, the adversary can calculate the individual hash for each outcome. He will then compare the received hash with all the three computed hash valuesand easily determine the input.

To avoid such predictions, the input is combined with another random input “r,” so that it becomes difficult to find X from H(r || X).

If “r” is chosen from a 256- bit distribution, the probability of finding the exact value of r is 1/2256. This random value “r” is called nonce in blockchain. A nonce is a random number that can be used only once.

This property can be used to commit a value. For example, the user has participated in an event where he has to commit to a betting value. The user is supposed to declare it without disclosing the actual value. This dilemma can be solved by using the preimage resistance property of a Hash Function. The value to be betted on can be hashed and the hash output can be declared publicly. So, looking at the hash, nobody can predict what you have bet on. Also, you can always prove the value if needed by computing the hash.

Second preimage resistance: This property states that it is not possible to find 2 values which have a similar hash i.e. given an input X and its hash H(X), it is infeasible to find Y, such that H(X) = H(Y). This implies that a hash function which is collision resistant is second preimage resistant too.

The “puzzle friendliness” property implies that the only way to get to a solution is to traverse through all the possible options in the input space.

GivenH(r || X) = Z, the puzzle friendliness property means that it is difficult to find a value “Y” that exactly hashes to “Z.” i.e. H(r´ || Y) = Z, where r and r´ are part of the random input.

In the previous example, if “Z” is an n-bits output, then it has taken just one value out of 2n possible values. Note carefully that a part of your input, say “r,” is from a high min-entropy distribution, which has to be appended with your input X. Now comes the interesting part of designing a search puzzle.

Let’s assume that Z is a n-bits output and is a set of 2n possible values. The goal is to find a value of r such that when hashed appended with X, it falls within the output set of 2n values i.e. within Z. This situation is similar to a search puzzle. The smaller the output space, the harder the problem. The only solution is to try all possible values of r so that the output falls in a specific range.

For an n-bit hash value output, an average effort of 2n is needed to break preimage and second preimage resistance, and 2n/2 for collision resistance.

2.3.1 A Heads-up on Different Hash Functions

The most popular and widely used hash function was MD4 which belongs to the message digest (MD) family. There are other variations of MD like MD5 and MD6, RIPEMD,etc.

The MD family of algorithms is used for checking the integrity of a message. It takes an input of 512 bit blocks to produce a 128-bit message digest checksum. The message and message digest are sent along to the receiver. The receiver on receiving the message calculates the checksum and compares it with the sent message digest.

Secure Hash Algorithm (SHA) is another such hash function family which is widely used. It consists of four algorithms namely SHA-0, SHA-1, SHA-2, and SHA-3. The algorithm was named SHA however as new versions were invented with improved features, the algorithm was suffixed with a version number.

SHA-1 was found to solve some irregularities in SHA-0. Both were 160-bit hash functions that consumed 512-bit block sizes. SHA-1 was designed by the National Security Agency (NSA) for use in the digital signature algorithm (DSA) as well as with various security tools and Internet protocols such as SSL, SSH, TSL, etc. It was used for consistency checks in version control systems such as Mercurial, Git, etc.Some cryptographic weaknesses were found in the algorithm in 2005so it was deprecated after the year 2010.

2.3.1.1SHA-2

SHA-2 has many variants such as SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, and SHA-512/256 where SHA-256 and SHA-512 are theprimitive hash functions; while the other variants are derived from them. The SHA-2 family of hash functions are widely used in applications such as SSL, SSH, TSL, PGP, MIME, etc.

SHA-224 is a truncated version of SHA-256 with a different initial value or initialization vector (IV). However, both produce the same bit length hash outputs.

SHA-224 computation is a two-step process:

- SHA-256 value is computed with a different IV instead of the default used in SHA-256.
- The resulting 256-bit hash value is truncated to 224-bits

Algorithm	Truncated version of
SHA-224	SHA-256
SHA-382	SHA-512
SHA-512/224, , SHA-512/256	SHA-512

Truncations are performed to accommodate applications that have a requirement of a certain length of output.

Truncation value can be determined by the security level expected.

160 bits are needed for collision resistance whereas 80 bits are necessary for preimage-resistance. Collision resistance decreases with truncation.

Truncation also helps maintain backward compatibility with older applications.

Example: SHA- 224 provides 112-bit security that can match the key length of triple-DES (3DES).

SHA-256 is based on a 32-bit word and SHA- 512 is based on a 64-bit word. So, on a 64-bit architecture, SHA-512 and all its truncated variants can be computed faster with a better level of security compared with SHA-1 or other SHA-256 variants.

Table 2-3 is a tabular representation taken from the NIST paper that represents SHA-1 and different SHA-2 algorithms properties :

Algorithm	Message Size (bits)	Block Size (bits)	Word Size (bits)	Message Digest Size (bits)
SHA-1	$< 2^{64}$	512	32	160
SHA-224	$< 2^{64}$	512	32	224
SHA-256	$< 2^{64}$	512	32	256
SHA-384	$< 2^{128}$	1024	64	384
SHA-512	$< 2^{128}$	1024	64	512
SHA-512/224	$< 2^{128}$	1024	64	224
SHA-512/256	$< 2^{128}$	1024	64	256

Table 2-3. SHA-1 & SHA-2 Hash Function in a Nutshell

2.3.1.2SHA-256 and SHA-512

SHA-256 belongs to the SHA-2 family of hash functions and is called so because it produces a 256-bit hash value as output. It is used in bitcoins.

Hash functions take arbitrary length input and produce a fixed size output. The arbitrary length input is broken into fixed length blocks before it is fed to the compression function. A construction method is used to iterate through the compression function by constructing fixed-sized input blocks from arbitrary length input data and produce a fixed length output. Merkle-Damgård construction, tree construction, and sponge construction are examples of construction methods. It is proven that if the underlying compression function is collision resistant, then the overall hash function with any construction method should also be collision resistant.

The construction method that SHA-256 uses is the Merkle-Damgård construction, so let us see how it works in Figure 2-9.

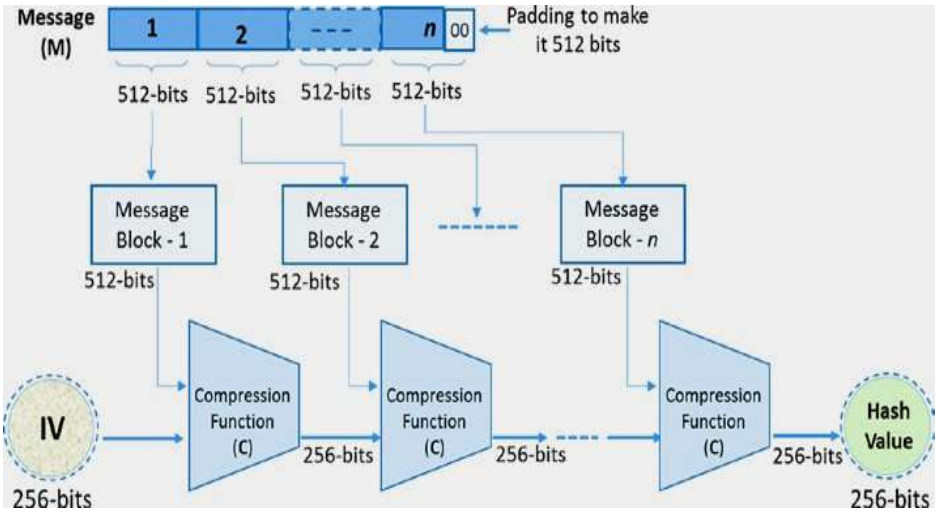


Figure 2-9. Merkle-Damgård construction for SHA-256
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

The message is divided into 512-bit blocks. If the message is not an exact multiple of 512 bits it is padded to make it 512 bits.

The 512-bit blocks are further divided into 16 blocks of 32-bit words (16 × 32 = 512).

Each block goes through 64 rounds of round function where each 32-bit word goes through a series of operations. The round functions are a combination of some common functions such as XOR, AND, OR, NOT, Bit-wise Left/Right Shift, etc.

The steps and the operations of SHA-256 are similar in SHA-512.However, there are 80 rounds of round functions in SHA-512 and the word length is 64 bits. The block size in SHA-512 is 1024 bits, which gets further divided into 16 blocks of 64-bit words The output message digest is 512 bits in length, that is, eight blocks of 64-bit words.

2.3.1.3RIPEMD

RACE Integrity Primitives Evaluation Message Digest (RIPEMD) hash function is a variant of the MD4 hash function. It is used in bitcoins.

RIPEMD was originally 128 bits but RIPEMD-160 was developed later. There exist 128-, 256-, and 320-bit versions of this algorithm, called RIPEMD-128, RIPEMD-256, and RIPEMD-320, respectively,

RIPEMD-160 is a cryptographic hash function whose compression function is based on the Merkle–Damgård construction. The input is broken into 512-bit blocks and padding is applied when the input bits are not a multiple of 512. The 160-bit hash value output is usually represented as 40-digit hexadecimal numbers.

The compression function is made up of 80 stages, made up of two parallel lines of five rounds of 16 steps each (5 × 16 = 80). The compression function works on sixteen 32-bit words (512-bit blocks).

Bitcoin uses both SHA-256 and RIPEMD-160 hashes together for address generation. SHA-256 is used for address generation whereas RIPEMD-160 is used to reduce the hash value to 160 bits.

2.3.1.4SHA-3

SHA or the Keccak (pronounced as "ket-chak") algorithm was standardized by the NIST in 2015. SHA-3 used a different construction method called sponge construction.SHA-3 has variants like SHA3-224, SHA3-256, SHA3-384, SHA3 -512, and two extendable-output functions (XOFs), called SHAKE128 and SHAKE256 to provide backward compatibility with SHA-2. XOFs are used for their capability to give an output that can be extended to any desired length.The following diagram (Figure 2-10) shows how SHA-3 (Keccak algorithm) is designed at a high level.

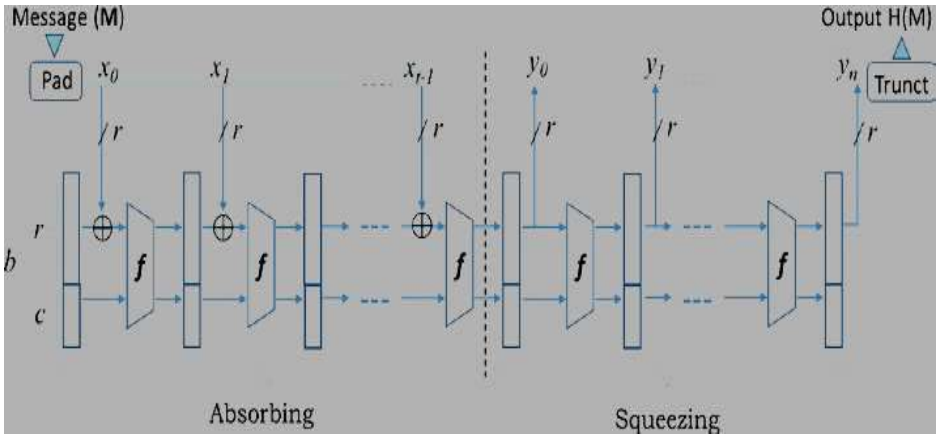


Figure 2-10. Sponge construction for SHA-3
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Steps in SHA-3

As you can see in Figure 2-10, the message is first divided into blocks (Xi) of size r bits. If the input data is not a multiple of r bits, then it is padded to Xi.

Padding is performed as below

Xi = m || P 1 {0}* 1

"P" is a predetermined bit string followed by a leading and trailing 1 and some number of zeros. Table 2-4 shows the various values of P.

Mode	Output Length	P	1 {0}* 1
SHA3-224	224	11001	1 {0}* 1
SHA3-256	256	11101	1 {0}* 1
SHA3-384	384	11001	1 {0}* 1
SHA3 -512	512	11101	1 {0}* 1
Variable Length (XOFs)	Arbitrary	1111	1 {0}* 1

Table 2-4. Padding in SHA-3 variants
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

SHA-3 sponge construction consists of two phases namely the "absorbing" phase for input, and the "squeezing" phase for output using keccak-3. The Absorbing phase consists of various operations of the algorithm whereas the Squeezingphase gives an output of a configurable length.

The SHA-3 is designed to be tunable for its security strength, input, and output sizes with the help of tuning parameters "r" and "c" to tradeoff between security and performance.

r + c = b where r is bitrate and "b" represents the width of the state
$b = 25 \times 2^{\ell}$ where "ℓ" can take on values between 0 and 6
Hence, "b"can be {25, 50, 100, 200, 400, 800 and 1600}.
"c" is the capacity which satisfies the condition r + c = b $\in$ {25, 50, 100, 200, 400, 800, 1600}

In SHA-3, the exponent value ℓ is fixed to be "6," so the value of b is 1600 bits.There are two permissible bitrate values for r : r = 1344 which gives c = 256 and r = 1088 which gives c = 512.

The core engine of the algorithm Keccak-f is called "Keccak-f Permutation". It consists of "n" rounds, where "n" is computed as: $n = 12 + 2\ell$. Since the value of ℓ is 6 for SHA-3, there will be 24 rounds in each Keccak-f. Every round takes "b" bits (r + c) input and produces the same number of "b" bits as output.In each round, the input "b" is called a state. This state array "b" can be represented as a three-dimensional (3-D) array $b = (5 \times 5 \times w)$, where word size $w = 2\ell$. So, w = 64 bits, which means 5 × 5 = 25 words of 64 bitseach. Recall that ℓ = 6 for SHA-3, so $b = 5 \times 5 \times 64 = 1600$. The 3-D array can be shown as in Figure 2-11.

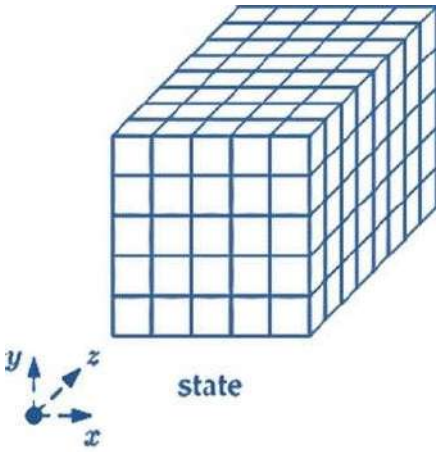


Figure 2-11. State array representationin SHA-3
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Each round consists of a sequence of five steps and the state array gets manipulated in each of those steps as shown in Figure 2-12.

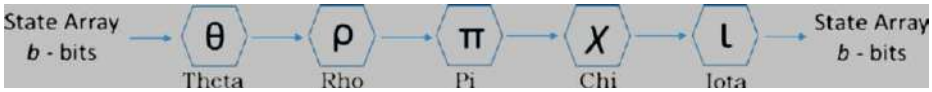


Figure 2-12. The five steps in each SHA-3 round
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Theta (θ) step: It performs the XOR operation to provide minor diffusion.

Rho (ρ) step: It performs bitwise rotation of each of the 25 words.

Pi (π) step: It performs permutation of each of the 25 words.

Chi (χ) step: In this step, bits are replaced by combining those with their two subsequent bits in their rows.

Iota (ι) step: It XORs a round constant into one word of the state to break the symmetry.

The last round of Keccak-f produces the y0 output, which is enough for SHA-2 replacement mode, i.e., the output with 224, 256, 384, and 512 bits. Note that the least significant bits of y0 are used for the desired length output. In case of variable length output, along with y0, other output bits of y1, y2, y3… can also be used.

Compared to SHA-2, the performance of SHA-3 is good in software and is excellent in hardware.

2.3.1.5Applications of Hash Functions

Hash functions are used in verifying the integrity and authenticity of information.

Hash functions can also be used to index data in hash tables which speeds up the process of searching. Hash values are shorter than original data so the search is faster.

It can also be used for authenticating users without storing the passwords locally. The hash of the password is calculated and stored locally. Whenever a user enters a password, its hash is calculated and further compared to the stored hash. If both the hash matches, the user logs in successfully.

Hash function can be used to implement PRNG.Bitcoin uses hash functions as a proof of work (PoW) algorithm to verify transactions and also to generate addresses.The two most important applications are digital signatures and in MACs such as hash-based message authentication codes (HMACs).

2.3.1.6Code Examples of Hash Functions

-*- coding: utf-8 -*- import hashlib

hashlib module is a popular module to do hashing in python

#Constructors of md5(), sha1(), sha224(), sha256(), sha384(), and sha512() present in hashlib

md=hashlib.md5()

md.update("The quick brown fox jumps over the lazy dog") print md.digest()

print "Digest Size:", md.digest_size, "\n", "Block Size: ", md.block_size

Comparing digest of SHA224, SHA256,SHA384,SHA512

print "Digest SHA224", hashlib.sha224("The quick brown fox jumps over the lazy dog").hexdigest()

print "Digest SHA256", hashlib.sha256("The quick brown fox jumps over the lazy dog").hexdigest()

print "Digest SHA384", hashlib.sha384("The quick brown fox jumps over the lazy dog").hexdigest()

print "Digest SHA512", hashlib.sha512("The quick brown fox jumps over the lazy dog").hexdigest()

All hashoutputs are unique

RIPEMD160 160 bit hashing example h = hashlib.new('ripemd160')

h.update("The quick brown fox jumps over the lazy dog") h.hexdigest()

#Key derivation Alogithm:

#Native hashing algorithms are not resistant against brutefore attack.

#Key deviation algorithms are used for securing password hashing.

import hashlib, binascii algorithm='sha256' password='HomeWifi'

salt='salt' # salt is random data that can be used as an additional input to a one-way function

nu_rounds=1000

key_length=64 #dklen is the length of the derived key

```
dk=hashlib.pbkdf2_hmac(algorithm,password,salt,nu_rounds,
dklen=key_length)
print 'derieved key: ',dk
print 'derieved key in hexadeximal :', binascii.hexlify(dk)
# Check properties for hash import hashlib
input = "Sample Input Text" for i in xrange(20):
# add the iterator to the end of the text input_text = input + str(i)
# show the input and hash result
print input_text, ':',      hashlib.sha256(input_text). hexdigest()
#coderef:BeginningBlockchain-Singhal,Dhameja,Panda
```

2.3.2 MAC and HMAC

HMAC is used to provide message authentication using Symmetric Key and message integrity using hash functions. The sender sends the message and MAC together for the receiver to verify and trust it. The receiver uses the symmetric key K to compute the MAC of the received message. The computed MAC iscompared with the received MAC for verification.MAC = H(key || message). HMAC is a technique to turn the hash functions into MACs. They are widely used in RFID-based systems. In SSL/TLS, HMAC is used to allow client and server to verify and ensure that the exchanged data has not been altered during transmission.

2.3.2.1 MAC Strategies

MAC-then-Encrypt:

MAC is computed on the cleartext, appended to the data, and then the mac and data are encrypted together. This scheme does not provide integrity of the ciphertext. At the receiving end, the message decryption has to happen first to be able to check the integrity of the message. It ensures the integrity of the plaintext, however. TLS uses this scheme of MAC to ensure that the client-server communication session is secured.

Encrypt-and-MAC: This technique requires the encryption and MAC computation of the message or the cleartext, and then appending the MAC at the end of the encrypted message or ciphertext. MAC is computed on the cleartext, so integrity of the cleartext can be assured but not of the ciphertext, which leaves scope for some attacks. Unlike the previous scheme, integrity of the cleartext can be verified. SSH (Secure Shell)uses this MAC scheme.

Encrypt-then-MAC: This technique requires that the cleartext needs to be encrypted first, and then compute the MAC on the ciphertext. This MAC of the ciphertext is then appended to the ciphertext itself. This scheme ensures integrity of the ciphertext, so it is possible to first check the integrity and if valid then decrypt it. It easily filters out the invalid ciphertexts, whichmakes it efficient in many cases. Also, since MAC is in ciphertext, in no way does it reveal information about the plaintext. It is usually the most ideal of all schemes and has wider implementations. It is used in IPsec.

2.4 ASYMMETRIC KEY CRYPTOGRAPHY

Asymmetric key cryptography is also known as “public key cryptography”. Every user has a pair of keys i.e. a public key and a private key.One key is used for encryption and another for decryption. Both the keys are different. Asymmetric key cryptography can be used for confidentiality or as digital signatures. The public key can be shared with everyone but the private key is kept secret.Since, the key is public and usually stored in a public repository it is called public key cryptography.

Public key cryptography is used forEncryption.

Let’s assume that Alice wants to send a message to Bob.

Alice encrypts the plaintext message m using encryption algorithm E and the public key PukBob to prepare the ciphertext c.

c = E(PukBob, m)

Send the ciphertext c to Bob.

Bob receives the encrypted text c and decrypts it using the decryption algorithm Dand his own private key PrkBob to get the original plaintext m.

m = D(PrkBob, c)

Such a system can be represented as shown in Figure 2-13.

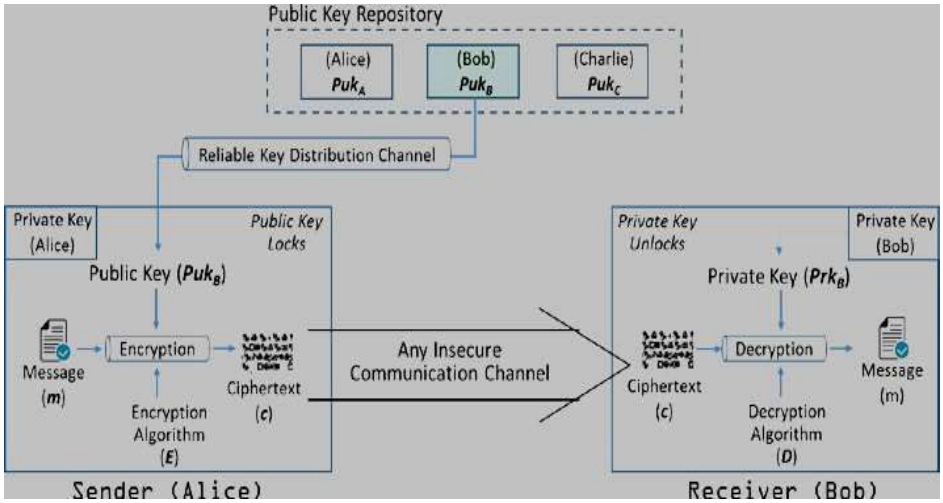


Figure 2-13. Asymmetric cryptography for confidentiality #imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Public key cryptography can also be used for authentication.

The sender encrypts the message with his own private key. The receiver on receiving the encrypted text decrypts it using the sender’s public key thereby authenticating that the message originally came from the sender.Since, a private key is used for encrypting a message it is called a digital signature.

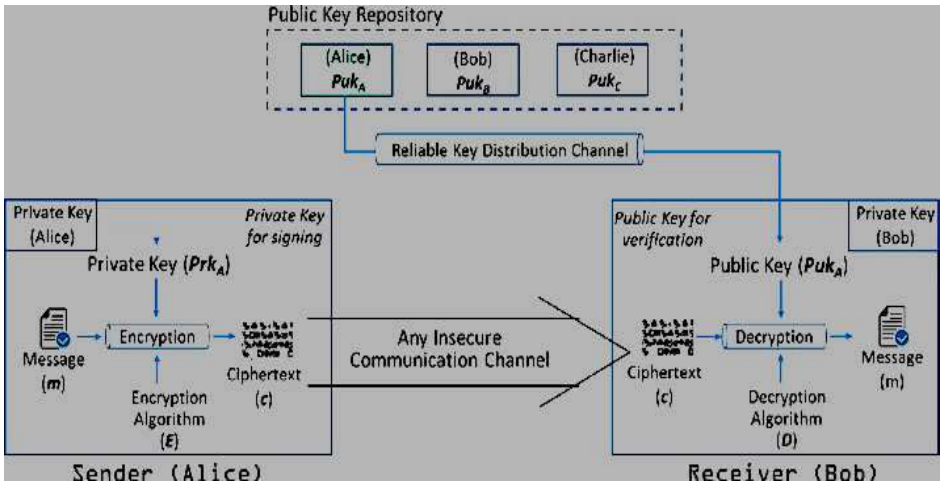


Figure 2-14. Asymmetric cryptography for authentication #imgref:BeginningBlockchain-Singhal,Dhameja,Panda

In the example in Figure 2-14, the message was prepared using Alice's private key, so it could be ensured that it only came from Alice. So, the entire message served as a digital signature. Note that both confidentiality and authentication are desirable. To facilitate this, public key encryption has to be used twice. The message should first be encrypted with the sender's private key to provide a digital signature. Then it should be encrypted with the receiver's public key to provide confidentiality. It can be represented as:
c = E[PukBob, E(PrkAlice, m)]
m = D[PukAlice, D(PrkBob, c)]

As you can see, the decryption happens in just its reverse order.

In the real world, App stores such as Google Play or Apple App Store require that the software apps should be digitally signed before they get published.

Public keys are known and accessible to everyone. They can be used to encrypt the message or to verify the signatures.

Private keys are extremely private to individuals. They are used to decrypt the message or to create signatures.

In asymmetric or public key cryptography, there is no key distribution problem, as exchanging the agreed upon key is no longer needed.

The biggest challenge with this scheme is authentication of the public key. How to ensure that the public key used is really the public key of the intended recipient and not of an intruder or eavesdropper? To solve this, a trusted third party called Public Key Infrastructure (PKI) is used. PKIs assure the authenticity of public keys by the process of attestation or notarization of user identity. PKIs provide verified public keys by embedding them in a security certificate by digitally signing them.

The public key encryption is a one-way function or a trapdoor function because both the keys are needed for the scheme to work. The encryption key cannot be used for decryption and vice versa.

2.4.1 RSA

The RSA algorithm, named after Ron Rivest, Adi Shamir, and Leonard Adleman, is possibly one of the most widely used cryptographic algorithms. It is based on the practical difficulty of factoring very large numbers. In RSA, plaintext and ciphertext are integers between 0 and n − 1 for some n.RSA is a block cipher.

Modular Arithmetic

Let m be a positive integer called modulus. Two integers a and b are congruent modulo m if:

$a \equiv b \pmod{m}$, which implies $a - b = m \cdot k$ for some integer k. Example: if $a \equiv 16 \pmod{10}$ then a can have the following solutions: a = . . ., −24, −14, −4, 6, 16, 26, 36, 46

Any of these numbers subtracted by 16 is divisible by 10. For example,

−24 −16 = −40, which is divisible by 10. Note that $a \equiv 36 \pmod{10}$ can also have the same solutions of a.

As per the Quotient-Remainder theorem, only a unique solution of "a" exists that satisfies the condition: $0 \leq a < m$. In the example $a \equiv 16 \pmod{10}$, only the value 6 satisfies the condition $0 \leq 6 < 10$. This is what will be used in the encryption/decryption process of the RSA algorithm.

Let us now look at the Inverse Modulus. If b is an inverse to a modulo m, then it can be represented as:

$a\,b \equiv 1 \pmod{m}$, which implies that $a\,b - 1 = m \cdot k$ for some integer k. Example: 3 has inverse 7 modulo 10 since

$3 \cdot 7 = 1 \pmod{10}$ => 21 − 1 = 20, which is divisible by 10.

Generation of Key Pairs

In the RSA scheme, the public key consists of (e, n) where n is called the modulus and e is called the public exponent. Similarly, the private key consists of (d, n), where n is the same modulus and d is the private exponent.

Let us see how these keys get generated along with an example:

1. Generate a pair of two large prime numbers p and q.

   Let us take two small prime numbers as an example here for the sake of easy understanding. So, let the two primes be p = 7 and q = 17.

2. Compute the RSA modulus (n) as n = pq.

   This n should be a large number, typically a minimum of 512 bits. In our example, the modulus (n) = pq = 119.

3. Find a public exponent e such that $1 < e < (p - 1)(q - 1)$ and there must be no common factor for e and $(p - 1)(q - 1)$ except 1. It implies that e and $(p - 1)(q - 1)$ are coprime. Note that there can be multiple values that satisfy this condition and can be taken as e, but any one should be taken.In our example, $(p - 1)(q - 1) = 6 \times 16 = 96$. So, e can be relatively prime to and less than 96. Let us take e to be 5.
4. The pair of numbers (e, n) form the public key and should be made public. So, the public key is (5, 119).
5. Calculate the private exponent d using p, q, and e considering the number d is the inverse of e modulo $(p - 1)(q - 1)$. This implies that d when multiplied by

   e is equal to 1 modulo $(p - 1)(q - 1)$ and $d < (p - 1)(q - 1)$. It can be represented as:

   $e\ d = 1 \bmod (p - 1)(q - 1)$

   In our example, we have to find d such that the above equation is satisfied which means, $5\ d = 1 \bmod 96$ and also $d < 96$.Solving for multiple values of d, we can see that d = 77 satisfies our condition. See the math:$77 \times 5 = 385$ and $385 - 1 = 384$ is divisible by 96 because$4 \times 96 + 1 = 385$
6. The private key is (77, 119).

Encryption/Decryption Using Key Pair
Encryption:
$c = m \cdot e \pmod n$ given the public key (e, n) and the plaintext message m.
Decryption :
$m = c \cdot d \pmod n$ given the private key (d, n) and the ciphertext c.

RSA at work:

- The sender wants to send a text message to the receiver whose public key is known and is say (e, n).The sender breaks the text message into blocks that can be represented as a series of numbers less than n.
- The ciphertext equivalents of plaintext can be found using $c = m\ e \pmod n$. If the plaintext (m) is 19 and the public key is (5, 119) with e = 5 and n = 119, then the ciphertext c will be 195(mod 119) = 2, 476, 099(mod 119) = 66, which is the remainder and 20,807 is the quotient. So, c = 66
- When the ciphertext 66 is received at the receiver's end, it needs to be decrypted to get the plaintext using $m = c\ d \pmod n$.
- The receiver already has the private key (d, n) withd = 77 and n = 119, and received the ciphertext c = 66 by the sender. So, the receiver can easily retrieve the plaintext using these values as m = 6,677(mod 119) = 19

The RSA scheme is based on this practical difficulty of factoring large numbers. If p and q are not large enough, or the publickey e is small, then the strength of RSA goes down. Currently, RSA keys are typically between 1024 and 2048 bits long. The computational overhead of the RSA cryptography increases with the size of the keys. Hence, in situations where the amount of data is huge, it is advisable to use a symmetric encryption technique and share the key using an asymmetric encryption technique such as RSA.

Also, we looked at one of the aspects of RSA, that is, for encryption and decryption. RSA can also be used for authentication through digital signature. One can take the hash of the data, sign it using their own private key, and share it along with the data. The receiver can check with the sender's public key and ensure that it was the sender who sent the data, and not someone else. This way, in addition to secure key transport, the public key encryption method RSA also offers authentication using a digital signature. RSA is widely being used with HTTPS on web browsers, emails, VPNs, and satellite TV. It is also used to digitally sign many commercial applications or the apps in app stores. SSH also uses public key cryptography; when you connect to an SSH server, it broadcasts a public key that can be used to encrypt data to be sent to that server. The server can then decrypt the data using its private key.

2.4.2 Digital Signature Algorithm

The DSA was designed by the NSA as part of the Digital Signature Standard (DSS) and standardized by the NIST. Its primary objective is to sign messages digitally, and not encryption. RSA is for both key management and authentication whereas DSA is only for authentication. DSA is based on discrete logarithms. At a high level, DSA is used as shown in Figure 2-15.

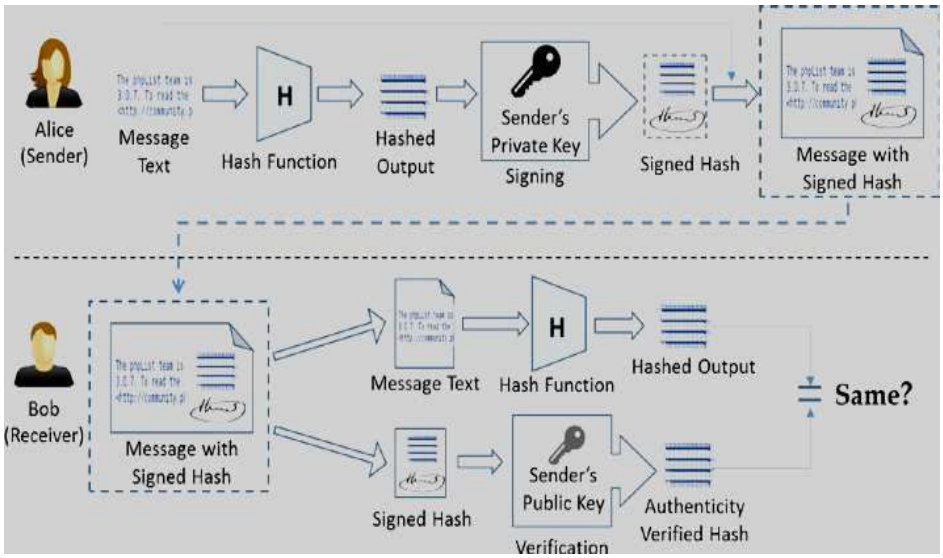


Figure 2-15. Digital Signature Algorithm (DSA)
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

As you can see in Figure 2-15, the message is first hashed and then signed. So, after the message is signed, the signed hash is tagged with the message and sent to the receiver. The receiver can then check the authenticity and find the hash. Also, hash the message to get the hash again and check if the two hashes match. This way, DSA provides the following security properties:

Authenticity: Signed by private key and verified by public key
Data integrity: Hashes will not match if the data is altered.

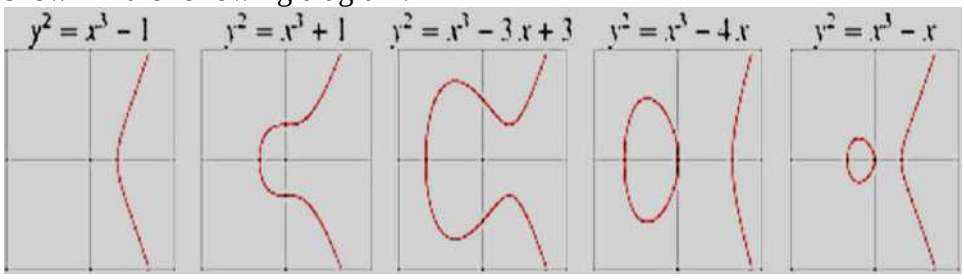
Non-repudiation: Since the sender signed it using a private key, they cannot deny later that they did not send the message. Non-repudiation is a property that is most desirable in situations where there are chances of a dispute over the exchange of data. For example, once an order is placed electronically, a purchaser cannot deny the purchase order if non-repudiation is enabled in such a situation.

A typical DSA scheme consists of three algorithms: (1) key generation,(2)signature generation, and (3) signature verification.

2.4.3 Elliptic Curve Cryptography

Elliptic curve cryptography (ECC) refers to a suite of cryptographic protocols and is based on the discrete logarithm problem. ECCoffers greater security using a smaller key size hence it is widely used in small embedded devices, sensors, Iot devices, etc.A 160-bit ECC key is considered to be as secure as a 1024-bit RSA key.

ECC is based on a mathematically related set of numbers on an elliptic curve over finite fields.The elliptic curve satisfies the following mathematical equation:
$y2 = x3 + ax + b$, where $4\ a3 + 27\ b2 \neq 0$
With different values of “a” and “b”, the curve takes different shapes as shown in the following diagram:



#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

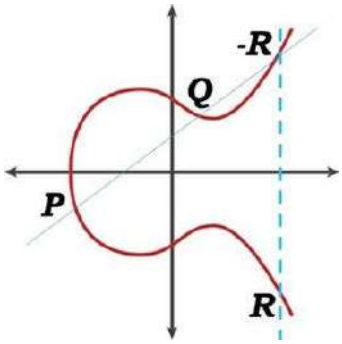
There are several important characteristics of elliptic curves that are used in cryptography, such as:

1. The curves are horizontally symmetrical. i.e., the curve on the X-axis is a mirror image of the curve on the Y-axis.

2. Any nonvertical line can intersect the curve in at most three places. Consider two points P and Q on the elliptic curve and draw a line

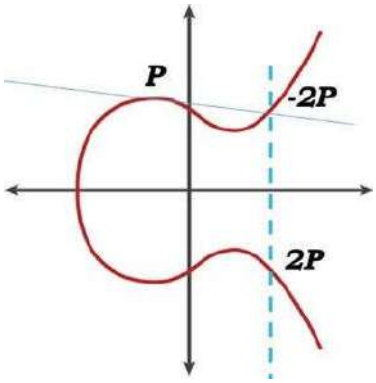
through them. This line intersects the curve at one more place. Let's name it (− R). If you draw a vertical line through (− R), it will cross the curve at, say, R, which is a reflection of the point (− R).

3. The third property implies that P + Q = R. This is called "point addition," which means adding two points on an elliptic curve will lead you to another point on the curve. Refer to the following diagram for a pictorial representation of these three properties.

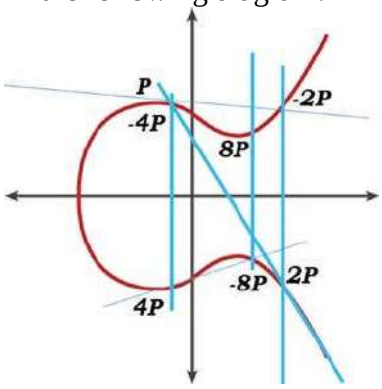


#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

We can apply this operation to the same point P, i.e., P and P (called "point doubling"). We can have an infinite number of lines passing through P. Let's consider the tangential line. The tangent line will cross the curve in one more point and a vertical line from there will cross the curve again to get to the final value. It can be shown as follows:



Hence it is can be seen that we can apply point doubling many times to the initial point and every time it will lead us to a different point on the curve as shown in the following diagram:



#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

When the initial and final point is given, there is no way to determine if point doubling was applied to reach the final resulting point. This is the discrete logarithm problem for ECC, where it states that given a point G and Q, where Q is a multiple of G, find “d” such that Q = d G. Here, Q is the public key and d is the private key.

The curve should be defined over a finite field which means that there is an upper limit on the maximum value on the X-axis. This value is represented as P and is called "modulo” value. It also defines the key size.

So, in order to define an ECC, the following domain parameters need to be defined:

The Curve Equation: $y2 = x3 + ax + b$, where $4\ a3 + 27\ b2 \neq 0$

P: The prime number, which specifies the finite field that the curve will be defined over (modulo value)

a and b: Coefficients that define the elliptic curve

G: Base point or the generator point on the curve. This is the point where all the point operations begin and it defines the cyclic subgroup.

n: The number of point operations on the curve until the resultant line is vertical. So, it is the order of G, i.e., the smallest positive number such that $nG = \infty$. It is normally prime.

h: It is called “cofactor,” which is equal to the order of the curve divided by n. It is an integer value and usually close to 1.

ECC is a great technique to generate the keys and is used along with other techniques for digital signatures and key exchange.

2.4.4 Elliptic Curve Digital Signature Algorithm

The ECDSA is a type of DSA that uses ECC for key generation. ECDSA can be a better alternative to RSA in terms of smaller key size, better security, and higher performance.

There are broadly three steps to ECDSA: key generation, signature generation, and signature verification.

Key Generation

Since the domain parameters (P, a, b, G, n, h) are pre-established, the curve and the base point are known by both parties. Also, the prime P that makes it a finite field is usually 160 bits. So, the sender, say, Alice does the following to generate the keys:
Select a random integer d in the interval $[1, n - 1]$
Compute Q = d G
Declare Q is the public key and keep d as the private key.

Signature Generation

Once the keys are generated, Alice, the sender, would use the private key “d” to sign the message (m). So, she would perform the following steps in the order specified to generate the signature:

Select a random number k in the interval [1, n − 1]
Compute k.G and find the new coordinates (x1, y1) and find r = x1 mod n
If r = 0, then start all over again
Compute e = SHA-1 (m)
Compute s = k −1 (e + d . r) mod n
If s = 0, then start all over again from the first step
Alice's signature for the message (m) would now be (r, s)

Signature Verification

Let us say Bob is the receiver here and has access to the domain parameters and the public key Q of the sender Alice. As a security measure, Bob should first verify that the data he has, which is the domain parameters, the signature, and Alice's public key Q are all valid. To verify Alice's signature on the message (m), Bob would perform the following operations in the order specified:
Verify that r and s are integers in the interval [1, n − 1]
Compute e = SHA-1 (m)
Compute w = s −1 mod n
Compute u1 = e w mod n, and u2 = r w mod n
Compute X = u1 G + u2 G, where X represents the coordinates, say (x2, y2)
Compute v = x1 mod n
Accept the signature if r = v, otherwise reject it

ECDSA is used in digital certificates. A digital certificate is a public key, bundled with the device ID and the certificate expiration date. This way, certificates enable us to check and confirm to whom the public key belongs and the device is a legitimate member of the network under consideration. These certificates are very important to prevent "impersonation attack" in key establishment protocols. Many TLS certificates are based on ECDSA key pairs.

2.4.5 Code Examples of Asymmetric Key Cryptography

```
# -*- coding: utf-8 -*- import Crypto
from Crypto.PublicKey import RSA from Crypto import Random
from hashlib import sha256
# Function to generate keys with default length 1024 def
generate_key(KEY_LENGTH=1024):
random_value= Random.new().read
keyPair=RSA.generate(KEY_LENGTH,random_value) return keyPair
#Generate Key for ALICE and BOB bobKey=generate_key()
aliceKey=generate_key()
#Print Public Key of Alice and Bob. This key could
sharedalicePK=aliceKey.publickey()
bobPK=bobKey.publickey()
print "Alice's Public Key:", alicePK print "Bob's Public Key:", bobPK
#Alice wants to send a secret message to Bob. Lets create a dummy message for Alice
```

```
secret_message="Alice's secret message to Bob" print "Message",
        secret_message
# Function to generate a signature def generate_signature(key,message):
message_hash=sha256(message).digest()
signature=key.sign(message_hash,'') return signature
# Lets generate a signature for secret message
alice_sign=generate_signature(aliceKey,secret_message)
# Before sending message in network, encrypt message using the Bob's
public key...
encrypted_for_bob      = bobPK.encrypt(secret_message, 32)
# Bob decrypts secret message using his own private key...
decrypted_message     = bobKey.decrypt(encrypted_for_bob) print
"Decrypted message:", decrypted_message
# Bob will use the following function to verify the signature from Alice
using her public key
def verify_signature(message,PublicKey,signature):
message_hash=sha256(message).digest()
verify = PublicKey.verify(message_hash,signature) return verify
# bob is verifying using decrypted message and alice's public key
print "Is alice's signature for decrypted message valid?",
verify_signature(decrypted_message,alicePK, alice_sign)
```

2.4.6 The ECDSA Algorithm Code

```
import ecdsa
# SECP256k1 is the Bitcoin elliptic curve
signingKey = ecdsa.SigningKey.generate(curve=ecdsa.SECP256k1) # Get
the verifying key
verifyingKey = signingKey.get_verifying_key()
# Generate The signature of a message signature =
signingKey.sign(b"signed message")
# Verify the signature is valid or invalid for a message
verifyingKey.verify(signature, b"signed message") # True.
Signature is valid
# Verify the signature is valid or invalid for a message assert
verifyingKey.verify(signature, b"message") # Throws an error. Signature
is invalid for message
#coderef:BeginningBlockchain-Singhal,Dhameja,Panda
```

2.5 Diffie-Hellman Key Exchange

It is one of the most popular ,simple and effective key exchange algorithmdeveloped for securely exchanging the cryptographic keys in an insecure encironment.

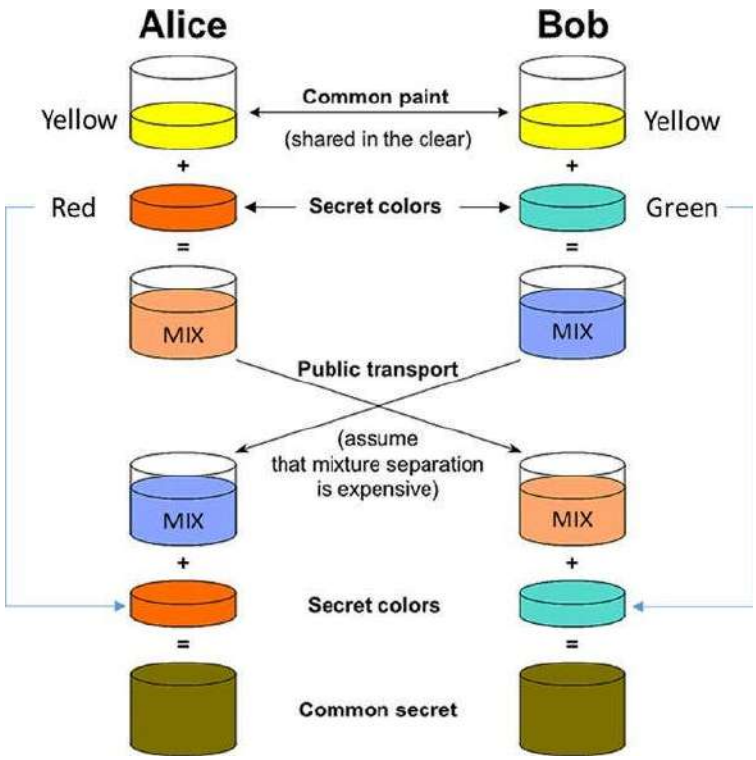


Figure 2-16. Diffie-Hellman key exchange illustration #imgref:BeginningBlockchain-Singhal,Dhameja,Panda

DH is based on the concept of sharing components used for key formation. The key is generated at both the ends. In the figure shown above, only the yellow color was shared between the two parties in the first step, which may represent any other color or a random number. Both parties then add their own secret to it and make a mixture. That mixture is again shared through the same insecured channel. Respective parties then add their secret to it and form their final common secret. In this example with colors, observe that the common secrets are the combination of the same sets of colors.

Mathematical Example of DH:

1. Alice and Bob agree on P = 23 and G = 9
2. Alice chooses private key a = 4, computes 94 mod 23 = 6 and sends it to Bob
3. Bob chooses private key b = 3, computes 93 mod 23 = 16 and sends it to Alice
4. Alice computes 164 mod 23 = 9
5. Bob computes 63 mod 23 = 9

Alice and Bob are able to generate the same secret key at their ends without sharing the actua key.

2.5.1 Code for DH:

```
/* Program to calculate the Keys for two parties using Diffie- Hellman Key exchange algorithm */
// function to return value of a ^ b mod P
long long int power(long long int a, long long int b, long long int P)
{
if (b == 1)
return a;
else
return (((long long int)pow(a, b)) % P);
}
//Main program for DH Key computation int main()
{
long long int P, G, x, a, y, b, ka, kb;
// Both the parties agree upon the public keys G and P P = 23; // A prime number P is taken
printf("The value of P : %lld\n", P);
G = 9; // A primitve root for P, G is taken printf("The value of G : %lld\n\n", G);
// Alice will choose the private key a a = 4; // a is the chosen private key
printf("The private key a for Alice : %lld\n", a); x = power(G, a, P); // gets the generated key
// Bob will choose the private key b b = 3; // b is the chosen private key
printf("The private key b for Bob : %lld\n\n", b); y = power(G, b, P); // gets the generated key
// Generating the secret key after the exchange of keys ka = power(y, a, P); // Secret key for Alice
kb = power(x, b, P); // Secret key for Bob
printf("Secret key for the Alice is : %lld\n", ka); printf("Secret Key for the Bob is : %lld\n", kb);
return 0;
}
#coderef:BeginningBlockchain-Singhal,Dhameja,Panda
```

2.6 SYMMETRIC VS. ASYMMETRIC KEY CRYPTOGRAPHY

Symmetric	**Asymmetric**
Private Key	Public Key
Key Exchange Problem	None
Simple	Compute intensive

Fast	Slow
Suitable for large data	Suitable for short messages
Permutation and substitution	Integer arithmetic
Purely for Encryption and Decryption	Key sharing
Trusted environment	Untrusted environment
Cannot be used for authentication	Can be used for authentication
More keys are required n * (n − 1)/2	Less number of keys = 2n

Number of Participants	Number of Symmetric Keys	Number of Asymmetric Keys
2	1	4
4	6	8
10	45	20
50	1225	100
100	4950	200
1000	499500	2000

Table 2-5. Key Requirements Comparison for Symmetric and Asymmetric Key Techniques

2.7 GAME THEORY

Game Theory is used in many real-life situations to solve complex problems. It is used in Bitcoins and many other blockchain solutions. It was introduced by John von Neumann to study economic decisions. However, it became popular after the theory of "Nash Equilibrium," by John Forbes Nash Jr.

Game Theory is based on situations where one or more parties are involved and the actions that they perform are going to influence the outcome just like when playing games. In a game there are two or more players. Each player wants to win the game so every player plans a strategy as to how to win the game.

For example, in cricket or football, there are two teams which are competing with each other. There is a referee overseeing the game and there are rules for the game. Every team plays to win. In real life, there are various situations where you have to take a decision which will be beneficial for you that is also like playing a game. When you are appearing for an interview, the chances of your selection can also be considered as a game. Whenever we are in such situations we think about all the possible outcomes for every action that we take and finally select the one which is going to benefit us the most.Games are not just about the action that an individual performs, it is also about the actions that others perform. for example, when you are appearing for an interview, the

outcome of the interview is not just about how good your performance is, it is also about how the other candidates perform.

Game theory is a study of strategies involved in complex games. It is the art of making the best move, or opting for the best strategy in a given situation based on the objective. To do so, one must understand the strategy of the opponent and also what the opponent thinks your move is going to be.

Let us take a simple example: There are two siblings, one elder and the other younger. Now, there are two toy cars that they got as a gift, one is red and the other is blue. The elder one wants the red car, but knows if he opts for that, then the younger one would cry for the same car. So, he opts for the bluecar and it turns out as expected, the younger one wants the same. Now, the elder one pretends to have sacrificed the blue car and gives it to the younger one and keeps the red car to himself. This is a win-win for both the parties, as this was the objective of the elder one. If the elder one wanted, he could simply have fought with the younger kid and got the red car if that was his objective. In the second case,the elder one would strategize where to hit so that the younger kid is not injured much but enough so that he gives up on the redcar. This is game theory: what is your objective and what should be your best move?

Another example: Imagine a vendor supplies vegetables to a town. There are three ways to get to the town. One Is a regular or the usual route which is a shorter route and better,second is a shorter route but it is narrow and third is a longer route that is wide. One day there is a block on the regular route. Now, the vendor has to choose a route. The second route is narrow and it will be the obvious choice by everyone. Hence, it may get congested. The third route is longer and may add to the fuel costs but the vendor will definitely reach faster. The vendor has to strategize his moves. If he reaches before everyone he can sell his vegetables at higher cost and recover the additional fuel expense. This is game theory: what is your best move for the objective you have in mind, which is usually finding an optimal solution.

The role that you play and your objective both play a vital role in formulating the strategy. Example: If you are an organizer of a sport event, and not a participant in the competition, then you would formulate a strategy where your objective could be that you want the participants to play by the rules and follow the protocol. This is because you do not care who wins at the end, you are just an organizer. You should consider if there could be a situation where a participant breaks a rule and loses one point but injures the opponent so much that they cannot compete any longer. So, you have to take into account what the participants can think and set your rules accordingly.

On the other hand, a participant would strategize the winning moves by taking into account the strengths and weaknesses of the opponent, and the rules imposed by the organizer because there could be penalties if you break the rules.

Game theory is the method of modeling real-life situations in the form of a game and analyzing what the best strategy or move of a person or an entity could be in a given situation for a desired outcome.

Game theory concepts can be used to design systems where the participants play by the rules without assuming emotional or moral values of them. It can help you build robust solutions and lets you test those with different interesting scenarios.

2.7.1 Nash Equilibrium

Games can be classified as cooperative/non-cooperative games, symmetric/ asymmetric games, zero-sum/non-zero-sum games, simultaneous / sequential games, etc. Nash equilibrium is related to cooperative/non-cooperative games.

In cooperative games, the players cooperate with each other and can work together to form an alliance. An external force can also be applied to ensure cooperative behavior among the players. In non-cooperative games, the players compete as individuals with no scope to form an alliance. The participants justlook after their own interests. Also, no external force is available to enforce cooperative behavior.

Nash equilibrium states that, in any non-cooperative games where the players know the strategies of each other, there exists at least one equilibrium where all the players play their best strategies to get the maximum profits and no side would benefit by changing their strategies. If you know the strategies of other players and you have your own strategy as well, if you cannot benefit by changing your own strategy, then this is the state of Nash equilibrium. Thus, each strategy in a Nash equilibrium is a best response to all other strategies in that equilibrium.

A player may strategize to win as an individual player, but not to defeat the opponent by ensuring the worst for the opponents.Also, any game when played repeatedly may eventually fall into the Nash equilibrium.

2.7.2 Prisoner’s Dilemma

Prisoner’s dilemma is an example of a non-zero-sum game. It is a symmetric game because changing the identities and strategies of the players does not change the payoff of the game.

Prisoner’s dilemma is about two criminals, Rob and Smith, caught by the cops for selling drugs. They are kept in different cells for interrogation. They are informed that the sentence for drug dealing is two years. The cops suspect these two could be involved in a robbery that happened last week. The cops have to strategize a way to get to the truth.

The cops go to Rob and give him three choices:

1. If Rob confesses the robbery and Smith does not, then his punishment would be reduced to just one year and Rob will get five years.

2. If Rob denies and Smith confesses, then Rob gets five years and Smith gets one year.
3. If both confess, then both get three years of imprisonment.

The cops give the same choice to Smith. This situation is called the prisoner's dilemma.

Rob and Smith are in two different cells and cannot communicate with each other. Both cannot decide to deny it and just get out in two years in jail for the drug dealing case, which seems to be the global optimum in this situation. The biggest problem is even if they could talk to each other, they may not really trust each other.

Each has two choices, confess or deny. Each knows that the other would choose what is best for him.

If he denies and the other confesses, then he is in trouble by getting five years of jail and the other gets just one year of jail. He certainly does not want to get into this situation.

If he confesses, then other has two choices: confess or deny.

Now Rob thinks that if he confesses, then whatever Smith does, he is not getting more than three years.

Let us state these scenarios for Rob.

Rob confesses and Smith denies—Rob gets one year, Smith gets five years (best case given Rob confesses)

Rob confesses and Smith also confesses—Both Rob and Smith get three years (worst case given Rob confesses)

This situation is called Nash equilibrium where each party has taken the best move, given the choices of the other party. This is definitely not the global optimum, but represents the best move as an individual.

Nash equilibrium is the most stable stage where changing your decision does not benefit you at all. It can be pictorially represented as shown in Figure 2-17.

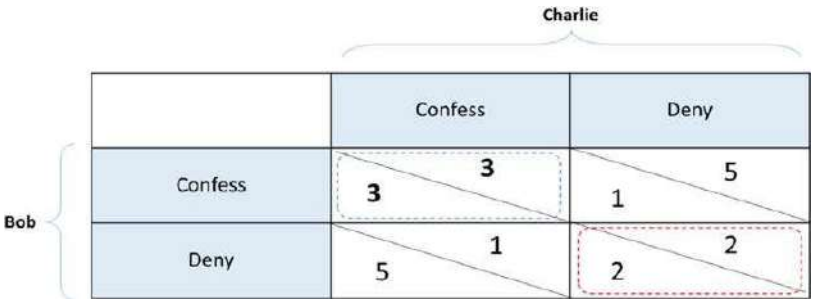


Figure 2-17. Prisoner's dilemma—payoff matrix
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

2.7.3 Byzantine Generals' Problem

The Byzantine Generals' Problem was a problem faced by the Byzantine army while attacking a city. Several army factions commanded by separate generals surrounded a city to win over it. The only chance of victory was when all the generals attacked the city together. However, the problem was how to reach a consensus i.e.either all the generals should attack or all of them should retreat.If some of them attack and some retreat, then chances are greater they would lose the battle.

Let assume there are five factions of the Byzantine army surrounding a city. They would attack the city if at least three out of five generals are willing to attack, but retreat otherwise. If there is a traitor among the generals, he will vote for attack with the generals willing to attack and vote for retreat with the generals willing to retreat.He can do so because there is no central coordination amongst the generals. There can be various situations based on the number of traitors like:

- ✔ Two generals attack the city and get outnumbered and defeated.
- ✔ There can be more than one traitor.
- ✔ Message co-ordination between the generals
- ✔ The messenger is caught/killed/bribed by the city commander
- ✔ Traitor general forges a different message and fools other generals
- ✔ Finding the honest and traitor generals

As you can see, there are so many challenges that need to be addressed for a coordinated attack on the city. It can be pictorially represented as in Figure 2-18.

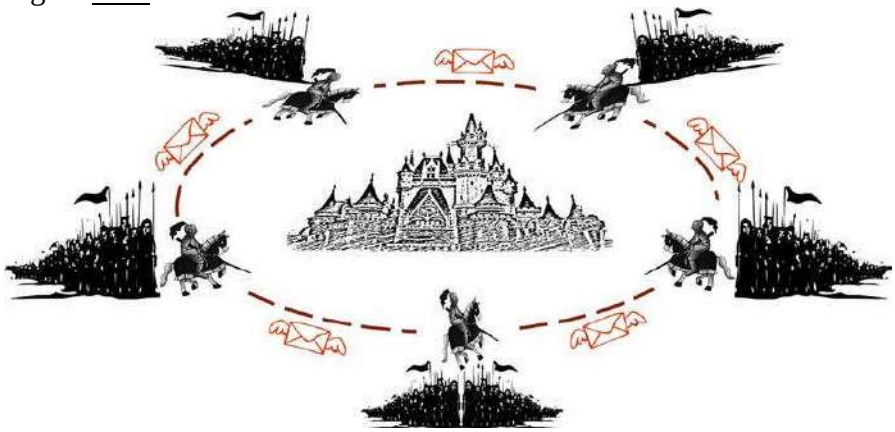


Figure 2-18. Byzantine army attacking the city
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

A group of people reaching consensus on some voting agenda or maintaining the consistent state of a distributed or decentralized database, or maintaining the consistent state of blockchain copies across nodes in a network are a few examples similar to the Byzantine Generals' Problem.

2.7.4 Zero-Sum Games

A zero-sum game is a game in which one player's gain is equivalent to another player's loss, one wins exactly the same point as the opponent loses.The choices made by players cannot increase nor decrease

the available resources in a given situation. Poker, Chess, Go, etc. are a few examples of zero-sum games. Only one person wins and the opponent loses in a zero sum game, such as tennis, badminton, etc. Financial instruments such as swaps, forwards, and options can also be described as zero-sum instruments. The amount is deducted from one account to be added to another account.

Insurance is an example of a zero-sum game. We pay an insurance premium to the insurance companies to guard against some difficult situations such as accidents, hospitalization, death, etc. We are compensated by the insurance companies when we face such tough situations. However, everyone who pays the premium does not meet with an accident or get hospitalized. If anyone does, they might need a lot of money compared with the premium they pay. So, the insurance company can make a profit by investing our premium amount and getting returns on it.

An interview is an example of a zero sum game. A candidate who is selected is at the cost of others rejection.

2.7.5 Why to Study Game Theory

Game theory is a revolutionary interdisciplinary phenomenon bringing together psychology, economics, mathematics, philosophy, and an extensive mix of various other academic areas.

Game theory is a part of our life because we take decisions everyday by thinking about the probable outcome. Game theory helps us in thinking differently.

In many real-world situations, the participants or the players are faced with a decision matrix similar to that of a “prisoner’s dilemma.” So, learning these concepts not only helps us formulate the problems in a more mathematical way, but also enables us to make the best move. It lets us identify aspects that each participant should consider before choosing a strategic action in any given interaction. It tells us to identify the type of game first; who are the players, what are their objectives or goals, what could be their actions, etc., to be able to take the best action. Much decision-making in real life involves different parties; game theory provides the basis for rational decision-making..

2.8 COMPUTER SCIENCE ENGINEERING

Computer science stitches the various components of cryptography, game theory, and many others to build a blockchain.

2.8.1 The Blockchain

A blockchain is actually a data structure. It is a chain of blocks linked together. A block can be made up of a single or multiple transactions.A hash pointer is a cryptographic hash of a data block. (Figure 2-19). In blockchain, hash pointers point to the previous block thus creating a link that cannot be tampered.

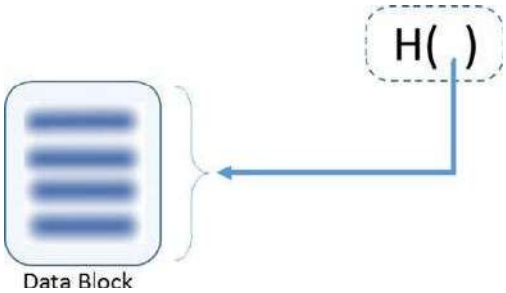


Figure 2-19. Hash pointer for a block of transactions #imgref:BeginningBlockchain-Singhal,Dhameja,Panda

The hash of the previous block is stored in the current block header, and the hash of the current block and its block header will be stored in the next block's header. This creates the blockchain as we can see in Figure 2-20.

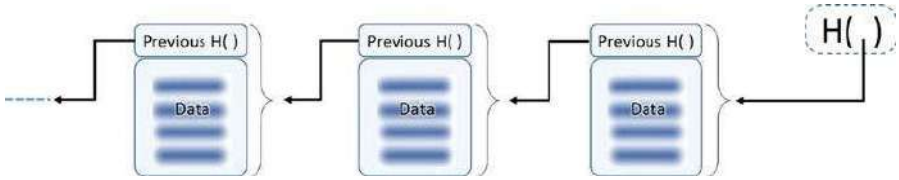
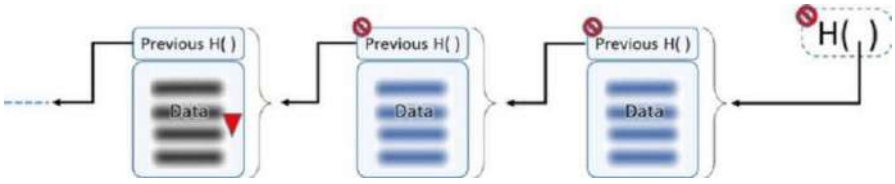


Figure 2-20. Blocks in a blockchain linked through hash pointers #imgref:BeginningBlockchain-Singhal,Dhameja,Panda

The first block in a blockchain is called the genesis block. Since, all the blocks are linked together with hashes it is impossible to tamper the contents of a block without changing the consecutive blocks.

Let us focus on Figure 2-21 to understand how it is not possible to alter the data in any way.



#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Figure 2-21. Any attempt in changing Header or Block content breaks the entire chain. Assume that you altered the data in block-178. If you do so, the hash that is stored in the block header of block-179 would not match.So one has to keep calculating and changing the hash of all the consecutive blocks until it reaches the latest block. It is extremely infeasible to do so. Also, the blockchain state is stored by many nodes in the blockchain,so all the nodes have to be updated which is extremely unlikely. Thus, hashing makes it a tamper-proof blockchain data structure.
The hash is calculated using either the SHA2 or SHA3 family of hash functions.It produces a 256-bit hash output which is represented using hexadecimal characters.

2.8.2 Merkle Trees

A Merkle tree is a binary hash tree. It is named after its inventor Ralph Merkle. Merkle trees are constructed by hashing paired data (usually transactions at the leaf level), then again hashing the hashed outputs all the way up to the root node, called the Merkle root. It is constructed bottom-up. In Bitcoin, the transactions of a single block become the leaves in a merkle tree. A typical Merkle tree can be represented as in Figure 2-22.

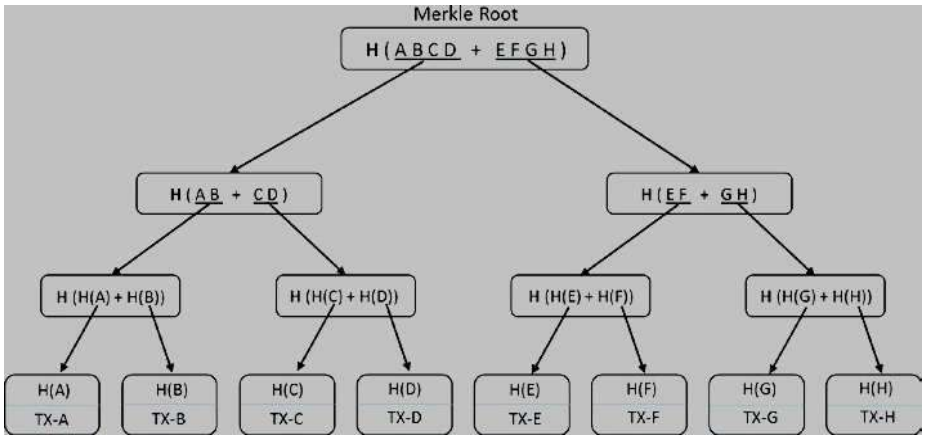


Figure 2-22. Merkle tree representation #imgref:BeginningBlockchain-Singhal,Dhameja,Panda

The Merkle tree is tamper-proof. A slight change in data at any level in the tree, the hash will not mash the hash stored one level up until the root node. It is really difficult for an adversary to change all the hashes in the entire tree. The hashes in the tree change even if the order of the transactions is changed.Merkle is a binary tree so if the number of transactions in a block is odd then the last transaction is duplicated to make it even.

Merkle trees are used to verify if a specific transaction belongs to a particular block. If there are “n” transactions in a Merkle tree (leaf items), then this verification takes just Log (n) time as shown in Figure 2-23.

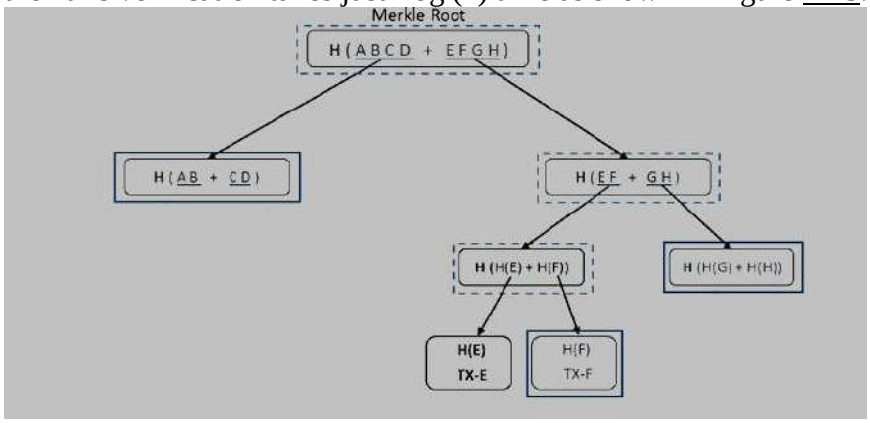


Figure 2-23. Verification in Merkle tree #imgref:BeginningBlockchain-Singhal,Dhameja,Panda

A subset of transactions is needed to verify if a transaction or any other leaf item belongs to a Merkle tree, we do not need all as we can see in the diagram in Figure 2-23. One can just start with the transaction to verify along with its sibling leaf item, calculate the hash of those two, and see if it matches their parent hash. Then continue with that parent hash and its sibling at that level and hash them together to get their parent hash. Continuing this process all the way to the top root hash is the quickest possible way for transaction verification. No of computations needed for verification of transactions is $\log_2 N$. Hence, for 8 transaction elements, only three computations (log2 8 = 3) would be required for verification.

Imagine a situation in a blockchain where each block has a lot of transactions. Since it is a blockchain, the hash of the previous block is already there; now, including the Merkle root of all the transactions in a block can help in quicker verification of the transactions. If we have to verify a transaction that is claimed to be from, say, block-22456, we can get the transactions of that block, verify the Merkle tree, and confirm quickly if that transaction is valid.

A node that has to verify if a certain transaction took place needs to verify: transaction as part of the block(Merkle), and block as part of the blockchain(Hash).

2.8.3 Code Snippet for Merkletree

```
# -*- coding: utf-8 -*- from hashlib import sha256
class MerkelTree(object): def init (self):
pass
def chunks(self,transaction,n):
#This function yeilds "n" number of transaction at time for i in range (0, len(transaction),number):
yield transaction[i:i+2]
def merkel_tree(self,transactions):
#Here we will find the merkel tree hash of all transactions passed to this fuction
#Problem is solved using recursion techqiue
# Given a list of transactions, we concatinate the hashes in groups of two and compute
# the hash of the group, then keep the hash of group.
We repeat this step till # we reach a single hash sub_tree=[]
for i in chunks(transactions,2): if len(i)==2:
hash = sha256(str(i[0]+i[1])).hexdigest() else:
hash = sha256(str(i[0]+i[0])).hexdigest() sub_tree.append(hash)
# When the sub_tree has only one hash then we reached our merkel tree hash.
#Otherwise, we call this fuction recursively if len(sub_tree) == 1:
return sub_tree[0] else:
```

```
return self.merkel_tree(sub_tree)
if name ==' main ': mk=MerkelTree()
merkel_hash= mk.merkel_tree(["TX1","TX2","TX3","TX4","TX5",
"TX6"])
print merkel_hash
#coderef:BeginningBlockchain-Singhal,Dhameja,Panda
```

2.9 PUTTING IT ALL TOGETHER

Cryptographic functions are deterministic. Given an input they always produce the same output. They are one-way and cannot be inverted. The hash value changes for even a small change in the input.

Public key cryptography can be used for authentication by digital signatures. It also helps in non-repudiation.

Game theory helps in designing sustainable robust systems. It helps in solving various situations that can arise when unknown entities are transacting with each other. The best way to handle such a system is to reward the participants playing by the rules and penalizing those who do not.

The blockchain data structure uses cryptographic hashes and merkle trees for building a tamper resistant chain of blocks. Merkle trees are specifically used for verification of transactions.

2.10 PROPERTIES OF BLOCKCHAIN SOLUTIONS

Immutability

A blockchain transaction is irreversible. Once a transaction is recorded, it cannot be altered. The transactions are broadcast to the network so all the nodes have a copy of the blockchain. As the number of blocks increase so does the immutability of the blockchain. It is not feasible for someone to alter the data of so many blocks in a series. A transaction that gets logged in the system remains forever in the system.

Forgery Resistant

Blockchain is decentralized in nature. Hence, it is prone to attack and forgery. Cryptographic hash and digital signatures used in blockchain ensure the system is forgery resistant. The transactions are signed using a private key and a hash is calculated of the same. Hence, it is impossible for anyone to forge it thus ensuring integrity and authentication.

Democratic

Blockchain does not have a centralized controller. All the nodes are treated equally. Hence, every participant has equal rights in any situation, and decisions are made when the majority reaches a consensus. Thus it is democratic in nature.

Double-Spend Resistant

When an amount is spent twice in more than one transaction, then it is called double spend. For example, you have 50$ in your account. You send 50$ to Alice and 40$ to Bob. This is double spend. This is difficult to track when there is no account balance system like in Bitcoin. In bitcoin, there is no notion of account balance. Every transaction has its source from other transactions you have received. Assume, you have received 100$ from Bob. You need to give 50$ to Alice. So, you have to give the 100$ transaction as input for spending 50$ to Alice. It is easy to prevent double-spend in a centralized system because the central authority is aware of all the transactions like our bank. So, the only way possible to prevent double-spend is to be aware of all the transactions. Hence, all the nodes in the blockchain have access to the entire blockchain right until the genesis block.

Consistent State of the Ledger

The state of the blockchain should be consistent across all the nodes of the network. To ensure the stability of the system, a consensus mechanism is used in blockchains.

Resilient

The network should be resilient enough to withstand temporary node failures, unavailability of some computing nodes at times, network latency and packet drops, etc.

Auditable

A blockchain is a chain of blocks that are linked together through hashes. Since the transaction blocks are linked back till the genesis block, auditability already exists and we have to ensure that it does not break at any cost. Also, if one wants to verify whether a transaction took place in the past, then such verification should be quicker.

2.11 BLOCKCHAIN TRANSACTIONS

Blockchain consists of blocks of transactions that are verified and then added to the block. Whenever an individual or an entity is making a transaction, they just have to broadcast it to the whole network. This transaction is validated by multiple nodes. Once validated, it is updated on all the nodes of the network as part of the blockchain. When the transactions happen every second, broadcasting individual transactions can become a costly affair. Hence, transactions are combined in blocks. It is also done to prevent a Sybil Attack. In a Sybil attack, people create replicas of their nodes to dominate the network.

Steps in blockchain transactions:

Every new transaction gets broadcast to all the nodes on the network so that all the computing nodes are aware of all transactions

Every transaction undergoes validation and authentication checks by the nodes. If it is valid, it is accepted, else rejected. The nodes further group

multiple transactions into blocks to share with the other nodes in the network. This is called proposing a block. Every node is given equal priority in the generation of new blocks. A consensus mechanism ensures that every node agrees upon a block. The blocks are time stamped in the order they arrive and get added to the blockchain.

Once the nodes in the network unanimously accept a block, then that block gets added to the blockchain by including the hash of the previous block. “Bloom filters” are widely used to test the membership of a transaction in a block.

2.12 Distributed Consensus Mechanisms

The biggest challenge in a decentralized system is achieving consensus about which node will propose the new block. The best strategy is that only one node should propose a block at a time and the rest of the nodes should validate the transactions in the block and add to their blockchains if transactions are valid. If any one node proposes a block and the rest of the nodes agree on it, then all those nodes add that block to their respective blockchains. The agreement about who proposes a block is called consensus, it comes from game theory. The system is designed in such a manner that players are rewarded for playing by the rules and penalized for bad behavior. Hence, the reward becomes a reason for everyone playing by the rules. Players would shy from breaking rules irrespective of the anonymity offered by bitcoins. The reward is the driving factor for a stable system. Players can use multiple identities to reap rewards.The goal of consensus is to ensure that the network is robust enough to sustain various types of attacks. The consensus algorithm has to fall into the Byzantine fault tolerant consensus mold to be able to get accepted.

2.13 PROOF OF WORK

The PoW consensus mechanism is used in Bitcoin. The idea behind the PoW algorithm is to essentially do a lot of computation for a block of transactions before it gets proposed to the whole network.APoW is actually a piece of data that is difficult to produce in terms of computation and time, but easy to verify.

PoW was initially used to prevent email spams. If a lot of computation is to be done before one can send an email, then spamming a lot of people would require a lot of computation to be performed which can act as a deterrent to an adversary. Similarly, in blockchain, a lot of computations need to be done before proposing a block. Hence, proposing a block is a lot of hard work which is rewarded. However, if someone is playing mischief and injecting a fraudulent transaction then rejection of that block by the rest of the nodes will be very costly.

The difficulty of the work before the block proposal should be adjustable so that there is a control over how fast the blocks can get generated.

Blockchain uses a concept of difficulty level. The user has to find a number below the difficulty level. It can be found by trying out a lot of numbers(puzzle friendliness). So, proposing a block is finding a value by trying all possible values to meet the criteria. The difficulty level is adjusted further after every new block to keep a check on how fast a new block is proposed.

Since, all the participating nodes are solving it, it is impossible to predict which node would solve it first. Once the puzzle is solved, that node proposes a block. In case of public blockchains, the nodes that are investing their computing resources are rewarded for their honest behavior.(block reward)

2.14 PROOF OF STAKE

The Proof of Stake (PoS) algorithm is another popular distributed consensus algorithm.It is used for validating blocks of transactions and not for mining new coins.

In PoS systems, the validators have to bond their stake,i.e. mortgage some amount as stake, to be able to participate in validating the transactions. The probability of a validator being selected is proportional to their stake; the more the amount at stake, the greater is their chance to validate a new block of transactions. A miner only needs to prove they own a certain percentage of all coins available at a certain time in a given currency system. For example, if aminer owns 5% of all Ether (ETH) in the Ethereum network, they would be able to mine 5% of all transactions across Ethereum. Decisions about which node gets to create the new block of transaction is based on the PoS algorithm being used. There are various types of PoS algorithms such as naive PoS, delegated PoS, chain-based PoS, BFT-style PoS, and CasperPoS, to name a few. Delegated PoS (DPOS) is used by Bitshares and Casper PoS is being developed to be used in Ethereum.

The creator of a block in a PoS system is based on the amount at stake and henceis much faster compared with PoW systems. There are no block rewards in PoS. Validators receive transaction fees.ThePoS systems provide better protection against malicious attacks because an attacker has to put an amount at stake and executing an attack would risk the entire amount at stake. PoS is not compute intensive like PoW hence it saves a lot on resources like electricity and consuming CPU cycles.

2.15 PRACTICAL BYZANTINE FAULT TOLERANCE ALGORITHM (PBFT)

Hyperledger, Stellar, and Ripple use PBFT consensus. PBFT does not generate rewards like PoW. Every node has a replica of the internal state. The requests are broadcast to all participating nodes.On receiving a request, a node performs the computation based on their internal states.

The outcome of the computation is shared with all other nodes in the system. Hence, every node is aware of the computations done by other nodes. The results are compared and a final value is agreed and committed upon.Every Node is aware of the final value and a final consensus is achieved. This is demonstrated in Figure 2-24.

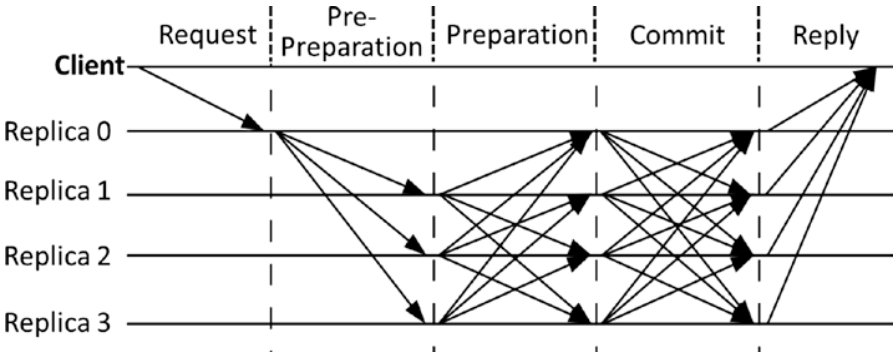


Figure 2-24. PBFT consensus approach
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

PBFT is efficient compared with other consensus algorithms. However, anonymity in the system may be compromised because of the way this algorithm is designed. It is one of the most widely used algorithms for consensus even in non-blockchain environments.

2.16 BLOCKCHAIN APPLICATIONS

Blockchain can be implemented fully for creating a decentralized system or partially by using blockchain as a backend. Bitcoin blockchain is an example of a decentralized blockchain application where every transaction is broadcast to the entire network. A web application is built and hosted in a centralized web server that makes Bitcoin blockchain updates when required.

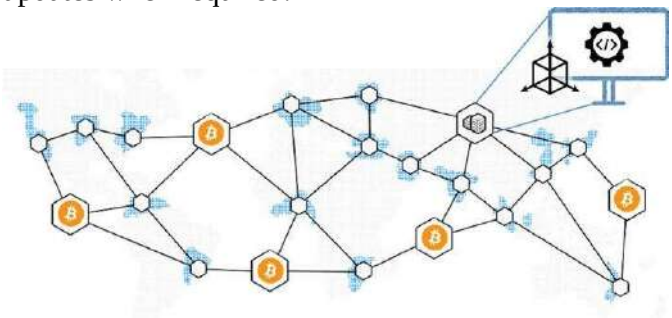


Figure 2-25. Bitcoin blockchain nodes
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Every node is self-sufficient and maintains its own copies of the blockchain database. Blockchain applications with no centralized servers are the purest decentralized applications; they are usually public blockchains. Public blockchains rarely use the infrastructure from cloud service providers whereas private blockchains use cloud.

There could be one or more web applications for different departments or actors with Blockchain as their backends. These blockchains are in sync with each other. This is an example of technical decentralization. However, it is politically centralized. The system is able to maintain transparency and trust because of the accessibility to a single source of truth. Take a look at Figure 2-26, which may resemble most of the blockchain POCs or applications being built on blockchain where blockchains are hosted by some cloud service provider by consuming their blockchain-as-a-Service (BaaS) offering.

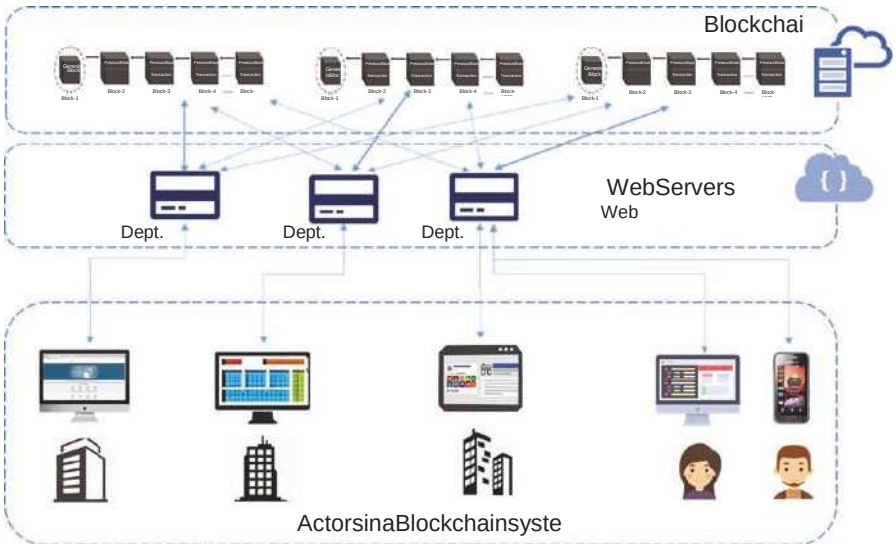


Figure 2-26. Cloud-powered blockchain system
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

One web application can handle requests from multiple different actors in the system with proper access control mechanisms. All the actors in the system have their own copies of blockchains which helps in maintaining transparency in the system as well to generate data-driven insights with ready access to data all the time.

The different “blockchains” maintained by different actors in the system are consistent by design because of consensus algorithms such as PoW, PoS, etc.

PoS is preferred over Pow by the private blockchains because of PoW’s heavy resource consumption, electricity and computing power requirements.

Decentralized applications (DApps) are being built on Ethereum blockchain networks. These applications could be permissioned on private Ethereum or could be permissionless on a public Ethereum network. Figure 2-27 gives a high-level understanding of how those applications might look.

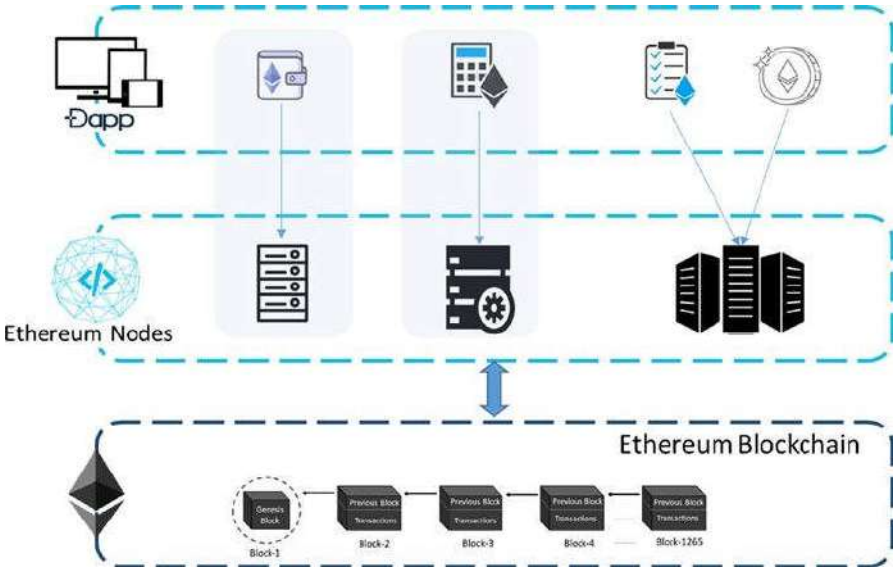


Figure 2-27. DApps on Ethereum network
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Blockchain applications can be Pure blockchain native applications or applications that treat blockchain as just a backend,and there are hybrid applications that use the legacy applications along with the blockchain for some specific purpose.

2.17 SCALING BLOCKCHAIN

Blockchains are difficult to scale. Bitcoins are not a replacement to fiat currencies.It cannot be used for day to day monetary transactions which are carried out by our debit or credit card.

Consensus protocols are used for agreement between all the communicating nodes for stability of the system. In a blockchain network, every node maintains its own copy of the entire blockchain, validates all transactions and blocks, serves requests from other nodes in the network, etc. to achieve decentralization. This can lead to latency. Increase in the number of nodes provides stability however it also increases the number of transactions hence adding to the load of computing and storage requirements. This is a common cause for concern in public blockchains. Private blockchains on the other hand can be easily scaled because the controlling entities could define and set node specifications with high computation power and more bandwidth or use off-chain computations.

2.17.1 Off-Chain Computation

Off-chain computation is outsourcing the resource intensive operations and limiting only the storage of outcomes on blockchain nodes. There are different variants of off-chain computation depending on people as well as the computation needs and limitations of the nodes involved. It can be considered as a layer on top of the blockchain which is responsible for processing involved in a blockchain. Off-chain computation can be done by a dedicated computational node(sidechain) or it can be distributed

by a centralized node or amongst a random group of nodes. Off-chain computation helps in scaling the blockchain, it isolates damages to the sidechain and prevents the main blockchain from any damages from a sidechain. The “Lightning Network” for Bitcoins is an example of a sidechain that helps in faster execution of transactions. “Zerocash,” in bitcoins is another example of a sidechain used for privacy. Off-chain computation does not affect a node's capability of carrying out or verifying transactions. A user with his own private key can carry out a transaction by signing it. Bitcoins do not have a concept of accounts. Every transaction has its origin or funding from another transaction. So, if a user is spending some amount, he has to show some previous transactions where he received that amount. Ethereum on the other hand has accounts so it is important to maintain state information.

Let's assume Alice and Bob carry out a lot of transactions in a month. All these individual transactions would have their state information which will be maintained by all the nodes in a stateful blockchain. The concept of “state channels" is introduced to address this challenge. The state channel is updated with utmost security using cryptography periodically or when a certain transaction threshold is reached. State channels are essentially a two-way communication channel between users, objects,or services.

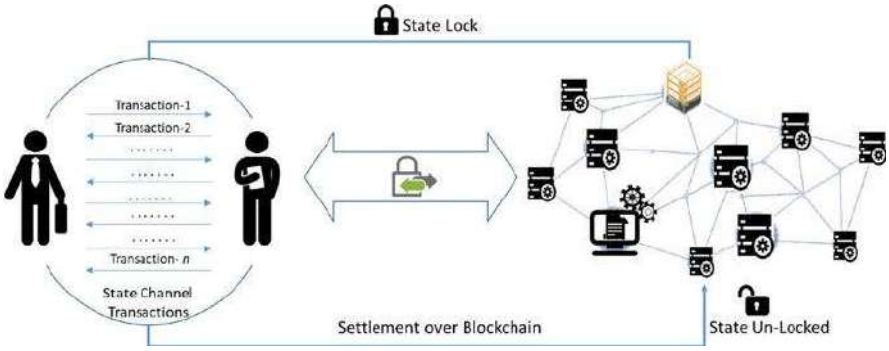


Figure 2-28. State channels for off-chain computation #imgref:BeginningBlockchain-Singhal,Dhameja,Panda

The off-chain state channels are private and confined to a group of participants. The state of blockchain for the participants is locked in the beginning by using MultiSig scheme or a smart contract-based locking.

The participants make cryptographically secured transactions among each other. Since the transactions are cryptographically signed, they can be verified and submitted to the blockchain.

The state channels could have a predefined lifespan, or could be bound to the amount of transactions being carried out in terms of volume/quantity or any other quantifiable measure. The final outcome of the transactions is saved on the blockchain and that unlocks the state as the final step.The Lightning Network is an Off-chain computation network for Bitcoin whereas the “Raiden Network” was designed for Ethereum blockchain.

2.17.2 Sharding Blockchain State

Sharding is a concept of slicing databases for easy processing. Operations like Disk read/write always lead to bottlenecks while dealing with huge data sets. The data is partitioned across multiple disks so that the read/ write could be performed in parallel leading to reduced latency. This technique is called sharding. Take a look at Figure 2-29.

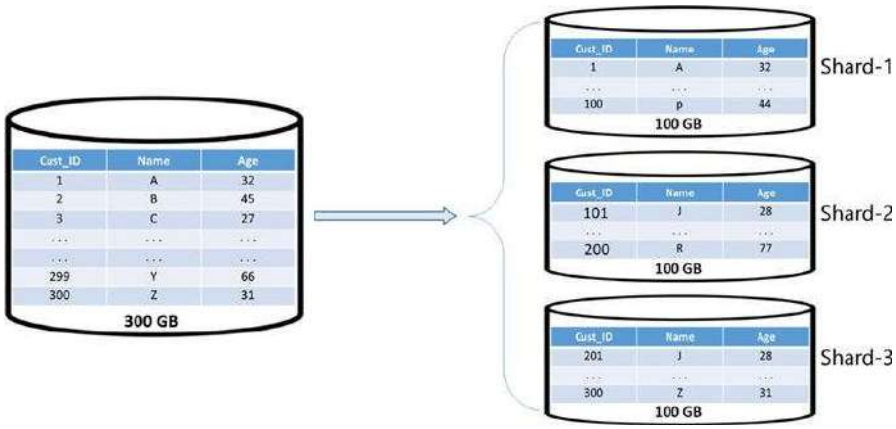


Figure 2-29. Database sharding example
#imgref:BeginningBlockchain-Singhal,Dhameja,Panda

Notice in Figure 2-29 , a300GB database table is partitioned into three shards of 100GB each and stored on separate server instances. The same concept can be applied for blockchain. The complete blockchain state is divided into different shards which contain their own substates i.e. a node need not store the entire blockchain, it can just store portions or shards relevant to it. When a transaction occurs, it is routed to only specific nodes depending on which shards they affect. It is not mandatory for all the nodes to perform calculations and verifications for each and every transaction. A mechanism or a protocol could be defined for communication between shards when multiple shards are needed to process transactions. Each blockchain can implement its variant of sharding.For example, using unique account wise shards(applicable in Ethereum).

2.18 SUMMARY

This chapter deals with the core fundamentals of cryptography, game theory, and computer science engineering. These concepts play an important role in Blockchain implementations. We learned various cryptographic algorithms used in blockchain. These techniques help in solving various issues related to security of the transmission of data. Game theory presents different perspectives when working in a trustless system.

2.19 QUESTIONS:

1. What is symmetric key cryptography?
2. What is asymmetric key cryptography?
3. How does game theory influence Blockchain?
4. What is ECDSA?
5. What is Off-chain computation and sharding?

2.20 REFERENCES:

A. Bikramaditya Singhal, Gautam Dhameja, Priyansu Sekhar Panda 2018 31

B. Singhal et al., Beginning Blockchain, https://doi.org/10.1007/978-1-4842-3444-0_2

Unit II

3

ETHEREUM

Unit Structure

3.0 Objectives
3.1 Introduction
3.2 Three parts of Block chain
3.3 Ether as currency and commodity
3.4 Building Trustless systems
3.5 Smart Contracts
3.6 Ethereum Virtual Machine
3.6.1Wallets as a computing metaphor
3.6.2The Bank teller metaphor
3.6.3Breaking with banking history
3.6.4How encryption leads to trust system
3.6.5 Using parity with Geth
3.6.6 Anonymity in Cryptocurrency
3.6.7 The Mist browser Installation
3.7 Central Bank Network of Yesterday
3.8 Virtual Machines
3.9 EVM Applications
1. State Machines
2. Guts of the EVM
3. Block, Mining place in the state transition function
4. Renting time on the EVM
5.Gas, Working with gas, Accounts, transactions and messages, transactions and messages, Estimating gas fees for operations, opcodes in the EVM
3.10 Summary
3.11 Questions
3.12 References

3.0 OBJECTIVES

At the end of this unit, the student will be to

- ✓ Describe the three parts of block chain
- ✓ Demonstrate Mist Browser
- ✓ Explain the EVM and EVM applications
- ✓ Discuss the issues of Ethereum

3.1 INTRODUCTION

1. A block chain is a fully-distributed, peer-to-peer software network which make use of cryptography to securely host applications, store data and easily transfer digital instruments of value that represents real-world money.

2. In Bitcoin and Ethereum, cryptography is used to conjur one secure computing environment out of thousands of similar machines, running with no central authority and no single owner.

3. The term “Ethereum” can be used refer to three distinct things: the Ethereum protocol, the Ethereum network created by computers using the protocol and the Ethereum project funding development of the two things listed in above line.

4. On the heels of bitcoin, Ethereum has become its own macrocosm, attracting enthusiast and engineers from numerous industries.

5. Many of civilization’s most nagging imperfections could become the domain of blockchain’s killer apps and the Ethereum protocol is widely considered to be the network where these distributed apps will spring up.

6. For programmers, the challenging thing about Ethereum isn’t usually the code, like most open source software projects, this one has on-ramps for people who already program in other environments.

7. For nonprogrammers, the challenge is divining how the ecosystem will develop and how the whole system will fit in to it. Claims that block chains will modernize the banking system, revolutionize insurance and lay waste to counterfeiting may be overblown, but by how much is again a question mark case.

8. Open source block chain networks such as Ethereum and Bitcoin are kits that allow us to pop up an economic system in software, complete with account management and a native unit of exchange to pass between accounts. Just like game monopoly.

9. People call these native units of exchange coins, tokens or cryptocurrencies, but they are no different from tokens in any other system, they are a form of money(or scrip) that is usable only within that system.

10. Block chains work something like mesh networks or local area networks(LANS), they are merely connected to other “peer” computers running the same software. When you want to make one of these peer-peer(P2P) networks accessible through a web browser, we need to use special software libraries such as web3.js to connect an application’s front end via a javascript API’s to its back end(the block chain).

11. In Ethereum, we can take this concept one step further by easily writing financial contracts with other users inside the system, financial contracts are nothing but smart contracts.

12. The key component is this idea of a Turing-complete block chain. As a data structure, it works kind of the same way that bitcoin works, except the difference in Ethereum is, it has this built-in programming language, phrase by VitalikButerin, inventor of Ethereum.

13. In Ethereum, smart contracts are written in the programming language, known as solidity. Turing completeness was an advantage that many developers quickly latched onto, but more important is Ethereum's ability to save state.

14. In computing, a simple definition of a stateful system is one that can detect changes to information and remember them over time.

15. The ability to engineer interactions between users in the future and under certain conditions is a powerful addition to a blockchain. It allows developers to introduce control flow in to cryptocurrency transaction programming. This is the biggest distinction between Ethereum and Bitcoin, but not the only one, as described here.

16. In Bitcoin, all transactions happen as soon as possible. Because of Bitcoin's lack of statefulness,it has to execute transactions all in one go. The block chain as envisioned by Bitcoin's creators was a distributed transaction ledger that kept a running tally of everyone's bitcoin balances in the network(In Bitcoin, the network is written in the uppercase and bitcoin the token in lowercase).

17. The common scripting language makes it more straightforward for blockchains that share the Ethereum protocol to share data with one another, enabling groups that use separate block chains to share information and value with each other.

18. In a telecommunication context, a protocol is a system of rules that describes how a computer (and it programmer) can connect to participate in and transmit information over a system or network, whereas in Ethereum, the protocol is designed for building decentralized applications with emphasis on rapid development time, security and interactivity.

3.2 THREE PARTS OF BLOCK CHAIN

1. A block chain can be thought of as a database that is distributed or duplicated across many computers. The innovation represented by the word blockchain is the specific ability of this network database to reconcile the order of transactions, even when a few nodes on the network receive transactions in various order.

2. This usually happens because of network latency due to physical distance, for example, a transaction created by a user buying a hot dog in Tokyo will be dispatched first to nodes in Japan.

3. By the time a node in new York gets word of this transaction a few milliseconds later, a nearby transaction in Brooklyn sneaks in “ahead” of the one in Tokyo. These inconsistencies are there in current distributed system, and this makes a challenge for removing inconsistencies for handling distributed types of task.

4. So, to remove the inconsistencies present in current distributed system transactions, power of block chain system is deployed with various technologies to crack the problem.

5. A block chain is really the combination of three technologies as follows

5.1 Peer-to-Peer networking- A group of computers such as the Bit torrent network that can communicate among themselves without relying on a single central authority and therefore not presenting a single point of failure.

5.2 Asymmetric cryptography- I.A way for these computers to send a message encrypted for specific recipients such that anyone can verify the sender’s authenticity, but only intended recipients can read the message contents. II. In Bitcoin and Ethereum, asymmetric cryptography is used to create a set of credentials for your account, to ensure that only you can transfer your tokens.

5.3 Cryptographic hashing-A way to generate a small, unique “fingerprint” for any data, allowing quick comparison of large datasets and a secure way to verify that data has not been altered in both bitcoin and Ethereum, the merkle tree data structure is used to record the canonical order of transactions, which is then hashed in to a “fingerprint” that serve as a basis of comparison for computers on the network and around which they can quickly synchronize.

6. The combination of these three elements grew out of experiments with digital cash in the 1990s and early 2000s. Adam Back released Hashcash in 2002, which pioneered the use of mining to send transactions. The pseudonymous Satoshi Nakamoto added distributed consensus to this innovation with the creation of Bitcoin in 2009.

7. Together, these three elements can mimic a simple database that is decentralized and stored in the nodes of the network. In the same way that a group of ants constitute a functioning colony, you can think of Bitcoin as a machine. In computing terms, it’s a virtual machine, the particulars of which we’ll get into later.

8. Ethereum adds in computer science terms, a trustful global object framework messaging system to the paradigm established by the bitcoin virtual machine.

9. Ethereum assumes many chains- The Bitcoin we know today is not the only large-scale deployment of bitcoin software, Litecoin, for example

uses the bitcoin software modified as do dozens more. Ethereum was built with the assumption that there may be many blockchains and thus there should be a set of protocols in place by which they can communicate.

10. Ethereum network will be a distributed network of decentralized systems, enabling many different cryptographic tokens of value, with various purposes and interpretations to be easily and quickly defined and then brought to life.

11. The value of a bitcoin is determined by the market for bitcoins. Sure, certain bitcoin-holding entities have obtained domestic money transmitter licenses and will redeem your bitcoins for US dollars, Euros, gold or other flat currency. But these entities are private businesses that charge fees and could go out of business at any time.

12. So, bitcoin and networks like it are vulnerable only to the extent that there is no "redeemer of last resort", no trusted (government or corporate) entity you can be sure will redeem your bitcoins or ether for US dollars in the future. Short of paying a private money changer, the only option for converting bitcoins to something of real value is to connect to an online exchange and trade the coins for flat currency, thus finding another buyer.

13. Just as the Bitcoin network moves bitcoin tokens, the Ethereum network moves ether tokens.

3.3 ETHER AS CURRENCY AND COMMODITY

1. It's commonly said that the bitcoin isn't backed by anything, and that's true. Of course, modern fiat currenciesaren't backed by anything either. But they're different: endorsed by a government, a fiat currency is held by default by anyone paying taxes and buying government bonds. Some international commodities sales are denominated in dollars, too (for example, oil) giving people another reason to hold dollars.

2. For cryptocurrencies, challenges to adoption remain. Today, these digital tokens remain a fast, secure, public payment layer on top of the existing fiat money system; an experimental deployment that might someday grow to replace the centralized payments networking technologies used by companies like Visa and MasterCard today.

3. However, incredible possibilities are on the horizon as governments and private institutional investors begin to create large markets for inancial products and services denominated in ryptocurrencies.Central banks may even adopt the technology. As of this writing, at least one country has issued a digital dollar using Bitcoin software known as Bardados.

4. Gresham's Law

4.1 A currency that can buy a lot of valuable securities and assets is a currency worth saving. The Ethereum network allows anyone to write a trustworthy, self-executing financial contract (smart contract) that will move ether in the future. Conceivably, this could allow financial contracts that project far into the future, giving stakeholders in the contract a reason to hold and use ether as a store of value.

4.2 Originally applied to gold and silver currency, Gresham's Law states that in an economy, "bad" money drives out "good." In other words, people save and hoard currencies they expect to appreciate in value, while spending currencies they expect to depreciate in value.

4.3 Although the law is named for a 16th-century English financier, the concept appears to date all the way back to Medieval writings, and indeed all the way back to ancient texts including Aristophanes' poem "The Frogs," usually dated to around 405 BC- Coins untouched with alloys, gold or silver, Each well minted, tested each and ringing clear. Yet we never use them! Others pass from hand to hand.

4.4 For millennia, people have saved the value of their work-product in a monetary instrument that will stay stable, appreciate in value, or inflate in price—not something prone to crashing in value.

4.5 Today, crypto currencies are volatile in price, and are accepted by only a handful of governments and corporations worldwide as of this writing. Few, if any, decentralized smart contracts are in use in businesses today.

4.6 But by the same token, flat currencies issued by central banks have an awful historical record, demonstrably prone to bubbles, depressions and manipulation. Can crypto currency ever be real money and will it be better than the money to which we are accustomed?.

5. The path to Better Money

5.1 Today, Bitcoin(denoted by the ticker symbol BTC) is used by people, governments and corporations to transfer value and buy products or services. Each time they send bitcoins, they pay a small fee to the network, which is denominated in bitcoins. Ether, denoted by the ticker symbol ETH, can be used similarly.

5.2 First ether, has another use, it can pay to run programs on Ethereum's network. These programs can move ether now or in the future or when certain conditions are met.

5.3 Because of its ability to pay for the execution of transactions in the future, ether can also be considered a commodity, like fuel for the network to run applications and services. So it has an additional dimension of intrinsic value over bitcoins, it is not just a store of value.

5.4 Today, the overwhelming usage of flat currencies might suggest that crypto currencies are worse money- that is more prone to

worthlessness in the long run and yet bitcoins and ether are famously hoarded by holders, and even held in a trust by atleast one company where grayscale, a subsidiary of digital currency group. Meanwhile, central banks in the west experiment with near-zero interest rates and quantitative easing, also known as money printing in ever more dangerous and desperate attempts to keep inflation and deflation in check.

5.5 Crypto currencies are being drawn in to the market by higher process to service genuine demand. This is reflected in the ever-increasing process of most cryptographic tokens, however volatile their prices in over period of time. This balancing act between hoarders, speculators and spenders creates a thriving and healthy marketplace for cryptocurrency and suggest that cryptotokens as an asset class are already serving the purpose of money and much more.

6. Cryptoeconomics and Security

6.1 One reason to bring up currencies and commodities in the discussion of smart contracts is to train ourself to think in terms of building economic systems in pure software, that's the promise of Ethereum.

6.2 The design of software systems with game theoretic rules constitutes the emerging field of cryptoeconomics, which may seem simple at first-an equity coin, for example-creates worlds of complexity when rendered in code. In fact, what makes systems like Ethereum and Bitcoin so secure is that they are not based on any hack-proof technology but rather rely on powerful financial incentives and disincentives to keep malefactors at bay.

6.3 These are attractive value propositions that every engineer and software designer should be excited about. But bootstrapping currency(or strip) coins is an altogether separate, added challenge to getting people excited about end-user applications.

6.4 Although the most obvious applications of this software might be found in financial services, future applications may also use the same levers-trust, transactions, money and scripting-for entirely other purposes.

7. Back to the Good Old Days

7.1 It's true that Bitcoin and Ethereum add a bit of complexity—economics—to writing software programs. But they are also simpler in some ways; working with decentralized protocols is similar to working with computers of the 1970's.

7.2 They were enormous and expensive shared resources, and individuals could rent time on these machines from a university or corporation that owned one. The Ethereum network functions as one large computer which executes programs in lockstep; it is a machine which is "virtualized" by a network of other machines.

7.3 Being composed of many private computers, the Ethereum Virtual Machine (EVM) itself can be said to be a shared computer which is ownerless.

7.4 Changes to the EVM are achieved through hard forking: persuading the entire community of node operators to upgrade to a new version of the Ethereum software.

7.5 Changes to the network can't simply be pushed by the core development team. They involve a political process of persuasion and exposition. This ownerless configuration is meant to maximize uptime and security, while minimizing the incentive for subterfuge.

8. Cryptochaos

8.1 Everybody who looks at blockchain development for the first time feels overwhelmed. It's a new technology, things are changing rapidly, and expertise in decentralized systems is rare.

8.2 Nobody knows what's coming next, but it's clear the technology is working—to the tune of over $26 billion USD (as of this writing), which is roughly the market capitalization of all cryptocurrencies combined. Retailers big, small, online, and offline are beginning to accept payments in digital coins.

8.3 The Ethereum project is built with new developers in mind, and gives you the tools to create unheard-of solutions to age-old problems. It's up to developer to figure out what to build with this powerful new toolset.

3.4 BUILDING TRUSTLESS SYSTEMS

1. With help solidity programming language, which is used for writing secure blockchain programs to improve or automate the end-user experience of all sorts of businesses and enable the creation of new kinds of products and services.

2. As we are seeing how the banking products and services we know today, which evolved over a thousand years of trial-error, can change, benefit or be brought to scale by trustless distributed or semi distributed systems.

3. Trustless is used in this context to mean "Not requiring faith that counterparties will operate honestly and without failure, thus impervious to fraud and other counterparty risks".

3.5 SMART CONTRACTS

1. Ethereum and that is the notion of a smart contract, some business logic that runs on the network, semi-autonomously moving value and enforcing payment agreements between parties.

2. Smart contracts are often equated to software applications, but this a reductive analogy; they're more like the concept of classes in conventional object-oriented programming.

3. When developers speak of "writing smart contracts," they are typically referring to the practice of writing code in the Solidity language to be executed on the Ethereum network. When the code is executed, units of value may be transferred as easily as data.

4. Objects and methods for value

4.1 In computing, an object is usually a little chunk of data—information—encapsulated in a particular structure or format. Often this data has associated instructions called methods indicating how the object can be used or accessed. Now let's imagine the information held in this object is valuable to someone, and this person would be willing to pay to trigger a method which displays it.

4.2 In the example given below, let's imagine a user wants to pay a small fee to use a cake recipe he or she discovered online. This recipe is the data object in our example. At the most literal level, the characteristics of the cake object, called attributes, are stored along with the methods at a certain address in the computer's memory.

4.3 The object below represents the attributes of a cake, and contains a method whereby the computer can display instructions for how to combine these ingredients to make the cake. Storing the information in this way makes it easy for the program and the programmer to swap in and out the attributes without needing to change the code for the display instructions.

4.4 Javascript code for cake recipe is shown here

```
var cake = {
firstIngredient: "milk",
secondIngredient: "eggs",
thirdIngredient: "cakemix",
bakeTime: 22
bakeTemp: 420
mixingInstructions: function() {
return "Add " this.firstIngredient + " to " + this.secondIngredient + " and stir with " + this.thirdIngredient + " and bake at " + bakeTemp + " for " + bakeTime + " minutes." ;
}
```

4.5 This is an example of how computers "move" data around to display useful results to their human users. In Ethereum, you can write functions that send money around, just as this little object's method called mixing Instructions, when executed, can display the mixing instructions for a cake.

4.6 Solidity code can be used on the back-end of an application to add micro-payments, user accounts, and functionality to even simple computer programs, without the need for third-party libraries.

4.7 Imagine for a moment that the mixing Instructions function cost a few cents in ether to execute. After the price of the cake recipe is deducted from the user's Ethereum wallet balance—which takes a few seconds, on average—your smart contract would call the mixing Instructions method and show the user how to make the cake. All this can be done without authentication, payment APIs, accounts, credit cards, extensive web forms, and all the typical work that comes with building an e-commerce application.

5. Content creation

5.1 The cake recipe example showcases another big area of potential for Ethereum: intellectual property, licensing, and content royalties.Today, selling content on the Web or through apps means dealing with powerful distributors including Apple, Google, and Amazon, who make punitive rules about selling digital content and levy large fees.

5.2 Ethereum makes it possible to facilitate micro transactions whereby a user pays only, say, $0.25 for a recipe—an amount that would be impractical to pay using fee-laden credit-card networks. There are challenges to content creators doing business this way today, including the price volatility of the ether token also.

6. Where's the Data

6.1 All transactions in Ethereum are stored on the blockchain, a canonical history of state changes stored on every single Ethereum node. When you pay for computing time on the Ethereum network, this includes the cost of running the transaction and for storage of the data included in your smart contract.

6.2 As soon as you execute your smart contract and the fees are paid from your ether balance, that data will then be included in the next block. Because the Ethereum network requires all nodes to keep a full state database of all contracts, any node can query the database locally.

7. What is Mining?

7.1 A distributed system has no single owner, machines are free to join the Ethereum network at will and begin validating transactions. This process is known as mining.

7.2 Mining nodes confer to arrive at a consensus about the order of transactions across the system, which is necessary to tabulate everyone's account balances on the fly, even as many transactions pass through the network. This process consumes electricity, which costs money, and so miners are paid a reward for each block they mine, about 5 ether.

8. Ether and Electricity Prices

8.1 Miners are paid this ether for mining, and also for running scripts on the network (in the form of gas). The cost associated with electricity expenditure of servers running on the Ethereum network is one of the factors that gives ether, as a crypto commodity, its intrinsic value—that is, someone paid real money to their electricity company to run their mining machine.

8.2 Specialized mining rigs, which use arrays of graphics cards to increase their odds of completing a block and getting paid, can run up electricity bills anywhere from $100 to $300 a month per machine, depending on rates as per area.

8.3 Mining is fundamental to both Bitcoin and Ethereum, and in principle works similarly in both networks, with a few caveats.

3.6 ETHEREUM VIRTUAL MACHINE

1. The Ethereum Virtual Machine (EVM)—the name for the system just described—can be programmed, and to what ends. It is written in a way that should make sense to both financial and technical thinkers, so that developers and domain experts can more easily arrive at a common understanding of what they should build together, and which tools are right for their project.

2. In the realm of cryptocurrency software, there are generally two essential types of client applications wallets and full nodes. Wallet usually denotes a lightweight node that connects to a block chain to perform basic functions, such as sending and receiving cryptocurrency. Full nodes are command-line interfaces that can perform the full gamut of operations allowed by the network.

3. Ethereum can refer to both the Ethereum protocol and the Ethereum network created by computers using the protocol. Operating a node on the network allows you to upload smart contracts. For sending and receiving cryptocurrencyall you need is a wallet application for your computer or smartphone.

4. Ethereum has several client applications, like mist browser is one of them, the mist browser, a user friendly wallet that can perform some of the duties of a full node-namely executing smart contracts.

5. Eventually, entire web-app-like programs will be accessible through Mist, with their back ends built on Ethereum; that’s why it’s called a browser. Today, it’s useful for sending and receiving the ether cryptocurrency. But tomorrow, it may also be a distribution point for consumer and enterprise software applications, almost like an App Store.

6. The term currency as in cryptocurrency refers to a fungible unit of value for the system, much like a token or scrip. The term fungible applied to a currency means “Mutually interchangeable”. In flat currency terms, one dollar can be said to fungible for another dollar.

3.6.1 Wallets as a Computing Metaphor

1 Wallets are software applications for desktop or mobile devices that hold your keys to the EVM. These keys correspond to an account, which is referred to by a long account address. In Ethereum, accounts do not store your name or any other personal information. They are pseudonymous. Anyone can generate an Ethereum account by connecting to the network with any Ethereum client (such as Mist).

2. If you’ve already downloaded an Ethereum wallet or full node on your computer or phone, you were probably prompted to create an account. The wallet application probably also asked you to create a password to protect your keys with encryption. As you can gather, these keys are an important part of sending and receiving ether.

3. Let’s begin by looking at your account address, also called a public key. Your public key has a matching private key that allows access to your account. This private key should be kept secret and not published anywhere.

4. Accounts in both Bitcoin and Ethereum are represented by long hexadecimal addresses. An Ethereum address looks like this: 0xB38AA74527aD855054DC17f4324FE9b4004C720C. In the Bitcoin protocol, the raw hexadecimal address is encoded in base 58 with a built-in version number and checksum, but underneath looks just like an Ethereum address. Here’s an example of a Bitcoin address: 1GDCKfdTo4yNDd9tEM4JsL8DnTVDw552Sy.

5. To receive ether or bitcoins, you must give the sender your address, which is why it’s called a public key. An account is a data object, an entry in the block chain ledger, indexed by its address, containing data about the state of that account, such as it balance. An address is a public key belonging to a particular user, it’s how user access their accounts. In practice, the addresses is technically the hash of a public key, not the public key itself.

6. In the EVM, asymmetric cryptography is used by the network to generate and recognize valid Ethereum addresses, and also to “digitally sign” transactions. In secure communications, asymmetric cryptography is used to encipher private communications, so that even if they are intercepted by enemies, they remain unreadable.

7. It’s important to note that ether is not contained in any particular machine or application. Your ether balance can be queried, and ether sent or received, by any computer running an Ethereum node or wallet. Even if the computer where your Mist wallet lives gets destroyed, never fear: all you need is your private key, and voila, you can access your ether from another node.

8. However, if you hand over your private keys to someone else, that person can access the EVM and pull your money out without you ever knowing. As far as the network is concerned, anyone with your private key is you.

9. Because the EVM is a global machine, it has no way of knowing which node you'll create a transaction from. Unlike today's web apps, Ethereum does not look for a "trusted" computer; it doesn't know your phone from any other phone. If this seems unusual, think of it like a bank ATM system, which provides account access for anyone holding your debit card number and your four-digit pin.

10. Losing your phone or computer to theft or destruction does not mean you lose your money, provided the following are true: You have backed up your private key. You didn't give your private key to anyone else.

3.6.2 The Bank Teller Metaphor

1. In a way, using a wallet or full node is like getting behind the bank teller's desk and being in control of your own money. Not in the sense that you can get paper cash, but in the sense that a bank teller controls a node within the bank's computer system that can execute transactions in a global database of transactions. A teller controls the bank's database, which connects to other bank databases.

2. In conventional banking, by extension, a paper check is a written instruction for the bank teller to make a transaction using the bank's computer system. On the check is your account number and a routing number.

3. In cryptocurrency, this legacy banking system—a hodgepodge of human and computer processes—is completely obviated by the use of an algorithmic consensus engine running on a peer-to-peer computer network.

4. Settlement and clearing of transactions happens on the network itself within seconds (or, with bitcoin, minutes) of the transaction being digitally signed and broadcast by a node. Thus it can be said in in cryptocurrency transaction that "the settlement is the trade."

5. Cryptocurrencies are different from the fiat currencies used by conventional banks, which are centralized. Your tokens are virtual, and your balance (along with that of everyone else who holds ether) is tabulated by the blockchain network. There is no tangible ether or bitcoin currency, although some third parties have created "collectible" coins preloaded with cryptocurrency.

6. Be extremely wary of any online service or organization that offers to hold, store, or act as custodian of ether, bitcoins, or any other cryptocurrency. The advantage of distributed public systems is to eliminate counterparties from transactions, and allow entities to transact on a peer-to-peer basis. The point is, you can hold these assets securely, without a custodian.

7. Do not use any wallet or online service that holds your private keys for you. Only use applications that store your private keys on your device.

8. The best way for new Ethereum programmers to visualize the concept of a blockchain is to imagine a paper transaction ledger that can be synchronized with other paper transaction ledgers around the world.

8.1 When a wallet application attempts to make a change to the database, the change is detected by the nearest Ethereum node, which then propagates the change around the network. Eventually, all the transactions are recorded on every ledger. In the abstract, this works something like the polygraph machine patented by John Isaac Hawkins in 1803. This was the first “copy machine,” although its name today is used to refer to so-called lie-detecting devices. Just like the polygraph, the blockchain is an apparatus for allowing many “machines” to change the state of a ledger in the same way, nearly simultaneously.

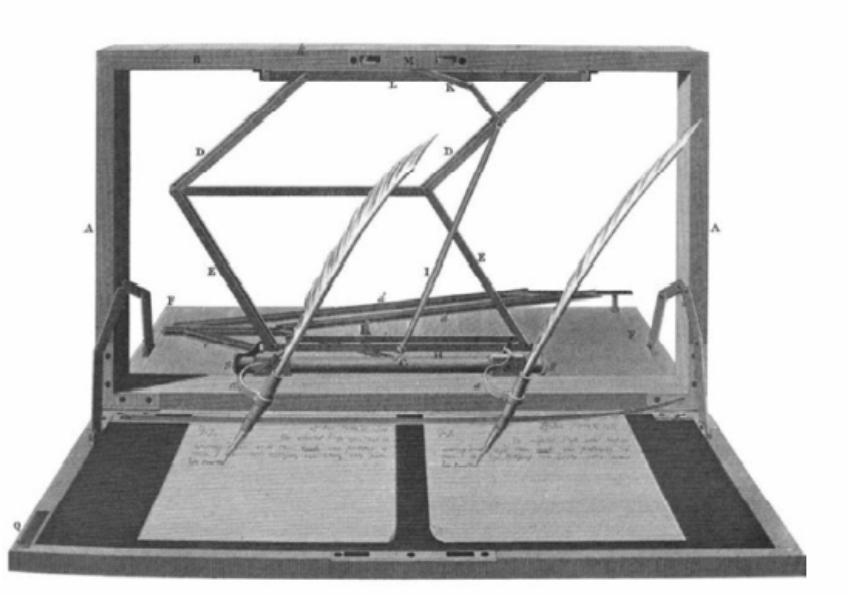


Fig 1 The Polygraph Machine

3.6.3 Breaking with Banking History

1. The only important to point out that (as with Bitcoin) no individual has the power to create more ether. This characteristic stands in stark contrast to the last 400 years of financial markets and central bankers, which reads like a history of large-scale scam artists.

2. Since the stock-jobbing days of the late 17th century in London’s Exchange Alley, entrepreneurs and scammers (then called stock projectors) have been selling equity in ventures both legitimate and not. Often they would secretly issue new shares to themselves and their confederates when the price would go up—known to Americans in the 19th century as watering the stock.

3. Over time, speculating on stocks became a pastime that people of all ages and backgrounds enjoyed on both sides of the Atlantic, and the

modern stock markets were born, with their processes and counterparties to act as middlemen who ensure trustworthy transactions..

4. But even with the banking regulations passed after the Great Depression, dishonest entrepreneurs still found ways to carve out secret stock pools, or unload the shares they had without the public knowing—only to let the business collapse after getting their money out.

5. Few times in modern history have speculative bubbles wiped out as much wealth and human progress as the crash of 1929 in the United States. However, similar depressive episodes in the United States and Europe (including the Panic of 1873–1879) were caused by someone, either central banks or investors themselves, messing with the base quantity of money, equities, or bonds in a large marketplace.

3.6.4 How Encryption leads to Trust

1. Asymmetric cryptography is a method of sending secure messages back and forth over a network, where the sender and the recipient do not trust the channel of communication. In the case of the EVM, those messages are transactions, being signed and sent to the network in order to change the state of some of its accounts. It's called "asymmetric" because each party has a pair of two different, but mathematically related, keys.

2. Public-key cryptography was developed for wartime communications, and when used properly, can be extremely secure. Unlike symmetric-key cryptographic, public key cryptographic communications don't require a secure channel between parties. This is essential in Bitcoin and Ethereum, because any computer running the protocol can join the network, without any vetting. However, the computational complexity involved in encrypting data makes it useful only for small data objects, like the alphanumeric string that becomes your private key. This is why encryption must be used sparingly.

3. At a high level, it can be said that Ethereum uses encryption to validate and verify that any and all changes made to account balances in the EVM are legitimate, and that no account has been increased (or decreased) erroneously. Here are some definitions that will help moving forward.

3.1 Symmetric Encryption- A process by which a snippet of plain text, usually held in a document, is smashed together with a shorter data string called a key to produce a ciphertext output. This output can be reversed, or decrypted, by the party that receives it, so long as they also have that same key. Trying to decode the message without the key would be, computationally speaking, immensely time-consuming and expensive—so much so that some kinds of encryption are considered practically unbreakable, even with huge computing resources.

3.2 Asymmetric Encryption-This way of encrypting information requires the program to issue two keys simultaneously, one that is public and one that you keep private. The public key is public in the sense that you can list it on your web site or social profile, such as an e-mail address.

3.3 Secure Messaging- Alice uses Bob's public key to encrypt a message. When he receives the ciphertext, he can decrypt it using his matching private key, ensuring that only Bob can read the message. This is called secure messaging. But it leaves a dangerous possibility open: anyone could send Bob a message claiming to be Alice. How does he know that Alice is the real sender of the message?.

3.4 Secure and Signed Messaging-If Alice wanted to assure Bob that she is the true sender, she would do things differently. First, she would take her plaintext message and encrypt it using her private key. Then, she would encrypt it again using Bob's public key. When Bob receives the message, he decrypts it first using his private key, but he's still left with ciphertext. He must decrypt it again using Alice's public key. This second layer of encryption assures him that Alice is indeed the sender, because presumably, nobody has Alice's private key but Alice. This is known as "secure and signed" messaging.

3.5 Digital Signature-For maximum security, Alice would take another step: she would hash the plaintext of her message, and attach it along with the message. She would then encrypt this bundle with her own private key, and again with Bob's public key. When Bob receives and decrypts the ciphertext he can run Alice's plaintext message through the same hashing algorithm Alice used. If for some reason the fingerprint of the message turns out differently, then it means the actual message text was damaged or altered an route.

3.6.5 Using Parity with Geth

1 Ethcore.io is a private Ethereum development company composed of a few former contributors to the Ethereum project, including Gavin Wood, another Ethereum project cofounder, who created the Solidity language and authored the Ethereum Yellow Paper.

2. He and his team have created a powerful node written in the Rust programming language. Parity works on macOS, Windows, Ubuntu, and in a Docker instance.

3.6.6 Anonymity in Cryptocurrency

1. Bitcoins and ether are not anonymous payment instruments. Anyone who knows your public key can look on the blockchain and see the dates and amounts of transactions coming in and out of your account. From this data, they might be able to put together a pattern of transactions from which they could deduce your activities. Federal authorities are already using machine-learning transactions to decode spending patterns on darkmarket sites such as AlphaBay.

2. Anonymity, secrecy, and privacy in cryptocurrency are generally conflated by newbies, sometimes with disastrous ends. Bitcoin and Ethereum addresses are pseudonymous by nature; they're not linked to your real name or information. But every transaction you send is public, in the sense that anyone can see the transaction on the blockchain. This is why public blockchains are touted for their transparency; if you know someone's public key, you can look up all their transactions..

3. Data within smart contracts themselves are encoded but not encrypted. Encryption is used only to hash large datasets and verify transaction senders and recipients. However, you can encrypt data yourself before putting it into an Ethereum smart contract, if you'd like to use the public Ethereum chains in a private manner.

3.6.7 The Mist Browser Installation

1. In simple term, Mist browser is user interface for Geth client. Geth is one of the Ethereum client that is used to connect to Ethereum networks and it provide command line utility to do various functions. Mist browser provide user interface to perform similar functions that we perform using Geth client command line utility.

2. To download the mist browser, use the link given here- https://github.com/ethereum/mist/releases

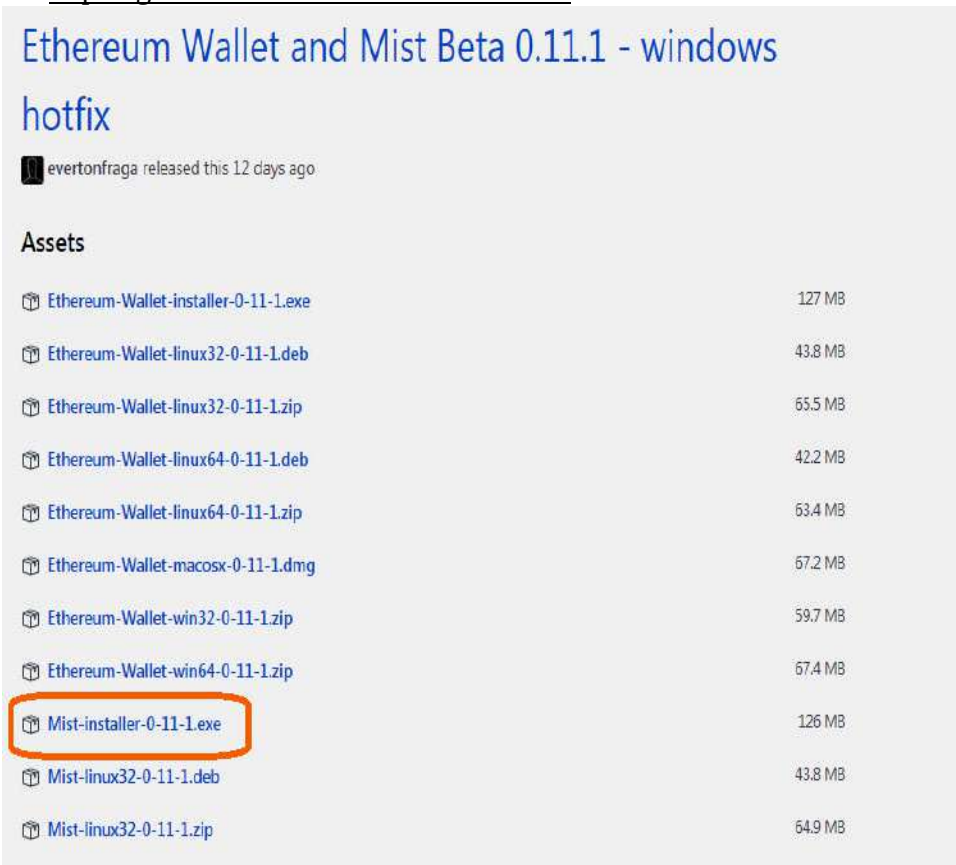


Fig 2 Choose the mist browser selected in orange color

3. Run the exe file and install the mist browser. Once you open the mist browser, it start connecting to peers/nodes and start downloading the block chain data. Click on launch application to launch the mist browser.

4. Once you open it, we can go to “develop” menu bar and check the “Ethereum Node” option that shows which version of Geth is being used.

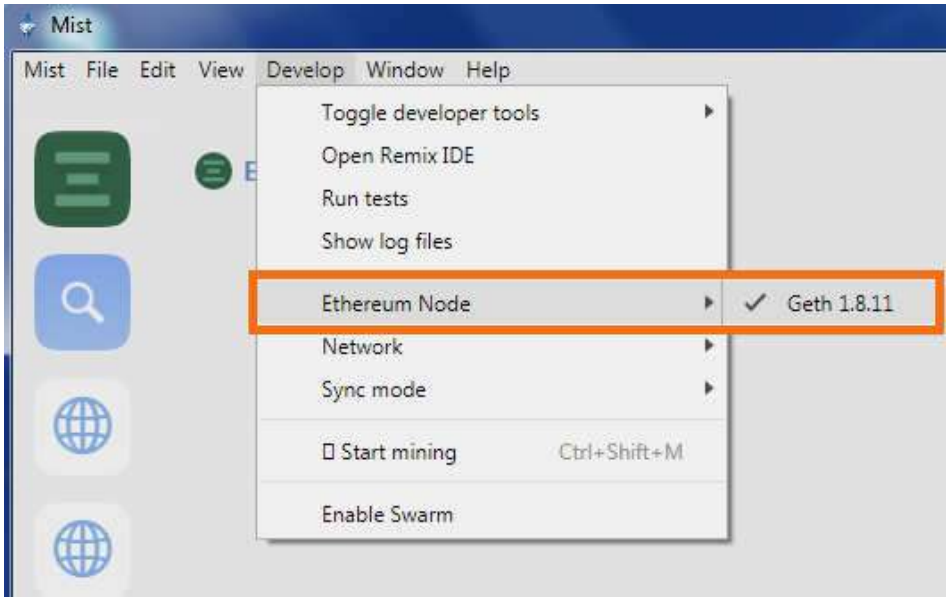


Fig 3 Mist Browser Ethereum Node

5. You can go to “Network” option under “develop” menu bar option to choose to which block chain network you want to connect.

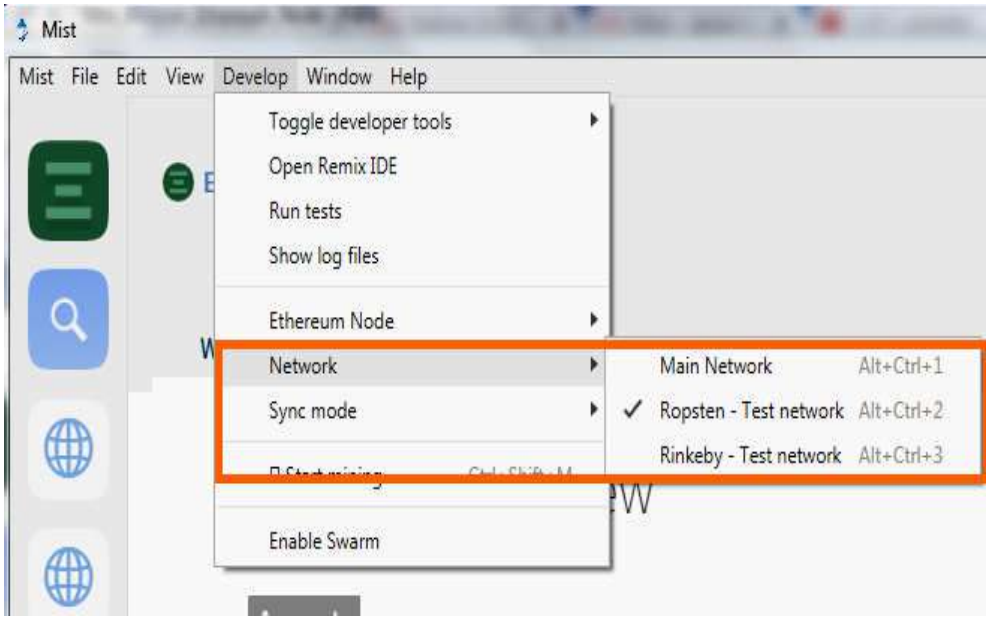


Fig 4 Mist Browser Network

6. You see one more option “Sync Mode” that show the mode that you would like to choose to connect with block chain network.

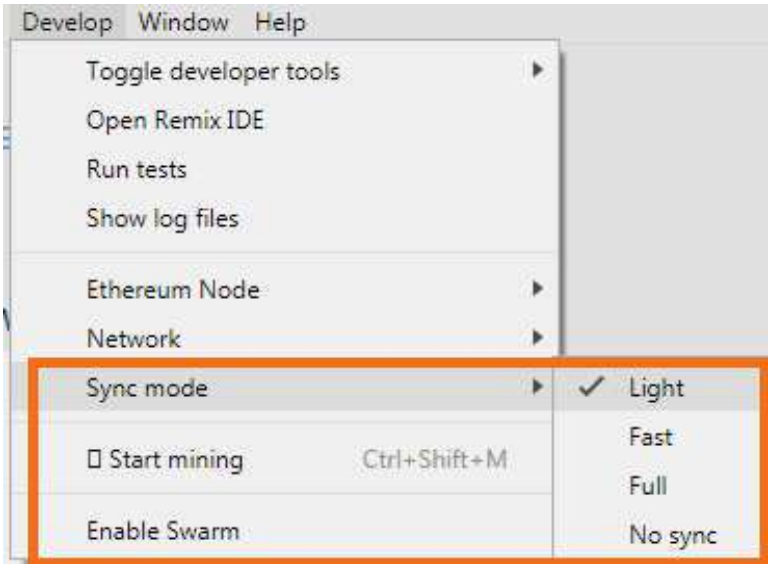


Fig 5 Mist Browser Sync Mode

7. You can also create accounts and develop smart contracts under wallet option as shown below.

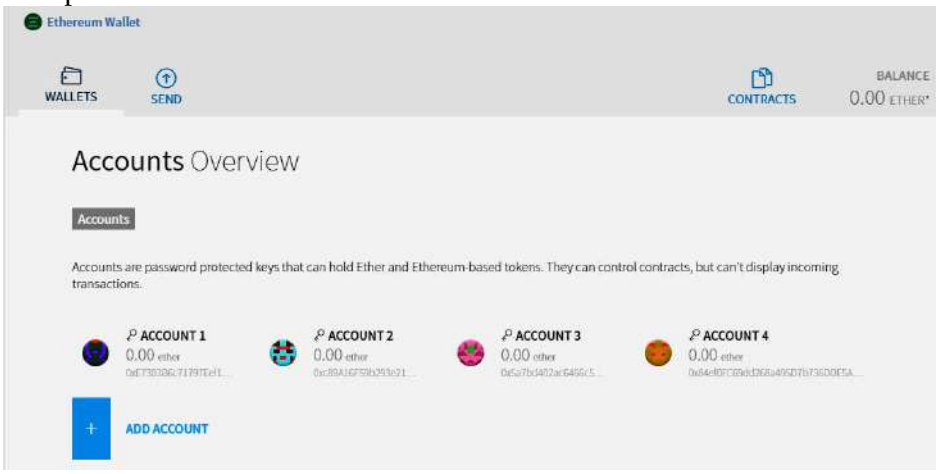


Fig 6 Mist browser Accounts

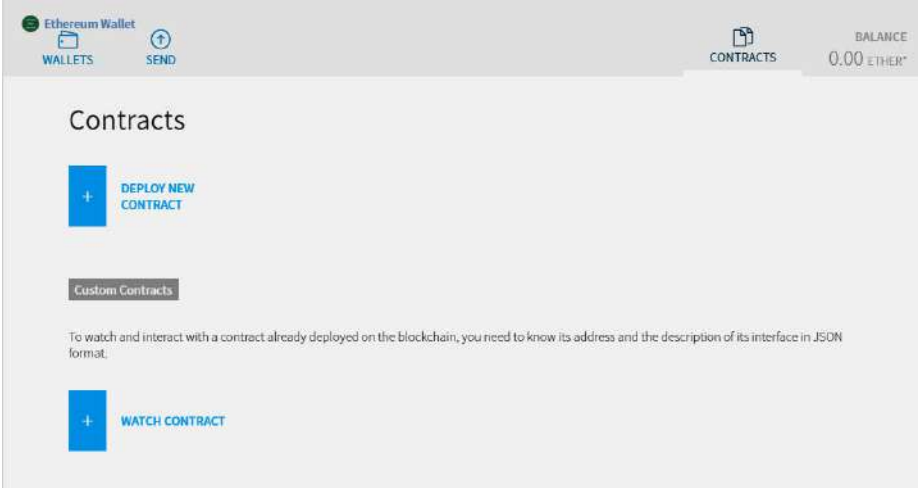


Fig 7 Mist Browser smart Contract

3.7 THE CENTRAL BANK NETWORK OF YESTERDAY

1. Today, corporations, insurers, universities, and other large institutions spend incredible amounts of money building and maintaining software services and IT for their own employees, and all their lines of business. Their various inflows and outflows are reconciled by large commercial banks, which have their own architecture, policy, codebase, databases, and layers of infrastructure. This, of course, is all on top of the Fedwire, which is the Federal Reserve's real-time gross settlement system, or RTGS.

2. The Federal Reserve is the central bank of the United States. The Fedwire is used by all Federal Reserve member banks to settle final payments in electronic US dollars. Any qualified state-chartered bank may become a member of the system by buying shares in it. Fedwire is owned and operated by the 12 Federal Reserve Banks themselves, and although it does charge fees, it isn't operated for profit.
3. This system processes unthinkable amounts of US dollars every day—trillions upon trillions. It has some great features, too: there's an overdraft system covering allexisting and approved accounts, and the system is famously reliable, even for remittances overseas. It has been in operation in some form or another for about 100 years.

4. As you can imagine, maintaining the security and reliability of the Fedwire software is extremely expensive. Yet, the cost of building and maintaining layers on top of an RTGS is higher still, owing to its security requirements. Ultimately, these costs are passed on to corporations who use commercial banks, in the form of fees. Those companies have their own IT infrastructure costs. In the aggregate these costs ultimately drive up prices and fees for consumers.

3.8 VIRTUAL MACHINES

1. Generally speaking, a virtual machine is an emulation of a computer system by another computer system. These emulations are based on the same computer architectures as the target of their emulation, but they're usually reproducing that architecture on different hardware than it may have been intended for.

2. Virtual machines can be created with hardware, software, or both. In the case of Ethereum, it's both. Rather than securely network thousands of discrete machines, as with Fedwire, Ethereum takes the approach of securely operating one very large machine that can encompass the whole Earth.

3. The Fedwire system is a settlement system with a user experience tailored to statechartered banks and their operators. It makes little or no concern for the end user of a retail bank, for example; that's the job of the retail bank.

4. Software developers will recognize Fedwire as a "platform for banks." What the bank chooses to build on top of Fedwire (the customer experience, the online banking tools, the brick-and-mortar branches, the financial products, the cross-selling) is what distinguishes it from other banks on the Fedwire system.

5. Ethereum is far more generalized. It allows anyone to spin up a network with as good or better security and reliability than Fedwire, and with the ability to make secure value transfers nearly instantly.

6. Developers can build any sort of financial products or business logic they want on top of this secure ledger, withautomated and immutable scripts, and without needing to pay the overheads dumped on them by the traditional centralized hosting and banking infrastructure.

7. But does it scale to the speed and size of a system like Fedwire? The answer is, yes, it can, but this will take several years. There are no direct or fixed limit neither for transaction sizes or block sizes. In Bitcoin, the size of the block is limited to 1MB, which works out to about 7 transactions per second. In Ethereum these limits increase and decrease in accordance with demand and network capacity.

8. However, this does not mean that blocks can be unlimited size. Recall that units of work in the Ethereum network are priced in gas. Thus, larger, more complex smart contracts cost more gas to store and execute.

9. An unbundling of banking services into ever smaller brands as the public Ethereum chain scales and is capable of processing more transactions, faster and faster. Laura Shin, author and host of the

blockchain-centric podcast Unchained, interviewed Adam Ludwin of San Francisco blockchain startup Chain in 2016 and wrote this:

9.1As for who owns the network, in the current system, if you go to Chase to deposit $50 cash, Chase holds that money, which was issued by the Federal Reserve, on its network. But Ludwin said you could imagine, instead of banks running the network, Fedwire, the current system for electronically settling payments between member banks, being reconstructed on a blockchain for which banks hold keys to make transfers.

9.2That could then lead to nonfinancial institutions being custodians of such currency. “With small enough amounts, you don’t need a bank,” said Ludwin. “Could Google, could Apple, could Facebook be holding small amounts of digital cash? Does that change the model of who a custodian is or could be? And the answer is yes.” It could also open up more avenues for peer-to-peer lending, reducing consumers’ reliance on banks for loan

10. The EVM may be coming into focus: a generalized, secure, ownerless virtual machine that offers cheap Fedwire-like functionality with a bunch of other magic on top. The EVM can run arbitrary computer programswritten in the Solidity language. These programs, given a particular input, will always produce the output the same way, with the same underlying state changes.

11. Solidity programs are capable of expressing all tasks accomplishable by computers, making them theoretically Turing complete. That means that the entire distributed network, every node, performs every program executed on the platform. When one user uploads a smart contract through their Ethereum node, it is included in the latest block and propagated around the network, where it is stored on every other node in the network.

12. The job of each and every node in the EVM to run the same code, as part of the block processing protocol. The nodes go through the block they are process and run any code enclosed within the transactions. Each node does this independently; it is not only highly parallelized, but highly redundant.

3.9 EVM APPLICATIONS

1. The EVM is a transaction singleton machine with shared state. In computing, this means it behaves like one giant data object, rather than what it is: a network of discrete machines, themselves singletons, in constant communication.

2. State machines.

2.1 The EVM, as we’ve discussed several times so far, is a state machine.

2.2 Digital V/s Analog

1. Foundational to the concept of a stateful computer is the idea of a switch that can be on or off. The 1s and 0s always referred to as the lingua franca of machines refer to arrays of metaphorical switches, so to speak, put in a certain configuration in order to code for specific letters, numbers, or other keyboard symbols. All of the symbols on a keyboard (and more) can be represented with just eight switches, which is why computing memory is stacked in multiples of eight. The so-called character code for a comma, for example, is 0010 1100.

3. State-ments

1. Individual snippets of code, when considered by themselves, fall broadly into two buckets: expressions and statements. Expressions are used to evaluate a particular condition; statements (note the root word!) are used to write information into the computer's memory. Together, expressions and statements let computers modify a database in a predictable way when specific conditions are met. This is the crux of automation, and it's the reason we find computers so useful.

4. Data's Role in State

1 Every time you change data in a computer's memory, you can think of its zillions of internal switchesas being in a slightly different configuration. State generally refers to the present condition of the system: the objective series of changes in information, across various memory addresses of the machine, that led to the current contents of its memory.

2. It's important to distinguish between an attribute and state. State is something that can change easily and predictably. Let's use the example of a car. Repainting a car is hard work, but it can be done. Paint color is an example of an attribute. In pseudocode, you might say the following about a car

Bodycolor=red

3. In computer programming, this is called a key/value pair. The key, bodyColor, has a value assigned to it, which is red. To change the value of this key, your code makes a new statement of the value to be something else

Bodycolor=green

4. Now let's say you instruct the computer that the color of this car will change frequently. In other words, you make the car's color a variable. Well, it can be said that the variable (in this case, the color) can have a state, which is a value that changes. But an individual value, such as green, has no state; green is simply green.

5. Working familiarity with the concept of state transition will help nonprogrammers gain insight into the truly hard problems incumbent in the design of decentralized systems.

3. Guts of the EVM Work

1 The computer itself is running a state function, constantly checking for changes to its state. It's like an overeager intern who wonders thousands of times per second if any new work has landed on his desk.

2. When new instructions are triggered, the computer runs code and may write new data to its memory. It's important to note that each state change must be based on the last state change; a computer doesn't just toss information into memory addresses willy-nilly.

3. The EVM runs a loop continuously that attempts to execute whatever instructions are at the current program counter (whatever program is "on deck" to be processed). The program counter works like a delicatessen queue: each program takes a number and waits its turn.

4. This loop has a few jobs: it calculates the cost of gas for each instruction; and it uses memory, if necessary, to execute the transaction if the preamble calculation succeeds. This loop repeats until the VM either finishes running all the code on deck, or it throws an exception, or error, and that transaction is rolled back.

5. The EVM Constantly checks for Transactions- the EVM has a constant history of all transactions within their memory banks, leading all the way back to the very first transaction. Unlike people, who have to deal with imperfect memory, a computer's state (as it exists today) is the specific outcome of every single state-change that has taken place inside that machine since it was first switched on.

6. Creating a Common Machine Narrative of What

6.1 Transactions, therefore, represent a kind of machine narrative—a computationally valid arc between one state and another.

6.2 As Gavin Wood's Ethereum Yellow Paper says: There exist far more invalid state changes than valid state changes. Invalid state changes might, e.g., be things such as reducing an account balance without an equal and opposite increase elsewhere. A valid state transition is one which comes about through a transaction.

7. Cryptographic Hashing- Hash Functions

7.1 Generally speaking, the purpose of hash functions, in the context of a blockchain, is to compare large datasets quickly and evaluate whether their contents are similar. A oneway algorithm processes the entire block's transactions into 32 bytes of data—a hash, or string, of letters and numbers that contains no discernible information about the transactions within.

7.2 The hash creates an unmistakable signature for a block, allowing the next block to build on top of it. Unlike the ciphertext that results from encryption, which can be decrypted, the result of a hash cannot be "un-hashed.

4. Block : The History Changes

4.1 Transactions and state changes in the Ethereum network are segmented into blocks, and then hashed. Each block is verified and validated before the next canonical block can be placed on “top” of it.

4.2 In this way, nodes on the network do not need to individually evaluate the trustworthiness of every single block in the history of the Ethereum network, simply to compute the present balances of the accounts on the network.

4.3 They merely verify that its “parent block” is the most recent canonical block. They do this quickly by looking to see that the new block contains the correct hash of its parent’s transactions and state.

4.4 All the blocks strung together, and including the genesis block, an honorific describing the first block the network mined after coming online, are called the blockchain. In some circles, you will hear the blockchain referred to as a distributed ledger or distributed ledger technology (DLT).

4.5 Ledger is an accurate description, as the chain contains every transaction in the history of the network, making it effectively a giant, balanced book of accounts. However, most so-called digital ledgers do not use proof of work to secure the network, as Bitcoin and Ethereum do.

4.6 Understanding Block Time- In Bitcoin, a block is 10 minutes. This so-called block time is derived from constants hardcoded into Bitcoin’s issuance scheme, with a total of 21 million coins to be released from 2009 to 2024, and rewards halving every four years and n Ethereum, block time is not a function of the issuance schedule of ether. Instead, block time is a variable that is kept as low as possible, for the sake of speedy transaction confirmation.

4.5 The Drawbacks of Short blocks- It’s important to note that Bitcoin’s long confirmation times make retail commerce and other practical applications difficult. When blocks are shorter and transactions move faster, user experience is better. However, shorter blocks and faster transactions make it more likely that a given node will get the order of transactions wrong, because it may not have heard about some transactions originating from far away.

4.6 Solo Node Block Chain- when you spin up your own blockchainit’s possible to use the Ethereum protocol with a single machine. It will process your transactions just fine, as long as one or more nodes are mining on the chain. But if someone knocks that machine offline, your chain is inaccessible, and transactions stop going through.For this reason, despite Ethereum being free and open software, the necessity for many, many nodes to create a resilient network causes developers to converge and work (for the most part) as one community, on a small number of public chains.

4.7 Distributed Security-The distributed nature of the Ethereum Virtual Machine, and the fact that it is composed of many nodes around the world, means that it must be purpose-built to solve the diffmatching problem that can arise when there are many near-simultaneous changes to the same database, from many users, all over the world.The EVM's resilience and security arise from the large number of machines mining on the network, incentivized by the earning of fees denominated in ether or bitcoins.

5. Mining's Place in the State Transition Function

5.1 Mining is the process of using computational work to nominate a block—that miner's version of recent transaction history—as the canonical block for this, the most recent block on the chain.

5.2 . Mining achieves the consensus required to make valid state changes, and the miners are paid for contributing to the consensus building. This is how ether and bitcoin are "created."

5.3 Each time a new block is created, it is downloaded, processed, and validated by node on the network. During processing, each node executes all the transactions contained therein. the Ethereum state transition function can be defined as the following six steps. For each transaction in a block, the EVM performs the following:

5.3.1 Check whether the transaction is in the right format. Does it have the right number of values? Is the signature valid? Does the nonce—a transaction counter—on the transaction match the nonce on the account? If any of these are missing, return an error.

5.3.2 Calculate the transaction fee by multiplying the amount of work requiredby the gas price. Then deduct the fee from the user's account balance, and increment the sender's nonce (transaction counter). If there's not enough ether in the account, return an error.

5.3.3 Initialize the gas payment; from this point forward, take off a certain amount of gas per byte processed in the transaction.

5.3.4 Transfer the value of the transaction—the amount being sent—to the receiving account.If the receiving account doesn't exist yet, it will be created. (Offline Ethereum nodes can generate addresses, so the network may not hear of a given address until a transaction takes place.) If the receiving address is a contract address, run the contract's code. This continues either until the code finishes executing or the gas payment runs out.

5.3.5 If the sending account doesn't have enough ether to complete the transaction, or the gas runs out, all changes from this transaction are rolled back. A caveat are the fees, which still go to the miner and are not refunded.

5.3.6 If the transaction throws an error for any other reason, refund the gas to the sender and send any fees associated with gas used to the miner.

6. Renting Time on the EVM

6.1 For every instruction the EVM executes, there must be a cost associated, to ensure the system isn't jammed up by useless spam contracts.

6.2 Every time an instruction executes, an internal counter keeps track of the fees incurred, which are charged to the user. Each time the user initiates a transaction, that user's wallet reserves a small portion (selected by the user) to pay these fees.

6.3 After a transaction has been broadcast to the network from a given node—let's say Bob sends Alice some ether from his computer—the network propagates the transaction around so that all the nodes can include it in the latest block.

6.4 Gas is a unit of work used to measure how computationally expensive an Ethereum operation will be. Gas costs are paid with small amounts of ether. The purpose of gas is twofold. First, it guarantees a prepaid reward for the miners that execute code and secure the network, even if the execution fails for some reason. Second, it works around the halting problem and ensures that execution can't go on longer than the time it prepaid for.

6.5 Gas is a unit of work; it's not a subcurrency, and you can't hold or hoard it. It simply measures how much effort each step of a transaction will be, in computational terms.

6.6 Gas costs ensure that computation time on the network is appropriately priced. This works differently in Bitcoin, where the fee is based on the size of the transaction in kilobytes. Because Solidity code can be arbitrarily complex, a short snippet of instructions could generate a lot of computational work, whereas a long snippet could generate less. That's why fees in the EVM are based on the amount of work being done, not on the size of the transaction.

7. Working with Gas

7.1 Unfortunately, the term gas creates some confusion. Every transaction requires a STARTGAS value. This value is referred to as gasLimit in the Yellow Paper and often just as gas in Geth and Web3.js.

7.2 Every transaction also requires the user to specify a gas price.

7.3 The amount stipulated in STARTGAS, multiplied by the gas price, is held in escrow while your transaction executes.

7.4 If the gas price you offer for a transaction is too low, nodes won't process your transaction, and it will sit unprocessed on the network.

7.5 If your gas price is acceptable to the network, but the gas cost runs over what's available in your wallet balance, the transaction fails and is rolled back; this failed transaction is recorded to the blockchain, and you get a refund of any STARTGAS not used in the transaction.

7.6 Using excessive STARTGAS does not cause your transactions to be processed more quickly, and in some cases may make your transaction less appealing to miners.

7.7 Accounts, Transactions and Messages

1 Ethereum has two types of accounts: Externally owned accounts Contracts accounts.

2. Externally Owned Accounts

2.1 An externally owned account (EOA) is also known as an account controlled by a pair of private keys, which may be held by a person or an external server. These accounts cannot hold EVM code. Characteristics of an EOA include the following: • Contains a balance of ether • Capable of sending transactions • Controlled by the account's private keys • Has no code associated with it • A key/value database contained in each account, where keys and values are both 32-byte strings.

3. Contract Accounts

3.1 Contract accounts are not controlled by humans. They store instructions and are activated by external accounts or other contract accounts. Contract accounts have the following characteristics: • Have an ether balance • Hold some contract code in memory • Can be triggered by humans (sending a transaction) or other contracts sending a message • When executed, can perform complex operations • Have their own persistent state and can call other contracts • Have no owner after being released to the EVM • A key/value database contained in each account, where keys and values are both 32-byte strings.

8. Transactions and Messages

8.1 A transaction in the EVM is a cryptographically signed data package storing a messagewhich tells the EVM to transfer ether, create a new contract, trigger an existing one, or perform some calculations.

8.2 Characteristics of Transactions
A recipient address; specifying no recipient (and attaching smart contract data) is the method for uploading new smart contracts. As you'll see, a contract address is returned so that the user knows where to access this contract in the future. • A signature identifying the sender • A value field showing the amount being sent • An optional data field, for a message (if

this is being sent to a contract address) • A STARTGAS value, indicating the maximum number of computational steps the transaction are prepaid • A GASPRICE value, representing the fee the sender is willing to pay for gas.

9. Characteristic of Messages

9.1 Messages are virtual objects that are never serialized and exist only in the EVM. When a miner is paid in the Ethereum network, this is accomplished by way of a message to increment the miner's payment address; it does not constitute a transaction.

9.2 A message is sent when a contract is being run by the EVM, and it executes the CALL or DELEGATECALL opcodes.

9.3 Messages are sent to other contract accounts, which in turn run the code enclosed in the message. Thus, contracts can have relationships with each other. A message contains the following: • The sender address of the message • The recipient address of the message • The value field (indicating how much ether, if any, is being sent) • An optional data field (containing input data for the contract) • A STARTGAS value limiting the amount of gas the message can use.

10 Estimating Gas Fees for Operations- Transactions need to provide enough STARTGAS to cover all computation and storage.

Table 1 Costs of Common EVM operations

Operation Name	Gas Cost	Description
step	1	Default amount per execution cycle
stop	0	Free
suicide	0	Free
sha3	20	SHA-3 hash function
sload	20	Gets from permanent storage
sstore	100	Puts into permanent storage
balance	20	Queries account balance
create	100	Contract creation
call	20	Initiating a read-only call
memory	1	Every additional word when expanding memory
txdata	5	Every byte of data or code for a transaction
transaction	500	Base fee transaction
contract creation	53,000	Changed in homestead from 21,000

11. Opcodes in the EVM

11.1 In Ethereum and Bitcoin, things work differently. Because the network is also a global machine, the "methods" you use to make calls across the network are just machine-language codes, of the ilk used inside an individual computer.

Table 2 List of some of the Opcodes. Many more are there for exploration

0s: Stop and Arithmetic Operations		
0x00	STOP	Halts execution.
0x01	ADD	Addition operation.
0x02	MUL	Multiplication operation.
0x03	SUB	Subtraction operation.
0x04	DIV	Integer division operation.
0x05	SDIV	Signed integer.
0x06	MOD	Modulo.
0x07	SMOD	Signed modulo.
0x08	ADDMOD	Modulo.
0x09	MULMOD	Modulo.
0x0a	EXP	Exponential operation.
0x0b	SIGNEXTEND	Extend length of 2s (complement signed integer).
10s: Comparison and Bitwise Logic Operations		
0x10	LT	Lesser-than comparison.
0x11	GT	Greater-than comparison.
0x12	SLT	Signed less-than comparison.
0x13	SGT	Signed greater-than comparison.
0x14	EQ	Equality comparison.

3.10 SUMMARY

In this chapter, we learned that Ethereum offers another approach to building software, one in which security and trust are baked in at the protocol level. This may have a substantial global impact. As the world digitizes, large-scale systems become increasingly mission-critical for all kinds of organizations—not just in banking and insurance, but also in city services, retail, logistics, content distribution, journalism, apparel manufacturing, and any other industry that has provenance or payments in play.

3.11 REFERENCES

[1] Chris Dannen- "Introducing Ethereum and Solidity"- Foundations of Crytocurrency and Blockchain Programming for Beginners by Apress.

3.12 QUESTIONS

Q1. Describe Block Chain
Q2. Specify the Characteristics of Ethereum
Q3. List down the installation steps of MIST browser.
Q4. Why Gas is important?
Q5. List down the opcodes of EVM?

4

SOLIDITY PROGRAMMING

Unit Structure

4.0 Objectives
4.1 Introductions
4.2 Global Banking Made real
4.3 Complementary currency
4.4 Programming the EVM
4.5 Design Rationale
4.6 Importance of formal proofs
4.7 Testing formatting solidity files, Tips for Reading Code
4.8 Statements and expressions in solidity
4.9 Value types, Global special variables, Units and Functions
4.10 Summary
4.11 Questions
4.12 References

4.0 OBJECTIVE

At the end of this unit, the student will be able to

- ✓ Define the solidity programming.
- ✓ Illustrate the concept of Global banking, complementary currency.
- ✓ Summarize the programming construct of EVM and solidity.
- ✓ Give the method of testing the solidity codes.

4.1 INTRODUCTION

1. Solidity is a new programming language used to write programs called smart contracts, which can be run by the EVM. This new language is a hodgepodge of conventions from networking, assembly language and web development.

2. Imagine a person on a beach in another country. The person took a trip here in a whim and breezed past the currency exchange booth in the airport, figuring use of credit or debit card while visiting- no need for cash. But in a rush he forgot to bring sunglasses. A vendor walking along the beach has a pair that happen to be a style of a person. In fact, they are better than the pairs passing in the duty free area of the airport. He doesn't have a credit card reader-just only having Android phone and that person don't have any local currency. He give a person a little card with an email address and a phone number, to buy a sunglasses later on.

3. Think about this scenario for a time being and see the power of protocol-based digital currency. Why a person send this man a text message or an email or even call him on the phone, but cant send him money the same way?.

4. In general, a computing environment is an infinite loop that repeatedly carries out whatever operation is current in the system's program counter(jumping the queue in the program counter is where the JUMP opcodes derive their name).

5. The program counter iterates one by one until the end of that particular programs is reached. The machine exits the execution loop only if it encounters (throws) an error or hits an instruction designating the machine to STOP or Return a result or value.

6.These operations have access to three types of space in which to store data:

6.1 The stack, a container in which values can be added or removed (pushed or Popped). Stack values are defined within a method.

6.2 Dynamic memory, also known as the heap, an infinitely expandable byte array. This resets when the program finishes.

6.3 A key/value store for account balances and in the case of contract addresses, solidity code.

7. Solidity contracts can also access certain attributes about the incoming message such as the value, sender and data of the incoming message as well as the data from the block header.

4.2 GLOBAL BANKING MADE REAL

1. The banks of the world have computer systems that while upgraded and mostly modern are the descendants of machines that predate the Internet and certainly the world wide web. As a result they are architected to be safe. There is no single global banking network but rather an interconnected mass of an interconnected mass of national systems and private banking software stacks with their own risks.

2. Extra-Large Infrastructure

2.1 A system such as Ethereum has nodes all over the world, being operated by private individuals who are paid for their activity in the form of mining fess, denominated in ether.

2.2 As a result, cryptocurrency protocols have the power to elevate financial transactions to the level of convenience we now enjoy with our telecommunications. But question here is how does a decentralized system of peer-to-peer nodes run "programs".

3. Worldwide Currency

3.1 The idea of a universal cryptocurrency seems to rest on the assumption that every human on earth will eventually download a cryptocurrency wallet on to their cell phone. However, such a pipe dream is not the roadmap for Ethereum.

3.2 Instead, the Ethereum core developer have chosen to make it easy for third parties to create complementary currencies or custom tokens that will be branded and used for special purposes(similar to credit card rewards points today).

3.3 These third parties(whether existing corporations, startups, municipalities, universities or nongovernmental organizations) could rely upon the public chain or large permissioned chains, to push around many different types of tokens, much the way that the global banking system is equipped to handle many different currencies.

3.4 It's unlikely that most people's first experience with ether will be for the sake of cryptocurrency experimentation. It's more likely they will end up holding digital tokens or points as part of a brand loyalty program, university program or employer-sponsored system.

3.5 Sports stadiums, theme parks, city summer camps, shopping malls, large office parks-anywhere there's a community exchanging money, a complementary currency might make sense.

4.3 COMPLEMENTARY CURRENCY

1. Why would a country ever need more than one form of money? In the decades leading up to the establishment of the federal reserve, the united states present day central bank, many local currencies circulated.

2. These paper bills generally represented gold on deposit somewhere and were thus local by nature, a certificate for gold is worth little if the redeeming institution is thousands of miles away.

3. In the period before widespread, systematic private money systems(a period of American history known as the wildcat banking era), many printing houses made their primary incomes from printing money with various ant counterfeiting features to rival their competing printing houses.

4. Benjamin franklin was one such printer who enriched himself on the printing of complementary currencies. In fact, he was known for his ant counterfeiting measures that went above and beyond.

5. According to the Smithsonian institution, he once printed an official issuance of local Pennsylvania currency with the name of the state spelled wrong, in the hopes of foiling counterfeiters who assumed those bills must be fake. Many of franklin's colonial bills bore the words to counterfeit is death.

6. The term complementary currency refers to a medium of exchange functioning alongside national flat currency, meeting a need the national coin cannot. These currencies generally have four purposes.

6.1 To promote local economic development within a small community.
6.2 To build social capital in that community.
6.3 To nurture more sustainable lifestyles.
6.4 To meet needs that mainstream money does not.

7. Solidity programming allows anyone to create a complementary currency, with a simple token contract. Those tokens can have whatever parameters the situation calls for, when we deploy a token contract using solidity as a programming language.

8. The Promise of solidity
8.1 Solidity is a high-level contract oriented language with similaritiesto javascript and C languages. It allows us to develop contracts and compile to EVM bytecode.

8.2 It is currently the flagship language of Ethereum. Although it's the most popular language library to be written for the EVM, it was not the first and probably will not be the last.

8.3 There are four languages in the Ethereum protocol at the same level of abstraction, but the community has slowly converged on solidity, which has edged out serpent(similar to python), Lisp like language(LLL) and mutan, the latter of which is deprecated.

8.4 Learning solidity enables us to moves tokens of value in any Ethereum-based system and because Ethereum and solidity itself are free and open source technology, clever minds will likely alter and re-release it or deploy it privately.

8.5 Browser Compiler

1. The most common way to test solidity is by using the browser-based compiler.

2. Sample code for browser compiler is given as follows

```
pragma solidity ^0.4.0;
contract Ballot {

   struct Voter {
uint weight;
      bool voted;
      uint8 vote;
      address delegate;
   }
   struct Proposal {
```

```
uintvoteCount;
    }

    address chairperson;
mapping(address => Voter) voters;
Proposal[] proposals;

    /// Create a new ballot with $(_numProposals) different proposals.
    function Ballot(uint8 _numProposals) public {
        chairperson = msg.sender;
        voters[chairperson].weight = 1;
proposals.length = _numProposals;
    }

    /// Give $(toVoter) the right to vote on this ballot.
    /// May only be called by $(chairperson).
    function giveRightToVote(address toVoter) public {
        if (msg.sender != chairperson || voters[toVoter].voted) return;
        voters[toVoter].weight = 1;
    }

    /// Delegate your vote to the voter $(to).
    function delegate(address to) public {
        Voter storage sender = voters[msg.sender]; // assigns reference
        if (sender.voted) return;
        while (voters[to].delegate != address(0) && voters[to].delegate !=
msg.sender)
            to = voters[to].delegate;
        if (to == msg.sender) return;
sender.voted = true;
sender.delegate = to;
        Voter storage delegateTo = voters[to];
        if (delegateTo.voted)
proposals[delegateTo.vote].voteCount += sender.weight;
        else
delegateTo.weight += sender.weight;
    }

    /// Give a single vote to proposal $(toProposal).
    function vote(uint8 toProposal) public {
        Voter storage sender = voters[msg.sender];
```

```
        if (sender.voted || toProposal>= proposals.length) return;
sender.voted = true;
sender.vote = toProposal;
        proposals[toProposal].voteCount += sender.weight;
    }
    function winningProposal() public constant returns (uint8
_winningProposal) {
        uint256 winningVoteCount = 0;
        for (uint8 prop = 0; prop <proposals.length; prop++)
            if (proposals[prop].voteCount>winningVoteCount) {
winningVoteCount = proposals[prop].voteCount;
                _winningProposal = prop;
            }
    }
}
```

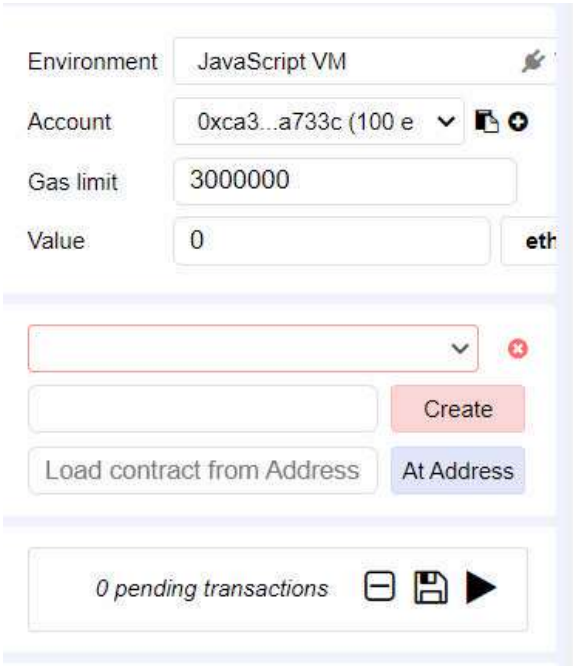


Fig 1 Output of code compiling and executing on browser compiler

4.4 PROGRAMMING THE EVM

1. Sometimes it's easier to learn a new habit than to break an old one. Many conventions in distributed application programming will strike today's web and native application programmers as odd or quirky.
2. Plus, they may already be professionally or personally invested in other languages or subject areas. So don't feel like the whole world has a head start on you if you are just starting out.
3. New coders can approach Ethereum without preexisting assumptions. Better yet they will find a system they can understand from top to bottom.

4. Not all hackers nor even software engineers know the intricacies of the underlying networks in the layers below their application hosting provider.

5. In conventional web applications, there are many individual servers with databases, communicating and sharing data over a network. This data may be manipulated by applications that live on still other servers. Even more servers may be in the mix to balance surges in demand.

6. A server is a computer that acts in a dedicated role, as part of a certain kind of service you want to offer people via the web. Some servers hold data (For example spreadsheets of information, such as customer names and addresses) in what are known as databases. Some servers run applications that other computers can access over the network.

7. In Ethereum, the network is the database and this network can run applications available to everyone on it. So you end up learning quite a bit about all three.

8. Watching a blockchain explorer report new transactions is something of a marvel when you know what's happening underneath. Although learning Ethereum may seem like a lot of work, it would be much more work to understand today's Web with a similar breadth and depth.

9. The following reasons urge users to begin experimenting with solidity

9.1 Easy Deployment

1. In Ethereum, you don't necessarily have much of the hassle of deploying and scaling a normal web application. All the required smart contracts for the back-end of distributed app also known as a dapp, can be neatly bundled up in a few documents and sent to the EVM and your program is available instantly to anyone on earth who installs an Ethereum wallet or command-line code.

2. Today, developers may want to build "hybrid" Ethereum dapps that are accessible through normal Web browsers, in which case adding ether payments is just adding more work. But by the time the network is complete in 2-3 years, it will be able to far easier to host all the components of an application using the Ethereum protocol.

3. In business jargon, time to value, or TTV, is the amount of time that passes from the moment the customer requests something to the moment the customer gets it. This something can be tangible or intangible. But a low TTV suggests that it is easy to think up a product or service and deliver it quickly to the people who want to use it.

4. In Ethereum, it is fast and inexpensive (if not yet easy) to develop and deploy unalterable, always-up, uncensorable applications that move real value over arbitrary distances. And everything is free, except the gas costs generated by your programs, and your own time (and computer).

5. For software engineers, service providers, system administrators, and product managers, the long-term impact of working in the Ethereum

ecosystem means less brittle systems, faster product iterations, and far less time developing infrastructure to support new applications or services. In short, this may amount to a drastic reduction in TTV for enterprise software vendors and in-house teams alike.

9.2 The case for writing Business Logic in Solidity

1. Because of its novel characteristics, the fate of Ethereum in 2017 and beyond doesn't necessarily rest on the mainstream popularity or adoption of today's Ethereum clients.

2. Instead, it relies on popularity with developers, brands, corporations, organizations, governments, and other institutions that are in a position to create an Ethereum token for their community, and perhaps even their own branded wallet.

3. They might do so in the interest of quickly and safely rolling out cool new products and services with ultra-low overhead. This also goes for large marketing campaigns, which must be deployed faster and faster today to keep up with the speed of Internet meme culture.

4. The frictionless nature of the payments in cryptonetworks makes it easier than ever to build a seamless sales and marketing experience for customers, with payments built in.

5. A complementary currency is also a highly valuable tool for use in rewards programs, membership clubs, and large retail districts. Customers who hold money in the form of a branded coin are apt to spend more regularly on that brand, just as frequent flyers today stay loyal to the airline miles and credit-card point schemes that give them the best bang for the buck.

6. Today, loyalty programs can be obscure and even slightly scammy. But the transparency of a blockchain-based loyalty coin would make it as good as any other form of cryptocurrency—meaning it might be traded on exchanges or accepted by other parties as payment.

9.3 Code, Deploy and Relax

1. Many Ethereum-enabled applications might be used through the Mist wallet, or another Ethereum-native application running a node under the hood. For developers of client applications, adding compatibility with new Ethereum-based tokens is trivial, meaning that a high degree of overlap and intercompatibility will exist between Ethereum wallets and tokens, just as there are many IMAP- and POP-compatible e-mail clients today.

2. It's also possible to create an Ethereum program today that is accessible through the regular old Web, with a little bit of work. However, deployment will be made increasingly easy with the use of new third-party frameworks.

3. However, this isn't to say that conventional web apps will go away. Many individuals and organizations have enormous resources invested in legacy web apps. That said, the Ethereum network makes it far easier and cheaper to roll out and operate applications at large scale, as

you'll see, tempting more and more teams to consider decentralizing their applications.

4.5 DESIGN RATIONALE

1. The solidity programming language has a syntax like javascript, but it is specially designed to compile in to bytecode for the Ethereum virtual machine. The EVM runs code that is fully deterministic the same algorithm with the same inputs will always yield the same results.

2. Solidity is statically typed, supports inheritance, libraries and complex user-defined types, among other features. Conscientious use of types can help programmers understand how their programs will execute.

3. Data types are exactly what they sound like. A programmer has the option of telling the machine what type of data to expect: for example, will it be a number or a string of letters? Loosely typed languages don't require the programmer to be specific; strongly typed languages do.

4. Writing Loops in Solidity

4.1 Loops are foundational to control flow in programming—that is, the codification of if-this-then-that contingencies or do-this-while-doing-that concurrencies. In most programming languages, loops are initiated with similar syntax. Solidity adheres to all the same syntactical regularities as JavaScript and C when it comes to loops.

4.2 An iterator loop is an object that enables a programmer to move through a container or list. Sometimes, iterators are used to instruct the computer to perform the same operation a certain number of times, or on a number of elements in the code.

4.3 A general-purpose loop has the same syntax in JavaScript, C, and Solidity. It instructs the computer to count up from 0 to 10: for (i = 0; i< 10; i++) {...}.

4.4 the EVM allows looping in two ways. You can write loops in Solidity, or you can create them using JUMP and JUMPI instructions. This jumps ahead a specified number of steps in the program counter. Recall that the program counter keeps track of the number and order of computational steps in a given program as it is being executed on the EVM.

4.5 This is just one way that Solidity and EVM opcodes can be used together to create a contract that is mostly expressive and readable, but also cheap to run. It's important to point out that because of the way gas price is calculated, some functionality might be easier to enforce or less expensive to execute if written using opcodes, and this can be especially useful if you're writing your own language library.

5. Expressiveness and Security

5.1 The adjective expressive is used in computer science to mean code that is easy for a human programmer to write and to understand. Expressive languages are the bridge between human thought patterns and machine execution patterns.

5.2 For a language to be expressive, its various constructs must be intuitively readable, and its boilerplate code (such as keywords, special variables, and opcodes) must use human-readable words that help programmers remember what they represent.

5.3 Expressive languages must be compiled down into something more machine-friendly before they can be run, and this requires work on the part of the computer. After all, expressive languages tend to be harder to reason about (harder to predict the behavior of), whereas more-restricted, lower-level, less-abstract languages make that reasoning easier.

5.4 The final frontier is smart contracts that can be easily formally verified, but also written in an expressive high-level language such as Solidity. This problem begs for automation, and indeed, automated formal verification is now on the horizon—a fact that computer scientists must be excited about, and that Ethereum developers will unknowingly benefit from.

6. The importance of Formal Proofs

6.1 If you learn Solidity programming, you may encounter the curiosity of other developers, who will get right to the point: how do you prevent someone writing an infinite loop and locking up the machine?.

6.2 Far from being a niche argument, this is the most relevant issue related to software engineering's role in the world today: can human beings make a free, openly accessible virtual computer that other human beings can't spoil? If the answer is yes, then it stands in stark defiance of the theory of the tragedy of the commons.

6.3 Historical Impact of a shared global Resource

1. In economics, the tragedy of the commons is the idea that a shared resource can't last. Eventually, users acting in their own self-interests will deplete the resource, because it comes at no cost to themselves to do so. A scenario like this, whereby someone can enrich themselves or act profligately while externalizing the costs to other people, is known as a moral hazard.

2. Here's an example: In New York City in late 2016, the municipal government installed computing terminals on the streets of Lower Manhattan. These terminals offered free WiFi to passing pedestrians. However, these terminals also came with a small touchscreen allowing walk-up Internet access.

3. No sooner were these shared resources up and running before people were pulling up chairs, watching YouTube or pornography, and loitering for hours.

4. Program administrators were forced to quickly restrict the onscreen Internet access, and now the terminals serve mostly as just Wi-Fi hotspots.

5. Thus, the notion of an extremely inexpensive public computer such as the EVM is nothing short of fantastic. It can be accessed by anyone, with any computer, anywhere, and will run programs far into the future. Nobody owns it and nobody can tamper with it. It can even store your money for you.

6.4 How attackers bring down communities

1 Decentralized economies represent a nascent threat to all sorts of private vested interests around the world, especially in developing economies, where powerful people would prefer the world continue on without a solution to the tragedy of the commons (and thus, remain at the mercy of the latest autocrat or crazy mob).

2. A network as a community of people connecting with each other via computer. An attacker is someone who hates this group, and seeks to cause them grief at any expense.

6.5 Hypothetical Attack written in solidity

1. Imagine that an attacker wants to lock up the EVM with a super memory-intensive smart contract, written in Solidity.

2. Keep in mind that for the purposes of this example, the contract could also be written in any language created for the EVM, such as Serpent or even the lower-level EVM code, not just Solidity.

3. According to Rice's theorem, the behavioral properties of some computer programs are mathematically undecidable, meaning it is not possible to write another computer program that can definitively predict whether Solidity code you show it will ever terminate.

4. Thus, there is no way to write any kind of effective "gatekeeper" program that will swat down a hypothetically memory-hungry smart contract written by the attacker in this scenario.

5. Smart contracts are distinct from distributed applications, or dapps, even though both are distributed and application-like. A dapp is a GUI application that uses Ethereum smart contracts on the back end, in lieu of a conventional database and web application hosting provider. Dapps may be accessed through the Mist browser or over the Web.

6. The EVM deals with this reality in various ways, including a hard limit on the number of computational steps per block, its deterministic language, and gas costs. Nevertheless, gray areas will always be explored by attackers if a financial incentive exists, and at $1 billion market capitalization, there's significant incentive to crack the EVM and steal ether.

7. Although gray areas can't be engineered away at once, they can be dealt with in a series of protocol forks over time. As far as accidentally

destructive programs, it's upto the Ethereum community to develop patterns and practices that are conducive to straightforward, easy-to-prove contracts that can develop into boilerplate standards.

6.5 Automated Proofs to the Rescue?

1. Although it's not possible to create a gatekeeper that kicks out bad programs, it is increasingly feasible to produce provably correct programs by using a machine-checkable proof—an automated program that mathematically proves other programs.

2. Because smart contracts move money, they make great lab rats for automated mathematical proofs. The goal of this area of computer science and mathematics research is to ensure, in a systematic way, that source code satisfies a certain formal specification. It's a way for independent auditors to come in and mathematically verify that the program is actually doing what it's supposed to do.

3. Automating the proving process is a boon for businesses but won't do much for the average programmer learning Solidity. Proofs merely show you whether what you intended to happen actually did happen in the program.

4. If your program doesn't prove out, there is no way for an automated system to tell you how to write it better.

5. Nevertheless, the point of exploring this topic is to signal that Ethereum networks may indeed one day carry high volumes of automated money-moving bots pushing around trillions of dollars safely; and that developing these bots may not be as slow, risky, and obscure a process as it is today.

6. Determinism in practice

6.1 Combining the concepts from the preceding sections, you can see that in some ways, the whole idea of Turing completeness may be an idealized concept of limited usefulness when designing a public system in the real world.

6.2 Thus, it could also be said that in practice, the EVM is not really Turing complete, because the bounded nature of execution in Solidity contracts could soon make it possible to theoretically predict the behavior of any program the EVM will run.

6.3 Bitcoin escapes none of these issues. The gray areas that exist between expressive languages and machine languages exist for Bitcoin's scripting language too, which is also compiled down at runtime into machine code.

7. Lots in Translation

7.1 A human can perform a mathematical proof on only a high-level, abstract language—that is, a human-readable programming language such as Solidity. Performing such a proof on assembly code or

machine code would be next to impossible for even the most dedicated mathematical minds.

7.2 The compilation process—the transmission of human-readable code into lower level machine code—sacrifices a lot of (human interpretable) information about how to reason about the program.

7.3 It also sacrifices information that would be useful to anautomated theorem prover. Thus, some ambiguity is always introduced into the process. Today, you can never be fully sure that even a mathematically proven smart contract written in Solidity will still be provable after being compiled.

4.7 TESTING ,FORMATTING SOLIDITY FILES

1. The way to prevent ambiguous code from losing your money is to test vigorously. The Ethereum network comes with a testnet called Ropsten that uses play ether, which costs nothing and can be drawn from a faucet quickly in a sandbox-like environment.

2. In reality, Ropsten is no different from the main chain. It is simply a different chain that was designated for testing. Like the Titanic and its sister ship the Britannic, they are identical except for the names, as is every other chain someone spins up.

3. Command Line Optional

3.1 Keep in mind that most of the important functions of Ethereum can be done in the Mist wallet: sending and receiving ether, tracking tokens, and deploying contracts. Using Geth (or the other command-line clients) is a good choice for developers who intend to learn to write dapps.

3.2 If you can't read or write code, don't worry. A tutorial on syntax and structure follows this example that will help you reason about what the code is doing. In the next chapter, we'll deploy a standard Ethereum token, with zero coding required.

3.3 There are only three requirements for deploying a simple contract in Solidity:

1. A text editor such as TextEdit on macOS, Gedit on Ubuntu, or Notepad on Windows. Be sure to switch to plain-text mode, which strips away all fonts, underlining, bold, hyperlinks, and italics. (Never use rich text to write code!).

2. The Mist Wallet.

3. The browser solidity compiler located at https://ethereum. github.io/browser-solidity/ or available at the following shortlink: http://compiler.eth.guide

3.4. Upload a contract is to copy-paste your solidity code from your text editing application in to the solidity browser compiler.

3.5. From there, you'll compile the code into bytecode, and copy-paste

that bytecode into Mist. It's really very easy, but let's not get bogged down in the logistics just yet.

3.6 For example Solidity program will look like this as given below

```
contract PiggyBank {

        address creator;
        uint deposits;

// Declaring this function as public makes it accessible to other users and
smart contracts.
        function PiggyBank() public
        {
            creator = msg.sender;
            deposits = 0;
        }

// Check whether any ether has been deposited. When it is deposited, the
number of deposits go up and the total count is returned

        function deposit() payable returns (uint)
        {
            if(msg.value > 0)
                 deposits = deposits + 1;
            return getNumberOfDeposits();
        }
        function getNumberOfDeposits() constant returns (uint)
        {
            return deposits;
        }

// When the external account that instantiated this contract calls it again,
it terminates and sends back its balance.
        function kill()
        {
            if (msg.sender == creator)
                selfdestruct(creator);
        }
}
```

3.7 Formatting Solidity Files

1. Every Solidity file should have (but does not require) a version pragma, a statement indicating which Solidity version this contract was written in.

2. Over time, this should prevent older contracts from being rejected by future versions of the compiler.

3. Before writing any solidity program, a programmer should write version in top of the code. Here for example version is pragma solidity ^0.4.7

4. Tips for reading code

   Here are seven facts that will make this contract more legible for beginners

4.1 Computers read code from top to bottom, left to right, just like English speakers. Putting one line before another generally means the computer will see that instruction first.

4.2 Typically programs takes an input and return some kind of output. Computable functions (mathematical functions that can be performed by a computer) are defined as functions that can be written as algorithms.

4.3 Algorithms take in data, perform an operation on it, and return some kind of output. Programs are algorithms with other algorithms nested in them.

4.4 An algorithm is like a machine: you can reuse it many times. Thus, writing algorithmic instructions—programming—will strike you as being a lot like writing Mad Libs, which the computer will later autocomplete with information that a user (or in Ethereum, a contract) gives it, via a transaction or message call.

4.5 Operators are the symbols between the English words, such as the equal sign, plus sign, and minus sign.

4.6 Types are the nouns of computer programming. So when you see a type, you know what is allowed in that space of the Mad Lib. A common type in Solidity is an address.

5. The EVM is much closer to this original kind of computer, but it's suited to thinking about sophisticated accounting and fiscal reconciliation. Recall that databases are merely spreadsheets themselves, and computer programs manipulate these databases. Thus, when you declare something, you are telling the computer to put it in the spreadsheet—specifically, to put it in the stack.

6. The computer will figure out, on its own, how much memory to have ready to store the values in any temporary, or so-called dynamic, computations—small, pivotal logical statements used to compute contingencies such as if-then. (It's important to define the stack and heap in order to see that this is where the danger of memory-hog programs lies: in asking the computer to use more dynamic memory than it has to spare).

4.8 STATEMENTS AND EXPRESSIONS IN SOLIDITY

1. Some functions produce a value, such as a number, or an answer to a true/false question. What exactly this value can be is determined by Solidity's types, mentioned earlier; the true/false value is called a Boolean.

2.What is an Expression?

2.1 Functions that produce a value are known as expression functions. Because expressions evaluate to a value of one type or another, in programming they can be used in place of values.

2.2 Other functions are declarative, and lead to the creation of a dedicated space in the computer's memory, which will be used each time it runs this routine. These declarative functions are important because they are crucial to writing statements.

3. What is a Statement?

3.1 A statement tells the computer to perform an action. The computer uses expressions to figure out how to take this action, and when. Thus, computer programs are composed of statements, and statements are often composed of expressions (or other statements).

4. Functions, public and private

4.1 In JavaScript and Solidity, you can use semicolons to chain statements, and tell the computer that another statement is coming up in the code: function first(); function second()

4.2 In Solidity, you can also declare whether you want certain functions to be available outside that program. These designations are as follows:

• public: Visible externally and internally (an accessor function for storage/state variables is created) • private: Visible only in the current contract (default

4.3 Functions written in Solidity code are not public by default. You must declare them as public when you make them, or they will not be available to contracts outside of the one they're in.

4.9 VALUE TYPES, GLOBAL SPECIAL VARIABLES, UNITS AND FUNCTIONS

1. Value Types- When writing solidity code, we can tell the computer what type of value to expect in each algorithmic instruction.

2. Booleans- Known in code as bool, the boleans are true/false expressions that evaluate to true or false.

3. Addresses- The address type holds a 20-byte value, which is the soze of an Ethereum address(40 hex characters or 160 bits). Address types also have member types.

4. Members of Addresses- These two members allow user to query the balance of an account or to transfer ether to an account. Be careful with transfer in smart contracts. It's better to use a pattern where the recipient is allowed to withdraw the money, than to have a contract initiating transfers 1)Balance 2) transfer

5. Address Related Keywords

Keywords come with the solidity language

5.1 <address>.balance (uint256): Returns the balance of the address in power.

5.2 <address>.send(uint256 amount) returns (bool): Sends given amount of weightage to address, and returns false on failure.

5.3 this(current contract's type): Explicitly converts to the address

5.4 selfdestruct(address recipient): Destroys the current contract, sending its funds to the given address.

6. Less-Common Value Types

6.1 Several other value types may be useful
1. Dynamically sized byte arrays
2. Fixed-point numbers
3. Rational and integer lietrals
4. String Literals
5. Hexadecimal Literals
6. Enums

7. Complex(Reference) Types

7.1 Generally speaking, types in Solidity are allotted 256 bits of memory in the EVM's storage; that's 2,048 characters. Types that are any longer than that can incur more-significant gas costs to move around. You'll need to choose carefully when assigning persistent storage in the EVM's stack. Here are the complex types that exceed 256 bits:

- Arrays
- Array Literals / Inline Arrays
- Structs
- Mappings

7.2 Arrays, structs, and other complex types have a data location that can be used by Solidity programmers to manipulate whether they are stored dynamically in memory or persistently stored.

8. Global Special Variables, Units and Functions

8.1 Global special variable can be called by any solidity smart contract on the EVM, they are built in to the language.

8.2Most of them return information about the Ethereum chain. Units of time and ether are also globally available. Literal numbers can take a suffix of power, finney, szabo or ether and will auto-convert between sub-denominations of Ether. Ether currency numbers without a suffix are assumed to be power.

8.3 Time-related suffixes can be used after literal numbers to convert between units of time. Here, seconds are the base unit, and units are treated as general units. Owing to the existence of leap years, be careful when using these suffixes to calculate time, as not all years have 365 days, and not days have 24 hours.

1 == 1 seconds

1 minutes == 60 seconds

1 hours == 60 minutes

1 days == 24 hours

1 weeks = 7 days

1 years = 365 days

8.4 Block and Transaction Properties

- block.blockhash(uintblockNumber) returns (bytes32): Hash of the given block, works for only the 256 most recent blocks .
- block.coinbase (address): Current block miner's address

- block.difficulty (uint): Current block difficulty
- block.gaslimit (uint): Current block gas limit
- block.number (uint): Current block number
- block.timestamp (uint): Current block timestamp
- msg.data (bytes): Complete call data.
- msg.gas (uint): Remaining gas.
- msg.sender (address): Sender of the message (current call).
- msg.sig (bytes4): First 4 bytes of the call data (function identifier).
- msg.value (uint): Number of wei sent with the message.
- now (uint): Current block timestamp (alias for block. timestamp).
- tx.gasprice (uint): Gas price of the transaction.
- tx.origin (address): Sender of the transaction (full call chain).

9 Operator Cheat Sheet

Table 1 Shows the operators we can use in solidity expressions

Precedence	Description	Operator
1	Postfix increment and decrement	++, --
	Function-like call	<func>(<args...>)
	Array subscripting	<array>[<index>]
	Member access	<object>.<member>
	Parentheses	(<statement>)
2	Prefix increment and decrement	++, --
	Unary plus and minus	+, -
	Unary operations	delete
	Logical NOT	!
	Bitwise NOT	~
3	Exponentiation	**
4	Multiplication, division, and modulo	*, /, %
5	Addition and subtraction	+, -
6	Bitwise shift operators	<<, >>
7	Bitwise AND	&

Precedence	Description	Operator
8	Bitwise XOR	^
9	Bitwise OR	|
10	Inequality operators	<, >, <=, >=
11	Equality operator, does-not-equal operator	==, !=
12	Logical AND	&&
13	Logical OR	||
14	Ternary operator	<conditional> ? <if-true> : <if-false>
15	Assignment operators	=, |=, ^=, &=, <<=, >>=, +=, -=, *=, /=, %=
16	Comma operator	,

10. Global Functions

1. In general in Solidity, special functions are mainly be used to provide information about the blockchain, but some can also perform mathematical and cryptographic functions. They are as follows:

- keccak256(...) returns (bytes32): Computes the Ethereum-SHA-3 (Keccak-256) hash of the (tightly packed) arguments.
- sha3(...) returns (bytes32): An alias to keccak256().
- sha256(...) returns (bytes32): Computes the SHA-256 hash of the (tightly packed) arguments. “Tightly packed” means that the arguments are concatenated without padding.To see how to add padding to arguments, see the following URL: http:// solidity.readthedocs.io/en/develop/units-and-globalvariables.html#mathematical-and-cryptographic-functions
- ripemd160(...) returns (bytes20): Computes the RIPEMD-160 hash of the (tightly packed) arguments
- ecrecover(bytes32 hash, uint8 v, bytes32 r, bytes32 s) returns (address): Recovers address associated with the public key from elliptic curve signature, returns 0 on error.
- addmod(uint x, uint y, uint k) returns (uint): Computes (x + y) % k, where the addition is performed with arbitrary precision and does not wrap around at 2**256.
- mulmod(uint x, uint y, uint k) returns (uint): Computes (x * y) % k, where the multiplication is performed with arbitrary precision and does not wrap around at 2**256.
- this (current contract's type): The current contract, explicitly convertible to its address.

2. It’s also worth mentioning contract-related variables that can be useful in writing solidity contracts-

2.1Super: The contract one level higher in the inheritance hierarchy.

2.2 selfdestruct(address recipient): Destroys the current contract, sending its funds to the given address.

2.3 assert(bool condition): throws if the condition is not met.

2.4 revert(): abort execution and revert state changes.

4.10 SUMMARY

The first steps toward understanding the impact of programs written for the EVM. You also took a critical look at the way these programs can achieve a meaningful degree of Turing completeness without sacrificing the security of the network. We've only touched briefly on the formal mathematics that make these programs so exciting for enterprise information technology.

4.11 REFERENCES

[1] Chris Dannen- "Introducing Ethereum and Solidity"- Foundations of Crytocurrency and Blockchain Programming for Beginners by Apress

4.12 QUESTIONS

Q1. Explain the concept of Global banking?
Q2. Give the different value types of solidity
Q3.What is Complementary Currency?
Q4. What are the different types of operator used in solidity programming language?
Q5. What is testing and formatting of Solidity programming language?
Q6. Give the tips for Reading code?

Unit III

5

HYPERLEDGER

Unit Structure

5.0 Objectives
5.1 Introduction to Hyperledger.
5.2 What is Hyperledger Fabric?
    5.2.1 Hyperledger Fabric Layers
    5.2.2 Role of Peers
    5.2.3 Hyperledger Fabric Work Flow
    5.2.4 Application areas of Hyperledger Fabric
    5.2.5 Benefits of Hyperledger Fabric
5.3 Hyperledger Composer
    5.3.1 Advantages of Hyperledger composer
5.4 Installing, Deploying & Running the Hyperledger Fabric
5.5 Summary
5.6 Questions
5.7 References

5.0 OBJECTIVES

At the end of this unit, the student will be able to:

- Understand the concept of Hyperledger using Blockchain Technology.
- Understand the working of Hyperledger.
- Understand the applications of Hyperledger Fabric in various domains
- Understand the need of a composer in Hyperledger Fabric.
- Setup Environment for Hyperledger Fabric and build the network in Linux environment.

5.1 INTRODUCTION TO HYPERLEDGER

Hyperledger is an open source project under linux foundation where people can come and work on the platform to develop the blockchain related use-cases.

Figure 1: Hyperledger Fabric

Hyperledger provides the platform to create personalized blockchain service according to the need of business work. Unlike other platforms which only develop blockchain based software, Hyperledger has an advantage of creating secured and personalized blockchain networks.

The public blockchain requires every peer in the network to complete the process and run consensus at the same time. when the business requires confidentiality in the work, the public network fails to keep this as it does not support private and consortium networks.

Example:

Consider a situation when person X wants to buy medicine from person Y, who is a doctor living in another country. As the medicine requirement is of the one person's private need, they need to maintain the data confidentially. But Dr. Y is selling medicine on the network to so many people, in the case of public blockchain, every transaction will get updated in the network to all the peers. That's where hyperledger finds its significance. In the hyperledger, the parties are directly connected and the concerned people's ledger will be updated. Hence Providing privacy and confidentiality.

5.2 WHAT IS HYPERLEDGER FABRIC?

Hyperledger Fabric is a framework for developing Blockchain-based solutions for the enterprise. It is open-source and under the umbrella of the Linux Foundation, designed by IBM. It is a private chain, thus super-helpful for enterprises since you don't want to put your transactions for public display. Having private chains also means faster transactions.

It is a permissioned blockchain network that gets set by the organizations that intend to set up a consortium. The organizations that take part in building the Hyperledger Fabric network are called the "members".

Simple Steps involved are as follows:

1. A requirement for the contract can be initiated through an app.
2. The membership service involved in the network validates the contract.
3. The concerned two-peer has to produce a result and then send it to the consensus cloud. The generated result from both the peers has to be the same inorder to validate the contract.
4. Once it is validated, then the transaction will happen between the affiliated peers and their ledger will be updated.

Thus, when a business requires a confidentiality and private network for their transaction to happen, hyperledger paves the way.

5.2.1 Hyperledger Fabric Layers

Hyperledger-based technology works using these layers:

1. **Consensus layer**: It makes an agreement on order and confirms if the transactions in a block are correct.
2. **Smart Contract Layer:** It processes and authorizes transaction requests.
3. **Communication layer:** It manages peer-to-peer (P2P) message transport.
4. An **Application Programming Interface**(API) is required which allows other applications to communicate with the blockchain.
5. **Identity management services**, which validates the identities of users and systems.

5.2.2 Roles of peers:

- ➔ Each member organization in the blockchain network is responsible to set up their peers for participating in the network. All of these peers are configured with appropriate cryptographic materials like Certificate Authority (CA)and other information.
- ➔ Peers in the member organization receive transaction invocation requests from the clients inside the organization.
- ➔ A client can be any specific application/portal serving specific organization/business activities.
- ➔ Client application uses Hyperledger Fabric SDK or REST web service to interact with the Hyperledger Fabric network.
- ➔ Chaincode (similar to Ethereum Smart Contract) installed in peers causes to initiate transaction invocation requests.
- ➔ All the peers maintain their one ledger per channel that they are subscribed to. Hence Distributed Ledger Technology (DLT). But unlike Ethereum in Hyperledger Fabric blockchain network peers

have different roles.So not all peer nodes are same. There are different types of peer nodes with different roles in the network:

- Endorser peer
- Anchor peer
- Orderer peer

★ **Endorser peer**

Upon receiving the “transaction invocation request” from the Client application the Endorser peer

1. Validates the transaction. ie Check certificate details and roles of the requester.
2. Executes the Chaincode(ie Smart Contract) and simulates the outcome of the transaction. But it does not update the ledger.

At the end of the above two tasks the Endorser may approve or disapprove the transaction.

As only the Endorser node executes the Chaincode (Smart Contract) so there is no necessity to install Chaincode in each and every node of the network. Thus, increases the scalability of the network.

★ Anchor peer

Anchor peer or cluster of Anchor peers is configured at the time of Channel configuration. Just to remind you, in Hyperledger Fabric you can configure secret channels among the peers and transactions among the peers of that channel are visible only to them.

Anchor peer receives updates and broadcasts the updates to the other peers in the organization. Anchor peers are discoverable. So any peer marked as Anchor peer can be discovered by the Orderer peer or any other peer.

★ **Orderer peer**

Orderer peer is considered as the central communication channel for the Hyperledger Fabric network. Orderer peer/node is responsible for consistent Ledger state across the network. Orderer peer creates the block and delivers that to all the peers.

Ordereris built on top of a message oriented architecture. There are two options are currently available to implement Orderer peer:

1. **Solo**: Suitable for development. Single point failure. Solo should not be used for the production ready network.
2. **Kafka:** Production ready Hyperledger Fabric network uses Kafka as the Orderer implementation. Kafka is a messaging software that has a high throughput fault tolerant feature.

5.2.3 Hyperledger Fabric Workflow

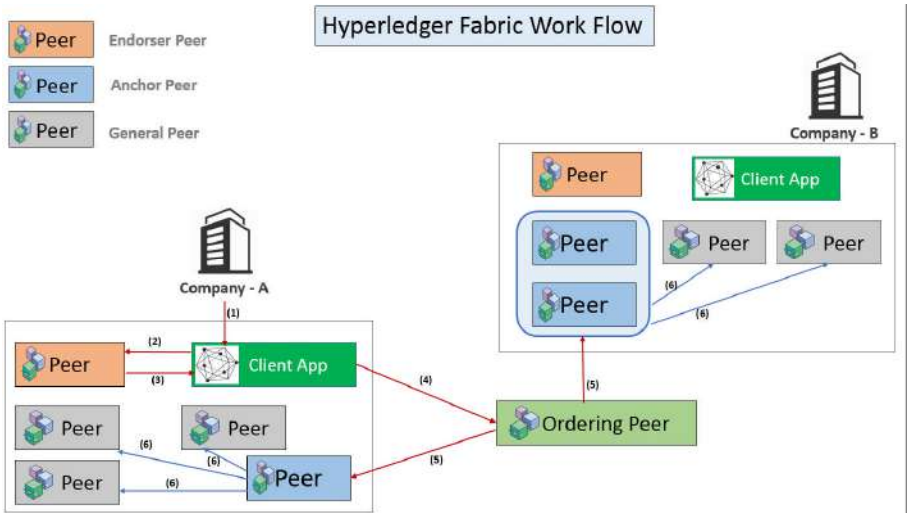


Figure 2 : Hyperledger Fabric Workflow

1. A participant in the member Organization invokes a transaction request through the client application.
2. Client application broadcasts the transaction invocation request to the Endorser peer.
3. Endorser peer checks the Certificate details and others to validate the transaction. Then it executes the Chaincode (ie. Smart Contract) and returns the Endorsement responses to the Client. Endorser peer sends transaction approval or rejection as part of the endorsement response.
4. Client now sends the approved transaction to the Orderer peer for this to be properly ordered and be included in a block.
5. The Orderer node includes the transaction into a block and forward the block to the Anchor nodes of different member Organizations of the Hyperledger Fabric network.
6. Anchor nodes then broadcast the block to the other peers inside their own organization. These individual peers then update their local ledger with the latest block. Thus all the network gets the ledger synced.

5.2.4 Application Areas of Hyperledger

- Financial Institutes
- Carbon Emission Reduction
- Healthcare
- Pharmaceutical Retailers
- London Stock Exchange Group
- TenneT Energy Community
- SAP
- Bank Guarantees
- Food Safety
- Banks
- Cryptocurrency

5.2.5 The benefits of Hyperledger Fabric

1. Permissioned network**: Establish decentralized trust in a network of known participants rather than an open network of anonymous participants.**
2. Confidential transactions**: Expose only the data you want to share to the parties you want to share it with.**
3. Pluggable architecture**: Tailor the blockchain to industry needs with a pluggable architecture rather than a one-size-fits-all approach.**
4. Easy to get started**: Program smart contracts in the languages your team works in today, instead of learning custom languages and architectures.**

5.3 HYPERLEDGER COMPOSER

- ➢ It is a set of collaboration tools for building simple blockchain business networks.
- ➢ Helps business owners and developers to create smart contracts and blockchain applications in order to solve business problems in a simple and speedy manner.
- ➢ It is built in Javascript, a platform-independent programming language that also supports the use of built-in libraries and uses available functions and scripts to make the utilities more scalable and reusable.
- ➢ Thus, Composer is an application development framework which simplifies and expedites the creation of Hyperledger fabric blockchain applications.
- ➢ One can easily define the business rules based on which blockchain transactions will be processed, the assets that are exchanged in blockchain-based use cases, and controls for participants, their identities, roles and access levels for performing the various kinds of transactions.
- ➢ Users ,with Hyperledger composer, can easily build and configure core components of the blockchain which include the network's digital assets, transaction logic, participants and access controls.
- ➢ Composer supports sharing, reusability and scalability of components across various organizations.
- ➢ One can easily generate the required scripts and APIs necessary for business implementation using Hyperledger Composer.
- ➢ It also supports use cases and real-time testing, which can even be performed through the web-based Composer playground without the need for local installations.
- ➢ Using Hyperledger Composer, it is possible for an individual to create and run a sample blockchain, and grant restricted permission to various participants.

For instance, one can easily build a "**Perishable Goods Network**" that facilitates trading of items like fruits and vegetables, include participants like farmers, shippers and importers, define individual roles for each participant, define and execute terms of agreement between the participants, track shipments, acknowledge, monitor and report status of goods at various stage in the supply chain, and payments management.

5.3.1 Advantages of Hyperledger Composer

Hyperledger Composer offers a lot of advantages as follows:

1. Faster creation of blockchain applications in easy steps.
2. Smooth and low-cost modeling and testing.
3. Allows the user to build, test and deploy various options and then implement the one that offers the best fit, and reusability of existing apps and APIs that reduce both effort and costs.
4. Maintenance of Core Data and Functionality of business network that includes the business model, transaction logic and access controls. As the archival of Business Network is central to Hyperledger Fabric composer.
5. It enables modeling the business requirements and functions, functional testing, as well as deployment testing on a live blockchain as it is an Web-based interface

5.4 HYPERLEDGER FABRIC INSTALLATIONS, DEPLOYING AND RUNNING THE FIRST TEST NETWORK

Steps to Install HyperLedger Fabric in Linux(Ubuntu)

1. Prerequisites
2. Linux Installations

1.Prerequisites

1. cURL □—□latest version
2. Docker□—□version 17.06.2-ce or greater
3. Docker Compose□—□version 1.14.0 or greater
4. Golang□—□version 1.11.x
5. Nodejs□—□version 8.x (other versions are not in support yet)
6. NPM□—□version 5.x
7. Python 2.7

Note:These prerequisites' versions are according to the fabric v1.4 documentation.

2. Linux(Ubuntu) installation steps

1. sudo apt-get install curl
2. sudo apt-get install golang-go

3. export GOPATH=$HOME/go
4. export PATH=$PATH:$GOPATH/bin
5. sudo apt-get install nodejs
6. sudo apt-get install npm
7. sudo apt-get install python
8. sudo apt-get install docker
9. curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
10. sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"
11. sudo apt-get update
12. apt-cache policy docker-ce
13. sudo apt-get install -y docker-ce
14. sudo apt-get install docker-compose
15. sudo apt-get upgrade

With the help of the above 15 steps, our environment is set up.

Next, we're going to download the samples of Fabric that have already been prepared to test it out. Enter the following two commands in your terminal.

16. sudo curl -sSL https://goo.gl/6wtTN5 |sudo bash -s 1.1.0

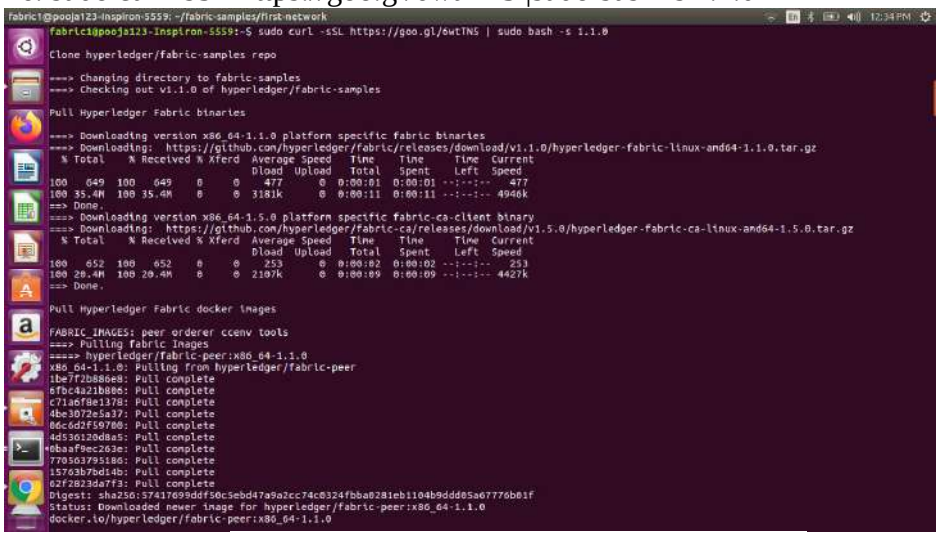


Figure 3 :Screenshot of execution of step 16

17. sudochmod 777 -R fabric-samples
18. cd fabric-samples/first-network
19. sudo ./byfn.sh generate

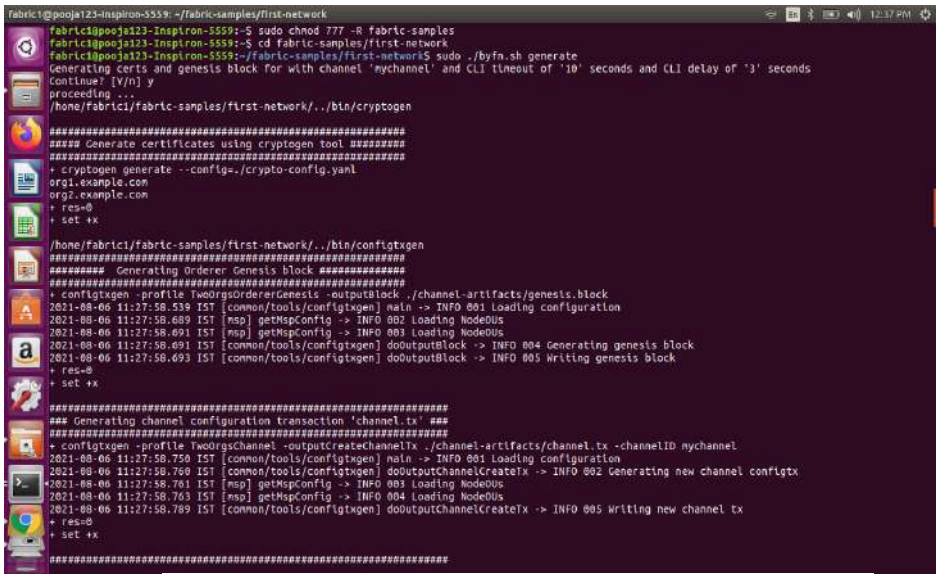


Figure 4: Screenshot of execution of step 17 to step 19

Now bring the blockchain network up with the following command.

20. sudo ./byfn.sh up

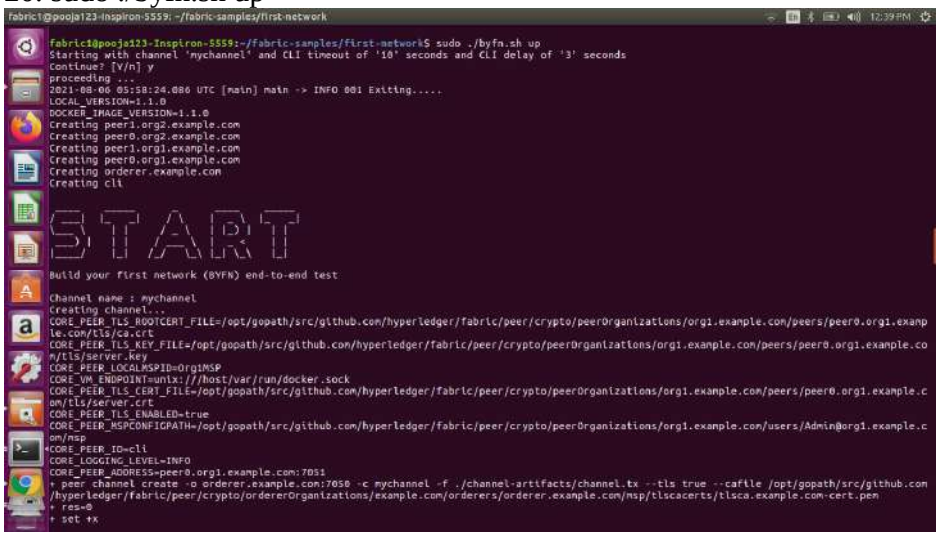


Figure 5: Screenshot of execution of step 20.

If everything worked, then you successfully created your first Fabric network! Congratulations!!!!.

For the time being you can bring the network down with the command:

21. sudo ./byfn.sh down

Thus, we reached a state where our computer can successfully use and deploy Hyperledger Fabric blockchain networks.

5.5 SUMMARY

In this Chapter, we Learnt about what Hyperledger Fabric is. We set up our computer to be able to deploy Fabric networks by installing the requirements and deploying the test fabric network. Now what?

With this foundation set, we can learn more about Fabric by creating more blockchain networks that suit our needs. We can test deploying smart contracts to our Fabric blockchain which will be taken care of in chapter 6.

5.6 REFERENCE:

1.**Installing python 2.7 in ubuntu: https://tecadmin.net/install-python-2-7-on-ubuntu-and-linuxmint/**

2. Step by step fabric installation:
https://www.srcmake.com/home/fabric

3. installation guide:Hyperledger Fabric Installation Guide! | Hacker Noon

5.7 QUESTIONS:

1. What is Hyperledger?
2. What is Hyperledger Fabric? Give its importance over simple blockchain.
3. Give working of Hyperledger Fabric with a suitable diagram.
4. Describe the role of different layers and peers involved in workflow of Hyperledger fabric.
5. What is the Hyperledger composer? Give its advantages.

6

SMART CONTRACTS AND TOKENS

Unit Structure

6.0. Objectives
6.1. Smart Contracts Defined
    6.1.1. How Do Smart Contracts Work?
    6.1.2. Benefits of smart contracts.
    6.1.3. Applications of Smart Contracts.
6.2. EVM As Back End.
6.3. Assets Backed by Anything
    6.3.1. *Bartering with Fiat Currency*
    6.3.2. Ether as Glass Beads
6.4. Cryptocurrency Is a Measure of Time
6.5. Function of Collectibles in Human Systems
6.7. Tokens as Category of Smart Contract
6.8.Playing with Contracts
6.9 Summary
6.10 Questions
6.11 References

6.0. OBJECTIVES

At the end of this unit, the student will be able to:

- ➢ Understand the concept of Smart Contract, its working and areas of applications.
- ➢ Understand the role of EVM in smart contracts.
- ➢ Understand the concept of cryptocurrency and its benefits in human Life.
- ➢ Understand the concept of tokens.
- ➢ Create and Deploy the token.

6.1. SMART CONTRACTS DEFINED

Smart contracts are simply programs stored on a blockchain that run when predetermined conditions are met. They typically are used to automate the execution of an agreement so that all participants can be immediately certain of the outcome, without any intermediary's

involvement or time loss. They can also automate a workflow, triggering the next action when conditions are met.

6.1.2. How do smart contracts work?

Smart contracts work by following simple **"if/when…then…"** statements that are written into code on a blockchain. A network of computers executes the actions when predetermined conditions have been met and verified. These actions could include releasing funds to the appropriate parties, registering a vehicle, sending notifications, or issuing a ticket. The blockchain is then updated when the transaction is completed. That means the transaction cannot be changed, and only parties who have been granted permission can see the results.

Within a smart contract, there can be as many stipulations as needed to satisfy the participants that the task will be completed satisfactorily. To establish the terms, participants must determine how transactions and their data are represented on the blockchain, agree on the "if/when...then…" rules that govern those transactions, explore all possible exceptions, and define a framework for resolving disputes.

Then the smart contract can be programmed by a developer – although increasingly, organizations that use blockchain for business provide templates, web interfaces, and other online tools to simplify structuring smart contracts.

6.1.2. Benefits of smart contracts

1. Speed, efficiency and accuracy

Once a condition is met, the contract is executed immediately. Because smart contracts are digital and automated, there's no paperwork to process and no time spent reconciling errors that often result from manually filling in documents.

2. Trust and transparency

Because there's no third party involved, and because encrypted records of transactions are shared across participants, there's no need to question whether information has been altered for personal benefit.

3. Security

Blockchain transaction records are encrypted, which makes them very hard to hack. Moreover, because each record is connected to the previous and subsequent records on a distributed ledger, hackers would have to alter the entire chain to change a single record.

4. Savings

Smart contracts remove the need for intermediaries to handle transactions and, by extension, their associated time delays and fees.

6.1.3.Applications of smart contracts

Explore how businesses benefit from smart contracts in active blockchain solutions

1. Safeguarding the efficacy of medications

Pharma Portal is a blockchain-based platform that tracks temperature-controlled pharmaceuticals through the supply chain to provide trusted, reliable and accurate data across multiple parties. Example: Sonoco & IBM

2. Increasing trust in retailer-supplier relationships

The Home Depot uses smart contracts on blockchain to quickly resolve disputes with vendors. Through real-time communication and increased visibility into the supply chain, they are building stronger relationships with suppliers, resulting in more time for critical work and innovation.

3. Making international trade faster and more efficient

Businesses are creating an ecosystem of trust for global trade. Using a blockchain-based platform, rules are standardized and trading options are simplified to reduce friction and risk while easing the trading process and expanding trade opportunities for participating companies and banks. Example:we.trade, the trade finance network convened by IBM Blockchain

6.2. EVM AS BACK END

Software apps, as they currently exist are typically discussed in two halves: the front end and the back end.

Back End: It usually refers to the database and the logic around interacting with it, which is where the program stores its information.

Front End: It usually refers to the part of the application the user sees: the interface with its various labels and controls.

Controls:controls is the general term for the little buttons, sliders, dials, hearts, stars, thumbs-up icons, and any other little thing you can click to make something happen.

The EVM is something like a replacement for the traditional application back end of a conventionally-hosted web or mobile application. Although the EVM itself is a fully fledged computer, it is not yet a complete end-to-end platform capable of hosting HTML/CSS interfaces; the most useful role it can play is as the back end to a distributed application.

Smart Contracts to Dapps

A smart contract is just a unit of functionality you upload to the EVM. The term distributed application, or dapp, typically describes a web- or

smartphone-accessible GUI application that uses the EVM as its back end. Unless it's a very simple dapp, its back end functionality will rely on several smart contracts.

6.3. ASSETS BACKED BY ANYTHING

In finance, an asset is a valuable resource that you expect will produce a benefit or value in the future. It can include physical natural resources or abstract financial Instruments.

By definition, the price of an asset should go up over time. (If it goes down, it is known as a depreciating asset.)

It can be said that cryptocurrencies are *assets backed by anything.* Let us try to understand this by an example given next.

6.3.1. Bartering with Fiat Currency

Let's say you are

- ➔ Alice, and you live in Japan. Assume you are paid in Japanese yen and that the prices of things such as rent, food, and basic services are denominated in yen. Now you want to pay someone in New York to do some translation work.
- ➔ On the other side we have Bob. Bob the translator uses US dollars; he holds them as savings; he pays taxes in USD, too.

This creates a problem. For most people, foreign currency is not much use, and exchanging it incurs high fees and risks of price slippage. Slippage refers to the price dropping before you have a chance to sell your lot. **Bob doesn't want yen, and Alice doesn't hold any dollars.**

The options available with alice and bob are:

- ➔ Alice and Bob may as well have cabbage and glass beads to barter.
- ➔ One of them can simply drive to the nearest bureau of exchange, probably at an international airport, that's not a parsimonious solution.
- ➔ Better option: Cryptocurrency, they need only establish a conversion rate, or multiplier,between their local currency and the cryptocurrency, and then convert the local price of the barter goods by using that multiplier. Whether they're using paper money or glass beads isn't relevant. For a trade to take place, they merely need to agree on a price.

6.3.2. Ether as Glass Beads

This example demonstrates one of the fundamental properties of ether and bitcoins:

1. They are standard accounting units of value, and simultaneously media of exchange themselves. For Example: Money also serves these functions, but in actuality, the medium of exchange (paper) is just a representation of value that exists in some bank's ledger. Here, they are one in the same value as the media.

2. These standard account units essentially tabulate themselves and balance the entire ledger anytime a payment moves from one place to another. This is another advantage over the money of today, which being inert has no "awareness" of other money in the system. As you may be imagining already, this makes s*mart contracts perfect for writing self-executing financial agreements.*

3. A derivative contract is a financial "bet" between two or more parties made on the value of the underlying asset. A derivative basically says that under certain conditions,Alice agrees to pay Bob a particular amount.

   What gives cryptocurrency the power to be used this way? Let's find out.

6.4. CRYPTOCURRENCY IS A MEASURE OF TIME

Because crypto assets and cryptocurrencies are impossible to counterfeit, this gives them an interesting property as a measure of time.But the point here is that these tokens are almost like the rings of a tree—their manufacture happens by a sophisticated process that cannot be "sped up."

Thus when trading with someone from a faraway economy, it becomes easy to trust prices denominated in cryptocurrency, because counterfeiting isn't possible.

Cryptocurrencies, currently, are not redeemable by any central authority for gold or fiat currency. However, they are classified as property or currency in a handful of countries.

Nevertheless, it can be said that cryptocurrency get their price from the marketplace:they are worth whatever someone in the marketplace will pay.As a result of its status as a decentralized digital medium of exchange, cryptocurrencies can be conceptualized as "assets backed by anything." It doesn't matter whether you're trading cattle, bananas, soybean futures, or private equities—the deal can be done in cryptocurrency. The only challenge is agreeing on a price.

Today, even if the buyer and seller agree to complete their transaction in cryptocurrency, it's likely they will quickly sell the cryptocurrency for local fiat money, to avoid price slippage. This will reduce if the prices of cryptocurrencies stabilize. Prices become more stable as the volume of transactions increases around the world, and the markets for trading cryptocurrencies become deeper, or more liquid.

Ether is similar to other cryptocurrencies such as bitcoins in this regard, but it does gain some intrinsic value from its usefulness in paying gas costs on the EVM.It makes ether more like a commodity such as oil or corn, which get their respective intrinsic values from their uses as fuel and food, respectively.

6.5. THE FUNCTION OF COLLECTIBLES IN HUMAN SYSTEMS

Keeping track of favors over time is a major function of money: to serve as a closed accounting system for a community to keep track of favors owed and favors given. This gets useful as bigger and bigger groups try to interact and cooperate.

Using collectibles to count favors is the essence of primordial accounting. Eventually,the value of these favors became abstracted, leading to the generalized instruments of value such as gold. This accounts for the modern-day association between wealth and esteem.

Ethereum and Bitcoin strike at the heart of a problem that is tens of thousands of years old, which is that reputation-accounting a natural human behavior, but also an imperfect one. Szabo continues:

Reputational beliefs can suffer from two major kinds of errors—errors of about which person did what, and errors in appraising the value or damages caused by that act. In both Homo sapiens neanderthalensis and Homo sapiens, with the same large brain size, it is quite likely that every local clan members kept track of every other local clan member's favors....

Between clans within a tribe both favor tracking and collectibles were used.

Two clans within a tribe exchanging collectibles within a closed system is something like a private bank database. Or a private blockchain. Szabo writes:

> *Between tribes, collectibles entirely replaced reputation as the enforcer of reciprocation, although violence still played a major role in enforcing rights as well as being a high transaction cost that prevented most kinds of trade.*

Just like the banks of today, human groups of yesteryear had trouble trading outside their accounting system. Whose money system do you use? Who keeps track of inter-tribe favors? No wonder there was so much bloodshed: the opportunity for cheating is just too persistent.

Platforms for High-Value Digital Collectibles

In a digital context, a reliable store of time has incredible potential as a platform for digital collectibles: valuable items that can be displayed, worn, or hung in one's personal space—either online or in real life—and that are not possible to knock off, nor easily stolen from their rightful owner.

When most people think of the Internet of Things, they think of sensor motes, self-diagnosing industrial equipment, and driverless vehicles. The Internet of Value, a euphemism referring to blockchain technologies, one of the many metaphors used to represent Ethereum and Bitcoin conceptually. But rather than think abstractly, it may be more useful to think about the potential in terms of valuable artwork, jewelry, fashion, or premium goods that look much like today's, but feature verifiable provenance and ownership stored on a blockchain.

In the future, the ownership, value, and provenance of a physical thing may never be "forgotten" as long as the blockchain where it was inventoried is still up and running. There will be no Antiques Roadshow on TV in 100 years. (We could even write a smart contract to take that bet!)

6.6. TOKENS ARE A CATEGORY OF SMART CONTRACT

Generally speaking, the Ethereum protocol prides itself on being featureless, which is one reason that tokens (as a concept) overlap so heavily with smart contracts (as a concept). Tokens are just one application of smart contract functionality on the EVM.

That said, Ethereum does make provisions for one common use-case of smart contracts, which is a sub currency, a.k.a. token. In the hopes of making it easy to get up and running, the Ethereum developers have put an easy-to-use template inside the Mist wallet for quickly launching your own tokens. Presumably, other templates for common smart contracts will follow. But at present, the one we get out of the box is the ability to create a custom unit of value which can be passed around, alongside ether, within the EVM.

If you were to phrase the user-friendly token-making progress as an elevator pitch for its value proposition to users, it would be something like this: "ultra-secure digital monetary system with automatic ledger balancing delivered as a service."

Now that you've gotten a taste of the historic potential of Ethereum and Bitcoin to create a new era of crypto collectibles and smart devices, let's get back to the brass tacks of deploying a token in the wild.

Tokens as Social Contracts

Tokens are sometimes called coins. You also learned that tokens themselves are smart contracts. (With enough repetition, these terms will hopefully enter your natural vocabulary by the end of this book!) But tokens themselves (like all forms of money) can also be seen as social contracts, or agreements between groups of users. In plain English, the implicit agreement of a group using a token would be as follows: "We all agree this token is money in our community." It's also a tacit agreement not to counterfeit, undermining the system!

The closest thing we have to a social contract in software form today is probably the end-user license agreements, or EULAs, that users sign when they create an account on services such as Facebook, Twitter, iTunes, or Gmail. This agreement usually includes language barring activities such as spamming other users, which would degrade the user experience.

Thinking this way allows us to imagine how our digital media and digital goods today might become digital collectibles that are discussed, marketed, sold, and displayed inside the social networks of the future, in which online artifacts like selfies and podcasts can be sold, licensed, or rented for fees of arbitrary size.

Tokens Are a Great First App

When making a token, consider that it is only as valuable as the community using it believes it will be. Thus, it is far easier to launch a token into an existing community that already trades using some kind of money or scrip. However, making sub currencies is not the only use of a crypto asset. The concept of an asset is highly generalized. Assets, in the form of financial contracts or smart contracts, can be used to represent shares of equity, or lottery tickets, or just scrip within a local economy. The price can be determined by the market, or it can be pegged to another asset. The rules are largely up to you.

In Ethereum, tokens exist within, and rely upon, the public blockchain: you can create a sub currency of ether, but ether will always remain the privileged token with which miners and gas costs are paid. If you want a purely independent blockchain network, you can create your own private blockchain and be completely disconnected from the main Ethereum chain. Making a sub currency is easier and will satisfy most use cases for curious developers. If you're working at an institution interested in using its own blockchain, never fear: you can make your own private chain and crypto economy that is separate and distinct from the Ethereum public chain.

Note: For topics Creating a Token and Deploying the Contract kindly visit the first link provided in the reference section.

6.7. PLAYING WITH CONTRACTS

Now that your contract is deployed with an interface in Mist, you can activate it. To call a contract in the EVM, you do not necessarily need to send any ether; you can call it simply by sending zero ether to the contract address. Boom, now you are the owner! If this doesn't work, be sure that the contract was uploaded to the testnet, and that the Mist you are using to send the zero-ether transaction is also on the testnet. For the Owned contract, activation is a yes-or-no question. You can call it with zero ether or 100. In more-sophisticated contracts, the amount you send is vital to how the contract behaves subsequently after being called. Owned is just a reference contract that might live on the EVM, a pivotal public resource contract with lots of incoming references, for years and years. By working with a small smart contract, you can see how smart contracts are used piecemeal to cobble together entire distributed apps, largely using boilerplate code or public-use instances, enabling the end programmer to just write the most customized of functionality, reducing the room for error.

6.8 SUMMARY

In this chapter, you were able to deploy two separate smart contracts. In the process, you learned about the most basic application you can write for the EVM, a token contract. You also considered some of the unique properties of distributed programs by playing with owned.sol. By now, you should begin to see how powerful the Ethereum protocol can be, and how simple and easy it is to deploy contracts that leverage the power of the network.

Next, it's worth learning more about how the EVM network-database achieves consensus: a process known as proof-of-work mining.

6.9 REFERENCE:

1. Creating a Token, Deploying the Contract, Playing with Contracts.

Smart contracts introduction:
https://www.ibm.com/topics/smart-contracts
2. What is EVM
   - https://coinmarketcap.com/alexandria/glossary/ethereum-virtual-machine-evm
   - https://ethereum.org/en/developers/docs/evm/

6.10 QUESTIONS

1. Define Smart contracts.
2. Briefly Describe the working of smart contracts.
3. Give Benefits of Smart contracts.
4. List the Applications of smart contracts.
5. Explain EVM as Back End.
6. Explain “Cryptocurrency Is A Measure of Time”
7. Exercise
    a. Create a custom token with no code.
    b. Watch token
    c. Deploy a Simple Contract in 5 minutes

Unit IV

7

MINING ETHER

Unit Structure

7.0 Objectives
7.1 Why
7.2 Ether's Source
7.3 Defining Mining
7.4 Difficulty
7.5 Self-Regulation and Race for Profit
7.6 How Proof of Work Helps Regulate Block Time
7.7 DAG and Nonce
7.8 Faster Blocks
7.9 Stale Blocks
7.10 Difficulties
7.11 Ancestry of Blocks and Transactions
7.12 How Ethereum and Bitcoin Use Trees
7.13 Forking
7.14 Mining
7.15 Geth on Windows
7.16 Executing commands in the EVM via the Geth Console
7.17 Launching Geth with Flags
7.18 Mining on the Testnet
7.19 GPU Mining Rigs
7.20 Mining on Pool with Multiple GPUs
7.21 Summary
7.22 Questions
7.23 References

7.0 OBJECTIVES:

After completion of the chapter , students get idea on following :

1.Understanding the Ethereum framework

2. Detailed understanding on Mining ether using several installations

3. Understanding of Go Ethereum working ,Testnetdemostration

7.1 WHY

Mining is the process by which the Ethereum network reaches consensus about the order of transactions in a given period of time, which in turn allows the EVM to make valid state transitions.

Using mining, the idea of decentralized systems with security can be conceptualized. Also there will be fair understanding of how long these systems can continue and at each level how security will be monitored.

7.2 ETHER'S SOURCE

Ether is considered the native token of Ethereum because it gets created out of thin air during the mining process, as payment for mining work performed by computers.

Because mining is computationally intensive, it can generate large electricity costs for your home or office. Miners take their rewards seriously.

Mining rewards are accomplished through an account balance increase programmed into the EVM's state transition function. They are payable to whichever a random miner finds a block.

7.3 DEFINING MINING

Miners refers to a vast global network of computers, operated mostly by enthusiasts in their homes and offices, running Ethereum nodes that are paid in ether tokens for the work of executing smart contracts and validating the canonical order of transactions around the world.

The process of mining is undertaken by each individual node, but the term also refers to the collective effort of the network: individual nodes mine, and the network itself can be said to be secured by mining. Miners process transactions in groups known as blocks.

Mining can properly be defined as dedicating computational effort to the bolstering of a given version of history as the correct one.

The mining process is computationally demanding for nodes because it involves executing a memory intensive hashing algorithm known as a proof-of-work algorithm. The proof-of-work algorithm for the Ethereum protocol is Ethash, a new function created by the core developers in order to address the problem of mining centralization evident in Bitcoin.

The cryptographic proof which results from mining can be completed more quickly when more hashpower is applied. Therefore, miners often form mining pools to increase their chances of winning rewards, which they then split among the group.

Note: What's a block?

A block refers to the data object containing those transactions, stored on Ethereum nodes. Each time a node starts, it must download the blocks it missed while offline. Each block contains some metadata from the previous block, to prove it is authentic and build on the existing blockchain.

7.4 DIFFICULTY

Ethereum and Bitcoin are self-regulating networks. As a network gets more popular, more mining hash power joins in search of profits, and blocks might be found too quickly.

To stay within range of its ideal 15-second block time, a dynamically self-adjusting value called difficulty will increase.

If blocks are found too quickly or slowly, the system changes the difficulty to get within range of its ideal block time. As time progresses, network difficulty increases.

Network difficulty may decrease or go flat if miners begin to drop off the network or if overall hash power decreases.

You can think of this difficulty variable as being part of the incentive structure to get miners on the network as soon as possible and to stay there.

However, difficulty has another use in the EVM, as one of several factors used to determine a block's score, sometimes referred to as its heaviness.

The heaviest, or highest-scoring, path through the transaction data structure can be said to be the longest, the one that most miners have historically converged upon as the true root-to-leaf path.

7.5 SELF-REGULATION, AND THE RACE FOR PROFIT

Mining is designed to be a money-maker for the people who engage in it; they are paid for providing security to the network.

The first thing to know is that time is a factor! When a new crypto currency launches, miners rush to turn on their machines.

With less competition for fees in the early days, they earn more. Even better, tokens belonging to useful crypto networks usually inflate in price over their lifetime, so earning them earlier gives miners more opportunity to profit from appreciation.

7.6 HOW PROOF OF WORK HELPS REGULATE BLOCK TIME

Ethereum, like Bitcoin, currently uses a consensus protocol called Proof-of-work (PoW). This allows the nodes of the Ethereum network to agree on the state of all information recorded on the Ethereum blockchain, and prevents certain kinds of economic attacks

Proof of Work (PoW) is the mechanism that allows the decentralized Ethereum network to come to consensus, or agree on things like account balances and the order of transactions. This prevents users "double spending" their coins and ensures that the Ethereum chain is incredibly difficult to attack or overwrite

Proof-of-work is the underlying algorithm that sets the difficulty and rules for the work miners do. Mining is the "work" itself. It's the act of adding valid blocks to the chain. This is important because the chain's length helps the network spot the valid Ethereum chain and understand Ethereum's current state. The more "work" done, the longer the chain, and the higher the block number, the more certain the network can be of the current state of things

Block time defines the time it takes to mine a block. Both in bitcoin blockchain and Ethereum blockchain, there is an expected block time, and an average block time.

Miners who successfully create a block are rewarded in 2 freshly minted ETH and all the transaction fees within the block. A miner may also get 1.75ETH for an uncle block. This is a valid block, created simultaneously to the successful block, by another miner. This usually happens due to network latency

Anyone who can optimize for the proof-of-work algorithm can find valid blocks faster, causing uncles to lag further and further behind. In the Bitcoin network, a small group of hardware companies has acquired a disproportionately huge amount of power over the network by creating hardware specifically built to run the Bitcoin PoW algorithm. The centralisation of mining efforts is highly profitable in Bitcoin, because it allows these big miners to find blocks faster, reaping all the block rewards. Slower machines never get a chance to solve a block, and eventually, even their uncle blocks come in further and further behind the winning block.

In Ethereum, uncle blocks are required to bolster the winning block. As uncles lag more, it becomes harder for the network to find a true block, being that valid uncles are a requirement.

Enter the Ethash algorithm: The Ethereum protocol’s defense against mining hardware optimization. Ethash is a derivative of Dagger-Hashimoto, which is a memory hard algorithm that can’t be brute-forced

with a custom application-specific integrated circuit (ASIC), like the kind that are popular with Bitcoin mining enterprises.

Key to this algorithm memory-hardness is its reliance on a directed acyclic graph(DAG) file, which is essentially a 1 GB dataset created a new every 125 hours, or 30,000 blocks. This period of 30,000 blocks is also known as an epoch.

Directed acyclic graph is a technical term for a tree in which each node is allowed to have multiple parents, with ten levels including the root, and a total of up to 225 value

7.7 DAG AND NONCE

DAG stands for Directed Acyclic Graph, and it is an ever-important element within the structure of Ethereum mining. DAG is a dataset over 1GB in size that is used by all Ethash coins to find solutions along the blockchain. It is used in all Ethash coins, like Ethereum, EthereumClassic, Metaverse, Ubiq and other coins to provide a proof of work. DAG file is generated every mining epoch and it increases from epoch to epoch A nonce is the number of the transaction of the sender's address. Every transaction from an address is numbered sequentially, beginning with 0 for the first transaction. For example, if the nonce of a transaction is 10, it is the 11th transaction sent from the sender's address.

In effect, each node is playing a guessing game with itself, trying to guess a nonce that will validate the current block; if it guesses the right nonce, it wins the block reward. If not, it continues guessing until it gets word that another node on the network has found a winner. Then, it discards the block it was mining downloads the new block, and begins mining a new block on top of that one.

But the node gets both parameters of the guessing game, as well as a new pair of dice (so to speak) with each potential block as it rolls in. The rules of the guessing game are designed this way to prevent clever individual nodes from outsmarting the system in the pursuit of more mining rewards.

Therefore, you can think of the DAG file as a way of standardizing the solution time of the proof-of-work algorithm. It levels the playing field for miners, but more important, helps cluster block times around the 15-second mark by ensuring that—even with massive computing power—you can’t guess the correct nonce a whole lot faster than your competitors.

All the data a node needs to participate in the guessing came is drawn from the blockchain itself. In cryptography, an encryption seed can be used to help generate a pseudorandom number, thus increasing the randomness of whatever encrypted output the Ethash algorithm produces. In Ethereum and Bitcoin, each node gets the seed from looking at the hash of the last known winning block. In this way, the node must be mining on

the correct, canonical chain in order to play the game correctly. Performing proof of work on an erroneous block (say, an uncle) cannot yield a winning block. This is helpful if you're trying to reduce unfair advantage in a proof-of-work scheme, which could be used by a large pool of miners to highjack the network onto a version of the truth in which everyone's ether is transferred to the hijacker's accounts.

Here is the process by which a node sets itself up to perform the PoW guessing game:

1. From an encryption seed derived from the block header, the mining node creates a 16 MB pseudorandom cache.
2. In turn, the cache is used to generate a larger 1 GB dataset that should be consistent from node to node; this is the DAG. This dataset grows over time, in a linear fashion, and is stored by all full nodes.
3. Guessing the nonce requires the machine to grab random slices of the DAG dataset and hash them together. This works similarly to using a salt with the hash function. In cryptography, a random data chunk you toss into a one-way hash function is called a salt. Salts are like nonces: they make things more random, and thus more secure.

7.8 FASTER BLOCKS

Block time defines the time it takes to mine a block. Both in bitcoin blockchain and ethereum blockchain, there is an expected block time, and an average block time. In bitcoin, the expected block time is 10 minutes, while in ethereum it is between 10 to 19 seconds.

Believe it or not, all these modifications to the original Bitcoin paradigm were made in the service of faster block times. Block times as low as 3–5 seconds may be mathematically feasible In both Bitcoin and Ethereum, we've said that block time is an idealized period for collecting transactions. Why is this? The system works to keep blocks as near as possible to the ideal, much the way that the human body tries to preserve homeostasis. The Bitcoin protocol targets 10-minute block times, and Ethereum targets 15 seconds. Once a true block is found, it takes a short while for other nodes to find out about it. Up until they discard their orphan block and begin mining on the new one,they are actually competing against the new block instead of building upon it. Thus, the effort expended on the orphan is wasted. Think of it this way: if latency causes miners to hear about new blocks an average of one minute late, and new blocks come every 10 minutes, then the overall network is wasting roughly 10 percent of its has power.

Lengthening the time between blocks reduces this waste. In the opinion of some blockchain theorists, Satoshi Nakamoto chose this ratio because it seemed an acceptable level of waste. Ethereum's faster block time is desirable because it makes transactions confirm faster, but the Ethereum protocol has had to make provisions in its design for the

commensurate decrease in security brought on by faster block times Block time can be compared to settlement time in a securities trading, which in the United States, stands at three days after the trade date, also known as T+3. A proposal is under consideration by the SEC to quicken settlement time to T+2. In Bitcoin, which has no smart-contract execution, blocks take a theoretical 10 minutes on average, but in reality, transactions process this quickly only about 63 percent of the time. About 13 percent of the time, it takes longer than 20 minutes for a transaction to receive a confirmation. During this time, it's possible to reverse a transaction up to 20 percent of the time. While merely irksome for Bitcoin enthusiasts and businesses, these conditions are unacceptable for a smart-contracts platform designed to power distributed software applications, so Ethereum takes a slightly different approach to mining, in order to achieve faster block times.

Making Fast Blocks Work

We've already discussed how faster block times are more desirable from the perspective of user experience. However, they can also produce undesirable effects. Because nodes are located all over the world, it's hard for them to stay perfectly in sync. That's because information takes time to travel across the Internet from node to node, also known as latency. Although it may not seem like much time to humans, it's enough to create collisions in the transaction record where the books don't balance On average, it takes about 12 seconds for a transaction to propagate around the Ethereum or Bitcoin networks; in actuality, much of this time is consumed by the downloading of transactions to the node. In the intervening time before it hears about a new block being found, a miner may continue to work on an old block briefly, before discarding it for the new winner.

As described in the section above, uncles that receive mining effort after a valid block has already been found elsewhere in the network are also known as stale or extinct blocks. Faster block times create a higher likelihood of stale blocks, and stale blocks decrease the network's absolute strength against attacks. Worse yet, higher rates of stale blocks make it easier for mining pools to win increasing efficiency advantages over solo miners, consistently beating them out of mining rewards. At best, this is unfair, and at worst, it makes the network less expensive to attack.

7.9 STALE BLOCKS

Stale blocks, are blocks that are not accepted into the blockchain network due to a time lag in the acceptance of the block in question into the blockchain, as compared to the other qualifying block.

In Ethereum, as we've said already, orphans or stales have yet another name: they are called uncles, Uncle blocks are created in Ethereum blockchains when two blocks are mined and submitted to the ledger at roughly the same time. Only one can enter the ledger as a block, and the other does not. and they are counted toward the score, or weight,

of a block. The way this is done in the Ethereum protocol is similar to the blockchain scoring system proposed in the GHOST protocol, which was outlined in a paper by Aviv Zhoar and Yonatan Sompolinsky in December 2013.

The concept of GHOST is simple. Miners that find orphan and stale blocks get rewarded, but the reward is lower than for standard blocks Vitalik Buterin describes the way he has adapted the GHOST idea for Ethereum, and how it compares to Bitcoin:

The idea is that even though stale blocks are not currently counted as part of the total weight of the chain, they could be; hence they propose a blockchain scoring system which takes stale blocks into account even if they are not part of the main chain. As a result, even if the main chain is only 50 percent efficient or even 5 percent efficient, an attacker attempting to pull off a 51 percent attack would still need to overcome the weight of the entire network. This, theoretically, solves the efficiency issue all the way down to 1-second block times. However, there is a problem: the protocol, as described, only includes stales in the scoring of a blockchain; it does not assign the stales a block reward.

Uncle Rules and Rewards
The following are rules regarding uncles:

In Ethereum's implementation of GHOST, uncles that are validated along with a block receive 7/8 of the static block reward, or 4.375 ether. A maximum of two uncles are allowed per block. These two places are won on a first-come, first-served basis. No transaction fees are collected or paid out for uncle blocks, because users are paying these costs once already in the valid block, which actually executes their commands. Crucially, in order to be worthy of a reward, an uncle block must have an ancestor in common with the true block within the last seven generations. This implementation of GHOST solves the issue of security loss by including uncle blocks in the calculation of which block has the largest total proof of work backing it. The uncle rewards are intended to solve the second issue, centralization, by paying miners who contribute to the security of the network, even if they do not nominate a winning block.

7.10 DIFFICULTIES

Ethereum's "difficulty bomb" refers to the increasing difficulty level of puzzles in the mining algorithm used to reward miners with ether on its blockchain It's worth mentioning that the GHOST protocol (even as Ethereum has adapted it) is the subject of some criticism. Although its flaws are known, they are generally regarded to be harmless.

Fixing the GHOST implementation may not be worthwhile anyway, as it will be rendered deprecated when the Ethereum protocol moves away from a proof-of-work to what is known as a proof-of-stake consensus algorithm. One reason why cryptocurrency have value in the marketplace is that they are limited in issuance.

Today, 12.5 bitcoins are awarded per block (that is, every 10 minutes). This rate will continue until mid-020, when 6.25 bitcoins per block will be awarded for each block. Rewards halve this way every four years until approximately the year 2110–40, when 21 million bitcoins will have been issued. Ethereum achieves its limited issuance by planning to end the proof of work period entirely. The effective mining period for Ethereum will come to a close sometime in 2017–2018 when the Ethereum system makes the switch; one of the big selling points of proof of stake (or PoS) is that it does not require mining (and the accompanying energy expenditure) to reach consensus.

In an effort to force this transition, and simultaneously limit the issuance period for ether, the core developers have built in a difficulty bomb that makes proof-of-work mining less and less feasible beginning in the latter half of 2017, before finally becoming impossible in 2021. How this new proof-of-stake system will work is the subject of much research and debate within the community.

Miner's Winning Payout Structure

A successful miner of a winning block receives a flat payment, plus transaction fees, plus a share of the bounty of all uncles that helped it win. Thus it can be said the rewards in the Ethereum protocol are determined as follows:

1. A set block reward of 5.0 ether (for the miner that finds the winning block)
2. Fee payments of the gas expended within the block (for the miner that finds the winning block)
3. 1/32 ether per uncle of this block (for miners that find uncles)

Limits on Ancestry

The part of the protocol requiring uncles to be within seven blocks of the winning block to receive a partial award exists to make block history "forgettable" after a small number of blocks. The number seven was picked because it offers a reasonable amount of time for a miner to find an uncle, but not so long that it imposes centralization risks.

7.11 ANCESTRY OF BLOCKS AND TRANSACTIONS

Before a completed block can undergo processing and acceptance by the rest of the network, and before nodes can begin mining on top of a new block, each and every node must independently download and validate the block before beginning to mine in top of it.

Here are all the steps the block validator algorithm takes, in order:

1. Check if the previous block referenced exists and is valid.
2. Check that the timestamp of the block is greater than that of the referenced previous block and less than 15 minutes into the future.

3. Check that the block number, difficulty, transaction root, uncle root and gas limit (various low-level Ethereum-specific concepts) are valid.
4. Check that the nonce on the block is valid, showing the evidence of proof of work.
5. Apply all transactions in this now-validated block to the EVM state. If any errors are thrown, or if total gas exceeds the GASLIMIT, return an error and roll back the state change.
6. Add the block reward to the final state change.
7. Check that the Merkle tree root final state is equal to the final state root in the block header.

Only after these seven steps is a block canonized as valid and true! To make a blockchain, it would be theoretically possible to create block headers that directly contain data about every transaction, but this would pose scalability challenges and require immensely powerful hardware to run a node.

In Bitcoin and Ethereum, a data structure called a Merkle tree is used to avoid putting every single transaction in the header, which would be large and unwieldy. Ethereum adds a data structure representing the state of the EVM, called a state tree. Global state is presented in an Ethereum block by another tree structure known as a Patricia tree.

First and foremost, the role of tree structures is to help the node verify the data it receives inside blocks, such as the transaction ledger. Secondarily, their role is to do this fast, so that computers of all shapes and sizes can read the blockchain quickly.

In computer science, an associative array (or dictionary) refers to a collection of (key/value) pairs. In an associative array, the association between keys and values can be changed. This association is called a binding.

Operations associated with dictionaries include the following:

- Adding key/value pairs to the collection
- Removing pairs from the collection
- Modifying existing pairs
- Looking up a value associated with a given key

Hash tables, search trees, and other specialized tree structures are common solutions to the dictionary problem, where a dictionary is a generic term for a database of records. Solving dictionary problems involves methodologies for querying for a key (a word) and calling up its value (a definition).

7.12 HOW ETHEREUM AND BITCOIN USE TREES

In mathematics, a tree is an ordered data structure used to store an associative array of keys and values.

A radix tree is a variant that is compressed, requiring less memory. In a normal radix tree, each character in the key describes a path through the data structure to get to the corresponding value, like a set of directions.

Creating a Merkle tree requires hashing a large number of "chunks" of transaction data together until they become only one: a root hash. In Ethereum and Bitcoin, the Merkle tree structure is used to record the transaction ledger in each block. The root for the Merkle tree is hashed in with other metadata and included in the header of the subsequent block.

Thus, it can be said that each additional transaction (within each block) irrevocably changes the Merkle root; even one wrong transaction will make the root hash look completely different and thus, obviously wrong.

This is how blocks can prove their legitimate ancestry to the block validator algorithm, which is part of the overall block processing routine.

For a Bitcoin client, determining the status of a single transaction is as easy as looking at the header of the most recent block of the main chain. There, the client should find the Merkle proof showing that the root hash for the block contains the transaction in one of its Merkle trees.

The Merkle root is a fingerprint of all the transactions, correctly ordered, that have occurred in the blockchain up until that block.

Merkle-Patricia Trees

Thanks to the block header, it's quick and easy for a node to look for, read, or verify block data. In Bitcoin, the block header is an 80-byte chunk of data that includes the Merkle root as well as five other things.

The Bitcoin block header contains:

- A hash of the previous block header
- A timestamp
- A mining difficulty value
- A proof-of-work nonce
- A root hash for the Merkle tree containing the transactions for that block

Merkle trees are ideal for storing transaction ledgers, but that's about it. From the perspective of the EVM, one limitation of the Merkle tree is that although it can prove or disprove the inclusion of transactions in the root hash, it can't prove or query the current state of the network, such as a given user's account holdings.

Contents of an Ethereum Block Header

To remedy this shortcoming and allow the EVM to run stateful contracts, every blockheader in Ethereum contains not just one Merkle (transaction) tree, but three trees forthree kinds of objects:

- Transaction tree
- Receipts tree (data showing the outcome of each transaction)
- State tree

To make this possible, the Ethereum protocol combines the Merkle tree with theother tree structure we described above, the Patricia tree. This tree structure is fullydeterministic: two Patricia trees with the same (key/value) bindings will always have thesame root hash, providing increased efficiency for common database operations such asinserts, lookups, and deletes. It is therefore possible for Ethereum clients to get verifiableanswers to all sorts of queries it makes to the network, such as the following:

- Has transaction X been included in block? (Handled by the transaction tree.)
- Tell me all instances of event Y in the last 30 days. (Handled by the receipts tree.)
- What is the current balance of contract account Z? (Handled by the state tree.)

7.13 FORKING

A network of miners may split in two, if they cannot agree on the longest, heaviest chain.

▪There's much ado about forking in the cryptocurrency community, where it seems to imply the fracture of a community of humans along with a loss of consensus in the machine network. In reality, nascent forks are constantly happening.

Sometimes one branch dies, sometimes both die, and sometimes one lives on to propagate a winning nephew block.

A fork occurs when two valid blocks point to the same parent, but some of the miners see one, and the rest see the other.

Effectively, this creates two versions of "the truth," ensuring that these two groups can no longer be said to be on the same network. A state fork is a much bigger deal than a protocol fork. In a protocol fork, no data is changed, but miners may adjust parameters or update code on their nodes to make them perform to a modified specification that the community has agreed is an overall improvement. Protocol forks can thus be said to be voluntary, whereas state forks are not necessarily so.

In Ethereum, these constant budding forks are resolved within four blocks, as a matter of mathematical certainty, as one chain finds a winner, gets longer, and begins to "pull" other nodes toward it with the incentive of not only the miner fee for finding and executing the correct block, but all the added incentive of collecting the uncle block rewards.

Sometimes a node will find the "right" chain after already receiving a reward for about one to three blocks. Once the node jumps to a better, longer, more winning chain, that mining reward may disappear.

However, this all happens within four blocks—that is, one minute—so these small errata are considered no big deal.

Deliberate forks are typically deployed by attackers in order to double-spend funds: to make money out of thin air by simultaneously sending one balance to many accounts.

In fact, anyone with more than 50 percent of the hash power can engender a “hostile” deliberate fork, so to speak. In a double spend attack, an attacker operating a fleet of miners, with a large amount of hash power, sends an ether transaction to purchase a product.

After getting hold of the product, the attacker puts together an erroneous block with a second transaction. This second transaction attempts to send the same funds back to the attacker. He or she then creates a block at the same level as the block which contained the original transaction, but containing the second transaction instead, and dedicates all possible hash power to mining on the fork.

Should the attacker have more than 50 percent of has power, the double spend is guaranteed to succeed eventually at any block depth. Below 50 percent it’s far less prone to succeed.

But this attack is still feared enough that, in practice, most exchanges and other institutions who use ether wait for several confirmations before considering the transfer complete.

7.14 MINING

Because a distributed system has no single owner, machines are free to join the Ethereum network at will and begin validating transactions. This process is known as mining.

Mining nodes confer to arrive at a consensus about the order of transactions across the system, which is necessary to tabulate everyone’s account balances on the fly, even as many transactions pass through the network. This process consumes electricity, which costs money, and so miners are paid a reward for each block they mine: about 5 ether.

Mining is the process of using computational work to nominate a block—that miner’s version of recent transaction history—as the canonical block for this, the most recent block on the chain.

Mining is the process by which the Ethereum network reaches consensus about the order of transactions in a given period of time, which in turn allows the EVM to make valid state transitions

Mining is important because it is the process by which consensus is reached in the system, and by which ether is created.

Bitcoin also uses mining to reach consensus, but the way things work in Ethereum is a little bit different, owing to its ability to execute smart contracts.

7.15 GETH ON WINDOWS

Geth is such a great tool for learning, and because it's fairly easy to install,

Installing Geth on Windows
Download the latest stable binary.
Extract geth.exe from zip, open command Terminal and type this:
chdir<path to extracted binary>
open geth.exe

7.16 EXECUTING COMMANDS IN THE EVM VIA THE GETH CONSOLE

The formula for Geth commands is:
geth [options] command [command options] [arguments...]
To restart Geth with the console, type the following: geth console

Your Geth client should be running with the console enabled, giving you a command prompt.

Let's create an account by using a JavaScript API call. Choose a password.

In the console, type this, then hit Enter:
personal.newAccount("your_new_account_password_here")

You can check out all your accounts in the console by typing the following:**personal.listAccounts**

7.17 LAUNCHING GETH WITH FLAGS

Another popular way to get things done at the Geth command line is to launch Geth with certain flags

To start Geth on the testnet, type this: geth --testnet
You'll see text output, except that this mining is taking place on the testnet. Press Control+C to stop it

7.18 MINING ON THE TESTNET

One note about mining, Why is this? Actually, there is no need for Mist to mine on the main net and take up your computer's resources, because your contracts will execute without you mining. This is because there are currently thousands of nodes already mining on the public Ethereum chain, and being paid real ether to do so.

While there may coincidentally be others mining on the testnet while you are testing your contracts, there may also not be. Because

there’s no real financial incentive to leave a miner running on the testnet, you might find yourself in a lull, with nobody else on the testnet. This is why Mist allows testnet mining along with its GUI contract deployment interface

7.19 GPU MINING RIGS

Most ether mining is done with specialized GPU miners like the ones in Figure 7.19, which are operated by me. Two of the machines pictured are running the Claymore Dualminer, a custom mining program written by a Bitcointalk.org forum member named Claymore, and which mines both ether and another cryptocurrency simultaneously on multi-GPU rigs.The third and fourth rigs pictured here are running ethOS, a special Linux distro specifically created for rigs mining Ethereum, Zcash, or Monero. This is a far easier solution if you’re building from scratch. However, this is easiest done on Ubuntu. If you’re running Ubuntu and you’d like to mine with multiple GPUs, it’s easiest done with AMD hardware.

Figure 7-19. *Four Ethereum miners running in the author’s basement*
*Credits :*http://ethosdistro.com.

Once your video cards are physically installed, a few quick commands are all that are needed. In Ubuntu 14.04, open your Terminal and type the following:

-? sudo apt-get -y update
-? sudo apt-get -y upgrade -f
-? sudo apt-get install fglrx-updates
-? sudoamdconfig --adapter=all –initial

Then reboot. Next, enable OpenCL by entering the following Terminal commands:

-export GO_OPENCL=true
-export GPU_MAX_ALLOC_PERCENT=100
-export GPU_SINGLE_ALLOC_PERCENT=100

You can check that the configuration worked correctly by opening the Terminal back up again and typing this: aticonfig --list-adapters

You should now see your AMD graphics cards in a list. The card denoted with an asterix (*) is the computer's default video output. If you see a black screen, your monitor may be plugged into the wrong video card

7.20 MINING ON POOL WITH MULTIPLE GPUS

Competition for mining rewards is intense. You can think of your miner's chances of finding a winning block as being represented by the ratio of your miner's hashing power to network difficulty.

People who are mining for profit seek to gain an edge by using powerful hardware to improve their chances. As Ethereum becomes more popular, time passes, and mining hashpower on the network increases, mining becomes less and less appealing for most users.

If for no other reason than to mine new cryptocurrencies in the future. If you have hardware accessible, there's no reason not to experiment with mining, even if buying ether outright may be cheaper than mining it in some localities.

Once downloaded, extract the archive and make the qt.miner script executable:

-tar zxvf qtminer.tgz
-cd ./qtminer
-chmod +x qtminer.sh

Finally, start QTMiner with the following command, where address is the Ethereum address you want to be paid mining rewards, and name is the name of this particular mining rig: ./qtminer.sh -s us1.ethermine.org:4444 -u address.name -G.

To check your earnings without opening Mist, which can take forever to sync, go to Ethermine.org and enter the same Ethereum address

7.21 SUMMARY

In this chapter everything is discussed about PoW for Block regulation, stale blocks, forking , mining using various tools like

Geth,EVM and demonstrating it on testnet.It discusses on how to create and work on testnet , also how certain blocks behave as stale blocks and how they are useful in the mining process.Understanding on how the ethereum execution happens on windows platform and detailed understanding of the Go Ethereum application.

7.22 REFERENCES

[1] Chris Dannen- “Introducing Ethereum and Solidity”- Foundations of Cryptocurrency and Blockchain Programming for Beginners by Apress.

7.23 QUESTIONS

1.How Proof of Work Helps Regulate Block Time
2.Explain DAG and Nonce
3.Differentiate between Faster Blocks and Stale Blocks
4.How Ethereum and Bitcoin Use Trees
5.Explain Forking and Mining
6. Enlist steps on executing commands in the EVM via the Geth Console
7. Demonstrate Mining on the Testnet
8. Explain the concept of GPU Mining Rigs and Mining on Pool with Multiple GPUs

8

CRYPTOECONOMICS

Unit Structure

8.0 Objectives
8.1 Introduction
8.2 Usefulness of cryptoeconomics
8.3 Speed of blocks
8.4 Ether Issuance scheme
8.5 Common Attack Scenarios
8.6 Summary
8.7 Questions
8.8 References

8.0 OBJECTIVES :

After completion of the chapter , students get idea on following:

1.Understanding the crypto economics concept and its applications

2. Understanding common attack scenarios

8.1 INTRODUCTION

The study of economic activity conducted across secure computer networks is known as cryptoeconomics.

Game theory is used in economics, defense planning, psychology, political science, biology, and even the study of gambling (!) as a methodology for studying, analyzing, and predicting the behavior of humans and computers working inside a known system.

The digital communication age we know today was ushered in by cryptanalysis, also known as code-breaking.

Cryptography makes it possible to keep the meaning of those signals private to the sender and recipient, even when messages travel across the globe, riding along many networks along the way—some of which may be equipped with a spying apparatus.

The field of economics typically studies interactions between people, sometimes in hostile contexts such as war.

The emerging field of cryptoeconomics is the study of economic activity conducted across network protocols in an adversarial environment.

The domains of cryptoeconomics include the following:
- Online trust
- Online reputation
- Cryptographically secure communication
- Decentralized applications
- Currency or assets as a web service (so to speak)
- Peer-to-peer financial contracts (smart contracts)
- Network database consensus protocols
- Antispam and anti-Sybil attack algorithms

Applied cryptoeconomics is creating a game-like system with workable incentives and disincentives, which create a stable tension that keeps the network up and running.

8.2 USEFULNESS OF CRYPTOECONOMICS

a.Engineering a layer of defence:

Applied cryptoeconomics is about engineering a layer of defense between public networks and attackers of all sizes.

It combines game theoretical system design, encryption, and cryptographic hashing to protect a commonly used, commonly operated resource—in this case, a global transaction state machine.

b.Mining Pool

Mining pools contribute to centralization, which is why any pool with larger than 25 percent hashpower is approaching the threshold of network threat.

Should two such pools emerge, they might quickly get control of a network.

c.Encryption

Encryption turns a human-readable string of letters or numbers into an unreadable blob of random letters and numbers with one important caveat.

The ciphertext that comes out of encryption algorithms does not have a fixed length.

Pretty Good Privacy (PGP) and Advanced Encryption Standard (AES) are popular algorithms for doing.

8.3 SPEED OF BLOCKS

Subroutines in the mining process are engineered to maintain that block time.

The latency for Bitcoin nodes around the world, about 95 percent of them can be reached in 12.6 seconds, as measured by an academic team in 2013.

This number is proportional to block size, so in a “faster” block time currency, you could have a more responsive network.

Fast blocks are less secure in the near term, for reasons that we won’t get into here.

But in their favor, they produce fast confirmation times; in order words, they benefit from more granularity of information.

While nodes may be easier to fool initially, they are drawn powerfully toward the “true” chain within a few generations. The idea that faster blocks are proportionally less secure than slower blocks is false.

8.4 ETHER ISSUANCE SCHEME

Ether is created by the network to pay miners.

Some ether was pre-sold in mid- 2014 to bootstrap the funding of the network.

Approximately 60 million ETH were sold at prices varying from 1,000 to 2,000 ETH per bitcoin.

(About 10 percent was allocated to the Ethereum Foundation, and another 10 percent was maintained as a reserve at the time of the presale.)

From the presale forward, the system will issue 15.6 million ether per year in the form of rewards paid to miners.

Ether never stops being issued, but the amount issued per year is a smaller and smaller percentage of the overall pool.

Thus, ether’s issuance scheme is inflationary (in terms of quantity, not price) until approximately 2025, and deflationary in quantity thereafter.

The price of ether is whatever the market dictates, and is predicated mostly on demand of time on the EVM.

8.5 COMMON ATTACK SCENARIOS

The state transition function is bounded to a limited number of computational steps per block. If execution runs longer, it is cut off, and those state changes are reverted. However, fees are still paid to the miners for these rolled-back changes.

The rationale for this design decision in the protocol becomes apparent when viewed through a cryptoeconomic lens.

The Ethereum White Paper uses the following examples to demonstrate the usefulness of its specification when the network is under attack:

a. If an attacker sends a miner a contract containing an infinite loop, it will eventually run out of gas. However, the transaction is still valid in the sense that the miner can claim a fee from the attacker for each computational step the program took.

b. Even if an attacker tries to pay the appropriate gas fee to keep the miner working, the miner will see that the STARTGAS value is excessively high and will know ahead of time that the computation will take too many steps.

c. Imagine that an attacker is careful with his gas payment: the attacker sends contract code with just enough to make a withdrawal, but not enough to make the balance of the account go down. This is similar to a double-spend attack, in that it creates money out of thin air. However, in Ethereum, this transaction would be entirely rolled back because it ran out of gas in the middle

8.6 SUMMARY

In this chapter everything is discussed about cryptoeconomics, its applications, working and common attacking scenarios.It focuses on the idea of the attacked scenario in decentralised applications as compared to the client server architecture. Also, how the speed in which blocks are mined decides the architecture of economics in the cryptocurrency environment.It discusses the case studies and scenarios on how the architecture can be bridged depending on the attacker perspective.

8.7 REFERENCES

Chris Dannen- “Introducing Ethereum and Solidity”- Foundations of Cryptocurrency and Blockchain Programming for Beginners by Apress.

8.8 QUESTIONS

Q1. Describe Cryptoeconomics
Q2. Enlist applications based on cryptoeconomics
Q3. How speed of blocks play an important role in mining
Q4. Explain common attacking scenarios in working of cryptocurrency

Unit V

9

BLOCKCHAIN APPLICATIONS DEVELOPMENT

Unit Structure

9.0 Objectives
9.1 Introduction
9.2 Decentralized Applications.
9.3 Blockchain Applications Development
9.4 Interacting with the Bitcoin Blockchain
9.5 Interacting programmatically with Ethereum –Sending Transactions
9.6 Creating a smart contract
9.7 Executing a smart contract function
9.8 Public versus private Blockchains
9.8.1 Public Blockchain
9.8.2 Private Blockchain
9.10 Decentralized Applications Architecture
9.11 Summary
9.12 Reference for further reading
9.13 Questions

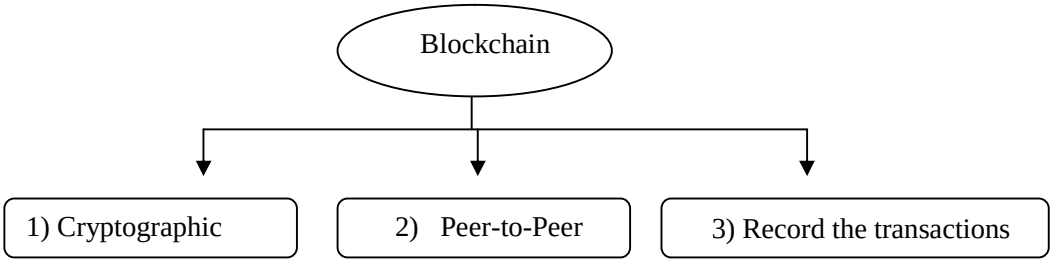
9.0 OBJECTIVES

- This chapter will help you to understand the basic concepts - of Blockchain and its applications.
- What is the need for Blockchain technology?
- And how does it will enhance the goal for recording digital information.
- Use of Bitcoin.
- Use of Ethereum software for creating smart contracts.
- Accessing the smart contract functions.
- Use of public and private Blockchains.

9.1 INTRODUCTION

As if we are aware about the new Blockchain technology which does not have standardized implementation. It is the technology that

maintains the transactions log (legder) for example the one used for Bitcoin which allows the recording of transactions on a spread log (distributed legder) via network of users. A Blockchain technology comprises of three prime technologies:



Figure(9.1 a)

9.2 DECENTRALIZED APPLICATIONS

When we talk about decentralized-it refers to the action or movement of command and decision making from a centralized system to a distributed network. For example: DApp –it is application software or a program that runs on a blockchain P2P network of computer rather on a single computer. In this application the data and records of operation are encrypted and a cryptographic token is required to access the application. The advantage of this application is that it's always accessible and do not have a single point of failure (SPOF) whereas as in classical application it was supported by a centralized database.

The DApp is supported by a smart contract that is connected to a Blockchain. A smart contract comprises of backend only and it is a small part of the whole DApp which is require in creating a decentralized application on a smart contract system. For example for creating a smart contract that is DAPP the most popular one is Ethereum.

9.3 BLOCKCHAIN APPLICATIONS DEVELOPMENT

Blockchain application development helps to increase reliability and speed up the exchange of information. It is a decentralized digital log that helps in saving the transactions which takes place on infinite numbers of computers around the globe.

In this application the transaction are grouped in blocks and its records one transaction after other in blocks of chain that is why it is named as Blockchain. The links between blocks and its content are protected via cryptography, so that the previous transaction cannot be destroyed, changed or created a new one.

In this the transactions are trusted without a central authority or a middleware. The ability of Blockchain to record, store, and move any kind

of data or records with great efforts in controlling the process in a decentralized manner has influenced the interest from new starts up and overall financial services industry.

Listed some of its interest areas are:

9.3.1 Blockchain in capital markets: - using this technology we can freely speed up and streamline the entire trade and its process.

9.3.2 Blockchain for cross – border payments:- this can be achieve by speeding up and simplifying the process which helps in reducing cost and cutting down the use of middlemen.

9.3.3 Blockchain to enhance digital identity: - in this technology the identify is shared to whom and how when it's online transactions moved to a blockchain enabled framework.

In this technology the users are able to choose how they identify themselves.

9.3.4 Blockcahin in loyalty and rewards: - The blockchain technology gives many benefits like maintaining transparency and tractability of various transactions.

9.4 INTERACTING WITH THE BITCOIN BLOCKCHAIN:

A Bitcoin is nothing a digital currency which can be managed without any central bank or any administrator. It can be easily sent from one user to another user via peer-to-peer networks. Each Bitcoin is a computer file which is saved or stored in a digital wallet app on a Smartphone or computer. So various transactions which take place are recorded in a public list which is called as Blockchain. Bitcoin is the first crypto currency which can be easily buy, sell and exchange directly without any intermediary like for example-bank.

Blockchain is a chain of blocks that contains information. It is a distributed ledger that is completely open to anyone. The interesting property of Blockchain is that once the data has been recorded in blockchain it becomes very difficult to change it so how does it works Each block contains some data and the hash of the block and the hash of the previous block. The data that is stored inside the block is dependent type of Blockchain for example –Bitcoin blockchain stored the information about the transaction such as sender and receiver.

9.5 INTERACTING PROGRAMMATICALLY WITH ETHEREUM –SENDING TRANSACTIONS:

An Ethereum transaction refers to an action initiated by an externally-owned account, in other words an account managed by a human, not a contract. It is a decentralized, open source blockchain with smart contract functionality. Ether is the native cryptocurrency of the

platform. It was created to enable developers to build and publish smart contracts and distributed applications that are DApps which can be used without the risk of downtime, fraud, or any interference of third party. It is also known as a distributed computing platform that supports developing decentralized Digital Applications (DApps) using Blockchain technology. It can process transaction over one million times per day. On an average day, it gets anywhere within 15 seconds and 5 minutes to transform a transaction if the standard gas price is paid.

There are some of the steps getting started as an Ethereun Developers are:

Step 1: Get a blockchain.
Step 2: Talk to a blockchain.
Step 3: Write some smart contracts.
Step 4: Deploy those smart contracts.
Step 5: Make s smart contract call.
Step 6: Setup your account.
Step 7: Transaction with your smart contracts.

9.6 CREATING A SMART CONTRACT

- A smart contract is simply a program that runs on the Ethereum blockchain.
- Its' a collection of code and data that reside at a specific address on the Ethereum blockchain.
- Smart contracts are a type of Ethereum account. This means that they have a balance and they can send transactions over the network. However there are not controlled by a user, instead they are deployed to the network and run as programmed.
- User accounts can then interact with a smart contract by submitting transactions that execute a function defined on the smart contract.
- Smart contracts can define rules, like a regular contract and automatically enforce them via the code.
- Smart contracts are simply programs that are stored on a blockchain that run when predetermined conditions are met.
- They typically are used to automate the execution of an agreement without any intermediary's involvement or time loss.
- A smart contract is an agreement between two people in the form of computer code.
- They run on the blockchain, so they are stored on a public database and cannot be changed.
- The transactions that happen in a smart contract are processed by the blockchain, which means they can be sent automatically without a third party involvement.

- A blockchain is a digital network built and maintained by distributed computers running specific pieces of software.
- A smart contract is a software program that adds layers of information onto digital transactions being executed on a blockchain.
- On blockchain, the goal of a smart contract is to simplify business and trade between both anonymous and identified parties, sometimes without the need for a middleman.
- A smart contract is computer code that can automatically monitor, execute and enforce a legal agreement.
- The aim of the smart contract is to provide security while transaction and reduce surplus transaction costs.
- It saves time and conflict and is also cheaper, faster and more secure way of payment as compared to the traditional system.

9.7 EXECUTING A SMART CONTRACT FUNCTIONS

Smart contracts get executed by the blockchain nodes, which process the transactions that are committed by the user. A transaction can be submitted to any node in the blockchain network, which then broadcasts it to the entire network so that all the nodes will see the transaction so this helps in maintaining transparency.

Firstly the terms of contract should be determined by the contractual parties. Once the contracts gets finalised, they are translated into programming code. So the code basically represents a number of different conditions statements that describe the possible flow of future transactions. The code which is created, it is then stored in the blockchain network and it is replicated among all the participants in the blockchain. After that the code is run and executed by all computers connected in the network. If a term or you can say certain conditions are met or satirised and it is verified by all participants of the blockchain network, then the applicable transactions is executed.

9.8 PUBLIC VERSUS PRIVATE BLOCKCHAIN

9.8.1 Public blockchain- is open networks that allow anyone to participate without any prior permission. In this type of blockchain anyone can join the network and read, write, or participate within the blockchain. A public blockchain is decentralized and does not have a single entity which controls the network. Data on a public blockchain are secure as it is not possible to modify or alter data once they have been validated on the blockchain.

Some features of public blockchain are:

- **High Security –** It is secure due to Mining.
- **Open Environment –** The public blockchain is open for all.
- **Anonymous Nature –** In public blockchain every one is anonymous. There is no need to use your real name or real identity, therefore everything would stay hidden, and no one can track you based on that.
- **No Regulations –** Public blockchain doesn't have any regulations that the nodes have to follow. So, there is no limit to how one can use this platform for their betterment
- **Full Transparency –** Public blockchain allow you to see the ledger anytime you want. There is no scope for any corruption or any discrepancies and everyone has to maintain the ledger and participate in consensus.
- **True Decentralization –** In this type of blockchain, there isn't a centralized entity. Thus, the responsibility of maintaining the network is solely on the nodes.
- **Full User Empowerment –** Typically, in any network user has to follow a lot of rules and regulations. In many cases, the rules might not even be a fair one. But not in public blockchain networks. Here, all of the users are empowered as there is no central authority to look over their every move.
- **Immutable –** When something is written to the blockchain, it cannot be changed.
- **Distributed –**The database is not centralized like in a client-server approach, and all nodes in the blockchain participate in the transaction validation.

9.8.2 A private blockchain- are managed by a network administrator and the participants need to consent to join the network. So to join a private blockchain one needs to take permission. There are one or more entities which control the network and this leads to third-parties involvement for performing the transaction. In this type of blockchain only entity participating in the transaction have knowledge about the transaction performed whereas others will not able to access it i.e. transactions are private.

Some of the features of private blockchain are:

- **Full Privacy –** It focus on privacy concerns.
- **Private Blockchain** is more centralized.
- **High Efficiency and Faster Transactions –**When you distribute the nodes locally, but also have much less nodes to participate in the ledger, the performance is faster.
- **Better Scalability** - Being able to add nodes and services on demand can provide a great advantage to the enterprise.

Difference between Public blockchain and Private blockchain:

Sr. No	Basis of Comparison	Public Blockchain	Private Blockchain
1	Access	Anyone can read, write and participant in a blockchain. It is public to everyone no need to take permission.	For writing and reading in this type of blockchain one need to take permission before joining the network so it is private.
2	Participants in network	Not aware about each other	Know each other
3	Centralized/ Decentralized	Public blockchain is a centralized	Private blockchain is a decentralized
4	Speed	Slow	Fast
5	Transaction	Transactions are lesser	Transactions are more
6	Security	It is more secure due to decentralized and active participation. As there is high number of nodes in the network it is very difficult to attack the system and get control over the network.	It is very liable to be hacked and attack over the network. Hence it is less secure.
7	Examples	Bitcoin, Ethereum, Monero, Zcash, Dash, Litecoin, Stellar, Steemit etc.	R3 (Banks), EWF (Energy), B3i (Insurance), Corda.

9.10 DECENTRALIZED APPLICATIONS ARCHITECTURE

Decentralized applications (dApps) are nothing it's a digital applications or programs code that are stored and run on a blockchain or P2P network of computers instead of a single computer. To build a DApps it requires a special system in order to achieve high security and reliability. It is a secure an unchangeable programs code running on a decentralized network with a mixture of front-end and back-end traditional technologies are called as DApps. To work with DApps we require tokens. These Tokens are nothing it's a smart contract which is written on top of the decentralized platform like Ethereum. Smart contracts are set of instructions (program codes) that are needed to be met to commit a particular transaction. By getting these token one can get different services on a web resource or mobile Apps. In short the client interacts with decentralized platform directly with the help of Ethereum "wallet" software like Metamask .

9.11 SUMMARY

This content will help you to understand the basics concept of Blockchain with its functionality and its execution. It will also help you to understand the concepts of smart contract how to develop with steps and its uses. How smart contracts help to achieve accuracy, transparency, security, storage and backups. How Go-green can be achieve via smart contracts by going paper free across the globe and increasing the business conscious.

9.12 REFERENCE FOR FURTHER READING

- https://ethereum.org/en/developers/docs/smart-contracts/
- https://corporatefinanceinstitute.com/resources/knowledge/deals/smart-contract/
- https://www.geeksforgeeks.org/difference-between-public-and-private-blockchain/

9.13 QUESTIONS

Q1. What are decentralized applications?
Q2. What are Ethereum?
Q3. What are smart contracts?
Q4. Explain public and private Blockchain?

MCQ QUESTIONS FOR PRACTICE

Q1. What is a blockchain?
a. A blockchain is a centralized digital ledger consisting of records called blocks.
b. A blockchain is a decentralized, distributed, digital ledger consisting of records called blocks.
c. A blockchain is a digital database consisting of records called class.
d. None of the above.

Q2. P2P stands for _____
a. Private to Public
b. Password to Private
c. Peer to Peer
d. None of the above

Q3. The maximum number of bitcoins that can be created is _____.
a. 11 Million
b. 25 Million
c. 21 Million
d. 100 Million

Q4. The process of creating new bitcoins is known as _____.

a. Financing
b. Sourcing
c. Mining
d. None of the above

Q5. _____ is used for storing bitcoins**.**

a. Block
b. Wallet
c. Both a and b
d. None of the above

❖❖❖❖

10

BUILDING AN ETHEREUM DAPP

Unit Structure

10.0 Objectives
10.1 Introduction
10.2 Building an Ethereum DApp: The DApp
10.3 The DApp, Setting Up a Private Ethereum Network
10.4 Creating the Smart Contract
10.5 Deploying the Smart Contract
10.6 Client 12 CO5 53 Application
10.7 DApp deployment: Seven Ways to Think About Smart Contracts
10.8 Dapp Contract Data Models
10.9 EVM back-end and front-end communication
10.10 JSONRPC, Web 3
10.11 JavaScript API
10.12 Using Meteor with the EVM
10.13 Executing Contracts in the Console
10.14 Recommendations for Prototyping
10.15 Third-Party Deployment Libraries
10.16 Creating Private Chains
10.17 Summary
10.18 Reference for reading
10.19 Bibliography
10.20 Questions

10.0 OBJECTIVES

This chapter will help you to understand more deeply about

1. Use of EVM
2. Use of Javascript API
3. Use and deployment of libraries.
4. The third party implications.

10.1 INTRODUCTION

This course content will help out to understand more deeply about blockchain. A brief idea about the smart contracts and its creation along with its deployment and execution.

10.2 BUILDING AN ETHEREUM DAPP: THE DAPP

To build a DApps it requires a special system in order to achieve high security and reliability. It is a secure and unchangeable programs code running on a decentralized network with a mixture of front-end and back-end traditional technologies are called DApps. To work with DApps we require tokens. Token is nothing but are smart contracts which is written on top of the decentralized platform like Ethereum. Smart contracts are nothing but a set of instructions (programs and code) that are needed to meet in order to commit a particular transaction. By getting these token one can get different services on a web resource or mobile app. In short the client interacts with the help of Ethereum "Wallet" software like Metamask.

10.3 THE DAPP, SETTING UP A PRIVATE ETHEREUM NETWORK:

Steps to create and test the private network

1. Install Ethereum
2. Create directory structure
3. Create accounts with keypairs (public/private)
4. Create a genesis configuration with the account details
5. Create genesis blocks for each node
6. Running a Bootnode
7. Running a miner
8. Running peer nodes
9. Attaching to a node

Step 1: Install ethereum

```
sudo add-apt-repository ppa:ethereum/ethereum
sudo apt update
sudo apt install ethereum
sudo apt install puppeth
```

Step 2: Create directory structure

Create the directory structure shown below, the data for each of the nodes will stored in the directories node1, node2, node2 and the log files written out to the directory "log".

```
pt@pt-XPS-13-9350:~/ethereum$ tree -L 2
.
├── data
│   ├── node1
│   ├── node2
│   └── node3
└── log
```

Step 3: Create the accounts

Create 3 new accounts for each of the nodes; the keystore is stored in a specific directory for each node.

Note: the value after "Public address of the key".

geth — datadir data/node1/ account new

geth — datadir data/node2/ account new

geth — datadir data/node3/ account new

Step 4: Genesis configuration

Create a genesis configuration using the account information from the previous step. The 3 accounts are to be pre-funded in the genesis configuration to enable you to test various functionality.

Step 5: Create genesis blocks for each Node

geth — datadir data/node1/ init privnet.json

geth — datadir data/node2/ init privnet.json

geth — datadir data/node3/ init privnet.json

Step 6: Running a Bootnode

Generate a key and start the boot node on any random port

bootnode -genkey boot.key

bootnode -nodekey -addr :8009

Copy the enode URL, this will be used by each node to find its peers.

Step 7: Running a miner

Every node can be a miner node by passing parameters to the geth command. For our example we will be using just one node as the miner node and the other nodes for testing RPC API call. We will be also using the account associated with the miner node for mining and rewards.

For demo purposes the password is the same for each account and stored in a file called password.txt.

The options to the command line use the boot node, the network id (in genesis config), unlock the account using the password in the password.txt file and "mine".

Step 8: Running peer nodes

Peer nodes are started with the details of

1. Data directory
2. RPC ports to use
3. Network id (in genesis file)
4. Bootnode to use
5. APIs that are available on the node

Step 9: Attaching to peer nodes

Connect to a node 1 and run commands if you want to test various APIs (admin,debug,eth,miner,net,personal,shh,txpool,web3)

geth attach http://localhost:8101

> admin.nodeInfo gives the details of Node 1

10.4 CREATING THE SMART CONTRACT

Ethereum are open-source, publicly distributed computing platform, featuring smart contracts functionality to build decentralised apps on top of this platform it allows decentralized apps to be build on it with the help of smart contracts functionality it was developed with solid butyrins as an extension to original core blockchain concept let look an analogy think of the ethereum network as a collection of ethereum network as a collection of decentralized and distributed vending machine now you go and you put token inside these vending machine and then specify the action that you want to be taken by pressing in the number now whenever each number is pressed a certain code gets executed inside the machine these are nothing but smart contracts and then service is provided to the concerned user on the network now the tokens that you have put inside the machine were actually ethers. Ethers are the cryptocurrency token of the ethereum network and they are used to pay for the transaction fee there's also another parameter called gas. Now we will understand how thereum differs from bitcoin blockhain on various fronts now firstly both these blockchain are conceptually dil +fferent from each other while bitcoin core idea was to bring a decentralized digital money that is not governed by anyone ethereum although a derivative extends beyond the realm of cryptocurrencies that is why it is also referred to as the second generation of blockchain now talking about cryptocurrency tokens bitcoin crypto is the inductor itself and it called Bitcoin cash whereas ethereum crypto token are called ether.

10.5 DEPLOYING THE SMART CONTRACT AND EXECUTING CONTRACTS IN THE CONSOLE

In the Developing Smart Contracts guide we set up our development environment.

If you don't already have this setup, please create and setup the project and then create and compile our Box smart contract.

With our project setup complete we're now ready to deploy a contract. We'll be deploying Box, from the Developing Smart Contracts guide. Make sure you have a copy of Box in contracts/Box.sol.

Hardhat doesn't currently have a native deployment system, instead we use scripts to deploy contracts.

We will create a script to deploy our Box contract. We will save this file as scripts/deploy.js.

```
// scripts/deploy.js
async function main () {
  // We get the contract to deploy
  const Box = await ethers.getContractFactory('Box');
  console.log('Deploying Box...');
  const box = await Box.deploy();
  await box.deployed();
  console.log('Box deployed to:', box.address);
}

main()
  .then(() => process.exit(0))
  .catch(error => {
    console.error(error);
    process.exit(1);
  });
```

We use ethers in our script, so we need to install it and the @nomiclabs/hardhat-ethers plugin.

```
$ npm install --save-dev @nomiclabs/hardhat-ethers ethers
```

We need to add in our configuration that we are using the @nomiclabs/hardhat-ethers plugin.

```
// hardhat.config.js
require('@nomiclabs/hardhat-ethers');

...
module.exports = {
...
};
```

Using the run command, we can deploy the Box contract to the local network (Hardhat Network):

```
$ npx hardhat run --network localhost scripts/deploy.js
Deploying Box...
Box deployed to: 0x5FbDB2315678afecb367f032d93F642f64180aa3
```

10.6 CLIENT 12 CO5 53 APPLICATION:

- Money Transfer and Payment Processing. ...
- Supply Chains Monitoring. ...
- Retail Programs Based on Loyalty Rewards. ...
- Digital IDs. ...
- Sharing of Data. ...
- Protection of Royalty and Copyright. ...
- Digital voting. ...
- Transfer of Real Estate, Land, and Auto Title.

10.7 DAPP DEPLOYMENT: SEVEN WAYS TO THINK ABOUT SMART CONTRACTS

Blockchain technology can help to improve the basic services such as in trade services. It is based on the decentralized, digitalised and distributed ledger model and due to these properties it is more secure and robust in nature. It creates decentralized records of transactions. It keeps immutable records of all transactions.
Six ways are:

- Blockchain and Bitcoin are not the same
- Data stored on blockchain is public
- On the blockchain, private information is visible to everybody
- There is only one blockchain
- Smart Contracts are legal documents
- Blockchain – a buzzword, nothing more

10.8 DAPP CONTRACT DATA MODELS

Ethereum is a network convention which allows the users to create and run **smart contracts** over a decentralized network. A smart contract consist set of code that runs specific operations and interacts with other smart contracts. Unlike Bitcoin which stores a number, Ethereum stores executable code. Ethereum removes the involvement of third party while executing a specific transaction. The third party is replaced by the set of codes which actually reduced time and money.

Dapp = frontend + smart contract backend

Some of the most promising Ethereum tokens and Dapps are laying the foundation for the future of the Internet, including:

- Basic Attention Token (BAT): used to improve privacy and value transfer between users, publishers, and advertisers. Used in the Brave browser.
- Golem (GNT): used to run code on one or many distributed compute nodes.
- Minds: a social media platform that improves value transfer between content creators and consumers.
- TokenSets: used to manage cryptocurrency assets via tokenized automated asset management strategies.
- Aave: used to earn interest on cryptocurrency deposits and borrow cryptocurrency assets.
- IDEX: a decentralized cryptocurrency exchange.

10.9 EVM BACK-END AND FRONT-END COMMUNICATION

EVM known as Ethereum Virtual Machine is a Turing complete software that runs on the ethereum network it enables anyone to runs a program regardless of the programming language and even non-specialized programmers can create a program on the EVM let me tell you how all of this works a user request 2 ethers for example the sender initiates a transaction message code and must pay for each step of the program that they activated including computation and the memory storage now this fee is paid in ethos which is taking from the wallet of the sender and then this transaction fee is collected by the nodes in the network that ran the EVM now each and every node of the network runs the EVM to execute the same instructions and this is how the blockchain database is maintained and updated by the many nodes connected to the network now all of thses nodes are actually the minor nodes that executes and verify the transaction and then ethereum blockchain turn they are rewarded with ethers for each successful block mining now once the transaction is verified and recorded the user gets 2 ethers in his wallet.

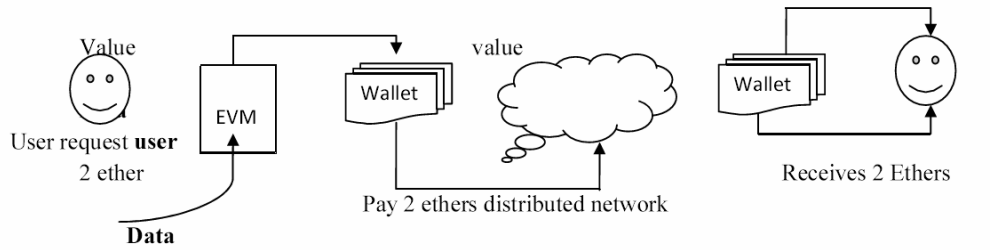


Figure (10.9 a)

10.10JSONRPC, Web

1) JSON-RPC is a simple RPC mechanism, similar to XML-RPC.

Protocol:unlike XML-RPC which is a client-server protocol, JSON-RPC is a peer-to-peer protocol. Its uses JSON (JavaScript Object Notation), as the serialization format and plain TCP streams or HTTP as transport mechanism.

JSON message types: it defines three messages types:

Request:method invokation with arguments encoded in JSON.

Response: Reply to method invokation containing the return arguments encoded in JSON.

Notification: Asynchronous request without response.

Specification: JSON-RPC is very simple that is JSON-RPC specification is very short.

2) JSON-RPC interactions JSON-RPC define 2 message exchange pattern that support most of the peer-to-peer interactions schemes. A Request-Response- sending JSON peer invokes a method on the remote JSON peer with a JSON request. The remote peer sends back a JSON response message.

10.11 JAVASCRIPT API

Application Programming Interface (API) it is a set of protocols used for building and integrating application software. It is used for app development and it enables the products and services to communicate with others without having any knowledge about its implementation, which results in time timing and money. It acts as a software middleware that allows two applications to communicate with each other. It sends the request to the provider and then provides the response back to the sender.

Web3.js

It is a collection of libraries which allow the users to perform various actions like sending Ether from one account to the other, read and write data from smart contracts, create smart contracts. In short this library is a collection of modules which contain specific functionality for the Ethereum ecosystem. By using the functionality of this module it allows one to interact with a local or remote ethereum node, by using either of the connection, HTTP, or IPC.It uses Ethereum Blockchain with JSON RPC (Remote Procedure Call) protocol for the interaction. As Ethereum is a peer-to-peer network of nodes that stores a copy of all the data and code on the blockchain, this API allows making requests to an individual Ethereum node with JSON RPC to read and write data to the network.

Ethers.js

It is a complete and compact general-purpose library that interacts with the Ethereum Blockchain and its ecosystem with the main features:

- It is an open-source platform.
- Keeps your private keys in your client, thus following a safe approach.
- It Import and export JSON wallets.
- Fully TypeScript ready, with definition files and full TypeScript source
- Complete functionality for all your Ethereum needs.

It is an alternative to Web3.js to build javascript frontend and interact with the Ethereum blockchain. This library is designed to make it simpler to write client-side JavaScript-based wallets, having the private key on the owner’s machine at all times.

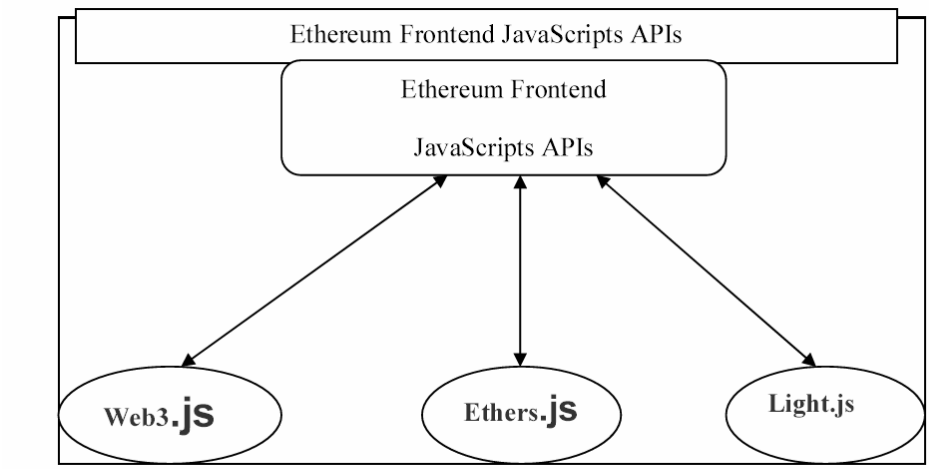
Light.js

This API provides a set of high-level tools for building light-client-efficient Dapps, primarily aimed for Dapp developers who are looking for an alternative to web3.js.

But what if you can't embed the light client in your application? Light.js works well with the remote full nodes, which makes it exceptional for all Dapps.

This library operates with the underlying goals.

- It picks the best pattern that works with light clients, listens to headers and makes API calls on a new header, making sure that the amount of network calls is not extreme.
- It put it into a high-level library so that Dapp developers use an easy API instead of following a well-known pattern**.**



Figure(10.11 a)

10.12 USING METEOR WITH THE EVM

Meteor is a full-stack JavaScript platform. It is used for developing modern web and mobile applications. It includes a set of technologies which helps in building connected-client reactive applications.

- Meteor allows you to develop in one language, JavaScript, in all environments: application server, web browser, and mobile device.
- Meteor embraces the ecosystem, bringing the best parts of the extremely active JavaScript community to you in a careful and considered way.
- Meteor provides full stack reactivity.

10.13 EXECUTING CONTRACTS IN CONSOLE.

Refer to 10.5

10.14 RECOMMENDATIONS FOR PROTOTYPING

1. Ethereum

One of the most ideal blockchain rapid prototyping platforms on the market today is Ethereum. With its robust smart contract flexibility and functionality, Ethereum has introduced multiple use cases.

At present, Ethereum is bringing a change in the consensus algorithm for quick Proof of Stake in the future.

2. Hyperledger Fabric

The Hyberledger Fabric is a B2B module hosted by Linux. The open-source project works with the aim to create codebase and enterprise-level distributed frameworks for databases.

Boasting of 185+ enterprises collaborating from a range of different industries, Hyperledger Fabric offers a grade solution that enables plug-and-play elements that wound the membership consensus and membership services.

3. R3 Corda

The R3 Corda platform is basically a consortium that consists of a few of the world's biggest financial institutions who have, together, developed a distributed platform of databases known as Corda.

While created mainly for the financial sector, Corda is even used to a great extent by healthcare, government, and supply chain industries.

This blockchain rapid prototyping tool has found popularization because of: A) Its consensus system that accounts the process of managing financial clauses, and B) Interoperability ease when integrating with legacy systems.

4. EOS

As a platform that is powered by native cryptocurrency, EOS emulates the attributes of an actual computer, including the GPUs and CPUs.

EOSIO mainly operates as a smart contract platform and a decentralized system that is intended for deployment of industrial-scale, decentralized use cases. The platform is known to offer all three elements — scalability, flexibility, and usability — that come together to make decentralized solutions a success.

5. Multichain

The platform is used for the development and deployment of a private blockchain — both inside or between two organizations. Its main purpose is to solve the blockchain deployment obstacle in the financial sector.

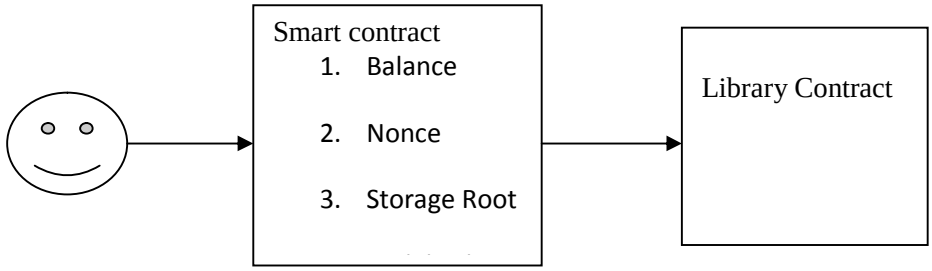
Offering control and privacy inside a private peer-to-peer chain, Multichain is the enhanced version of Bitcoin software used for conducting private financial transactions.

10.15 THIRD PARTY DEPLOYMENT LIBRARIES

Libraries are used to perform simple operations based on input and returns result. Libraries are not actually meant to change the state of the contract. Technically smart contracts are building block of a Dapp. In Ethereum smart contracts has address like external user account which helps to make interactions with the contract like calling methods , sending ether etc. Each contract has four properties.

1. Nonce : It's a count of number of transaction triggered from an account.
2. Balance: It's a number that tell about amount of ether this particular address holds
3. Storage root: Contract can store data, it's a root of tree which stores data from this contract
4. Codehash : It's hashed value of contract code**.**

How library works?



Figure(10.15 a)

Deployment of libraries: Library deployment is a bit different from regular smart contract deployment. There are two scenarios in the library deployment:

1. **Embedded Library:** If a smart contract is consuming a library which have only **internal functions** than EVM simply embeds library into the contract. Instead of using delegate call to call a function, it simply uses JUMP statement(normal method call). There is no need to separately deploy library in this scenario.
2. **Linked Library:** On the flip side, if a library contains **public or external functions** then library needs to be deployed. Deployment of library will generate unique address in blockchain. This address needs to be linked with calling contract.

10.16 CREATING PRIVATE CHAINS

Private chains are those blockchain which are managed by a single organization and this type of blockchain is permissioned blockchain.Its works on the access controls which restrict the users for participating in the network. In a private blockchain the entities whose who are participating in a transaction will have the knowledge about it whereas the others will not be able to access the network.

10.17 SUMMARY

This course will helps to understand the future trends of blockchain technology. This technology will helps you to encounter the issues like double spending and how it will improve the security, privacy, speed and reduced cost. How it can give visibility and traceability to any Business and individual control of data.

10.18 REFERENCE FOR READING

- https://docs.openzeppelin.com/learn/deploying-and-interacting#deploying-a-smart-contract
- https://www.marcopolonetwork.com/resources/essential-blockchain-technology-concepts/
- https://www.freecodecamp.org/news/what-is-a-dapp-a-guide-to-ethereum-dapps/

10.19 BIBLIOGRAPHY

- https://www.blockchain-council.org/blockchain/a-complete-guide-on-ethereum-frontend-javascript-apis/
- https://guide.meteor.com/
- https://docs.openzeppelin.com/learn/deploying-and-interacting#deploying-a-smart-contract
- https://medium.com/@pradeep_thomas/how-to-setup-your-own-private-ethereum-network-f80bc6aea088
- https://dzone.com/articles/the-list-of-best-blockchain-rapid-prototyping-tool
- https://www.freecodecamp.org/news/what-is-a-dapp-a-guide-to-ethereum-dapps/
- https://medium.com/coinmonks/all-you-should-know-about-libraries-in-solidity-dd8bc953eae7.

10. 20 QUESTIONS

Q1. What are Dapp?

Q2. What are EVM ?

Q3. What are private blockchain?

Q4. Does blockchain involves third party? If no explain?

Q5. What are smart contract.