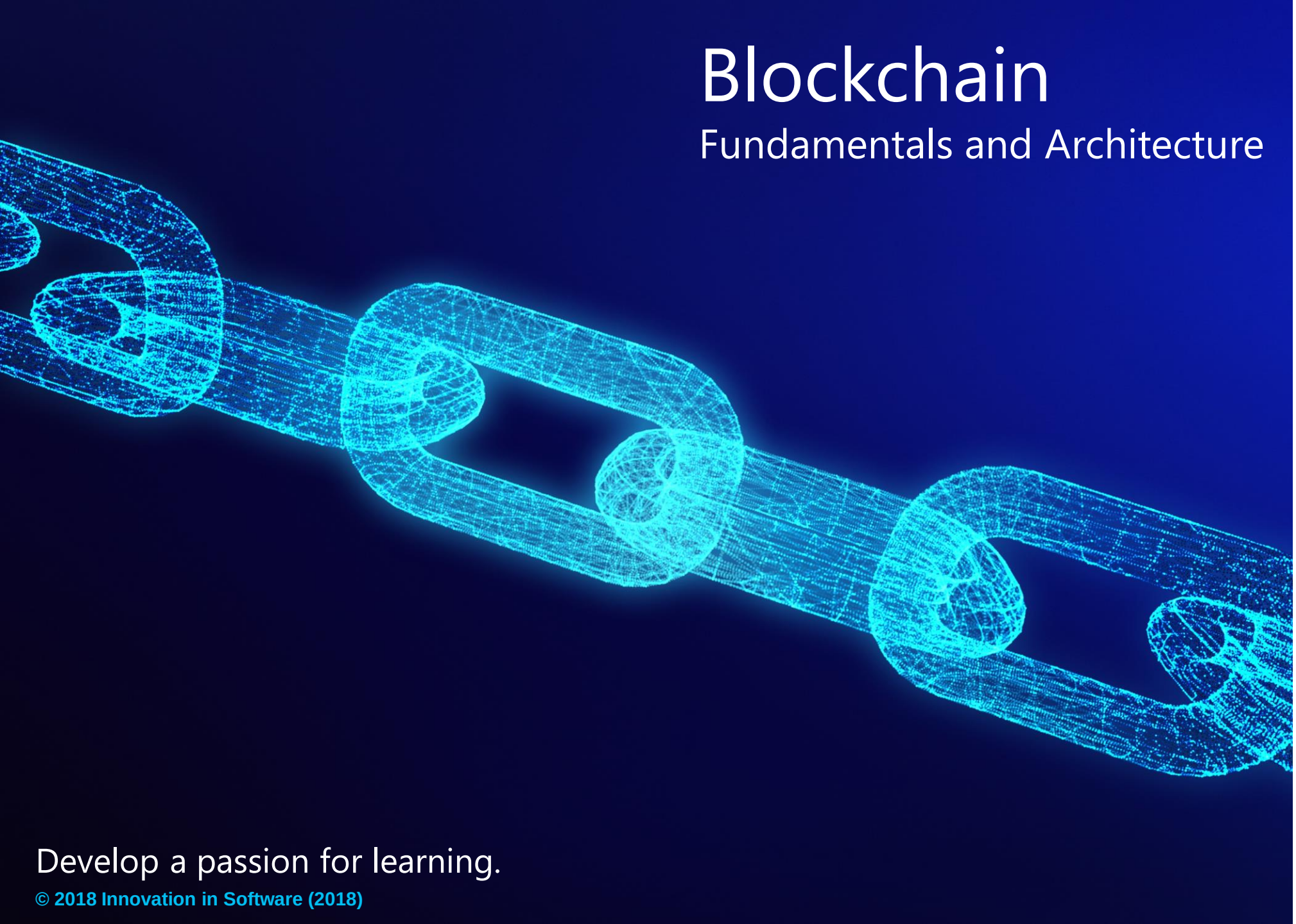
Blockchain

Fundamentals and Architecture



Develop a passion for learning.

EMERGING TECHNOLOGY

Blockchain has introduced the next generation of web applications.

What is Blockchain? The answer to that question is the source of considerable confusion. Blockchain is much more than Bitcoin, which is a cryptocurrency. With recent financial speculation on Bitcoin, interest in cryptocurrency has gone viral. Blockchain is part of the Bitcoin technology stack. There are a variety of implementations of Blockchain other than cryptocurrencies, such Ethereum and Hyperledger.

WHAT IS BLOCKCHAIN?

The implementation of Bitcoin is a mystery to many; even those actively speculating on the cryptocurrency. One reason is there is nothing tangible – *no coins*. If Bitcoin is a mystery, Blockchain is a true enigma. Misinformation on the web adds to the confusion. Here is a simple explanation: Blockchain is a decentralized web-based application that hosts a distributed ledger. Imagine a single accounting spreadsheet managed simultaneously by multiple Microsoft Excel web servers.

IMPLEMENTATIONS

Blockchain is a once in a generation disruptive technology based on a simple concept: distributed ledger. For technology, once in a generation means about every two years. Bitcoin is the best example and presently disrupting financial markets worldwide. Ethereum and smart contracts are another example and has the potential to disrupt a variety of industries, including legal, healthcare, election voting, smart locks, and distributed autonomous organizations (DAO).

POP QUIZ: WHAT IS THE ANSWER?

What is a distributed autonomous organization and the benefits?

10 MINUTES

CRYPTOCURRENCY BEFORE BITCOIN?

The last two years, our dependence on cash has lessened considerably. The majority of our interaction with money is digital (credit cards, ACH, debit cards, etc.) and may involve online commerce.

For most of us, our checking account is a bank statement (ledger) that is received periodically. We rarely if ever touch physical money. Even our bank statement is emailed to us.

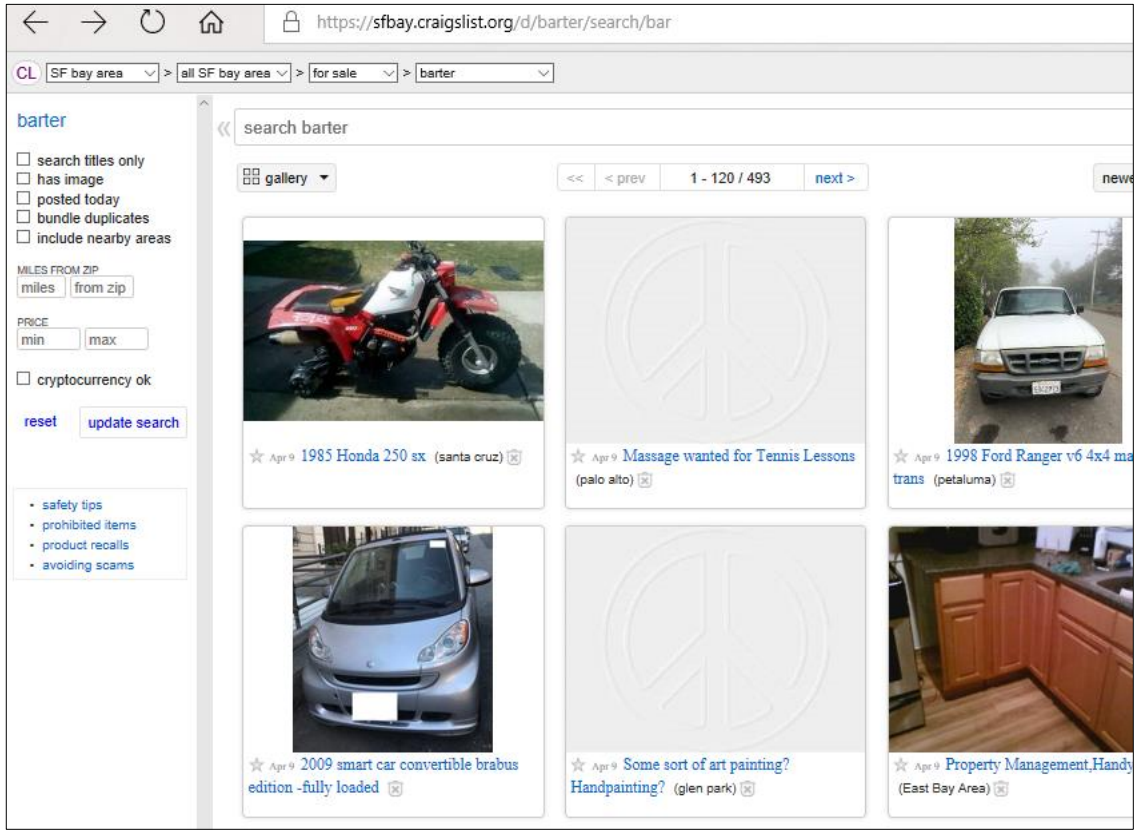
How is that different from cryptocurrency?

PAYPAL

Do you have a Paypal account? For many, this is there first experience with something similar to a cryptocurrency.

- All transactions are digital
- People can send you money
- You can send other money on the Paypal network
- *There is nothing physical*

NOT ALL ABOUT COINS



Blockchain has a much broader mission than exchanging monetary value. The distributed ledger of a blockchain can track *anything* of value. The only prerequisite is an ability to measure and then convey that value.

For example, craigslist.org has a barter section where people can exchange items of value without the transfer of cash. This is similar to a blockchain except there is a central, even though benevolent, authority.

LOGISTICS

Class Hours:

- Start time is 9am
- End time is 4:30pm
- Class times may vary slightly for specific classes
- Breaks mid-morning and afternoon (10 minutes)

Lunch:

- Lunch is 11:45am to 1pm
- Yes, 1 hour and 15 minutes
- Extra time for email, phone calls, or simply a walk.

Telecommunication:

- Turn off or set electronic devices to vibrate
- Reading or attending to devices can be distracting to other students
- Try to delay until breaks or after class

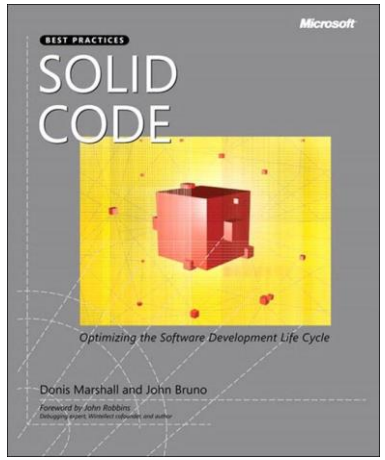
Miscellaneous

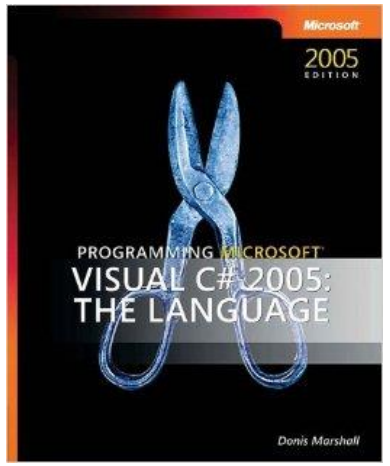
- Courseware
- Bathroom
- Fire drills

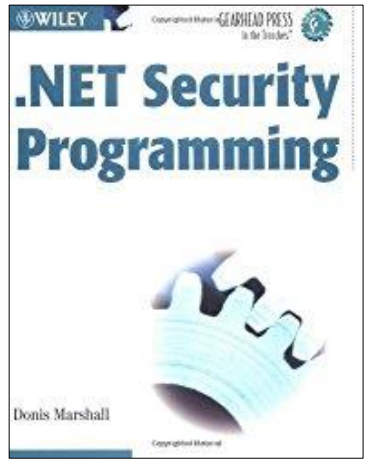
DONIS MARSHALL

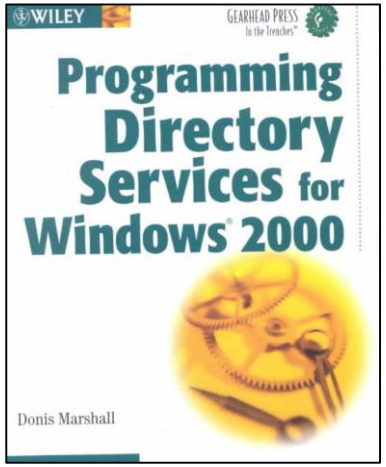
Blockchain Developer
Microsoft MVP
Microsoft Certified
Python, Java, Go,
and C++ developer
Author

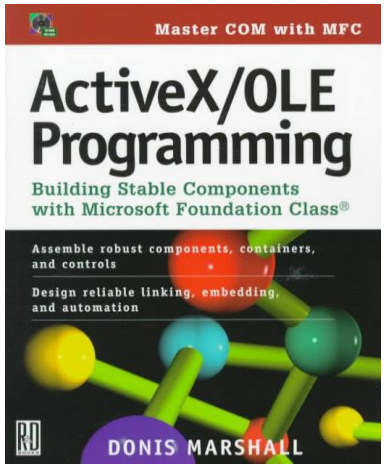
donis@innovationinsoftware.com

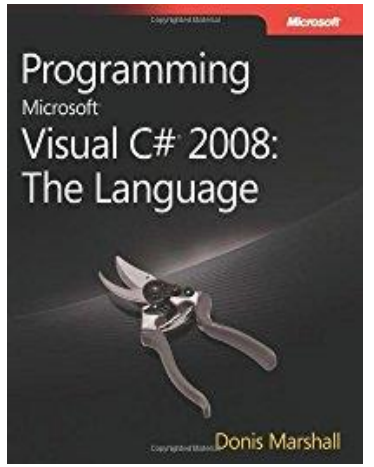












HISTORY LESSON

A secure, safe, distributed, and decentralized ledger was first described in the Journal of Cryptography 1991 written by Stuart Haber and W. Scott Stornetta. Here is the link.

https://link.springer.com/article/10.1007/BF00196791

This article describes a chain of secured blocks that form a ledger where value is transferred using transactions. The most known implementation of blockchain is Bitcoin.

The first person(s) to describe the concept of an actual blockchain was Satoshi Nakamoto in 2008. More about Satoshi Nakamoto later.

Blockchain 2.0 allows for the transfer of more than money with static transactions. This new methodology also supported programmable transactions which is the essence of smart contracts. The most recognize implementation of Blockchain 2.0 is Ethereum.

INTRODUCTION TO BLOCKCHAIN

Blockchain provides a technical framework for decentralized solutions.

Historically, centralized solutions have been more common than decentralized. The best example is banking where the bank manages the financial ledger of customers. The bank is the trusted authority and the clearing house of all transactions. But also a single-point of attack for assailants. For services rendered, the trusted authority charges fees for their services.

A non centralized solution removes the trusted authority in lieu of a peer to peer relationship. The result is a trustless network. The advantages include improved security, lower costs, scalability, and additional processing power. Yes, improved security in a trustless environment!

BLOCKCHAIN IMPLEMENTATIONS

Blockchain is in its relative infancy as a technology. Despite this, various technical solutions have already emerged. This is a short list.

- Ethereum
- Whisper
- Swarm
- Filecoin
- Hyperledger Fabric
- Bitcoin
- Litecoin

The complete list is soooooo much longer. There has been an explosion of interest in Blockchain recently.

ETHEREUM

Ethereum is a platform for running decentralized applications (Dapps) across nodes via smart contracts. Smart contracts are a misnomer; it is programmable code that is run as the target of a transaction. Transactions, powering Dapps, can be sent between users and contracts. They can also be sent between contracts.

Ethereum is a general purpose blockchain with an established ecosystem.

WHISPER

Whisper is a distributed messaging protocol that allows Decentralized Applications (Dapps) on the same blockchain to message each other. Consider this to be RPC for Dapps. The messages are temporary. Messages are topic based where nodes register topics of interest while filtering the remainder. Unicast, multicast, and broadcast of messages are allowed.

Whisper are used with dapps that collaborate on transactions.

SWARM

Swarm is a peer to peer (P2P) platform for distributed storage and replication where nodes can contribute their resources to the blockchain. This approach creates almost unlimited scaling. It is also used as distributed storage for Dapps.

This is an incentive based system where usage patterns affect availability.

Swarm is part of the Ethereum stack.

FILECOIN

Filecoin is a platform for decentralized storage versus distributed storage, such as Google Drive. It is an excellent tool to leverage unused storage accumulating on machines around the world.

Filecoin uses the InterPlanetary File System (IPFS), which is peer-to-peer hypermedia protocol, to realiably and securely transfer files. Each file is identified with a unique hash. The unique hash is used for indexing.

This is a compensation system where renumeration is done in Filecoin tokens (FIL).

HYPERLEDGER FABRIC

Hyperledger Fabric is a permissioned blockchain supporting global cross-industry transactions via smart contracts. Members of the Hyperledger Fabric network register through a Membership Service Provider (MSP).

Hyperledger Fabric is more extensible than other blockchain implementations. Instead of adopting a single universal standard there is support for pluggable implementations of different components.

Disparate ledgers can be created using channels. Groups of participants can create channels as partitions from other groups. Each channel can have separate rules that apply to particular participants.

BITCOIN

Bitcoin is a cryptocurrency and probably the most popular implementation of blockchain and has become a worldwide phenomenon. Ultimately, Bitcoin is a decentralized payment system not requiring centralized control from a traditional financial institution. Bitcoin is a public blockchain orchestrated by a series of nodes.

Bitcoin protocol was defined by Satoshi Nakamoto, who is an anonymous person or persons. You can download the protocol here:

https://bitcoin.org/bitcoin.pdf

BITCOIN 2

Bitcoin is a:

- Bitcoin protocol
- Decentralized peer to peer network
- Public transactional ledger
- Consensus network
- Set of rules and guidelines

BITCOIN - 3

Bitcoin is:

- Collaborative
- Secure
- Borderless

Bitcoins are created via a process called mining. Miners must solve a complex mathematical problem to process a Bitcoin transaction.

CODE EXAMPLES IN THIS COURSE

Example with prefix	Type
aulColors	array of unsigned long
bBusy	boolean
chInitial	char
dwLightYears	double word
nSize	integer
dbPi	double
pFoo	pointer
pszOwner	pointer to zero-terminated string
szLastName	zero-terminated string

Either actual or pseudo code is presented for the source in this course.

If the implementation is straightforward, then the exact code is presented. When the solution is discretionary, pseudo code is presented.

Hungarian notation is used for the pseudo code, where the prefix indicates the type. When ambiguous or not clear, comments are provided.

Lab 1- Cryptocurrency

CRYPTOCURRENCY

Blockchain is not a passing fad. New blockchain projects are being funded virtually everyday. There is an enormous amount of interest and investment in this sector of technology.

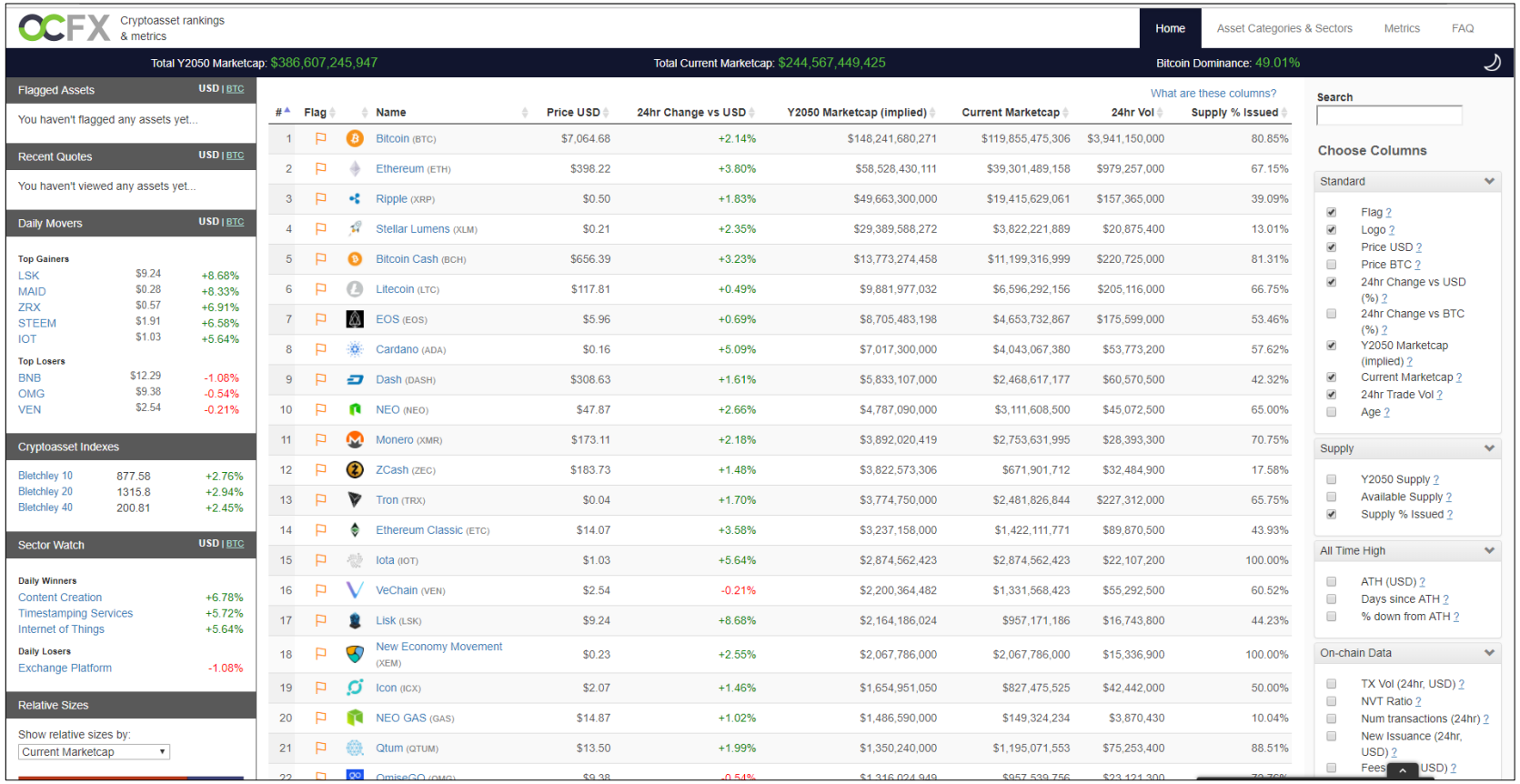
Probably the best example is the ecosystem that has sprouted up around cryptocurrency. Websites, APIs, SDK, analysts, and much more now exist to serve this segment and increasing exponentially.

Explore the websites presented on the next few pages. Even if you are not interested in cryptocurrencies, it will give you some understanding of the potential of blockchain solutions.

This is just a sampling of websites. There many more.

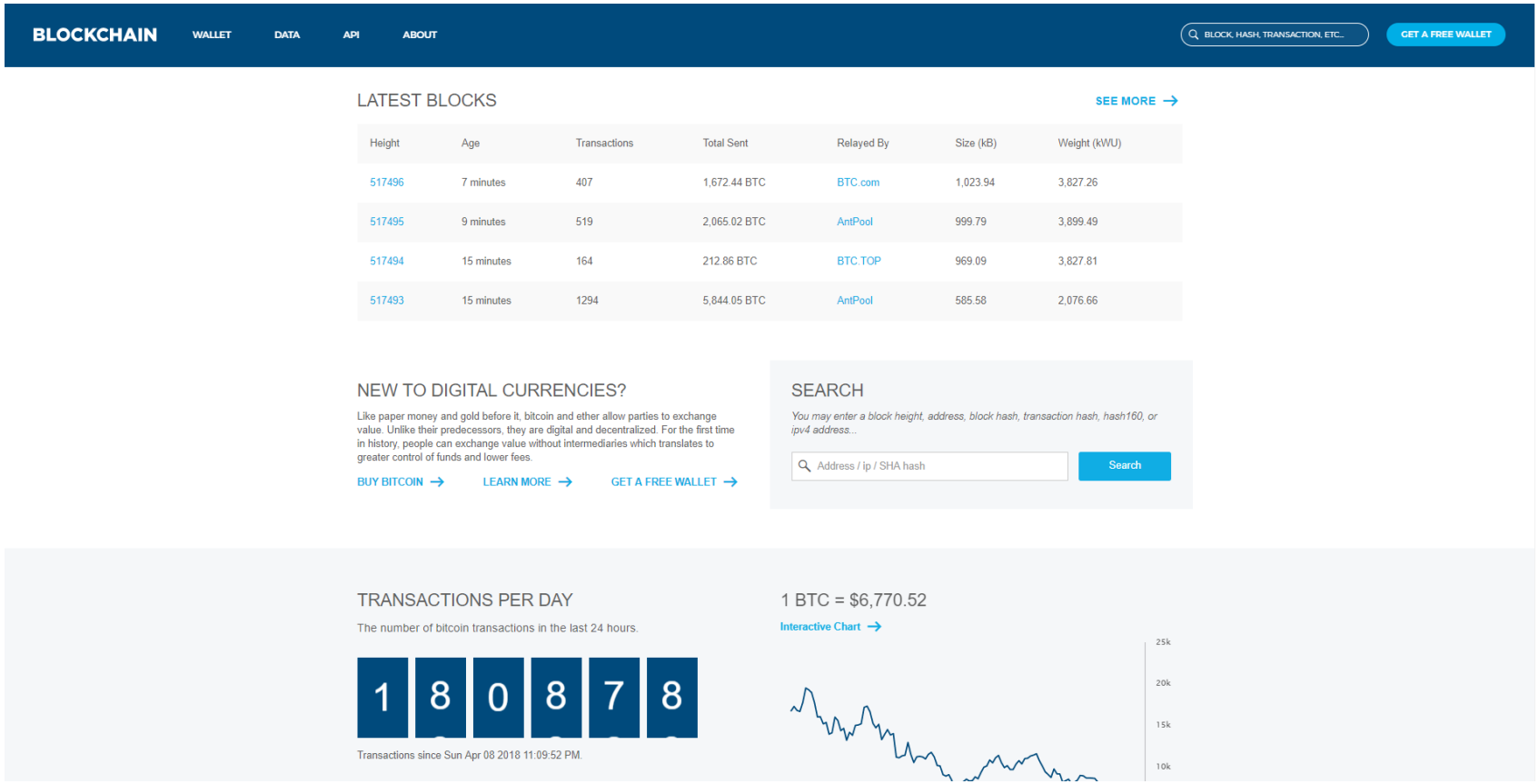
ONCHAINFX.COM

The most detail market analysis of cryptocurrencies. Click the links for each currency for more specific information.



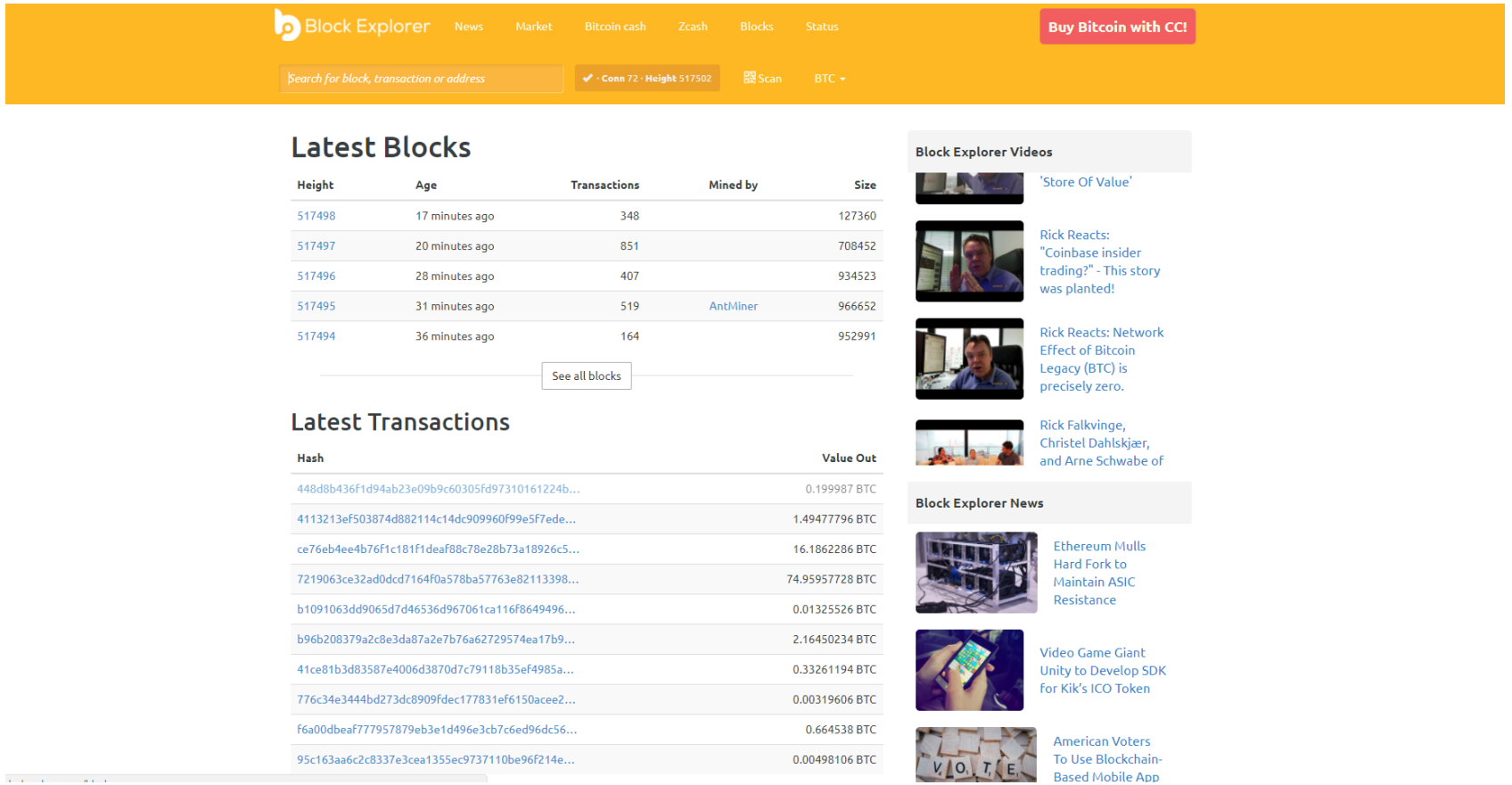
BLOCKCHAIN.INFO

BLOCKCHAIN.INFO is one of the most popular blockchain websites. Explore the options under the DATA menu. We will be using the search option in class.



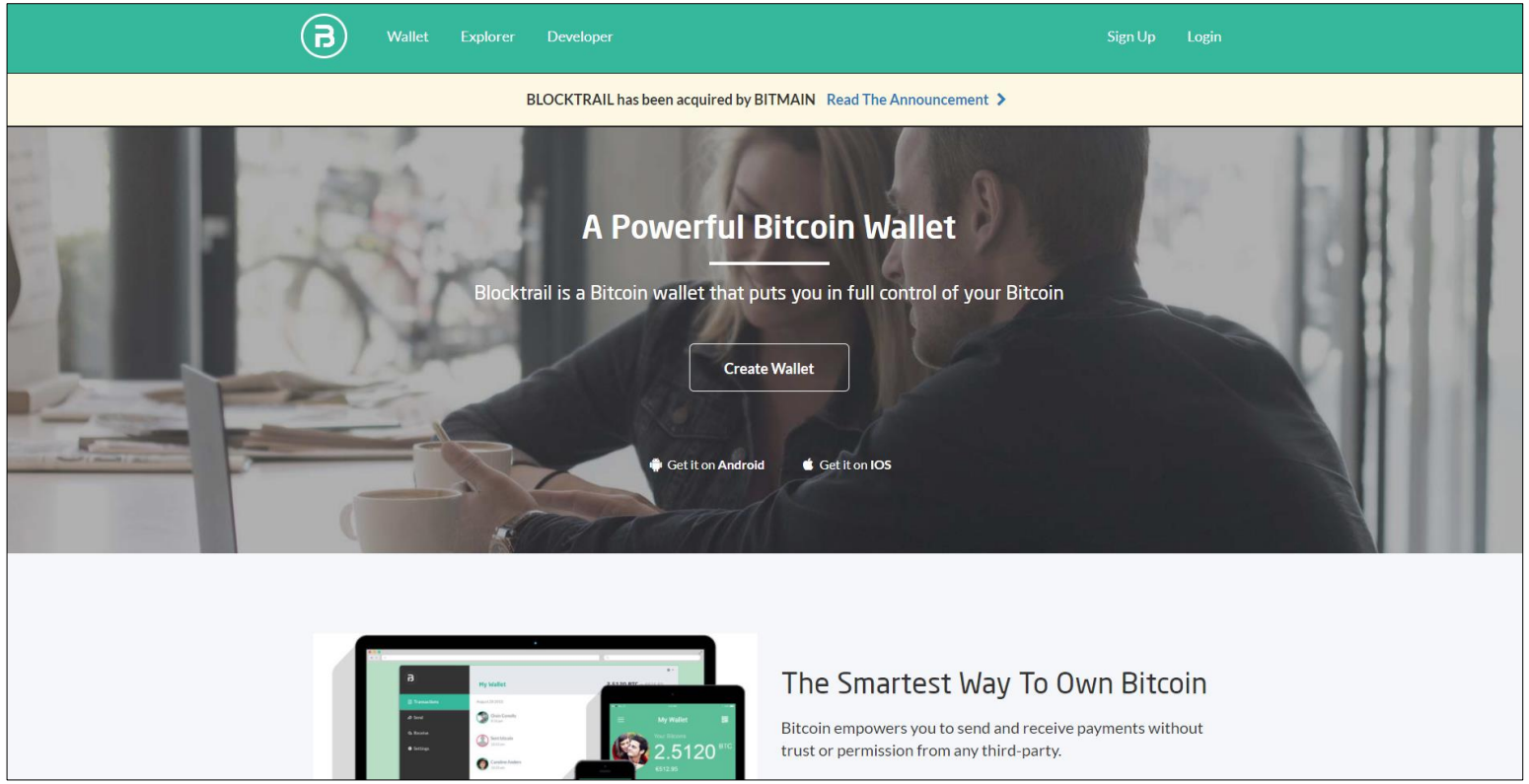
BLOCKEXPLORER.COM

BlockExplorer.com is another popular blockchain website. It provides market analysis of cryptocurrencies, technical detail of the Bitcoin blockchain, videos, and more.



BLOCKTRAIL.COM

BLOCKTRAIL.COM provides market analysis on cryptocurrencies. It has a straightforward GUI. Open the developer link to view programming tools, such as an API and SDK, that help simplify blockchain development.



BIT$CREENER.COM

Visit Bit$screener.com. It has a great mobile application for phones to monitor the market value of various cryptocurrencies.

Blockchain concepts

Develop a passion for learning.

BLOCKCHAIN – DEFINED AGAIN

Blockchain is a decentralized application that manages a public distributed ledger of transactions. Each transaction is a transfer of value – not necessarily monetary value. The value could for example be a lien for a vehicle. The blockchain is a secured, immutable, and order list of blocks (i.e., a chain of blocks). Each block contains one or more transactions, a sender and receiver, and an identifier.

There is a peer to peer network without a trusted central authority, the ultimate democracy, where nodes (computers) collaborate to manage the blockchain were decisions are made by consensus. There is not a central authority for authenticating and authorizing transactions.

Each participating node maintains a copy of the blockchain ledger. Since the ledger is replicated across several nodes, there is not a single point of attack, which is one of the many security features of the blockchain.

CENTRAL AUTHORITIES

Many businesses, particularly the financial sector, represent centralization of control in a trusted authority. This is most prevalent in industries or business where customers cannot trust one another, such as banking. The central authority provides a framework for trust, such as regulatory guidelines, the processing of transactions, coordinating activities, and authentication.

However, most trusted authorities require compensation for their services. The customer has the financial burden of supporting the central authority, which is sometimes arbitrary.

Central authority affects affordability, scalability, security, privacy, and policy.

Blockchain is a counterpoint to centralized authorities providing a frictionless solution for the digital economy.

AFFORDABILITY

Central authorities typically expect payment for services. Transaction fees are a common means of reimbursement for services render. For example, the transaction fee for a title search when purchasing a home. If the title search could be accomplished without a centralized authority and automated, the that fee could be greatly reduced or even eliminated. The collective cost savings would be considerable. This could be done with a blockchain where titles are kept on a public ledger.

Turing Pharmaceutical acquired the drug Daraprim and immediately raised the price 5,000%. Daraprim can be essential for those having a compromised immune system. The stated reason was to fund additional research. Martin Shkreli the President of Turing became rich but achieved an unparalleled level of infamy. Mr. Shkreli became known as the "most hated man in America".

https://stanford.io/2Ixr6Jp

PRIVACY

Central authorities are tasked with maintaining your privacy, which is different than security. Security is a break-in where a third party steals your personal and sometimes sensitive information. But users also rely on central authorities not to give away or sell personal information. This is a sacred trust placed with central authorities. However, this trust is often broken. With Blockchain, privacy is always maintained through strong cryptographic techniques and inherent to the model.

Facebook shared private information for 87 million loyal customers to Cambridge Analytica, which used the information for political purposes. This was done without the consent of users.

https://nyti.ms/2HP4Dr3

POLICY

A central authority can administer policies that affect users. This is a problem especially when policies are counter to the well-being of the user. Furthermore, some policies are subjective leaving room for interpretation and abuse. Most importantly this can lead to censorship of content or free expression. In Blockchain, policies are transparent and not subjective.

Governments often impact the policy of online companies. This means policies can depend on where a website is browsed. For example, the shadow of the Chinese government looms large on technology companies operating online in China. For example, content perceived critical of the Chinese government is often censored. Even the largest internet companies are not immune to this sort of pressure.

https://nyti.ms/2uRsO6v

DECENTRALIZED NETWORK

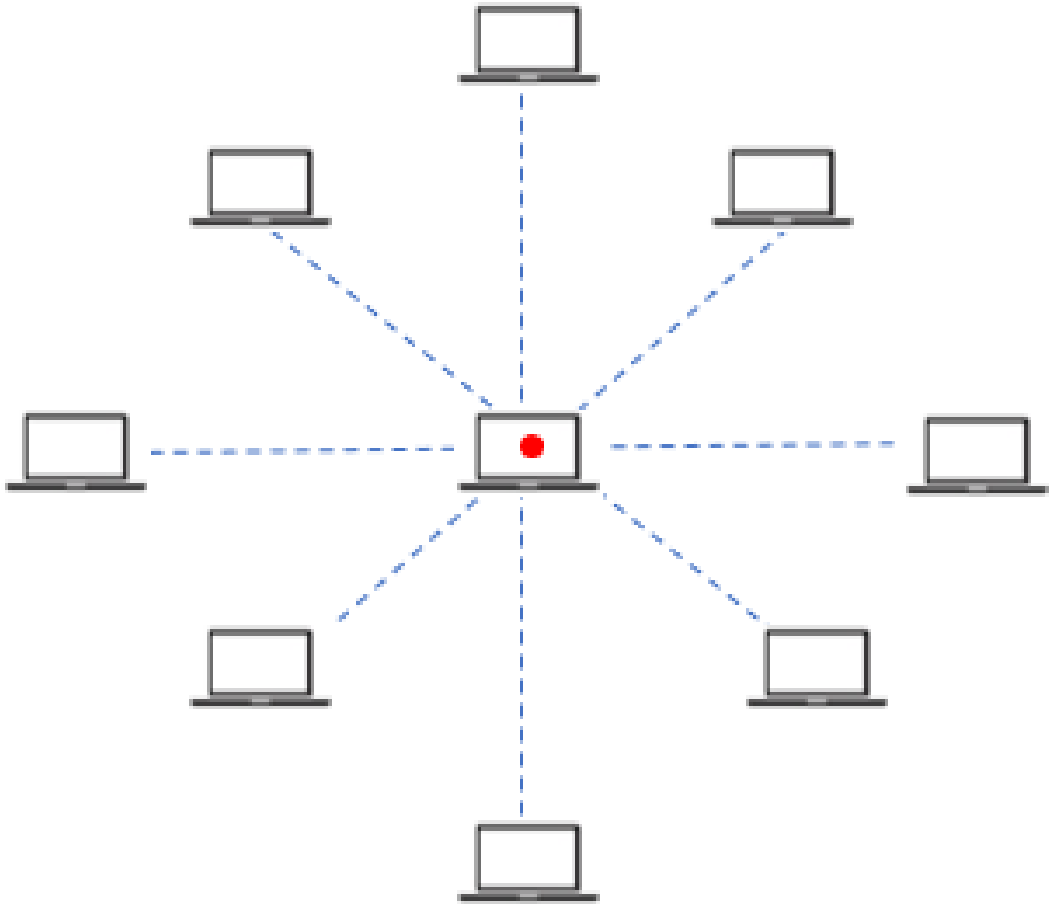
Blockchain is a decentralized network, which is one of the advantages. The three dominant models for network systems: is centralized, decentralized, and distributed.

Centralized: this is the satellite or client / server model. There is one primary computer (server) where work resides, while servicing requests from other computers (client/satellites). The satellites contribute limited or no processing power to the work.

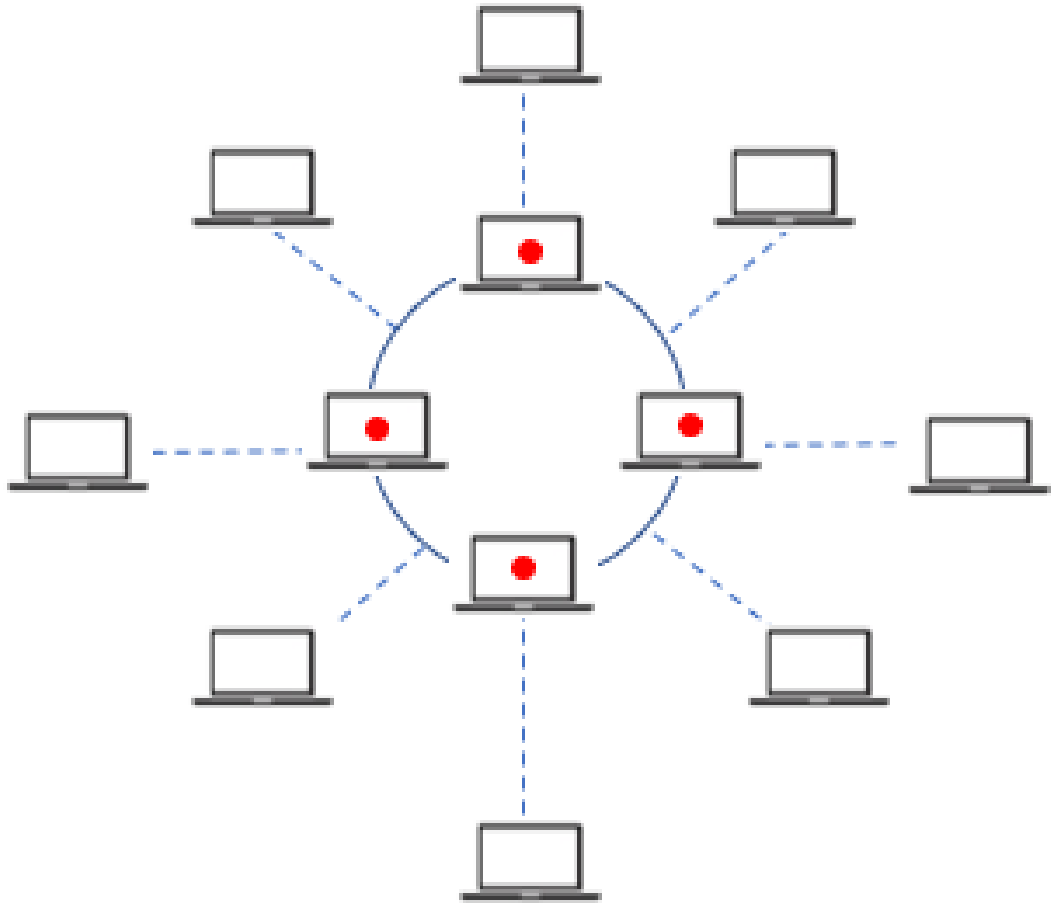
Distributed: this is the collaborative model where various aspects of the work are distributed across several computers. The work is decomposed into tasks which are then assigned to various servers.

Decentralized: there are several computers in a decentralized architecture – each working independently. The application may be running on multiple servers. The computers may communicate and make collective decisions about the state of the network.

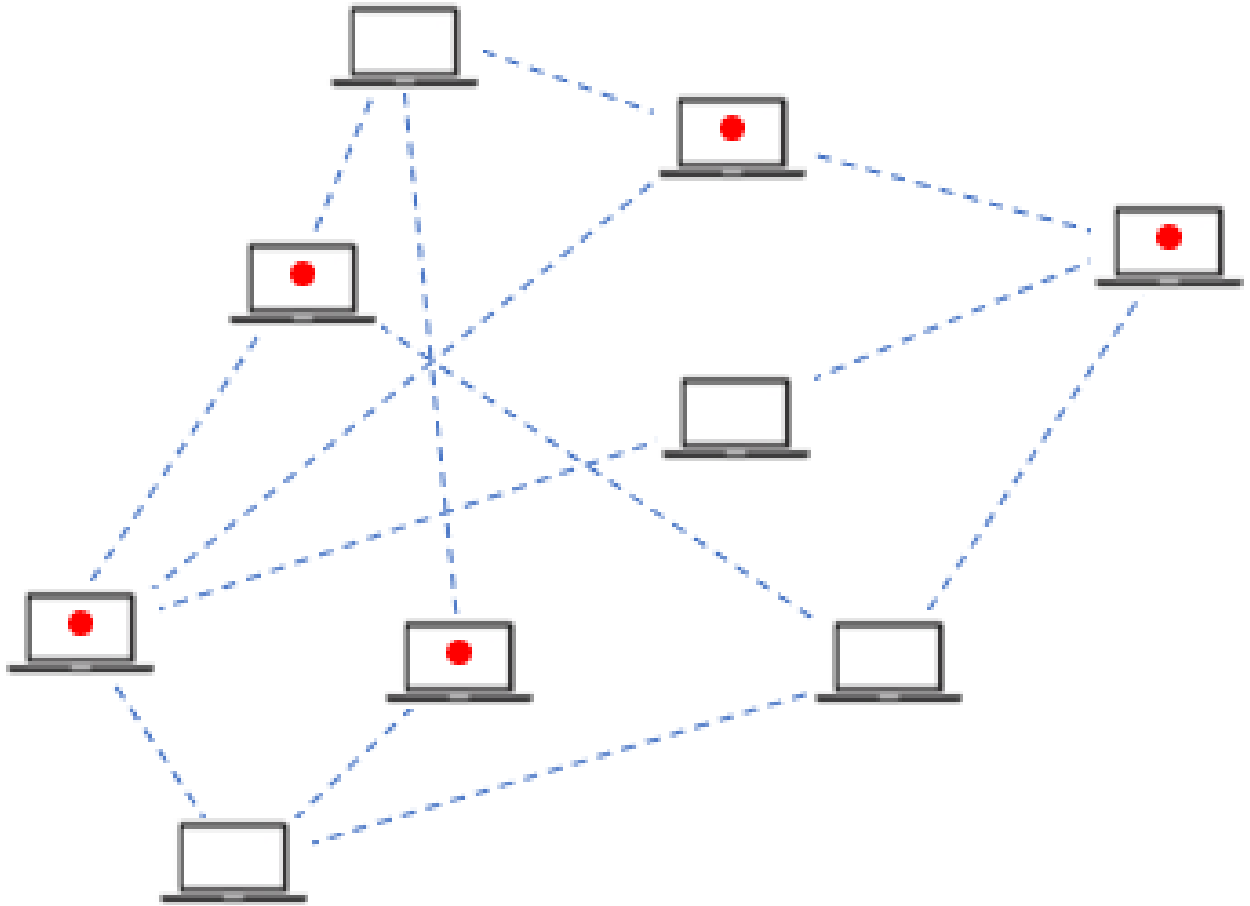
CENTRALIZED NETWORK - DIAGRAM



DISTRIBUTED NETWORK - DIAGRAM



DECENTRALIZED NETWORK - DIAGRAM



PROS AND CONS

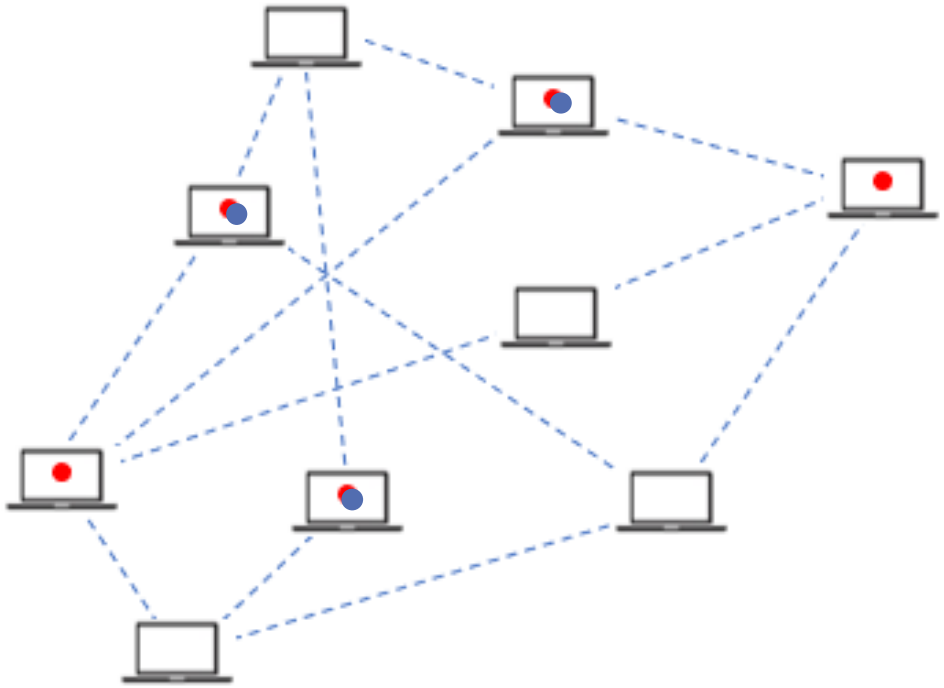
- Points of failure. Centralized systems are easier to manage but have a single point of failure.
- Fault / Tolerance. Centralized systems are highly unstable. Kill the central server to destabilize the entire network. Decentralized are resilient because of the redundancy of effort.
- Scalability. Centralized has limited scalability. Distributed is more scalable. Decentralized is infinitely scalability.
- Development. Centralized is simple and technically easier to develop. Decentralized requires more considerations and more technically complex.
- Security. Centralized is less secure with a single point of attack. Decentralized, especially blockchain, become more secure as the application scales.

PEER TO PEER

In a Peer-to-Peer (P2P) network, there are no master / slave relationship between computers. Responsibility does not reside with a central server, such as a client / server network. In a P2P network, all nodes (computer) share computing responsibility and work together collaboratively. This is the opposite of the client / server model where the main processing power remain with designated servers, which are handling requests for clients.

The blockchain is a P2P network.

DECENTRALIZED APPLICATION (DAPP)



Decentralized applications (Dapps) are when multiple instances of an application run independently on several computers on a P2P blockchain network.

More about Dapps in the module on Ethereum and smart contracts.

- Dapp instance
- Server node

ATTRIBUTES OF A BLOCKCHAIN

Blockchain is a protocol for a decentralized application hosting a distributed ledger. Blockchains that follow that protocol share similar attributes.

- No central authority
- Immutable
- Secure
- Transparent
- Privacy
- Scalable
- Available

NO CENTRAL AUTHORITY

Blockchain is a decentralize network of nodes that have various nodes, such as a miner, a full node, or simple payment verification (SPV). They work together and eliminate the need of a central authority. This includes sharing the responsibility for making the network both secure, immutable, and forming a consensus.

IMMUTABLE

The Blockchain manages a immutable chain of blocks and transactions. Examining the blockchain will expose all of the transactions since the blockchain was created. The blockchain is extendable. You can add but not change or delete blocks. A malicious node that changes the blockchain would be discovered by the other nodes with a copy of the same ledger. They will form a consensus to reject the change.

CONSENSUS

Speaking of consensus…

In the Blockchain architecture, there is no central authority trusted to make decisions on behalf of the network. Blockchain is a trustless network without a central authority. Instead decisions are made by consensus by the nodes. The majority of nodes must agree upon accepting a new block into the blockchain, which contains transactions.

SECURE

The Blockchain protocol assures the chain is securable.

- Each block is hashed – including the hash of the previous block.
- Blockchains are immutable which is verified by consensus.
- Transactions are signed
- Multiple copies of the valid block chain are kept. Attacks require an enormous amount of computational capability.

TRANSPARENT

Blockchains are public and the transactions are available for inspection. For example, anyone can inspect every bitcoin transaction on a blockchain since inception. Transparency contributes to a secure environment.

PRIVACY

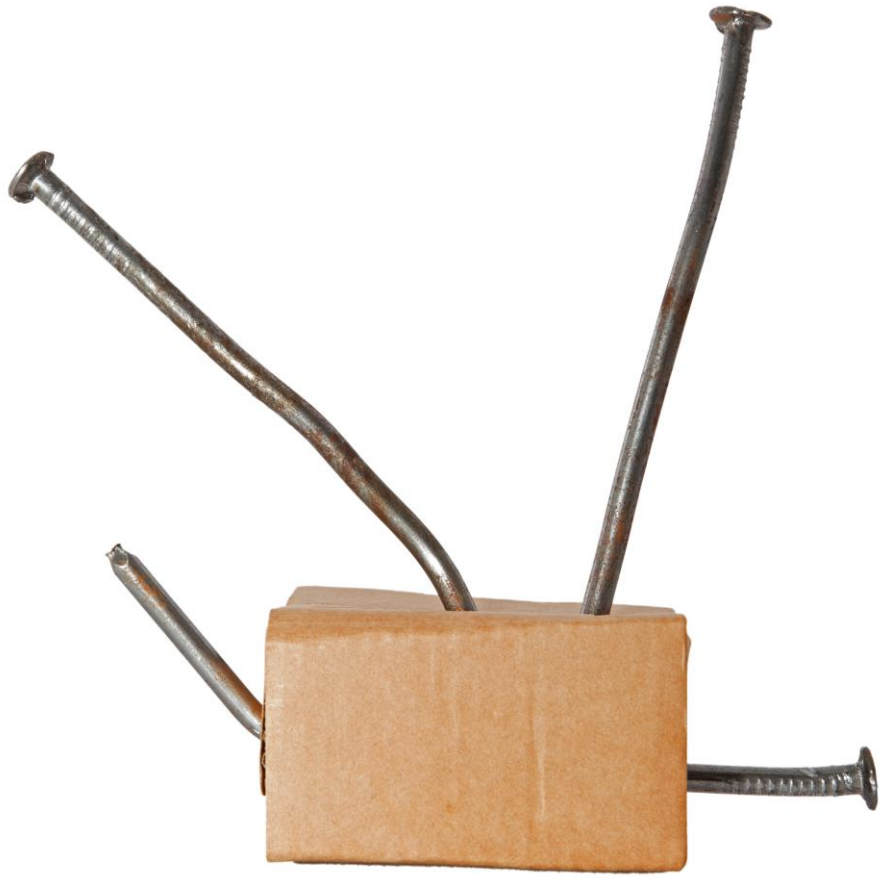
The Blockchain protocol supports transparency without compromising privacy. Bitcoin is an example. Every transaction is visible to inspection. You know where money is sent and the amount of the transaction. However, the sender and receiver are abstracted. In addition, privates keys are held in a secure location and not publicly available.

SCALABLE

Blockchains are scalable; just add additional nodes. However, running nodes cost money: computers, electricity, networking, and so on. Nodes receive incentives for their effort, which are periodically adjusted. Incentives can be increased to attract additional miners and full nodes as needed. This means that theoretically blockchains can scale indefinitely. It is simply a measure of compensation.

Of course for Bitcoin, subsidies are paid in coins.

RESILIENT



A blockchain may bend but unlikely to break. As a decentralized network with several nodes hosting the work, the blockchain eliminate dependencies. This is especially true as the various roles are replicated across the internet. Some nodes may be susceptible, from security attacks, internet, or machine failures, but it is unlikely that all of the nodes across a blockchain will fail simultaneously. For this reason, blockchains are incredibly resilient.

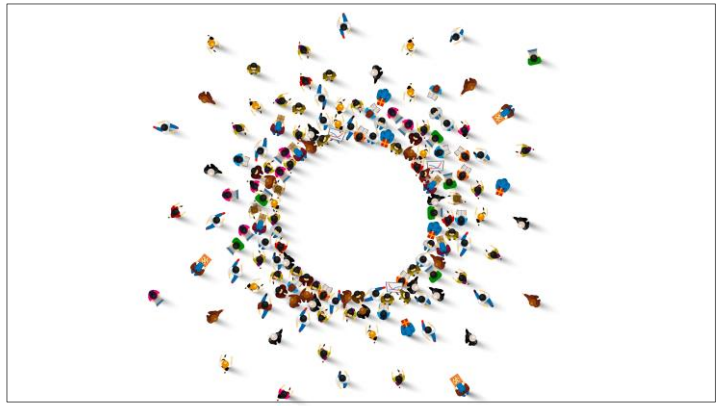
AVAILABLE

The blockchain is a web-based application. Like most web applications, a blockchain is open for business 24/7 – always available.

PUBLIC VERSUS PRIVATE

Public Blockchain

Private Blockchain



Blockchains can be either or public or private. Both are decentralized P2P networks that host a distributed ledger that is replicated across nodes. Public blockchains, such as Bitcoin, are accessible to anyone and highly scalable. Private blockchains are controlled by an entity, such as a company, that limits access to the network. That is the biggest difference between a private and public blockchain – a central authority.

For a private blockchain, the central authority grants access to the blockchain and assigns permissions. For that reason, private blockchains are also referred to as permissioned blockchains.

PUBLIC VERSUS PRIVATE - 2

Private blockchains may have different incentives. For example, miners may not be independent and therefore not require compensation.

Private blockchains can be more secure, such as residing behind a firewall. Everyone, such as the users, miners, and full nodes, could be in the same domain. This means some activities, such as Proof of Work (PoW), could require less work or not needed at all.

Hyperledger Fabric is an example of a permissioned blockchains.

PUBLIC VERSUS PRIVATE - ADVANTAGES

Private blockchain:

- Protected behind a firewall or other security
- More efficient because consensus *may not* be necessary
- More responsive since proof of work *may not* be required
- Not transparent
- Permissioned access

Public blockchain:

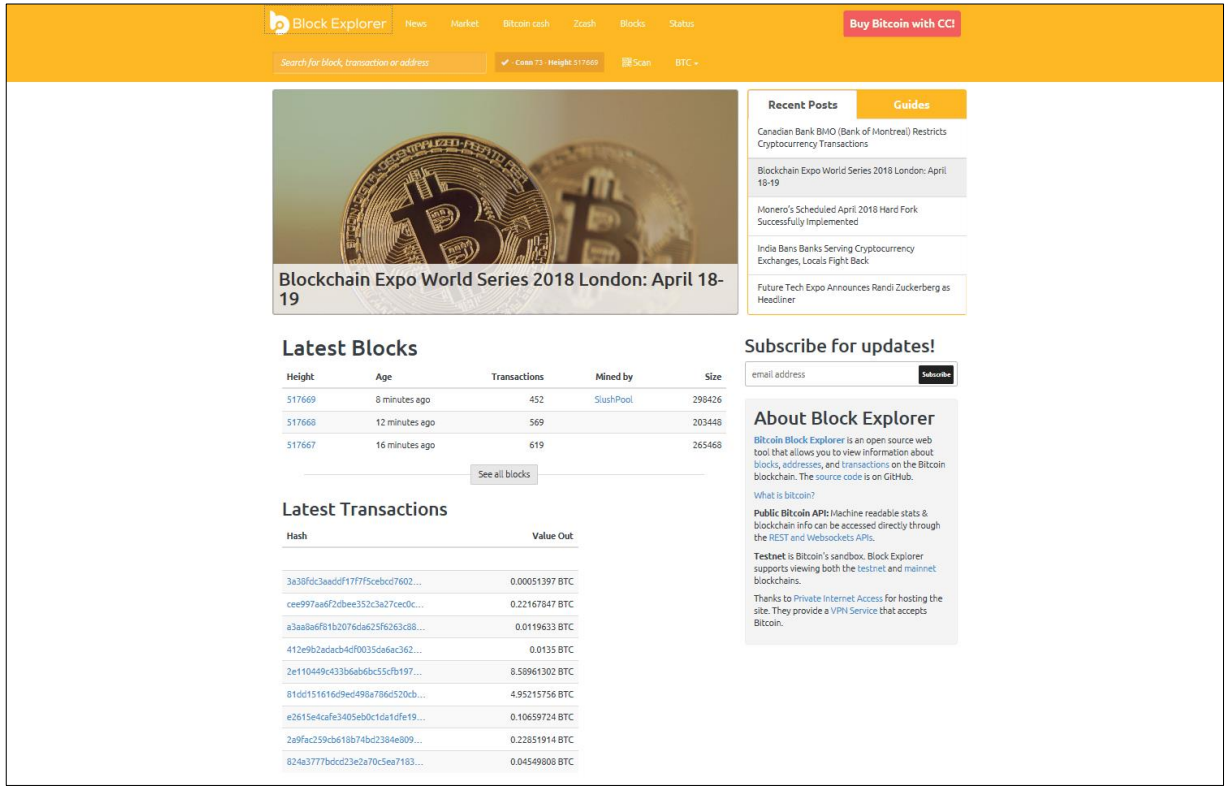
- No single point of failure
- Massively scalable
- More resilient
- Transparent
- Available to everyone

Lab 2- Explore a Blockchain

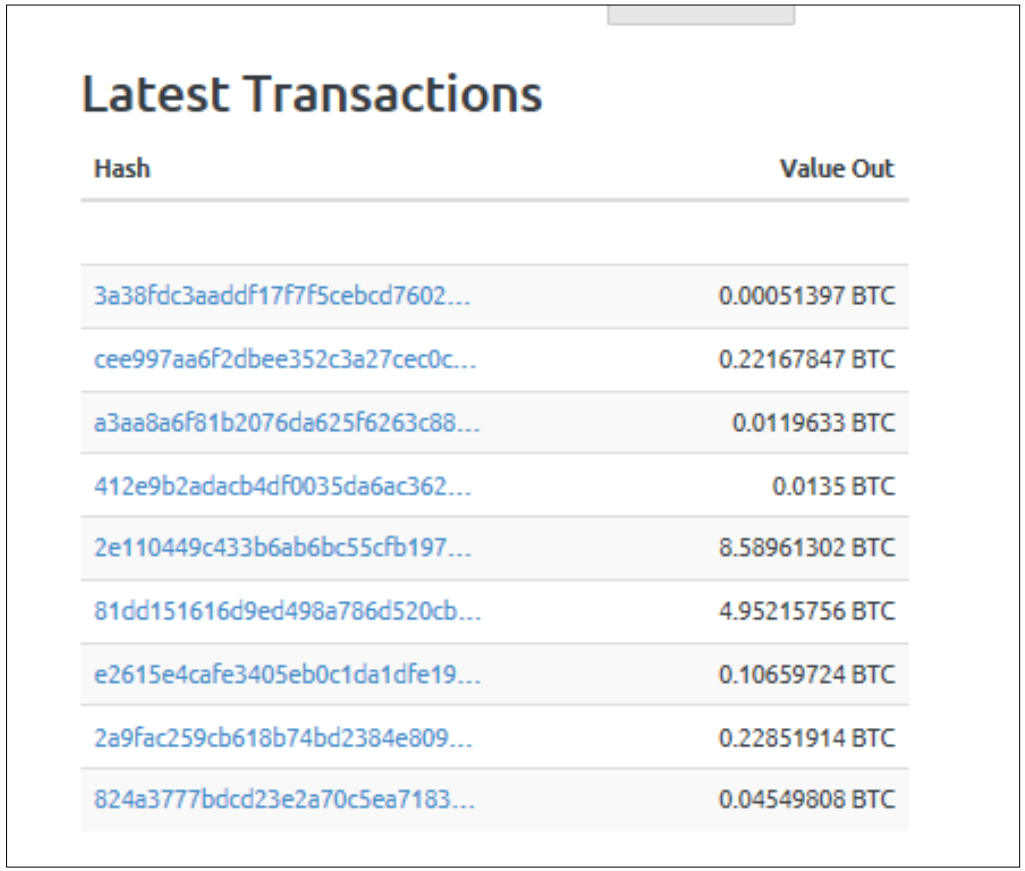
EXPLORE A BLOCKCHAIN - 1

In this lab, you will explore a blockchain using BlockExplorer.com. The goal of this lab is to learn how to navigate a blockchain and explore terminology. Links are provided to facilitate exploration.

Open BlockExplorer.com.



EXPLORE A BLOCKCHAIN - 2

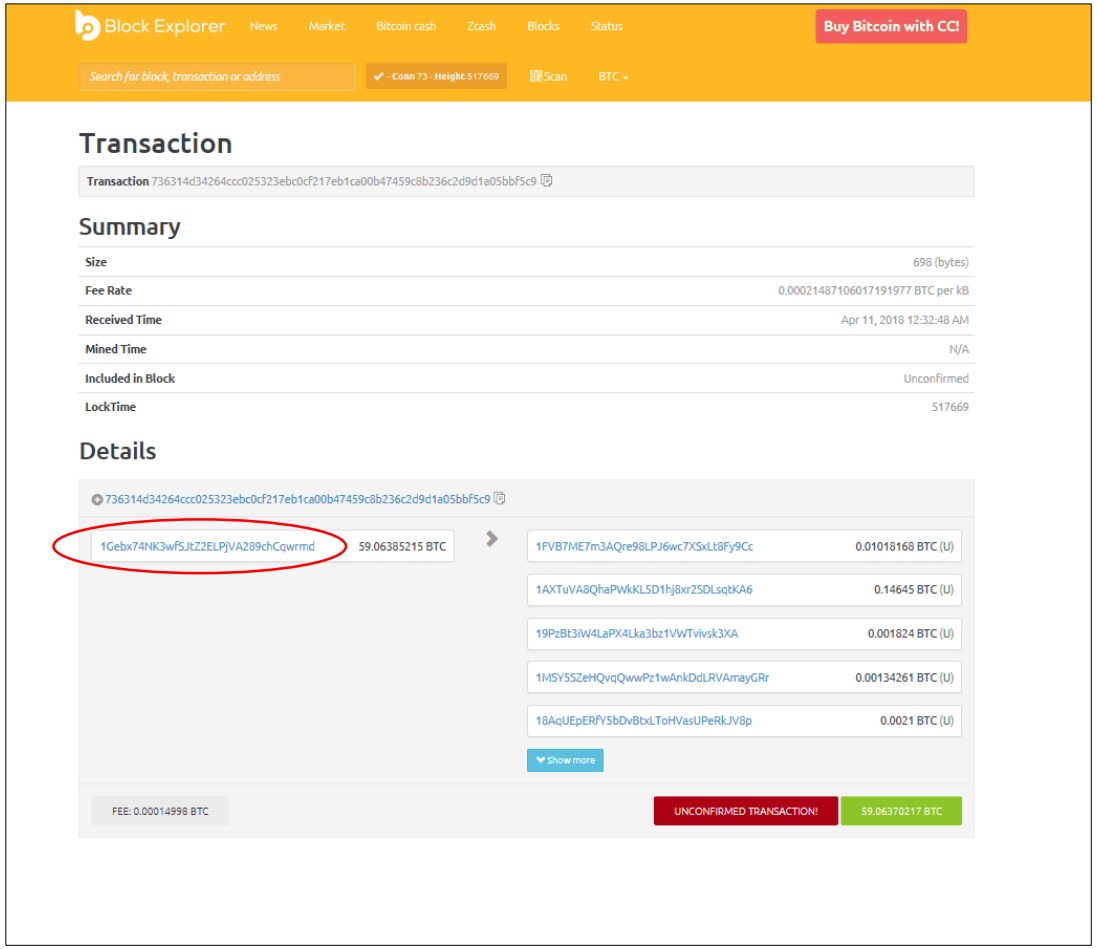


Find the list of “Latest Transactions”. You can watch additional transactions emerge.

Click on any transaction in the list.

You should see the Transaction window now.

EXPLORE A BLOCKCHAIN - 3

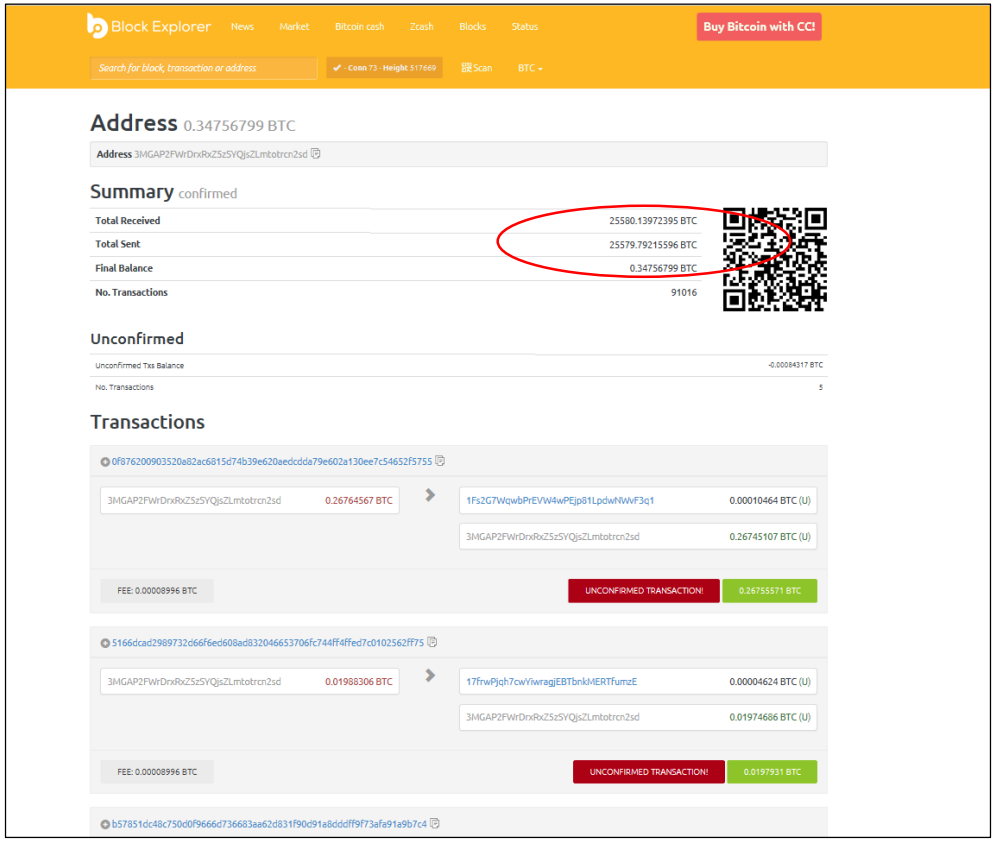


On the Transaction screen, look at the bottom for transaction details. Notice whether transaction is confirmed or not confirmed. If not confirmed, the transactions has not been mined into a block.

Keep selecting addresses, even switching transactions if necessary, until you find someone with money in there wallet (final balance).

You should be viewing the address page at the moment.

EXPLORE A BLOCKCHAIN - 4

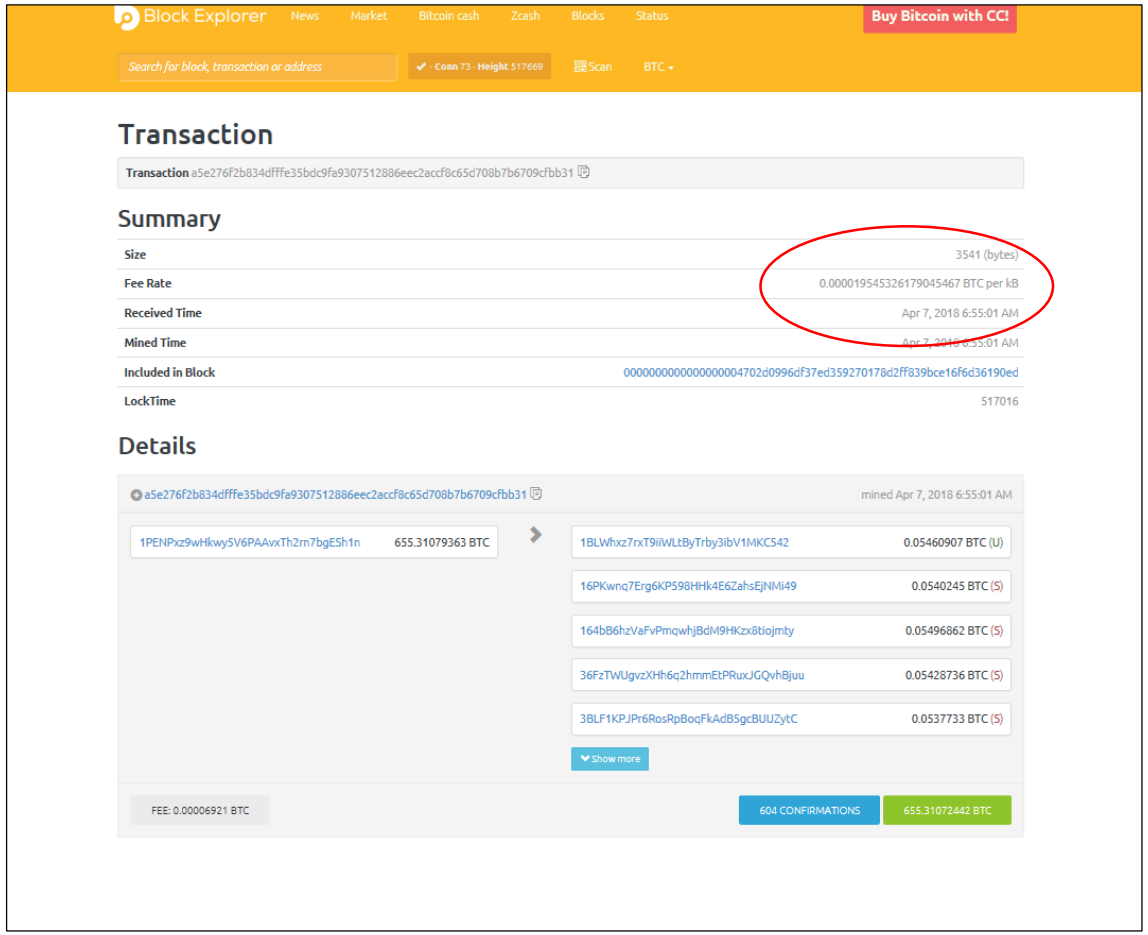


Do the math for this user! Does the amount received minus total sent equal the final balance?

Notice the transactions at the bottom. Some are confirmed; others are not.

Select the link for one of the confirmed transactions.

EXPLORE A BLOCKCHAIN - 5

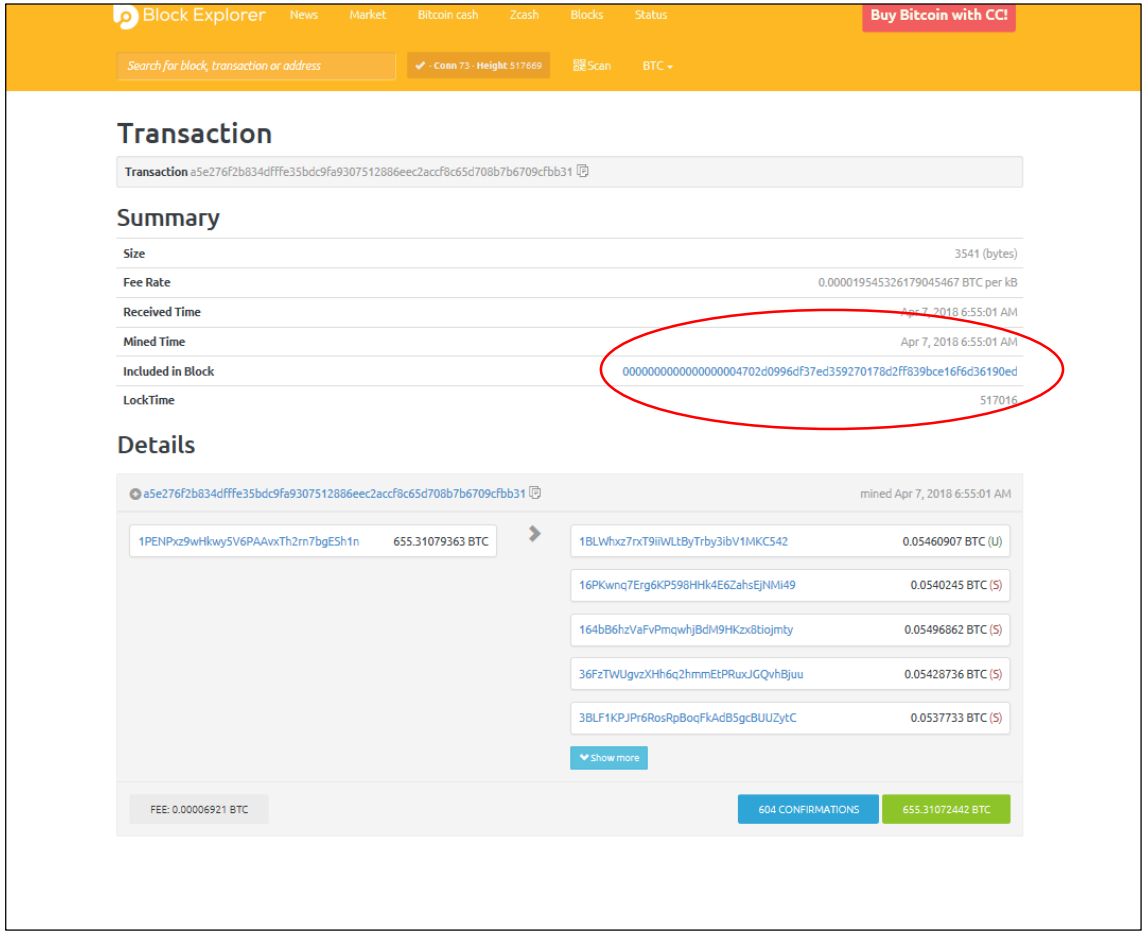


Do the math for this user! Does the amount received minus total sent equal the final balance?

Notice the transactions at the bottom. Some are confirmed; others are not.

Select the link for one of the confirmed transactions.

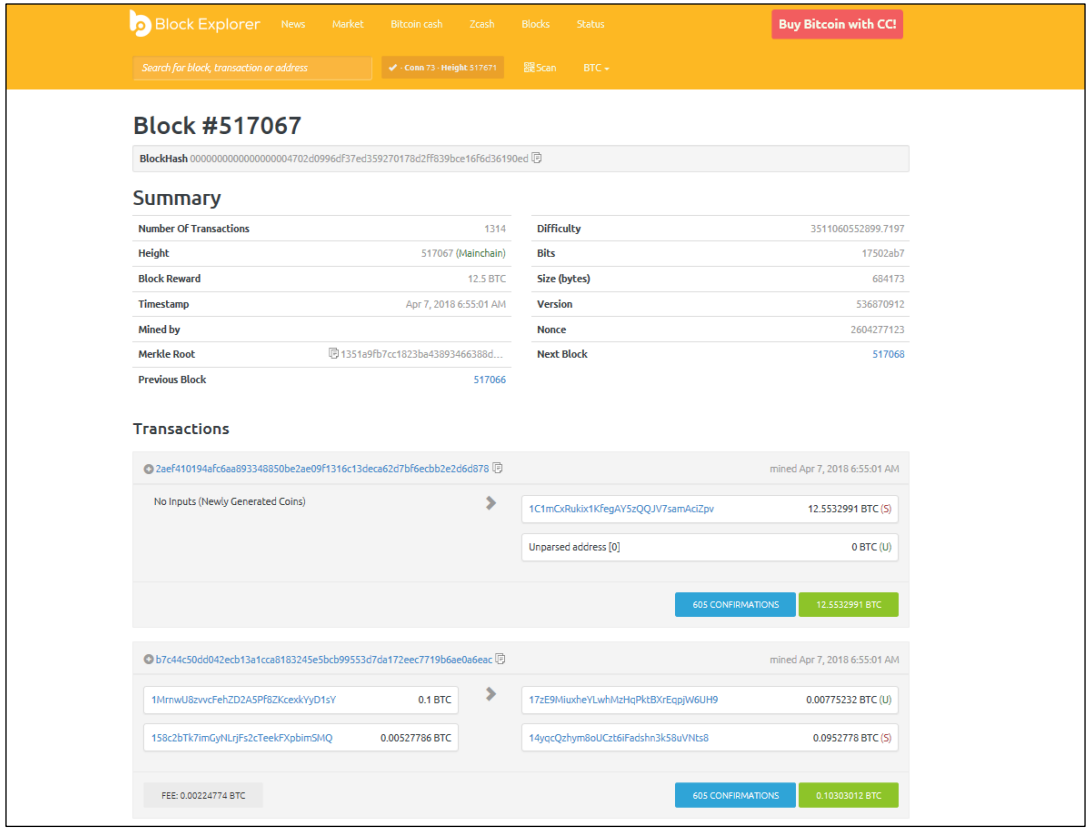
EXPLORE A BLOCKCHAIN - 5



Notice the details of the block, including when the block was mined. The date should be similar to the creation date of the transaction.

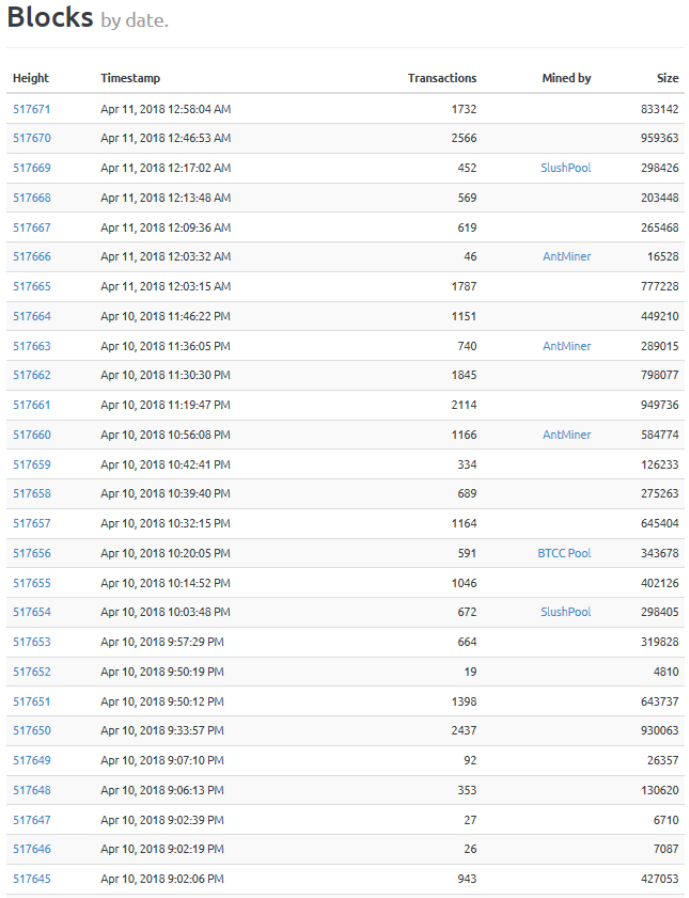
Also notice a link to the block containing this transaction. Let us few the block.

EXPLORE A BLOCKCHAIN - 6



View of the block.

EXPLORE A BLOCKCHAIN - 7



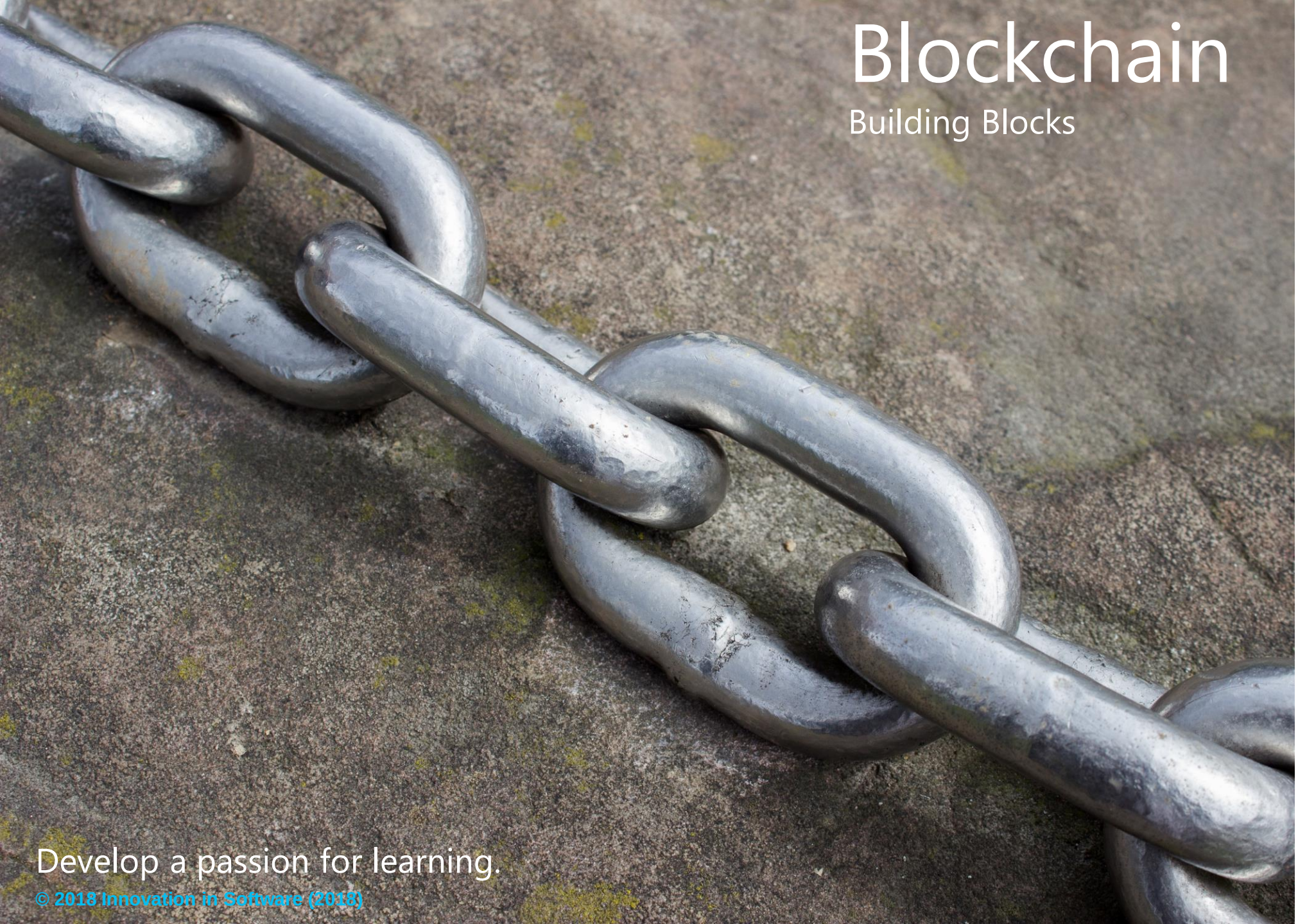
Blocks by date.

Height	Timestamp	Transactions	Mined by	Size
517671	Apr 11, 2018 12:58:04 AM	1732		833142
517670	Apr 11, 2018 12:46:53 AM	2566		959363
517669	Apr 11, 2018 12:17:02 AM	452	SlushPool	298426
517668	Apr 11, 2018 12:13:48 AM	569		203448
517667	Apr 11, 2018 12:09:36 AM	619		265468
517666	Apr 11, 2018 12:03:32 AM	46	AntMiner	16528
517665	Apr 11, 2018 12:03:15 AM	1787		777228
517664	Apr 10, 2018 11:46:22 PM	1151		449210
517663	Apr 10, 2018 11:36:05 PM	740	AntMiner	289015
517662	Apr 10, 2018 11:30:30 PM	1845		798077
517661	Apr 10, 2018 11:19:47 PM	2114		949736
517660	Apr 10, 2018 10:56:08 PM	1166	AntMiner	584774
517659	Apr 10, 2018 10:42:41 PM	334		126233
517658	Apr 10, 2018 10:39:40 PM	689		275263
517657	Apr 10, 2018 10:32:15 PM	1164		645404
517656	Apr 10, 2018 10:20:05 PM	591	BTCC Pool	343678
517655	Apr 10, 2018 10:14:52 PM	1046		402126
517654	Apr 10, 2018 10:03:48 PM	672	SlushPool	298405
517653	Apr 10, 2018 9:57:29 PM	664		319828
517652	Apr 10, 2018 9:50:19 PM	19		4810
517651	Apr 10, 2018 9:50:12 PM	1398		643737
517650	Apr 10, 2018 9:33:57 PM	2437		930063
517649	Apr 10, 2018 9:07:10 PM	92		26357
517648	Apr 10, 2018 9:06:13 PM	353		130620
517647	Apr 10, 2018 9:02:39 PM	27		6710
517646	Apr 10, 2018 9:02:19 PM	26		7087
517645	Apr 10, 2018 9:02:06 PM	943		427053

Choose the Blocks link for a listing of the blockchain.

Notice the column for Mined By. Some of the names of miners are included in this column.

Click on the link for one of the miners, such as SlushPool, for more information.

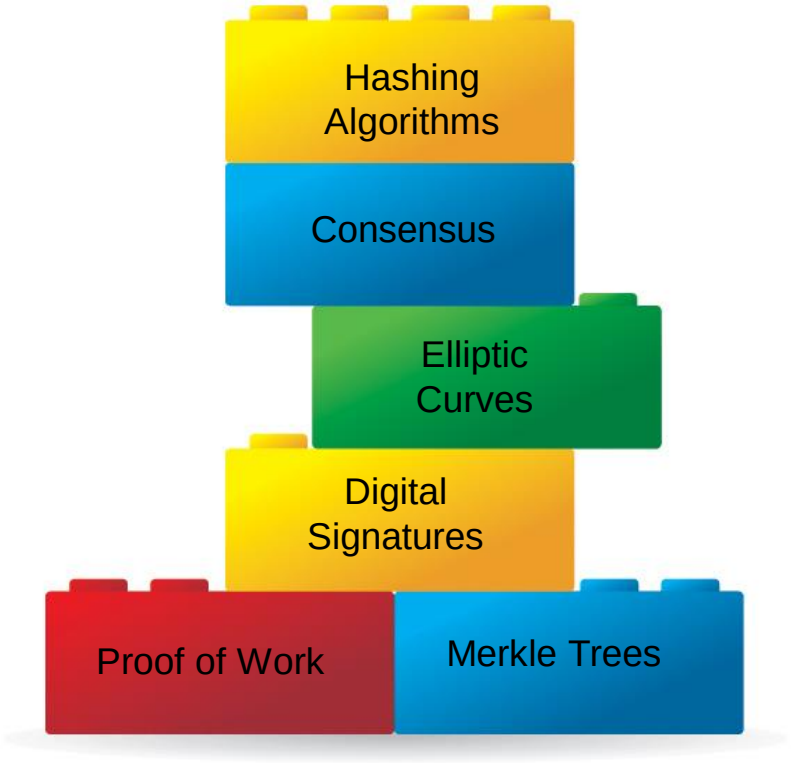


Blockchain

Building Blocks

Develop a passion for learning.

BUILDING BLOCKS



There are a variety of concepts, or building blocks, that are required to implement a blockchain. This module covers those topics, such as hashing algorithms, digital signatures, and more.

HASHING

Hashing transforms a byte array into a fix length and unique value, which is a called a digest. The digest cannot be reversed. Even the smallest change to the source data can change the results completely.

SHA-2 (Secure Hash Algorithm 2) is the one-way compression routine used with blockchains.

For verification of data integrity; you compare the hash to a previously calculated hash. If the hash has changed, the original data has been compromised. For example, computing and comparing the hash of a block can verify whether the blockchain, more than a specific block, has been modified or tampered with.

SHA256

The SHA-2 family includes six hash functions with digests (hash values) that are 224, 256, 384 or 512 bits:

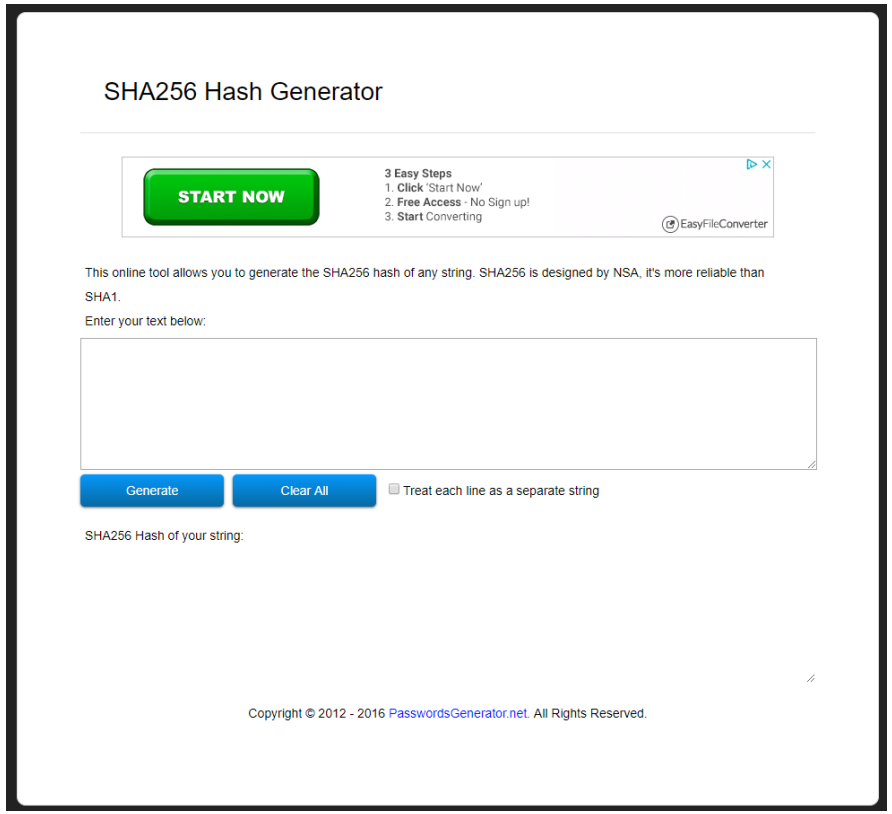
- SHA-224
- **SHA-256**
- SHA-384
- SHA-512
- SHA-512/224
- SHA-512/256

SHA-256 is the most commonly used hash algorithm in blockchain.

SHA-256

You can test the SHA-256 algorithm at this website.

http://passwordsgenerator.net/sha256-hash-generator/



HASHING - CODE

Here is a general purpose function that converts string data to a hash.

The string must be converted to a byte array and then later returned to a Unicode string.

The function Hash2HexString converts the hash to a human readable string.

```
public static String SetHash(String data)
            throws UnsupportedEncodingException,
            NoSuchAlgorithmException {


      byte[] hash=null;
      String stringHash;
    MessageDigest digest =
            MessageDigest.getInstance("SHA-256");
    hash = digest.digest(
            data.getBytes(StandardCharsets.UTF_8));
      stringHash=Utilities.Hash2HexString(hash);
      return stringHash;
}
```

HASHING - ANNOTATED

sole parameter is a Unicode string (1)

declare a byte array to store hash (2)

returns a message digest object for SHA-256 (3)

convert string to byte array which is then hashed (4)

convert hash to readable text (5)

```
`


public static String SetHash(String data)   1
          throws UnsupportedEncodingException,
          NoSuchAlgorithmException {

     byte[] hash=null;                  2
     String stringHash;
   MessageDigest digest =               3
          MessageDigest.getInstance("SHA-256");
   hash = digest.digest(
          data.getBytes(StandardCharsets.UTF_8));   4
     stringHash=Utilities.Hash2HexString(hash);     5
     return stringHash;
}
```

HUMAN READABLE HASH

```
public static String Hash2HexString(
          byte [] hash) {

     String buffer="";
     for(byte character: hash) {
          buffer+=Integer.toHexString(
          0xFF & character);
     }
     return buffer;
}
```

Hashes are a random sequence of byte values. As such, the hash may not be displayable. Hash2HexString iterates each character of the hash and incrementally transforms a hash into a displayable string.

This is the important code:

```
buffer+=Integer.toHexString(
     0xFF & character);
```

The 8-bit byte, which is signed in Java, is sign-extended to a 32-bit int. Apply the 0xFF mask to the byte undoes the sign extension.

You might also consider converting the hash to Base58.

WALLET

Wallets are an optional feature of blockchains. A wallet contains an address for a user or group. The wallet also has a calculated value, which represents the net value of the transactions sent to and from this address.

Some wallet implementations use the keypair from public key cryptography. The private key is used for authentication, while the public key is the wallet address. For that reason, you can distribute the public key to anyone wanting to send you something of value.

WALLET - 2

The first bitcoin address is:

1A1zP1eP5QGefi2DMPTfTL5SLmv7DivfNa,

and supposedly belongs to Satoshi Nakamoto.

DIGITAL SIGNATURES

Public-key cryptography is used extensively in blockchains.

- Confirm the identity of a sender
- Verify ownership of something of value
- Avoids sender repudiation

You sign data with your private key to create a digital signature. The signature can be confirmed later using the public key.

Digital signatures are not encryption and the original data cannot be discovered from the signed data.

DIGITAL FINGERPRINT

Digital fingerprints prove the authenticity of data (.i.e., data is not tampered with). Digital fingerprints refers to the output of a one-way function. This allows you to identify large amounts of data or files through a small fixed sized byte representation, which is called a digest. For Blockchain, hash are used for this purpose. Importantly, you cannot reverse engineer the source data from the digest.

ELLIPTIC CURVE CRYPTOGRAPHY

The public key created for blockchains must be truly unique. Collision could result in something of value being sent to the wrong person or destination. Private keys need to be sufficiently randomized to prevent potential problems. With elliptic curve cryptography the possible combinations are sufficiently large to virtually prevent any collisions. For elliptical curve algorithms a public key is a location on curve, represented by X and Y coordinates.

Using elliptic curve multiplication, a one way cryptography function, you can generate the public key.

Here is an excellent introduction to elliptic curve cryptography.

https://bit.ly/2fzPr8O

ELLIPTICAL CURVE - CODE

This code creates a public and private key using an elliptic curve. For convenience, a elliptic curve provider from Bouncycastle.org is used:

https://bit.ly/2EsZgf5

For the function KeyPairGenerator.getInstance, “ECDSA” selects the Elliptic Curve Digital Signature Algorithm from the Bouncycastle provider (“bc”).

```
KeyPairGenerator
keyGen=KeyPairGenerator.getInstance("ECDSA", "BC");

SecureRandom
random=SecureRandom.getInstance("SHA1PRNG");

ECGenParameterSpec ecSpec=new
ECGenParameterSpec("prime192v1");


keyGen.initialize(ecSpec, random);

KeyPair keyPair=keyGen.generateKeyPair();

privateKey=keyPair.getPrivate();

publicKey=keyPair.getPublic();
```

ELLIPTICAL CURVE - ANNOTATED

get an object for creating a public / private key pair (1)

get a pseudo random generator based on SHA1 (2

use the X9.62 curve prime192v1 1 (3)

Initialize the key generator for an elliptic curve and a source for randomness (4)

create public and private key (5)

extract the private key (6)

extract the public key (7)

```
KeyPairGenerator keyGen=KeyPairGenerator.getInstance(          1
      "ECDSA", "BC");
SecureRandom random=SecureRandom.getInstance("SHA1PRNG");      2
ECGenParameterSpec ecSpec=new
      ECGenParameterSpec("prime192v1");                        3

keyGen.initialize(ecSpec, random);                             4
KeyPair keyPair=keyGen.generateKeyPair();                      5
privateKey=keyPair.getPrivate();                               6
publicKey=keyPair.getPublic();                                 7
```

ENCRYPTION

Although Bitcoin does not use encryption, you may however leverage encryption in your blockchain to keep sensitive data secret. There is no rule against that.

You can use an encryption algorithm to translate data into a secret, which is called a cipher. Decryption is the process of converting a cipher back to the original data.

Symmetric encryption is done with a single key. The key is used for both encryption and decryption.

Asymmetric encryption requires a keypair. The private key is used to encrypt data, while the public is used decrypt.

ENCRYPTION - CODE

The goal of encryption algorithms is to prevent the decryption of a cipher without using the key. The likelihood should be equivalent to brute force, which is typically impractical. Longer keys increases the difficulty of decrypting a cipher.

```
import java.security.KeyPair;
import java.security.KeyPairGenerator;
import java.security.NoSuchAlgorithmException;
import java.security.PrivateKey;
import java.security.PublicKey;
import javax.crypto.Cipher;

public class Sample {
    public static void main(String [] args) throws Exception {
        // generate public and private keys
        KeyPair keyPair = buildKeyPair();
        PublicKey pubKey = keyPair.getPublic();
        PrivateKey privateKey = keyPair.getPrivate();
```

ENCRYPTION – CODE - CONTINUED

```
    // encrypt the message
    byte [] encrypted = encrypt(privateKey, "This is a secret message");
    System.out.println(new String(encrypted));  // <<encrypted message>>

    // decrypt the message
    byte[] secret = decrypt(pubKey, encrypted);
    System.out.println(new String(secret));     // This is a secret message
}

public static KeyPair buildKeyPair() throws NoSuchAlgorithmException {
    final int keySize = 2048;
    KeyPairGenerator keyPairGenerator = KeyPairGenerator.getInstance("RSA");
    keyPairGenerator.initialize(keySize);
    return keyPairGenerator.genKeyPair();
}
```

ENCRYPTION – CODE - CONTINUED

```
    public static byte[] encrypt(PrivateKey privateKey, String message) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.ENCRYPT_MODE, privateKey);

        return cipher.doFinal(message.getBytes());
    }

    public static byte[] decrypt(PublicKey publicKey, byte [] encrypted) throws Exception {
        Cipher cipher = Cipher.getInstance("RSA");
        cipher.init(Cipher.DECRYPT_MODE, publicKey);

        return cipher.doFinal(encrypted);
    }
}
```

RSA

RSA is the most popular algorithm for encrypting and decrypting data and was defined in 1978. RSA is a moniker for Ron Rivest, Adi Shamir, and Leonard Adleman, who together for published the RSA algorithm. It is an asymmetric algorithm relying on a keypair.

POP QUIZ: CRYPTOGRAPHY ALGORITHMS

What is the difference from using SHA-256 versus RSA for encryption and decryption?

5 MINUTES

CONSENSUS

The blockchain is a decentralized network without a central authority. How often have you heard this already? Without a trusted authority, there must be another mechanism to make decisions for the network. The nodes themselves make the important decisions though consensus. There are a variety of ways that consensus is used within a blockchain, including forking and accepting new blocks.

PROOF OF WORK

Work must be done to add a block to the blockchain. Completing the complex algorithm is considered *proof of work*. Performing proof of work to add a block to the blockchain is called mining. Proof of work is an investigation (mining) a miner (node) performs, which often requires installing a powerful computer or mining rig to solve a complex mathematical puzzle. This verifies not only the block but the transactions contained within the block.

Multiple miners on the network compete to be the first to find a solution for the mathematical problem pertaining to the candidate block. The first miner to solve the problem announces their solution simultaneously to the entire network.

Sometimes the work is made artificially *difficult* to protect the security of the blockchain. This makes it computational improbable for a hacker to attack the entire blockchain. Overtime mining becomes more efficient and requires less time to complete. For that reason, the complexity of work periodically increases to maintain a consistent amount of effort to add a block.

HASHCASH

Hashcash is part of the proof of work performed by the mining algorithm for various cryptocurrencies. Hashcash entails completing a predetermined amount of work that can be quickly verified. Hashcash was originally used to protect against email spam and denial of service attacks.

For email, a Hashcash stamp is added to the header, which requires some effort. This modest overhead can scale to an enormous amount of effort (and cost) for a spammer and acts as a deterrent.

Alternatives to Hashcash includes Cuckoo Cycle and Scrypt.

PROOF OF WORK - BITCOIN

For bitcoin, blocks are added about every ten minutes. This is controlled by the Hashcash mining algorithm that is implemented for blockchains. A block is mined when the current hash is prefixed with a certain number of zeroes. The numbers of zeros is the difficulty. This must be accomplished through brute force and iterating a nonce. For example, a difficulty of three would mine blocks until these results are found. More about this later.

```
0003WLR3k2X2

0000RA94212W

000Q112RT43E

000PEQ12T543
```

PROOF OF WORK – BITCOIN - CODE

the *difficult* parameter sets the level of difficulty (1)

creates the comparison string for determining if the mathematical puzzle has been resolved (2)

is a solution found? (3)

increase the nonce (4)

recalculate the hash (5)

mining completed (6)

```
public void mineBlock(int difficult) {          1
      String target=new String(                                  2
            new char[difficult]).replace('\0', '0');

      while(!hash.substring(0, difficult).equals(target)){       3
                  nonce++;                                       4
                  hash=calculateHash();                          5
      }
      System.out.println("Block mined!! : "+hash);               6
}
```

PROOF OF STAKE

Proof of Stake (PoS) is emerging as an alternative to Proof of Work where a consensus algorithm is used to validate blocks. In PoS, blocks are assigned to miners using a randomization algorithm. The selection is influence by the amount of cryptocurrency (the stake) held by the miner and how long. This replaces selecting miners by computation power with miners that have a stake in the health of the blockchain.

Randomization negates the arms race that has emerged with miners. Power centralizes on the miners with the most computational power, which itself creates a security vulnerability. The conundrum is that the powerful miners will receive most of the compensation. This disparity means that powerful miners can acquire more resources and increase their dominance.

GREENER SOLUTION

Proof of work encourages more and more computational power as a winning strategy for mining blocks. This consumes a lot of energy! Proof of Stake takes away the energy and computational power requirement of PoW and replaces it with stake

This is the reason that several cryptocurrencies have adopted PoS.

- Blackcoin
- Lisk
- Nxt
- Peercoin

PROOF OF STAKE - SCENARIO

There are four miners for a new cryptocurrency called VegasCoin:

- Bob 35 coins
- Rebecca 40 coins
- Sarah 5 coins
- Herbert 20 coins

The miners will be selected in a random but weighted order. Bob has a 35% chance of being selected, Rebecca 40%, and so on.

MERKLE TREE

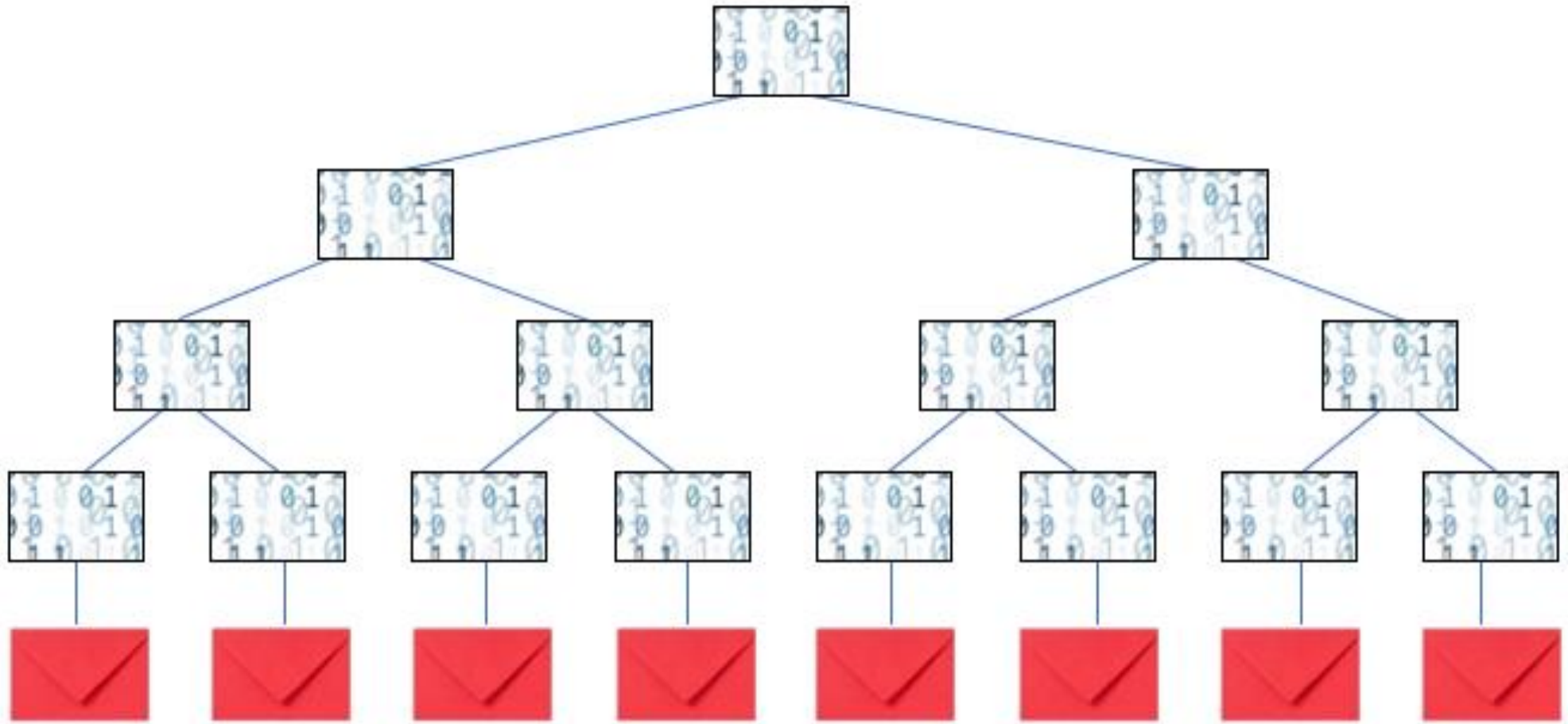
A Merkle Tree is a binary structure that efficiently and quickly allows the verification of large amounts of data. This structure has an important role in blockchain for secure verification of transactions. The Merkle Tree creates a digital fingerprint (hash) that is effective in determining if a transaction is contained within a block.

Merkle Trees are created by hashing pairs of nodes and then pairs of hashes until the top of the tree is reached – root hash. Hashing is typically done using a SHA-2 algorithm.

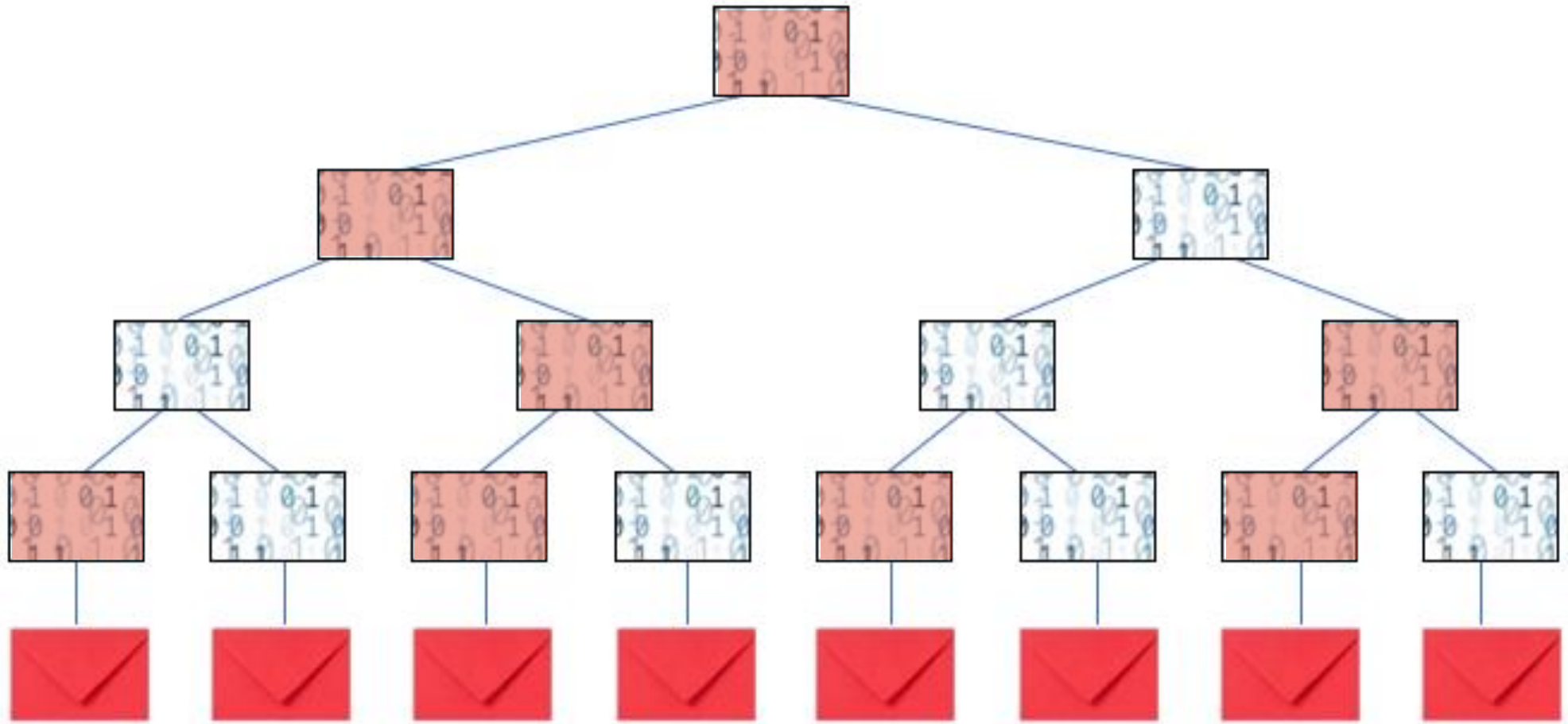
For a Bitcoin blockchain, the root hash is stored in the block header.

One advantage of a Merkle Tree is the capability to download and verify individual branches. Another advantage is that a Merkle Tree requires minimum memory to verify large numbers of transactions.

MERKLE TREE DIAGRAM



MERKLE TREE DIAGRAM - 2



SIMPLE PAYMENT VERIFICATION (SPV)

A Simplified Payment Verification (SPV) node is an efficiency that allows for the verification of transactions without downloading all the block data of the relevant blockchain. Merkle trees are used extensively by SPV nodes. The SPV node downloads only metadata, including the Merkle root. A SPV node downloads less than an gigabyte or two (depending on the size of the blockchain) versus tens of gigabytes of data.

As mentioned, SPV nodes do not have all the data. For Bitcoin, SPVs only receive block headers; but not the block body, which includes the transactions. Even without the block data (i.e., transactions), the SPV node can still verify a transaction with the assistance of a full node.

TRANSACTION VERIFICATION SPV

Scenario: A SPV node wants to confirm that a transaction exists within a block on the blockchain.

When the SPV receives a transaction, it can broadcast a request to the full nodes that exist on the network. The questions / message is “does this transaction exist”? Full nodes benefit from having all of the data from the blockchain. They can iterate through the blocks, recreating the Merkle Tree, until the transaction is located. The Merkle Tree is recreated using buddies.

Once found, the full node sends the block hash and Merkle buddies to the SPV node. The SPV can then confirm the transaction exists and the correct block.

Lab 3- Merkle Root

MERKLE TREE

In this lab, create a Merkle Tree. You will be provided the framework of an application to complete.

Merkle Trees are integral part of the blockchain.

Import various packages as needed.

MERKLE TREE

Create a class call MerkleTree.

Here is the method for hashing strings. You can enter “as is”.

```
public String getSHA2HexValue(String str) {
        byte[] cipher_byte;
        try{
            MessageDigest md = MessageDigest.getInstance("SHA-256");
            md.update(str.getBytes());
            cipher_byte = md.digest();
            StringBuilder sb = new StringBuilder(2 * cipher_byte.length);
            for(byte b: cipher_byte) {
              sb.append(String.format("%02x", b&0xff) );
            }
            return sb.toString();
        } catch (Exception e) {
                e.printStackTrace();
        }
        return "";
  }
```

MERKLE TREE 2

Before the getSHA2HexValue function, define another function called GetNextMerkleRow. The function has a single parameter: List<string>. It also returns a List<string>.

This function accepts the current row of the Merkle Tree and returns the next row. You will build the Merkle Tree bottom-up (i.e., starting with the bottom row).

Define a List<string> for storing the next row of the Merkle Tree.

Define an index counter.

MERKLE TREE 3

Still in the GetNextMerkleRow function, iterate elements of the current row in the Merkle Tree with a while loop. Continue until the count exceeds the elements along the row. Combine each item with a buddy and then hash.

- Get left element at index counter

```
String left = tempTxList.get(index);
```

- Get right element at index counter +1. If right element does not exist, assign empty string.
- Combine left and right string. Hash the resulting string with the helper function.
- Add to the list for the next row
- Increment the count

Return the next row in the Merkle Tree.

MERKLE TREE 4

At the top of the class, create the data members:

- Define *elements* as a variable which is a List<String>. This is the source elements for the Merkle Tree.
- Define root which is a string. This will contain the root hash of the Merkle Tree.

Create a public constructor for the MerkleTree class, which accepts a List<String> as a parameter. Initialize *elements* with the parameter. Set *root* to be an empty string.

Create an accessor function that returns the root string.

MERKLE TREE 5

Create a new function: Merkle_Tree. You will call Merkle_Tree to create the actual Merkle Tree.

In the function, first define a List<String> variable called tempTxList. Initialize with the source elements of the Merkle Tree: elements. In a for loop, copy the items from elements into tempTxList.

Get the bottom row of the Merkle Tree. Call GetNextMerkleRow with tempTxList as the parameter. Save the bottom row into a new temporary list: List<String>.

In a while loop, loop until there are no remaining rows in the Merkle Tree:

```
while (newTxList.size() != 1) {
```

Inside the loop, get the next row of the MerkleTree. Call GetNextMerkleRow with the temporary list and save results back into same variable.

After the while loop is complete, you are at the top of the Merkle Tree. Assign the first element of the temporary list to the root.

MERKLE TREE 6

Test the Merkle Tree with the following code:

```
List<String> tempTxList = new ArrayList<String>();
tempTxList.add("a");
tempTxList.add("b");
tempTxList.add("c");
tempTxList.add("d");
tempTxList.add("e");

MerkleTree merkleTree = new MerkleTree(tempTxList);
merkleTree.merkle_tree();
System.out.println("root : " + merkleTree.getRoot());
```

Blockchain / Blocks

Develop a passion for learning.

BLOCKCHAIN

The blockchain is an ordered, cross-linked, sequence of blocks; where each block contains transactions.

Blocks are uniquely identified with a SHA-256 cryptographical hash. Each block also references the previous block in the structure by its hash. The previous block is called the parent block, while the current block is the child.

Blockchains create a generational relationship for blocks in the chain. If you modified a parent block, the change will reverberate through all descendants, which is expensive. All the descendant hashes would have to be recalculated and proof of work redone. This computational overhead makes a blockchain virtually immutable. The longer the blockchain the more immutable.

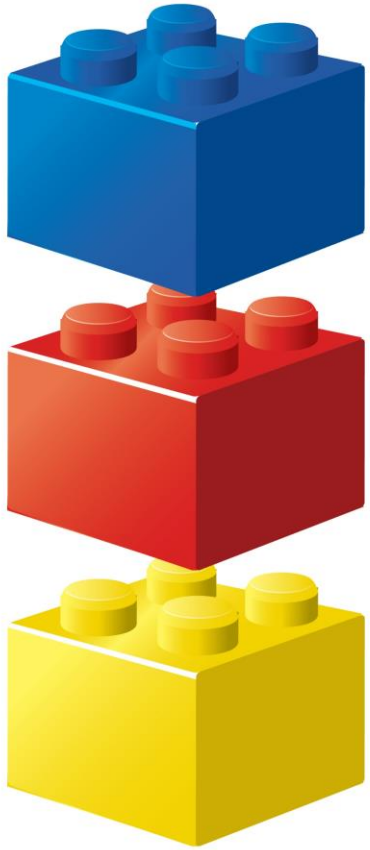
POP QUIZ: PARENTS AND CHILDREN

Can a blockchain block, have more than one parent?

Can a blockchain have more than one child?

10 MINUTES

GENESIS BLOCK



Every blockchain must have a first block. This is the called the genesis block. Starting with the latest block, previous hashes create a chain to the first block, which is the genesis block (i.e., Block 0). The previous hash for the genesis block is null.

The blockchain height is the distance from the genesis block to the top of the chain.

BLOCK DETAILS

The block contains transactions that have been broadcast to nodes of the blockchain. The transactions are the "data" of the block and represent transfer of value.

For Bitcoin, a block is separated into a block header and body. For example, hashes from previous blocks are stored in the block header, while the body contains transactions. The block is 80 bytes, while the average body is around 125kb. There are many other fields in a block.

.

BLOCK SCHEMA

Size	Field	Description
4 bytes	Block size	The size of the overall block
80 bytes	Block header	Block hasher, Merkle root, etc.
1-9 bytes	Transaction counter	Number of transactions in body
	Transactions	Transactions in block

This is the overall schema for blockchain blocks. Minor deviations for your particular blockchain is not only accepted but assumed. Consider the block schema to be the base type which you can extend as necessary.

BLOCK HEADER

Size	Field	Description
4 bytes	Version	Reference to a specific Bitcoin protocol
32 bytes	Hash (previous block)	Hash of the previous block
32 bytes	Merkle root	Reference to the transactions
4 bytes	Timestamp	Block creation time
4 bytes	Difficulty	POW difficulty
4 bytes	Nonce	POW counter

The block header includes a reference to the previous block. Version number to pair with the correct implementation. The difficulty, timestamp, and nonce are used for mining the block. Finally, the block contains a Merkle root, which is a reference to the payload (i.e., transactions).

Note: the block does not contain the hash of the current block.

POP QUIZ: BLOCK HASH

Is there is a practical reason that the hash of the current block is not contained in the header?

5 MINUTES

BLOCK IDENTIFIER

Blocks are identified with the hash or height.

The block hash is calculated using the SHA-256 algorithm with the block header as input. It is always calculated at the destination. However, the block hash is stored as the previous hash in the child block.

Some blockchains save the block hash in a separate list for quick indexing.

BLOCK IDENTIFIER - CONTINUED

Row height can also be used to identify a block within a blockchain. First block in block chain is block^0, which is the genesis block. This is the hash of block^0 of the Bitcoin blockchain:

000000000019d6689c085ae165831e934ff763ae46a2a6c172b3f1b60a8ce26f

For Bitcoin, the block height on January 1, 2017 was 446,000.

POP QUIZ: BLOCK HEIGHT

Is the block height for a specific block always unique?

Can you find the genesis block using either blockchain.info or blockexplorer.com?

https://blockchain.info/block/{blockHash}

https://blockexplorer.com/block/{blockHash}

5 MINUTES

HASH

Hash the elements of the block you do not want tampered with using the SHA-256 algorithm.

```
function CalculateHash
        szIntermediateResult=szPreviousHash+szTimeStamp+
             createMerkle(transactions)
      szCurrentHash=CreateHashAsString(szIntermediateResult)
      return szCurrentHash
```

BLOCKCHAIN

Creating a block chain is simply adding one block to the next.

```
function entryPoint
    arrayBlockchain
    block[1]=blockchain.add(new block, nothing)
    block[2]=blockchain.add(new block, block[1])
    block[3]=blockchain.add(new block, block[2])
```

IS BLOCK CHAIN VALID

You can confirm the correctness of the blockchain by iterating the blocks and examining the current and previous hashes. Rehash the blocks and compare to the saved hashes within the blocks to confirm that the immutable blockchain has not changed. Finally, confirm that each block is mined. Blockchain should not include any blocks that are not mined.

ISCHAINVALID

```
function IsChainValid
          blockCurrent=Block
          blockPrevious=Block
          szHashTarget=zeros[difficulty]

          for(increment i=1 to intSizeOfBlockhain)
                    currentBlock=Block(i)
                    previousBlock=Block(i-1)

                    if(!currentBlock.hash. ==  currentBlock.CalculateHash()
                              return false;

                    if(!previousBlock.hash == currentBlock.previousHash)
                              return false;

                    if(!currentBlock.hash.substring[0,3] == hashTarget)
                              return false;
```

DOUBLE SPENDING

Double spending is one of the important problems that central authority prevents. This means transferring the same value twice. With a cryptocurrency, this means spending available coins twice (i.e., overdrawing the account). With a real estate blockchain, that could mean selling a house twice to different people. As the central authority, a bank prevents that in the real world.

For a blockchain, double spending is preventing by the overhead of proof of work and consensus.

LONGEST BLOCKCHAIN

Blockchain hosts a distributed ledger, where the ledger is replicated to each node. The longest valid blockchain is accepted as the correct entity for the network. What prevents someone from modifying a block and then submitting a longer chain. Remember, the blockchain is supposed to be immutable. Proof of work prevents this type of interference. The hashcash proof of work means it takes considerable effort to create new blocks. Remining the entire chain to present an altered chain would require more effort than feasible; especially while other block are being added in realtime.

BLOCK FORKS

As mentioned, the block hash is unique; but block height is not necessarily unique. Block forks can create duplicate height. When do forks occur? Block forks can temporarily occur as the result of PoW. Two nodes complete POW almost simultaneously and broadcast the same block. When that occurs, the longest blockchain rules applies and the nodes select the appropriate block.

Lab 4- First Blockchain

FIRST BLOCKCHAIN - 1

In this lab, you will create your first blockchain. Have fun!

Import various packages as needed.

Download and install Gson for Eclipse. For help, watch this video:

https://bit.ly/2JBucO1

Create a new Java project. The initial class is *Block*. Define the initial data members:

- hash: string (not normally saved in block)
- previousHash: string
- data: string
- timestamp: long
- blockId:int
- count:static int

FIRST BLOCKCHAIN - 2

Also create four public accessor functions that return the value of each data members, except blockId and count, for example getHash.

Create a new class called StringUtil. In the class, define a function that will hash data. The function will accept a single string parameter and return a string (i.e., the hash).

Inside a try block:

- Convert the string parameter into bytes with getBytes function. The string is format is Unicode.
- Define a MessageDigest object with getInstance to create a one-way hash. The hash name is "SHA-256".
- Create the hash with MessageDigest.digest. The single parameter are the bytes from the string parameter. The result is the hash in bytes.
- Define and initialize an empty StringBuffer called hexString.

FIRST BLOCKCHAIN - 3

Still in the hashing function, convert the hash into a readable string.

- Define a for loop that iterates the bytes of the hash.
- Convert each byte to an hexadecimal character:

```
String hex=Integer.toHexString(0xFF & hash[i]);
if(hex.length() == 1) hexString.append('0');
hexString.append(hex);
```

Return hexString from function

If exception is raised, throw a new RuntimeException.

FIRST BLOCKCHAIN - 4

In the Block class, add a public function: CalculateHash. The function returns a string. The function will create a hash for the Block class.

- Define a string with combines previousHash+string version of timestamp+data
- Call the hashing utility with the string
- Return the hash from the method

Create a public constructor to initialize the Block class.

- Two parameters: data:string and previousHash:string
- Assign data and previousHash to the identical members of the class
- Assign timestamp the current date using the Date class
- Update hash with CalculateHash function
- Increment the static count variable
- Assign blockId the count.

FIRST BLOCKCHAIN - 5

Define a new class from main – the entry point for your application.

You need a blockchain! In the class, create an ArrayList generic of Block types.

In main, add blocks to ArrayList:

```
blockchain.add(new Block("Hi im the first block", "0"));
```

Add three more blocks to the blockchain using arbitrary data.

In for loop, display each block with the following data:

- blockId
- blockHash
- previousHash

FIRST BLOCKCHAIN - 6

Extra credit:

Earlier in the module, pseudo code was presented to test the validity of a blockchain. Create a function called IsChainValid. After creating the blockchain, test correctness of blockChain with IsChainValid. Display an appropriate message after calling function.

Mining Blocks Persistence

Develop a passion for learning.

MINING

Mining replaces the centralized authority for the blockchain through competitive computation. The miners create a distribute authority model where decisions are made by consensus.

Mining represents the clearinghouse for transactions in the blockchain; but also the security mechanism. For cryptocurrencies, trust in a secured blockchain is essential.

COMPENSATION

Miners are typically rewarded in some manner. For Bitcoin, miners receive two awards: coin for each block mined and transaction fees. This aligns miners with the success of the blockchain and increases the monetary supply of the coins.

The transaction fees is compensation for all the transactions included in the block. The reward is for completing PoW and incurring related energy costs, and adding a block to the blockchain.

THE MONEY SUPPLY

Miners receive compensation for mining blocks, which is essential the money supply. That is how newly minted bitcoins are created. For Bitcoin, the number of coins created through mining is halved about every 4 years. Originally, compensation was 50 bitcoins per block. Compensation was reduced to 25 bitcoins November 2012. July 2016 is was reduced to 12.5 bitcoins. This will continue to about 2140 when the maximum number of bitcoins will be reached: @ 21 million.

TRANSACTION FEES

Miners also earn money from transaction fees. Transaction fees are the differences of the transaction inputs and outputs. Transaction inputs and outputs are discussed in the later Transaction module. Presently, transaction fees represent about a half percent of the miner compensation. Not a lot! However, rewards will continue to diminish. As that occurs, transactions will have to increase under the current hashcash model.

CONSENSUS (AGAIN)

Miners make decisions for the blockchain through consensus.
Emergent because the consensus does not occur at a specific time.
Consensus emerges as the byproduct of asynchronous interaction of potential thousands of miners.

The consensus of:

- Independent validation of each transaction
- PoW on each block
- Verification of each block

TRANSACTION VERIFICATION

Blockchains generally have rules that enforce the correctness of the blockchain. The nodes are responsible to adhering to there rules.

For Bitcoin for example, transactions should adhere to rules before being propagated to the blockchain. Here are some of the rules:

- Transaction structure must be correct
- The lists of inputs and outputs should not be empty – with one exception
- Transactions must be less than 21 million bitcoins, which is a practical consideration
- Reject transactions where the some of input is less than the sum of output values

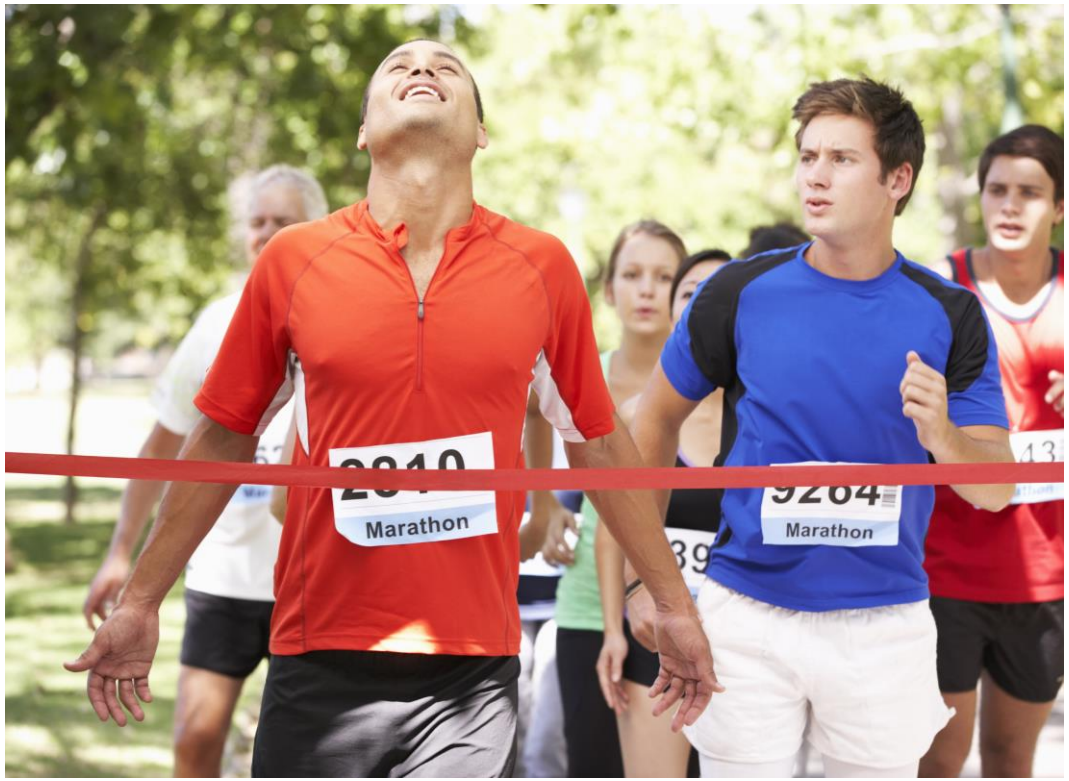
POP QUIZ: TRANSACTION VERIFICATION

What is the purpose of this rule:

For each input, if the referenced output exist in any other transaction in the pool, the transaction must be rejected.

5 MINUTES

COMPETITORS



Miners are in competition to successfully mine a block first and broadcast to the other nodes. However, this is more like a boxing match than a track and field race. In a boxing match, the boxers compete to win each round in succession. You can win points in every round.

For this reason, some nodes join mining pools. They win more consistently but for less money.

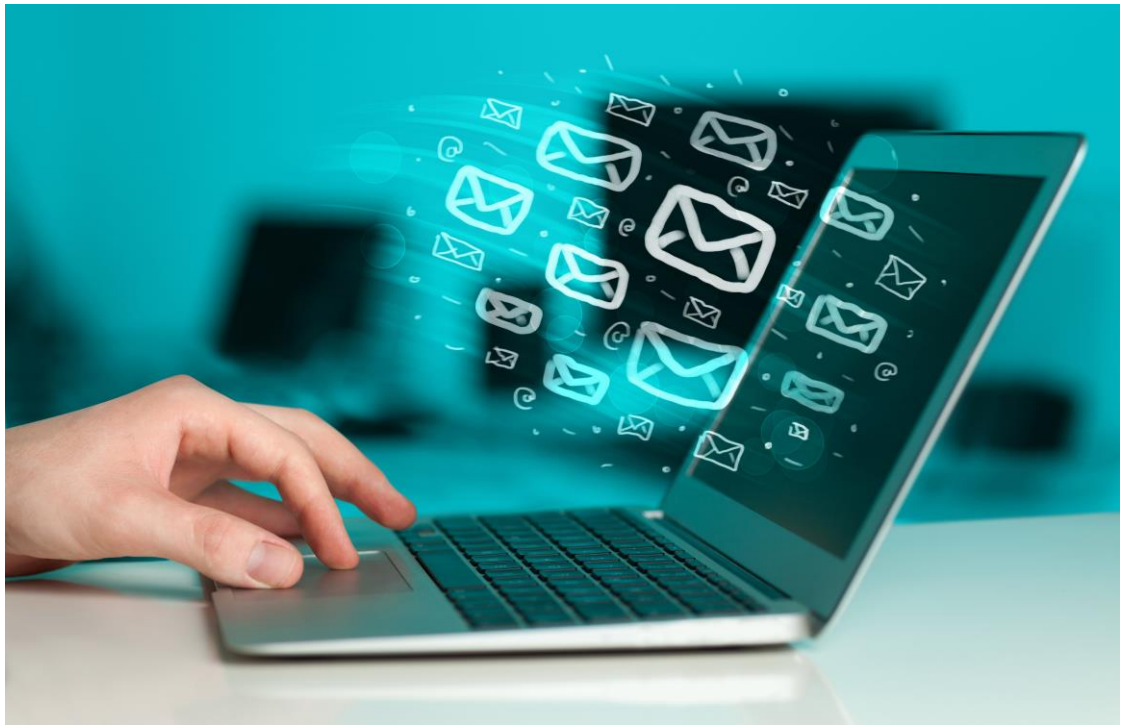
COMPETITORS - STEPS

Miners are in competition to successfully mine a block first and broadcast to the other nodes. If a miner receives a block while mining, then they lose that round!

The miner will perform some cleanup and independently verify the new block.

1. Remove transactions from the transaction pool that were included in the block
2. Remaining transactions are unconfirmed and can be included in a future candidate block

TRANSACTION POOL



Transactions that are verified are added to a transaction pool. Periodically, when a predefined threshold is met, the transaction are collected into a candidate block and mined. For Bitcoin, the threshold is every 10 minutes. If mining is successful, the block is broadcast to the other nodes.

COINBASE TRANSACTION

Miners receive compensation for their effort. The rewards is contained in a special transaction: coinbase transaction. For coinbase transactions, the address of the sender for the transaction is the miner. The compensation or reward includes both the mining and transaction fee.

Unlike other transactions, the coinbase transaction does not have a matching Unspent Transaction Output. More about this later.

What prevents a miner from awarding themselves a huge reward? The other nodes would notice and not accept the block. Problem caught and solved!

POW CODE (AGAIN)

```
public void mineBlock(int difficult) {
     String target=new String(
          new char[difficult]).replace('\0', '0');

     while(!hash.substring(0, difficult).equals(target)){
               nonce++;
               hash=calculateHash();
     }
     System.out.println("Block mined!! : "+hash);
}
```

Here is sample code for PoW as shown before. Increase the *difficult* parameter to lower the target value, which makes finding the solution harder and more time-consuming.

MORE ABOUT PROOF OF WORK ALGORITHM

The essential aspect of the PoW algorithm is the difficulty in predicting results based on inputs. For those reasons, the complex algorithm must be resolved through brute force.

PoW of work is finding a matching hash through brute force. A nonce is included in the input data. If the hash does not match, the nonce is increased and rehash. This process is repeated until a match is found.

MINING CODE

Transactions that are verified are added to a transaction pool. Periodically, when a predefined threshold is met, the transaction are collected into a candidate block and mined. For Bitcoin, the threshold is every 10 minutes. If mining is successful, the block is broadcast to the other nodes.

The computational ability of miners continues to scale. This means that more can be mined within 10 minutes. In addition, the number of people competing to solve the PoW continues to increase.

For Bitcoin, every 2,016 blocks all the nodes recalculate the PoW. How long did it take to calculate the last 2,016 blocks? If less than 20,160, difficulty is increased proportionally. If more, however unlikely, the difficulty is reduced.

MINER PROFITABILTY

The ability to mine a block is entirely independent of the number of transactions. This means the primary compensation, at the moment, is the reword for mining. Compensation is therefore linked to computational power and higher electricity costs. This is the primary factor into whether mining is profitable.

Here is a great article on the topic.

https://bit.ly/2GLWVhk

BLOCK CRITERIA

/As the block propagates across the network, each node applies a criteria to either accept or reject the block.

Here are some of the criteria:

- Block structure is valid
- Block has been mined
- For Bitcoin, the first transaction is coinbase
- All transactions in block are valid
- And so on

ADDING A BLOCK

These are the steps for adding a block to the blockchain.

- The node will attempt to locate the parent block from the previous hash
- Typically the parent is at the top of the blockchain
- xxxxxIf not found, the block is added to the orphan pool. When the parent is found in the future, the block is added then.
- Accepting the block to a blockchain is a “yes” vote from that node

PERSISTENCE

You must have a way to persist a blockchain. This is necessary for several reasons:

- When the blockchain resides in volatile memory and sometimes require rehydration
- Create a persistent blockchain
- Replication of the blockchain to other nodes
- Backup

Blockchains are typically persistent to either database or JSON.

DATABASE VERSUS JSON

For persisting the blockchain, databases are a popular option, such as SQLite and MySQL. JSON (JavaScript Object Notation) is another practical and effective option.

The details of SQLite and MySQL are beyond this course.

JSON solutions are convenient, implemented easily, and allows redundancy. JSON is clear text and easily transmitted around a public network. However, JSON is not the most compact format for data. This is particularly important when considering the vast number of transactions.

Make sure to conduct profiling before selecting your data strategy.

GSON

Google's GSON is a Java library that converts Java objects to JSON and the reverse. GSON has several advantages:

- Simple syntax
- Support for generics
- Custom representation of objects
- Well documented
- And more

SAVE BLOCKCHAIN CODE

In the sample code, blocks is an ArrayList<Block>, where Block is a structure that has fields for the previous hash, timestamp, and so on.

GSON is downloadable here:

https://github.com/google/gson

GSON will automatically handle the vagaries of the complex type and the generic type.

Basically, three lines of code to save the entire blockchain. Very cool!

```
public void SaveChain( ) throws IOException {

            try (Writer writer =
                        new FileWriter("Output.json")) {

                Gson gson = new GsonBuilder().create();


                gson.toJson(blocks, writer);


            }
}
```

SAVE BLOCKCHAIN CODE

Create file object for json (1)

create file (2)

write blockchain to file as json (3)

```
public void SaveChain( ) throws IOException {
            try (Writer writer =                    1
                    new FileWriter("Output.json")) {
                Gson gson = new GsonBuilder().create();   2

                gson.toJson(blocks, writer);   3

            }
}
```

READ BLOCKCHAIN

create JSON object (1)

create empty blockchain (2)

create stream for reading JSON from file (3)

associate file with input stream (4)

reflect type: arraylist of blocks (5)

read blocks from file (6)

Initialize blockchain with blocks (7)

```
Gson gson = new Gson();                                          1

ArrayList<Block> blocks=null;                                    2

try  {
      BufferedReader br = new BufferedReader(                    3

                  new FileReader("Output.json"));                4


                  java.lang.reflect.Type blockType =             5
                        new TypeToken<ArrayList<Block>>()
                        {}.getType();
      blocks = gson.fromJson(br,blockType);                      6

catch (IOException e) {
      e.printStackTrace();
}

Blockchain blockChain=new Blockchain(blocks);                    7

Block.SetSequence(blocks.size());
```

Lab 5- Mining

ADD MINING TO THE BLOCKCHAIN

Start with the minimal blockchain from the previous lab.

Add these two fields to the Block class.

difficulty: int

nonce:int

Initialize difficulty to three and the nonce to zero.

ADD MINING TO THE BLOCKCHAIN –2

Create a method called MineBlock with difficulty as the single parameter.

In the method, create several strings:

- Create a *target* string that contains a number of zeros. The number of zeros is defined by difficulty, which is the function parameter. This is the string you must resolved to.
- Create a string named *hash*. Initialized *hash* to string: "not mined". This is the current hash.
- Create a string *toHash*. Initialize to an empty string. This is the source string for hashing.

ADD MINING TO THE BLOCKCHAIN –3

Create a while loop. Iterate until hash is less than target. Inside of while loop:

- Iterate nonce
- Initialize toHash as previousHash+data+timestamp+nonce
- Create hash from toHash assign to hash variable

After while loop, update currentHash with mined hash.

ADD MINING TO THE BLOCKCHAIN –4

In main function, add four blocks with data: “a”, “b”, “c”, and “d”.

List current and previous hashes for each block.

Test the application with difficulty of 5 and then 7. What is the impact?

Develop a passion for learning.

DOORLOCK

I am a regular visitor to Starbucks and order a dry extra hot cappuccino, with *no chocolate*. Chocolate on top ruins everything.

Many Starbucks located in a urban area have a cipher lock on the bathroom door. You have to ask the barista *every* day for the key code. Since the key code remains valid for a day, the key code for today will not work tomorrow.

That is the best practice for blockchains. We will explain in this module.

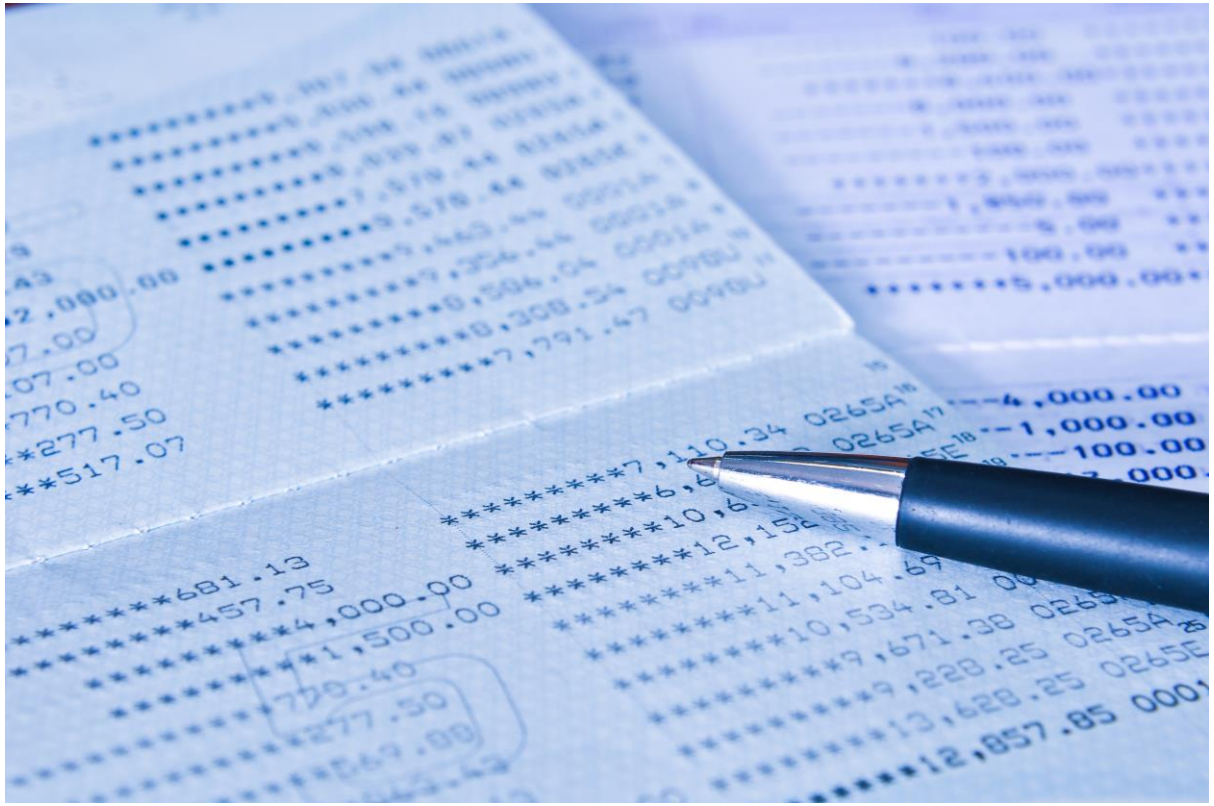
WALLETS

Transactions are used to transfer coins between owners. The wallet has one or more key pairs. This allows the wallet to receive or send value.

The wallet is also a software application able create and receive transactions. Here are some wallet tasks:

- Access net value of transactions
- Manage keys
- Tracking balance
- Creating and signing transactions

WALLETS – ARE THEY REAL?



"The wallet represents a user account, with a monetary balance, and keys available to that account." – said by many!

This is somewhat correct but wrong. The blockchain hosts a distributed ledger of input and output transactions. Nothing more. In a strict interpretation of the blockchain, there are no wallets.

However, even a bank account at the lowest level is simply credits and debits. My bank balance is an aggregation of those transactions. Everything, including the balance, is a derivation of that data.

KEYCHAIN

An address in a wallet is called a key. And a wallet can hold one or more keys.

There are two types of wallets: nondeterministic and deterministic. The type of wallet determines how the keys are created.

POP QUIZ: WALLET KEYS

What is the benefit of storing multiple keys in a wallet?

10 MINUTES

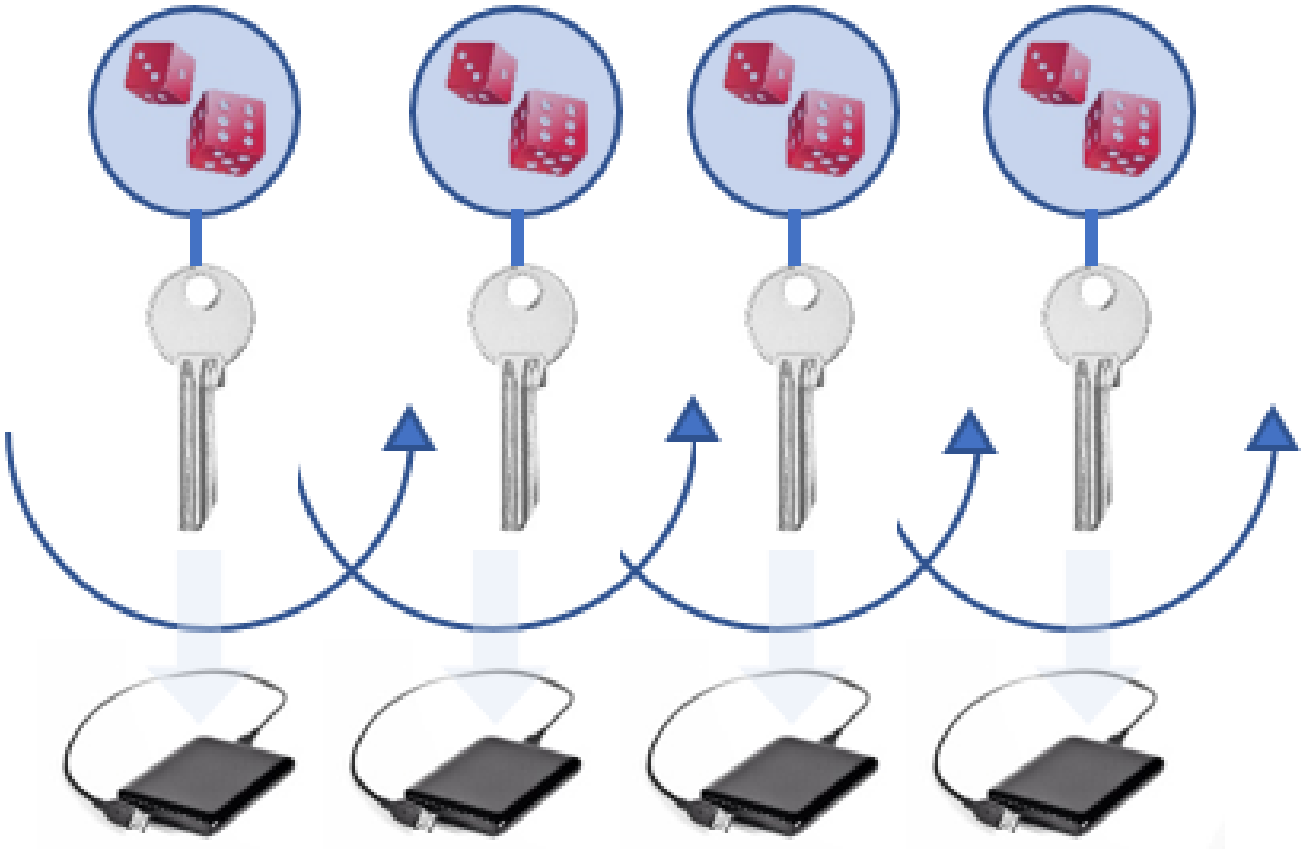
NON-DETERMINISTIC WALLET

A non-deterministic wallet contains unrelated private keys, which are generated independently from a random number. This type of wallet is appropriately called *Just a Bunch of Keys* (JBOK).

There are several shortcomings to a non-deterministic wallet:

- You must backup the wallet frequently
- Funds are lost forever if the wallet is corrupted
- Security is lessened from multiple use of keys
- The wallet is harder to maintain

NON-DETERMINISTIC DIAGRAM



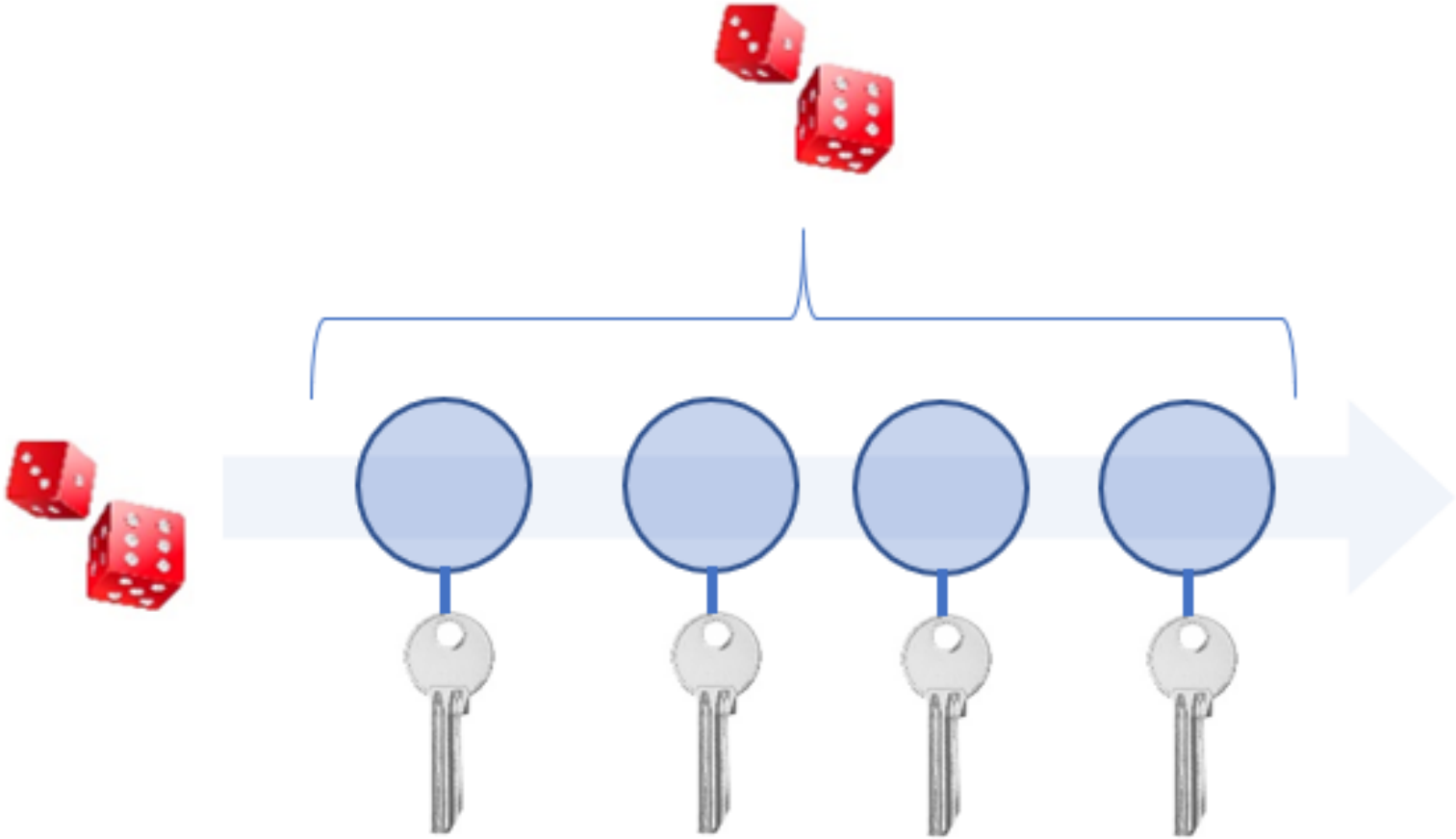
DETERMINISTIC WALLET

For a deterministic wallet, the private keys in the wallet are related. The keys are created using a one-way hashing. The keys are create using the same random seed.

Unlike the non-deterministic wallet, you do not have to backup the wallet frequently. The keys can be recreated using the same seed.

Instead of using a random seed, you can create a deterministic wallet from a mnemonic, which is essential a passphrase. This is the choice used by most cryptocurrencies today.

DETERMINISTIC WALLET - DIAGRAM



WALLET IN CODE

```
structure Wallet

     szPrivateKey
     szPublicKey

     UnspentOutputTransactions  // array of unused transactions
```

Here is an example wallet. It contains the public / private cryptographic keys as either a byte array or human readable string, the value of the wallet, which is the value of unused transactions. The keys can also be represented as a keypair.

Value can be anything: coins, document, real estate digital keys, or anything else of value.

BITCOIN ADDRESS

A bitcoin address is not a public key but created from a public key and consists of a series of letters and numbers, which is a 160-bit hash of the public key from the public / private keypair created along an elliptic curve. Anyone can have one or more bitcoin addresses.

1 – Create a private ECDSA key

18E14A7B6A307F426A94F8114701E7C8E774E7F9A47E2C2035DB29A206321725

2 - Take the corresponding public key

0450863AD64A87AE8A2FE83C1AF1A8403CB53F53E486D8511DAD8A04887E5B23522CD470243453A299FA9E77237716103ABC11A1DF38855ED6F2EE187E9C582BA6

3 - Perform SHA-256 hashing on the public key

600FFE422B4E00731A59557A5CCA46CC183944191006324A447BDB2D98D4B408

BITCOIN ADDRESS - 2

4 - Perform RIPEMD-160 hashing on the result of SHA-256

010966776006953D5567439E5E39F86A0D273BEE

5 - Add version byte in front of RIPEMD-160 hash

00010966776006953D5567439E5E39F86A0D273BEE

6 - Perform SHA-256 hash on the extended RIPEMD-160 result

445C7A8007A93D8733188288BB320A8FE2DEBD2AE1B47F0F50BC10BAE845C094

7 - Perform SHA-256 hash on the result of the previous SHA-256 hash

D61967F63C7DD183914A4AE452C9F6AD5D462CE3D277798075B107615C1A8A30

BITCOIN ADDRESS - 3

8 - Take the first 4 bytes of the second SHA-256 hash. This is the address checksum

D61967F6

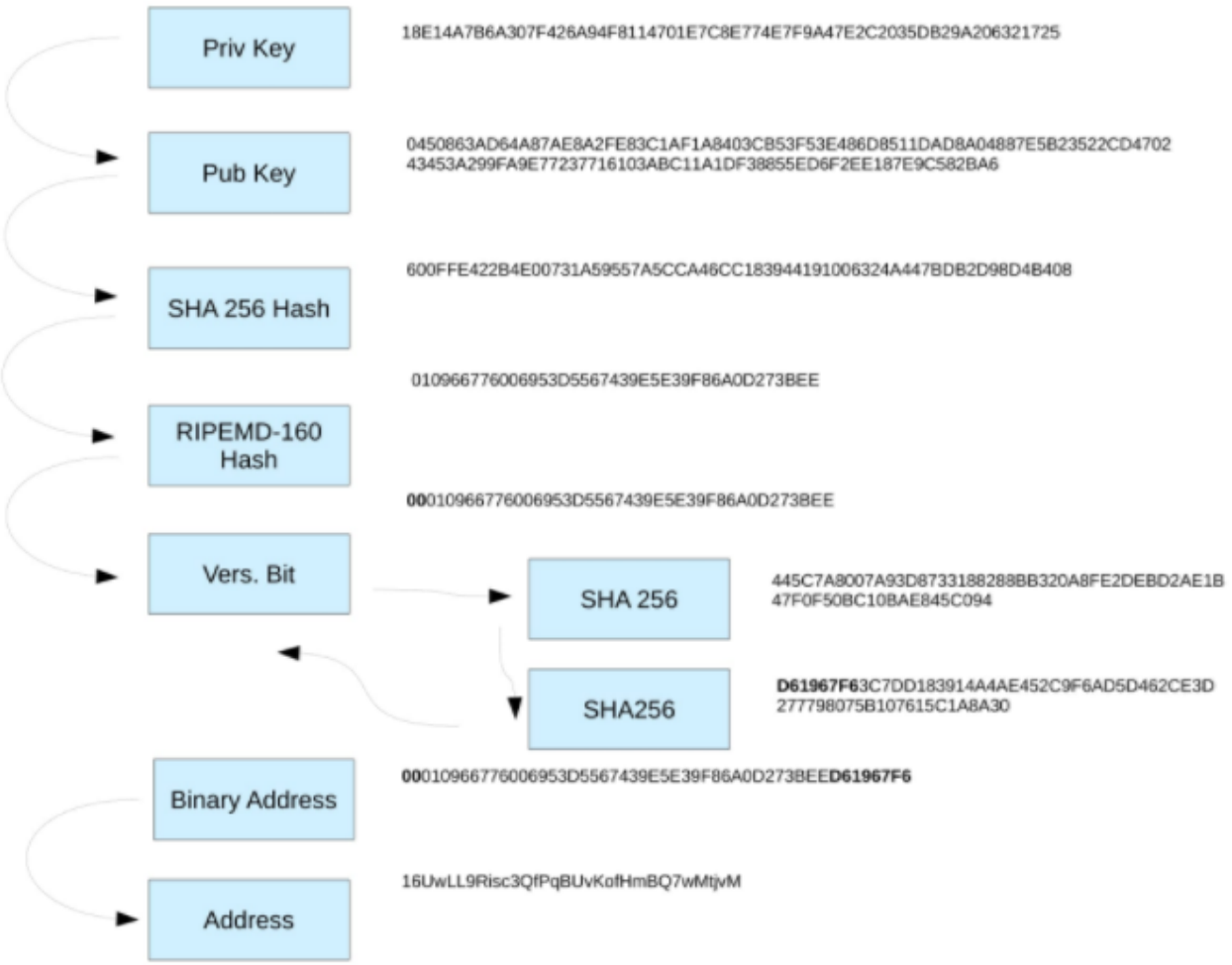
9 - Add the 4 checksum bytes from stage 7 at the end of extended RIPEMD-160 hash from stage 4. This is the 25-byte binary Bitcoin Address.

00010966776006953D5567439E5E39F86A0D273BEED61967F6

10 - Convert the result from a byte string into a base58 string using Base58Check encoding. This is the most commonly used Bitcoin Address format

16UwLL9Risc3QfPqBUvKofHmBQ7wMtjvM

BITCOIN ADDRESS DIAGRAM



Obtained from:

https://blockgeeks.com/guides/blockchain-address-101/

DIGITAL SIGNATURE

You can use asymmetric keys and public key cryptography to sign messages similar to signing a document with your unique signature. Remember, each user has a public and private key. The private key is kept secret.

Signing requires creating a message or digest of *target data* using a hash algorithm. The digest or fingerprint is then encrypted with your private key. You then send the target data, encrypted digest, and public key to someone else.

At the destination, the person decrypts the fingerprint to uncover the original hash. The data is then rehashed and compared. If identical, you confirm who sent the data and that the data is untampered.

DIGITAL SIGNATURE CODE

create JSON object (1)

create empty blockchain (2)

create stream for reading JSON from file (3)

associate reader with input stream (4)

reflect type: arraylist of blocks (5)

read blocks from file (6)

Initialize blockchain with blocks (7)

```
Gson gson = new Gson();                                          1

ArrayList<Block> blocks=null;                                    2

try  {
      BufferedReader br = new BufferedReader(                    3

                  new FileReader("Output.json"));                4


                  java.lang.reflect.Type blockType =             5
                        new TypeToken<ArrayList<Block>>()
                        {}.getType();
      blocks = gson.fromJson(br,blockType);                      6

catch (IOException e) {
      e.printStackTrace();
}

Blockchain blockChain=new Blockchain(blocks);                    7

Block.SetSequence(blocks.size());

TXOutput.count=blockChain.GetOutputCount();
```

DIGITAL SIGNATURE CODE - 2

create JSON object (1)

create empty blockchain (2)

create stream for reading JSON from file (3)

associate reader with input stream (4)

reflect type: arraylist of blocks (5)

read blocks from file (6)

Initialize blockchain with blocks (7)

```
Gson gson = new Gson();                                          1

ArrayList<Block> blocks=null;                                    2

try  {
      BufferedReader br = new BufferedReader(                    3

                  new FileReader("Output.json"));                4


                  java.lang.reflect.Type blockType =             5
                        new TypeToken<ArrayList<Block>>()
                        {}.getType();
      blocks = gson.fromJson(br,blockType);                      6

catch (IOException e) {
      e.printStackTrace();
}

Blockchain blockChain=new Blockchain(blocks);                    7

Block.SetSequence(blocks.size());

TXOutput.count=blockChain.GetOutputCount();
```

Lab 6- Keypairs

KEYPAIR

In this lab, you will create a keypair. The keypair will be used to sign and send the data (transaction). You will then receive and verify the data (transaction). This is a simulation of sending and receiving a transaction in a blockchain.

Create a new application with a block class. Here are the data members of the block class:

keyPair:KeyPair

signature: Byte[]

data: String

timestamp: String

KEYPAIR - 2

Created a nested class, TransactionData, within the Block class. Here are the members of the nested class:

- rawPublicKey: Byte[]
- signature: Byte[]
- data: String
- timeStamp: String

Create a member function call GetKeyPair that returns a KeyPair. In the function:

- With KeyPairGenerator class, call getInstance to create an instance from the "RSA" algorithm.
- Initialize the KeyPairGenerator instance for a 1024bit key size.
- Use the KeyPairGenerator to generate the keypair.

KEYPAIR - 3

Create a function to sign the block data. Here is the code:

```
public byte[] GetSignature() throws UnsupportedEncodingException, NoSuchAlgorithmException,
            InvalidKeyException, SignatureException {
    // Concatenate block data
    byte[] blockData = (data+timeStamp).getBytes("UTF8");
    //Create Signature object with "MD5WithRSA"
    Signature sig = Signature.getInstance("SHA1withRSA");
    sig.initSign(keyPair.getPrivate());
    // Update Signature with Block
    sig.update(blockData);
    // Get signature
    return sig.sign();
}
```

KEYPAIR - 4

Create the constructor for the Block class, which accepts *data* as the parameter. Data is a string. Here are the steps of the constructor:

- Initialize the data member with the parameter from the constructor.
- Call GetKeyPair function to initialize the keyPair data member.

Create function, GetRawPublicKey, that extracts the public key from keyPair as bytes.

- Call getPublic to derive the public key from the KeyPair of the class.
- Call getEncode to return the bytes from the public key.

KEYPAIR - 5

Next create a function, to send (persist) the transaction: SendDataAsFile.

Initialize TransactionData:

- Assign TransactionData.rawPublicKey with GetRawPublicKey function.
- Assign TransactionData.signature with GetSignature function.
- Assign TransactionData.data with data surrounding class.
- Assign TransactionData.timeStamp with timeStamp.

Write data to a JSON file. Here is the code.

```
try (Writer writer = new FileWriter("Output.json")) {          // Create file object for json
    Gson gson = new GsonBuilder().create();                    // Create file
    gson.toJson(transactionData, writer);
    // Write blockchain to file as json
}
catch (Exception e) {
e.printStackTrace();
}
```

KEYPAIR - 6

Create a function, to receive (rehydrate) the transaction: ReceiveAndVerify.

Create an instance of the transactionData.

Here is the detailed code to read the JSON file.

```
try  {
        Gson gson = new GsonBuilder().create();
        BufferedReader br = new BufferedReader(          // Create a new reader
        new FileReader("Output.json"));                  // Associate reader to input file
        // Reflect type for arraylist of blocks
        java.lang.reflect.Type dataType =
                new TypeToken<TransactionData>()
                {}.getType();
        transactionData = gson.fromJson(br,dataType);
} catch (IOException e) {
e.printStackTrace();
}
```

KEYPAIR - 7

Recreate the publicKey from the transactionData read from the JSON file:

- Create an instance of KeyFactory with the getInstance member method. Initialize for the “RSA” algorithm.
- Create a EncodedKeySpec object from X509EncodedKeySpec. Initialize the object with the transactionData.rawPublicKey.
- Recreate the publicKey with the KeyFactory instance. Call KeyFactory.generatePublic. The sole parameter is the EncodedKeySpec object.

KEYPAIR - 8

Recreate the signature:

- Create an instance of the Signature object for "SHA1withRSA".
- Initialize the signature object to used the publicKey with Signature.InitVerify.
- Update the signature object with the transaction data.

```
sig.update((transactionData.data+transactionData.timeStamp).getBytes("UTF8"));
```

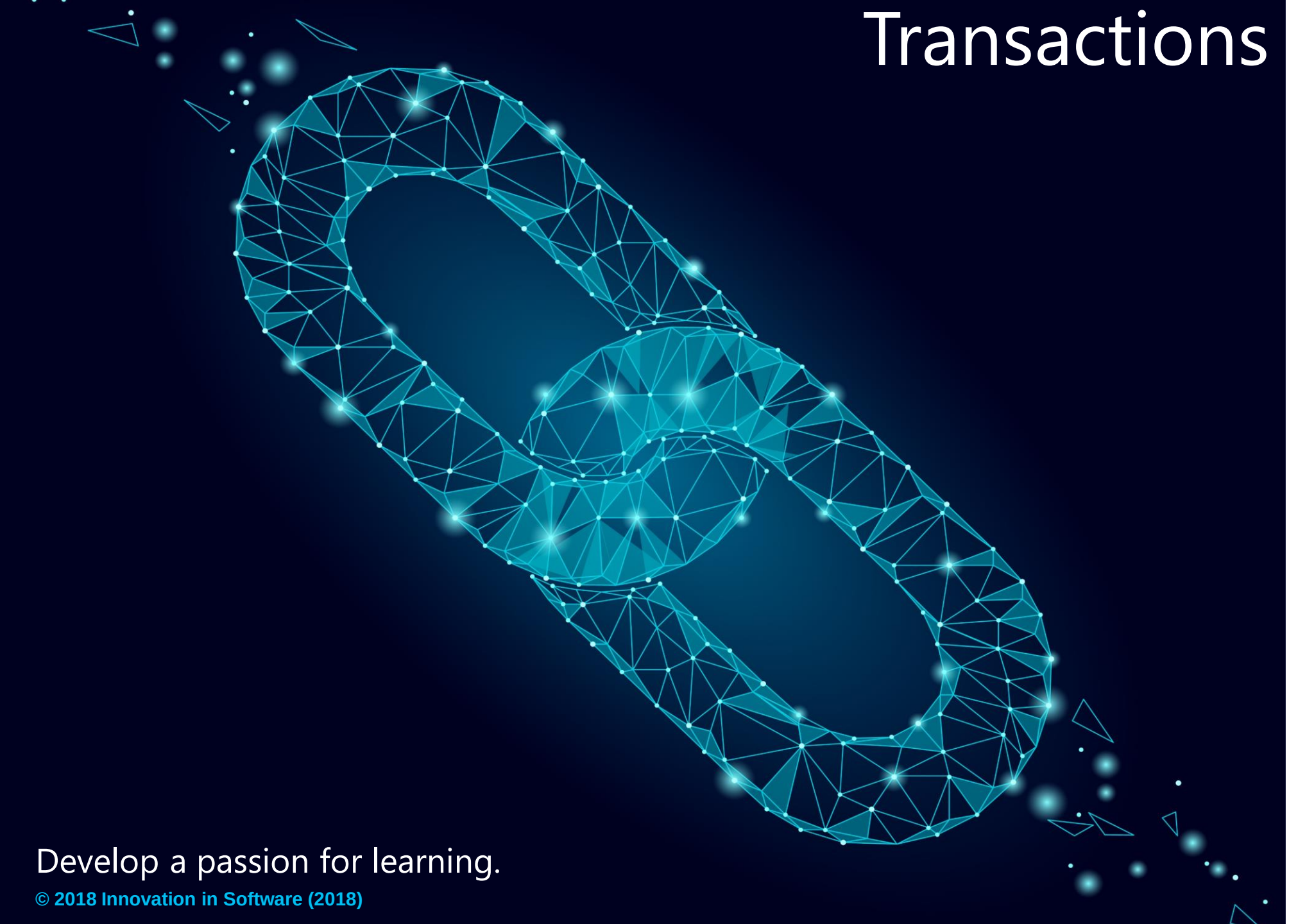
Verify the signature with the Signature.Verify method. The input parameter is the signature read from the transaction: transactionData.signature. If verified, display message that transaction is okay. If not, display message not verified.

KEYPAIR - 9

Here is code to test the ability to sign transactions.

```
public static void main(String [] args) throws InvalidKeyException,
        UnsupportedEncodingException,
        NoSuchAlgorithmException,
        SignatureException, InvalidKeySpecException {

    Block block=new Block("a");

    block.SendDataAsFile();

    block.ReceiveAndVerify();

}
```

Transactions



Develop a passion for learning.

WALLET

We just finished the conversation about Wallets. As mentioned, wallets don't actually contain value; but hold keys. Private keys are used to sign transactions, while public keys are sent as an address.

The value of a wallet is the net value of all of the transactions. What does that mean?

In this module, we will delve into the intricacies of transaction input, transaction outputs, and unspent transaction output (UTXO).

TRANSACTIONS

Transactions are the *gas* that makes blockchains run. That is a subtle nod towards Etheruum. The distributed ledger is a ledger of transactions. Similar to debits and credits, blockchain has transaction outputs and transaction inputs.

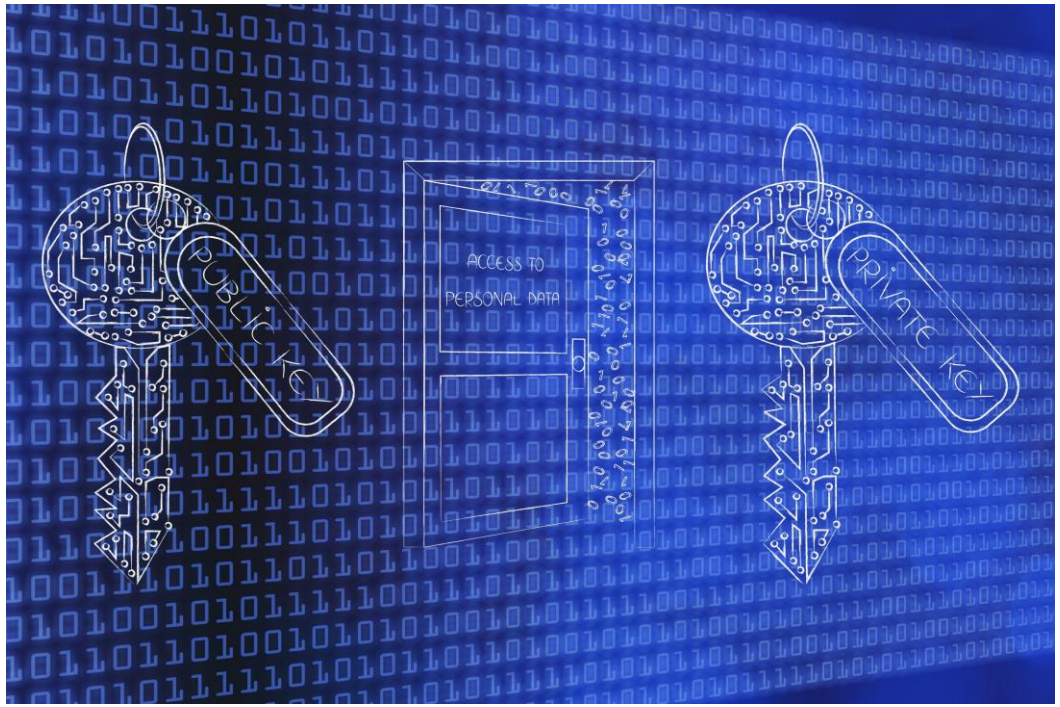
Transactions represent the transfer of value from one user to another. For that reason, there must be a mechanism for sending transactions in a secure and reliable manner.

POP QUIZ: WALLET KEYS

What are the three benefits to signing transactions?

10 MINUTES

PUBLIC PRIVATE KEY PAIR



When sending a transaction, we will sign a transaction with our private key. This prevents someone else from spending our hard owned money. The public key is sent with the transaction to allow anyone to authenticate our identity.

TRANSACTION TYPES

Two types of general purpose transactions:

- Transaction input (vin) – a reference to a previous output transaction (unspent) proven that money is available to be sent. Similar to a bank debit.
- Transaction output (vout) – amounts sent and the recipient. Similar to a bank credit.

Importantly, each transaction has a digital signature, where anyone can verify the owner.

TRANSACTION TYPES

Transaction are the equivalent of a blockchain money order. After receiving a transaction, you cannot spend the partial value of a transaction. You need to fully cash in the transaction and then send any remaining value (i.e., the change) back to yourself.

TRANSACTION OUTPUT

```
structure TransactionOutput

     Value // something of value
     szSender
```

You cannot spend value that you do not have! That does not work well unless you have overdraft protection. Guess what - Blockchain does not have overdraft protection!

Where do you get value in your wallet? Someone else must send you value in the form of a output transaction. You can spend the value from these transactions.

The balance in your account is the total value of unspent output transactions (UTXO). You cannot spend more than the available in the UTXO.

When you send value to someone, you must prove you have the available funds. That is done by referencing UTXO in the Transaction Input.

SATOSHI

For bitcoin, the value for a transaction is in satoshis, which is the smallest amount available in the Bitcoin blockchain. A satoshi is equivalent to a cent in USD.

A satoshi is a hundred millionth of a bitcoin, which is .00000001 bitcoin.

FULL LIST – BITCOIN DENOMINATION

Unit	Abbreviation	Decimal (BTC)	Alternate names	Info
Algorithmic maximum		20,999,999.9769		Calculation
tam-bitcoin		2,814,749.76710656		1,0000,0000 tonal
mega-bitcoin	MBTC	1,000,000		Rare in context
kilo-bitcoin	kBTC	1,000		Rare in context
hecto-bitcoin	hBTC	100		Rare
Initial block subsidy		50		Until block 210000[1]
bong-bitcoin	ᵇTBC	42.94967296		1,0000 tonal
Current block subsidy		12.5	block	As of block 420000
deca-bitcoin	daBTC	10		Rare
mill-bitcoin	ᵐTBC	2.68435456		1000 tonal
bitcoin	BTC	1	coin	SI base unit
san-bitcoin	ˢTBC	0.16777216		100 tonal
deci-bitcoin	dBTC	0.1		Rare
ton-bitcoin	ᵗTBC	0.01048576		10 tonal
centi-bitcoin	cBTC	0.01	bitcent	Formerly frequent[2]
milli-bitcoin	mBTC	0.001	millibit, millie	Occasional
bitcoin	TBC	0.00065536		Tonal base unit
bitcoin-ton	TBCᵗ	0.00004096		0.1 tonal
bitcoin-san	TBCˢ	0.00000256		0.01 tonal
micro-bitcoin	µBTC	0.000001	bit	Frequent
bitcoin-mill	TBCᵐ	0.00000016		0.001 tonal
		0.0000001	finney[3]	
bitcoin-bong	TBCᵇ	0.00000001		0.0001 tonal
	sat	0.00000001	satoshi	Blockchain value
	msat	0.00000000001	millisatoshi[4]	Payment channel value

https://en.bitcoin.it/wiki/Units

TRANSACTION INPUT

```
structure TransactionInput

     szID            // id of transaction
     nIndexOutput    // index of output transaction
     szOwner         // owner of output transaction
```

The transaction input references an transaction output. The transaction output, which is the money order equivalent, referenced in transaction input is consumed in *whole*.

The sender of the transaction output and the owner of the transaction input must be the same. You are not allowed to spend someone else's value– despite how much fun that may be.

TRANSACTION

```
structure Transaction

     szId
     vin        // array of transaction inputs
     vout       // array of transaction outputs
```

A transaction is a combination of transaction inputs and transaction outputs.

The transaction inputs reference the value being used in the transaction and reference previous outputs.

The transaction outputs is the value being sent. This may include sent value back to the sender.

Blocks of the blockchain contain an array of transactions collected for a transaction pool.

COINBASE TRANSACTION

What came first – the chicken or the egg? This is a well-known conundrum that also applies to blockchain. You need unspent output transactions (UTXO) to fund input transactions. However, where did the first unspent transfer originate?

Coinbase transactions solve this problem and are not associated with a matching UTXO transaction. Coinbase transactions are used to reward miners and increase the money supply. A coinbase transaction is where you create coins from nowhere and increase the money supply.

TRANSACTION PROPAGATION

I am a user that wants to send a transaction. If I have a wallet, that task is made much easier and is simply a function of the wallet.

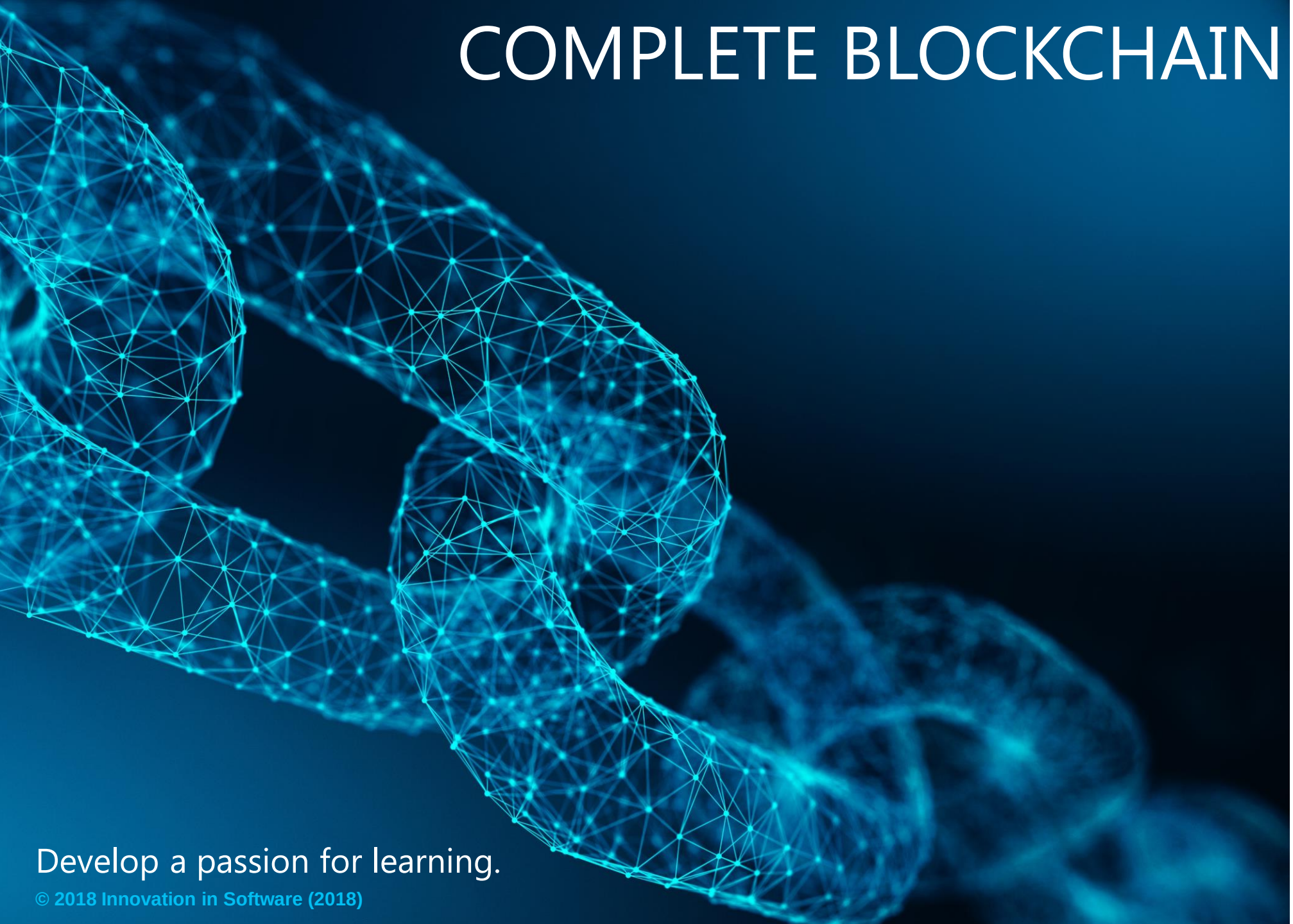
The wallet could send the transaction to a particular node. The node receiving the transaction would then broadcast the transactions to the remaining nodes. This process is called *flooding*. Within a few seconds every node will have access to the transaction.

TRANSACTION FEES

The transaction fee awarded miners is the difference of the vins and vouts transactions. Essentially, unclaimed change of a transaction belongs to the miner.

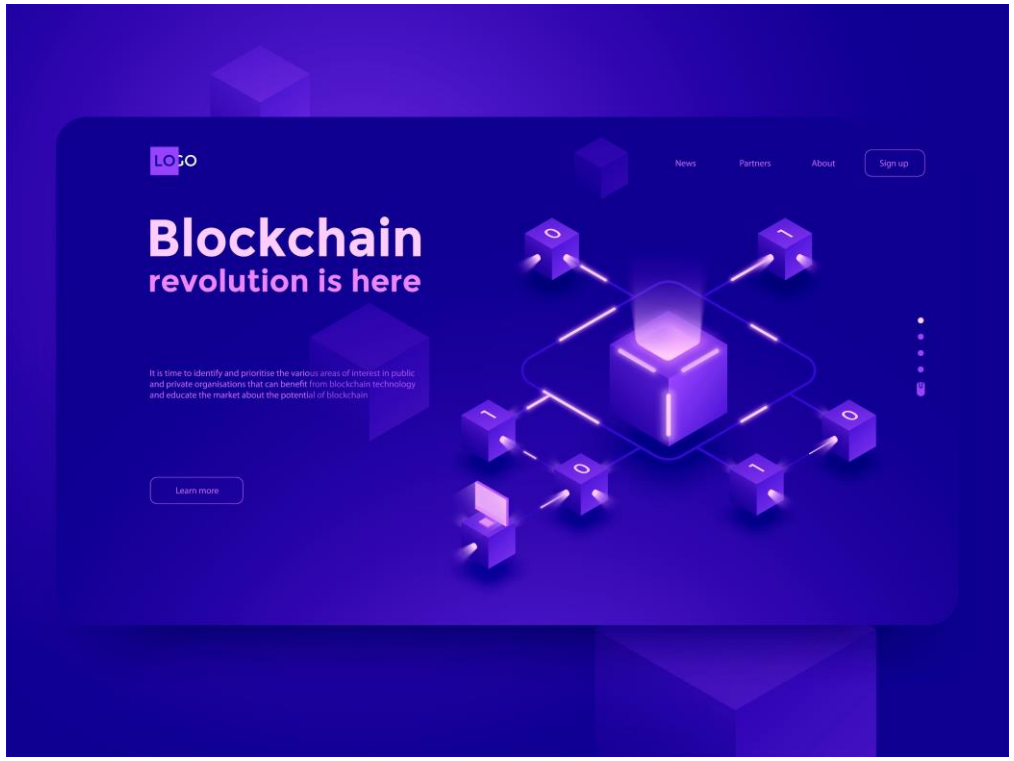
Prioritization. Transactions can be prioritized when added to a block. They are typically prioritized based on transaction fees. Naturally, transactions that offer higher fees receive higher priority. Having a prioritization algorithm for transactions is optional for a blockchain, including establishing the criteria.

COMPLETE BLOCKCHAIN



Develop a passion for learning.

COMPLETE BLOCKCHAIN



In this module you will create a complete blockchain:

- Key pairs
- Transactions
- Mining
- Account balances
- Security
- Persistence
- And much more

HANDS ON EXPERIENCE

This project will take around three hours. Pair programming is recommended for this lab. This is not one person typing and another watching; but two active participants implementing and researching, when necessary, the solution.

Feel free to take breaks during this lab as necessary.

CLASSES

You will create several classes:

- TXOutput: output transactions
- TXInput: input transactions
- Block: blockchain block and related functionality
- BlockChain: wrapper for blockchain and related functions
- Proof Of Work: proof of work algorithm
- Transaction: amalgamation of input and output transactions
- Utilities: Helper methods

RECOMMENDATIONS

- Paired programming
- Use blockchain terminology
- Self documenting code
- Use exiting code when possible
- Build often

UTILITIES CLASS

Create two methods in the Utilities class.

- **Hash2HexString.** Hash2HexString converts a hash to a displayable string. This method has one parameter, which is a byte array and returns a string.
- **SetHash**. SetHash converts a string to a hash and returns that hash as a displayable string. This method accepts a string as a parameter and returns a string.

TXOUTPUT CLASS

This class is for output transactions (i.e., sending money to someone).

- There are three fields:
  - coin value for transaction
  - person receiving the transaction
  - block transaction id
- Create a constructor that initializes the value and person fields. Transaction id is implemented as a count of total transactions. This is the row height.
- Create separate accessor methods to return the coin value, person, and transaction id.
- CanUnlockOutputWith. This method confirms the identity of the person. There is one string parameter and returns a Boolean. The string parameter is the expected identity. Returns true if user identifier matches the person of this transaction.

TXINPUT CLASS

This class is for input transactions. This is spent money from your perspective.

- There are two fields:

  transaction id of output id

  the person doing the transaction

- Create a constructor that initializes the two fields with the input parameters.
- CanUnlockOutputWith. This method confirms the identity of the person. There is one string parameter and returns a Boolean. The string parameter is the expected identity. Returns true if user identifier matches the person of this transaction.
- Create two accessor methods that return separately the transaction id and the person identity.

TRANSACTION CLASS

The Transaction class is a wrapper for the inputs and outputs of a transaction (debits and credits).

- There are three fields:

  transaction id

  array of input transactions

  array of output transactions

- Create a constructor that initializes the array of input and output transactions to constructor parameters.
- SetId. Initializes the transaction id to a count of transactions. Increments the count.
- NewCoinbaseTX. This method creates a coin base transaction. There are two parameters: the destination and arbitrary message. Returns a transaction.

  1. Initialize a TXInput array with one transaction. This is a empty transaction. Why?

  TXInput[] txInput= {new TXInput("", -1, message)};

  2. Initialize a TXOutput array with one transaction. This this should have a value of 100.
  3. Create a new transaction
  4. Return transaction

BLOCK CLASS

The Block class:

Here are the fields:

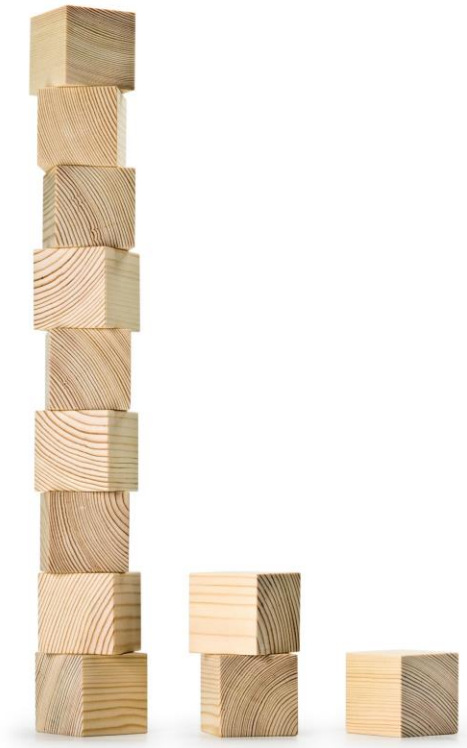
- Array of transactions
- Previous block hash
- Current block hash
- Block height
- Time stamp
- Static count

BLOCK CLASS - 2

Create a constructor. Initialize the various fields of the block class. Initialize the previousHash and transactions with constructor parameters. Calculate the remaining fields:

1. Initialize the time stamp to the current date
2. Calculate blockHeight by incrementing a count (static sequence data member)
3. Calculate the current hash for the block using Utilities.SetHash
4. Create an instance of the Proof of Work object. Pass in the current block as the parameter and a difficulty of 2.
5. Mine the current block using the Proof of Work object.

BLOCK CLASS - 3



- Create assessor methods for the current hash, previous hash, and blockId. Create a mutator for the sequence field.
- NewBlock. Accepts a string and array of transactions as parameters. Returns a Block. NewBlock is a static method. In the method create a new instance of a Block class. Return the new block.
- HashTransactions. Concatenate the transactions in the block and hash the results. Extra credit: create a Merkle tree from the transactions. Integrate the Merkle root into the class.
- DisplayTransactions: First, iterate and display the output transactions. Second, iterate and display input transactions.
- GetData. Return the concatenations of the previous hash+hash of the transactions+timestamp.

PROOFOFWORK CLASS

The Proof of Work class is used to mine blockchain blocks.

- Here are the fields:
  - difficulty
  - target block to mine
  - target for mining, such as “000”
  - Nonce
  - Maximum nonce value
- Create a constructor. Initialize target block and difficulty with constructor parameters. Assign mask as a series of zeros based on difficulty.
- **PrepareData.** PrepareData method that returns the block header (Block.GetData)+nonce. Nonce is incremented before returning from function.
- **Run.** Mine the block stored in the class.

BLOCKCHAIN CLASS

The Blockchain class manages an array of Blocks.

There are two constructors:

- The first constructor has one parameter – a list of blocks. Initialize Blockchain list with this parameter.
- The second constructor has one parameter, which is list of transactions. Create the genesis block with the transactions. Call the method CreateGenesisBlock.
- AddBlock. Adds a new block to the blockchain. The only parameter is an array of transactions. Returns nothing. Create a new block using Block.NewBlock with the transactions. Add the block to the list of blocks. Save the block to a Json file: BlockChain.SaveChain.
- CreateGenesisBlock. Create the first block of the blockchain. This method has a single parameter, which is an array of transactions.
    1. Create an array of transactions initialized with a single transaction, which is a coinbase transaction.
    2. Create a new block.
    3. Add the block to the blockchain.

BLOCKCHAIN CLASS - 2

- **ContainsInInputTx**. This method finds unused transactions. The only parameter is a TXOutput. Search for the output transaction in the list of input transactions.
  - Iterate the blocks of the blockchain.
  - Iterate the transactions of the block.
  - If the output transaction is assigned to a input transaction, return true. Otherwise, return false.
- **FindUTXO** . This method is finds unspent transactions for a particular person. The method has a single parameter, which is the person. Returns the UTXO for that person.
  1. Create an empty list of TXOutput transactions. This is the list of UTXO.
  2. Iterate the blocks of the blockchain.
  3. Iterate the transactions of each block.
  4. Iterate the array of TXOutput for each transaction.
  5. If ContainsInInputTx returns false, add to the UTXO list.

BLOCKCHAIN CLASS - 3

- NewOutputTransaction. This function sends money from one person to another. You are spending UTXO. There are three parameters: who is sending the money, who is receiving the money, and the amount being sent.
  - Create an empty list of TXInputs and TXOutputs.
  - Create an empty array of TXOutput, which will hold the UTXO. Initialize with FindUTXO.
  - Iterate the list of UTXO transactions. Keep a running total of the transaction value.
  - Create a new TXInput transaction for each UTXO transaction. Add to the TXInputs list. Break when the total is greater than the amount being spent.
  - Create a new TXOutput transaction for the total amount.
  - If there is *change*, create a new TXOutput transaction for that amount. Send to yourself.
  - Create a new Transaction from the list of TXInputs and TXOutputs. Set the Transaction id.
  - Return the Transaction.

BLOCKCHAIN CLASS - 4

- **GetBalance**. This method returns the balance for a person. There is one parameter, which identifies the person. Iterate and total the UTXO. Return the total.
- **SaveChain**. Here is the code to save the blockchain.

```
public void SaveChain( ) throws IOException {
try (Writer writer = new FileWriter("Output.json")) {
    Gson gson = new GsonBuilder().create();
    gson.toJson(blocks, writer)
}
```

BLOCKCHAIN CLASS - 5

ReadChain. Here is the code to read the blockchain.

```
public static Blockchain ReadChain( ) throws IOException {
Gson gson = new Gson();
ArrayList<Block> blocks=null;
try  {
        BufferedReader br = new BufferedReader(
        new FileReader("Output.json"));
        java.lang.reflect.Type blockType =
        new TypeToken<ArrayList<Block>>() {}.getType();
        blocks = gson.fromJson(br,blockType);
}       catch (IOException e) {
                e.printStackTrace();
}
Blockchain blockChain=new Blockchain(blocks);
Block.SetSequence(blocks.size());
TXOutput.count=blockChain.GetOutputCount();
return blockChain;
}
```

TEST BLOCKCHAIN

This is sample code to test your private blockchain.

```
public static void main(String [] args) throws Exception {
//CreateChain();
ReadBlockChain();
}
public static void CreateChain() throws Exception {

Transaction transaction1=Transaction.NewCoinbaseTX("Donis", "");
Blockchain blockChain=new Blockchain(new Transaction []
{transaction1});

System.out.println("Block 1\n\n");

Transaction transaction2=blockChain.NewUTXOTransaction(
                    "Donis", "Bob", 50);
blockChain.AddBlock(new Transaction[]{transaction2});
```

TEST BLOCKCHAIN - 2

This slide is a continuation of the previous slide.

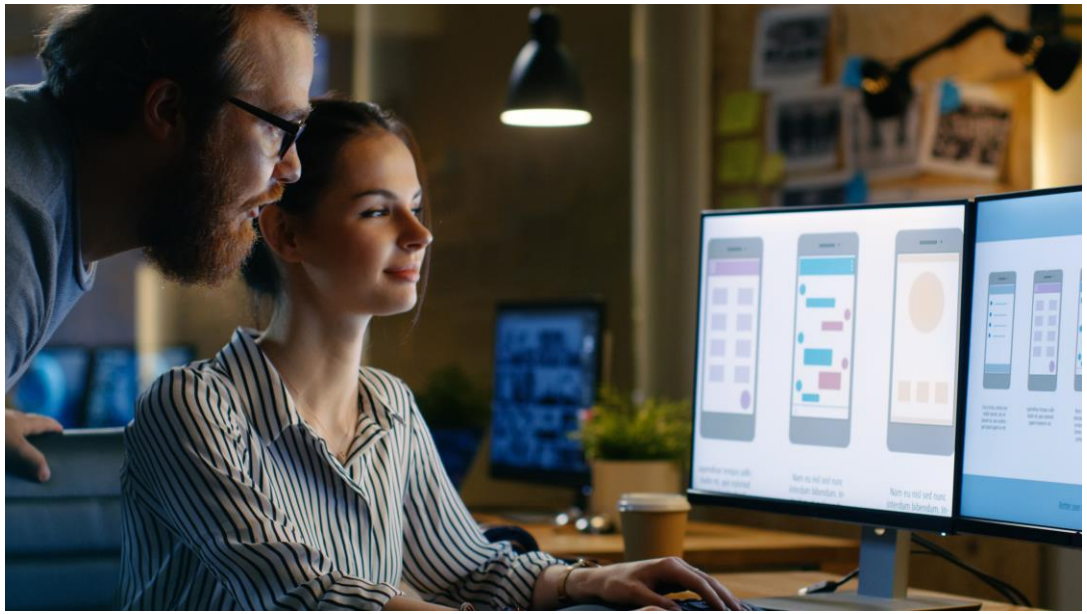
```
Transaction transaction3=blockChain.NewUTXOTransaction("Donis", "Fred", 25);
blockChain.AddBlock(new Transaction[]{transaction3});

Transaction transaction4=blockChain.NewUTXOTransaction("Bob", "Fred", 12);
blockChain.AddBlock(new Transaction[]{transaction4});

blockChain.SaveChain();
blockChain.DisplayChain();

int balance=blockChain.GetBalance("Donis");
System.out.println("The balance is Donis "+balance);

balance=blockChain.GetBalance("Bob");
System.out.println("The balance is Bob "+balance);

balance=blockChain.GetBalance("Fred");
System.out.println("The balance is Fred "+balance);
}
```

TEST BLOCKCHAIN - 3

This is a continuation of the previous slide.

```
Transaction transaction3=blockChain.NewUTXOTransaction("Donis", "Fred", 25);
blockChain.AddBlock(new Transaction[]{transaction3});


Transaction transaction4=blockChain.NewUTXOTransaction("Bob", "Fred", 12);
blockChain.AddBlock(new Transaction[]{transaction4});

blockChain.SaveChain();
blockChain.DisplayChain();

int balance=blockChain.GetBalance("Donis");
System.out.println("The balance is Donis "+balance);

balance=blockChain.GetBalance("Bob");
System.out.println("The balance is Bob "+balance);

balance=blockChain.GetBalance("Fred");
System.out.println("The balance is Fred "+balance);

}
```

TEST BLOCKCHAIN - 4



This is a continuation of the previous slide.

```
Transaction transaction3=blockChain.NewUTXOTransaction("Donis", "Fred", 25);
blockChain.AddBlock(new Transaction[]{transaction3});

Transaction transaction4=blockChain.NewUTXOTransaction("Bob", "Fred", 12);
blockChain.AddBlock(new Transaction[]{transaction4});

blockChain.SaveChain();
blockChain.DisplayChain();

int balance=blockChain.GetBalance("Donis");
System.out.println("The balance is Donis "+balance);

balance=blockChain.GetBalance("Bob");
System.out.println("The balance is Bob "+balance);

balance=blockChain.GetBalance("Fred");
System.out.println("The balance is Fred "+balance);

}
```