Bitcoin

Introduction

- Bitcoin has started a revolution with the introduction of the very first fully **decentralized digital currency**, and one that has proven to be extremely secure and stable.
  - sparked a great interest in academic and industrial research
  - introduced many new research areas.
- Since introduction in 2008, bitcoin has gained much popularity
- **Most successful digital currency** in the world with billions of dollars invested in it.
- Bitcoin is built on decades of research in the field of **cryptography**, **digital cash**, and **distributed computing**.

Introduction

- Early proposals to create digital cash go as far back as the early 1980s.
- In 1982, *David Chaum* proposed a scheme that used blind signatures to build untraceable digital currency.
  - Bank would issue digital money by signing a blind and random serial number presented to it by the user.
  - User could then use the digital token signed by the bank as currency.
  - Limitation in this scheme was that the bank had to keep track of all used serial numbers.
  - This was a central system by design and required to be trusted by the users.

Introduction

- Later on in 1990, *David Chaum* proposed a refined version named **e-cash** that not only used **blinded signature**, but also some **private identification data** to craft a message that was then sent to the bank.
  - This scheme allowed the **detection of double spending** but did not prevent it.
  - If the same token was used at two different locations, then the identity of the double spender would be revealed.
  - e-cash could only represent a fixed amount of money.
- *Adam Back's* **hashcash**, introduced in 1997, was originally proposed to thwart e-mail spam.
  - Solve a computational puzzle that was easy to verify but comparatively difficult to compute.

Introduction

- **B-money** was proposed by *Wei Dai* in 1998, which introduced the idea of using **Proof of Work** to create money.
  - **Major weakness** in the system was that an **adversary with higher computational power** could generate **unsolicited money** without allowing the network to adjust to an appropriate difficulty level.
  - System lacked details on the consensus mechanism between nodes and some security issues such as Sybil attacks were also not addressed.
- At the same time, *Nick Szabo* introduced the concept of **BitGold**,
  - Based on the Proof of Work mechanism but had the same problems as b-money with the exception that the network difficulty level was adjustable

Introduction

- *Tomas Sander* and *Ammon TaShama* introduced an **e-cash** scheme in 1999
  - For the first time, used **Merkle trees** to represent coins and **zero knowledge proofs** to prove the possession of coins.
  - Central bank was required that kept a record of all used serial numbers.
  - Scheme allowed users to be **fully anonymous** although at a computational cost.
- **RPOW (Reusable Proof of Work)** was introduced by *Hal Finney* in 2004 and used the hashcash scheme by *Adam Back* as a proof of computational resources spent to create the money.
  - This was also a central system that kept a central database to keep track of all used POW tokens.
  - This was an online system that used remote attestation made possible by a **trusted computing platform** (**TPM hardware**).

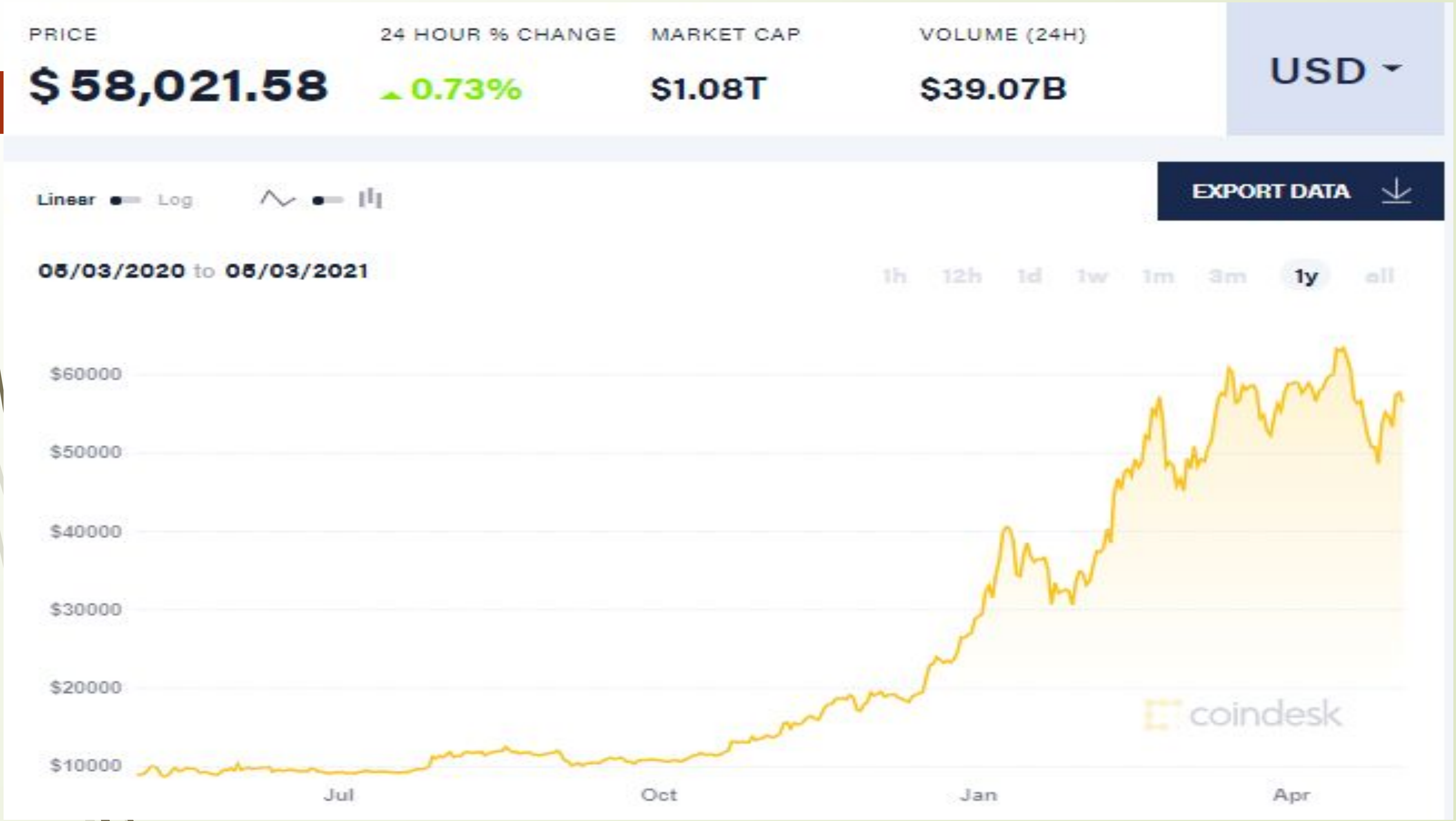
Bitcoin

- In 2008, a paper on bitcoin, ***Bitcoin: A Peer-to-Peer Electronic Cash System*** was written by *Satoshi Nakamoto*.
  - First key idea introduced was that purely **peer-to-peer electronic cash** that does need an intermediary bank to transfer payments between peers.
- Bitcoin is built on cryptographic research in **Merkle trees**, **hash functions**, **public key cryptography**, and **digital signatures**.
- BitGold, b-money, hashcash, and cryptographic time stamping provided the foundations for bitcoin invention.
  - All these technologies are cleverly combined in bitcoin to create the world's first decentralized currency.
- Key issue that has been addressed in bitcoin is an **elegant solution to the Byzantine Generals problem** along with **a practical solution of the double-spend problem**.

Milestones

- “Satoshi Nakamoto” created the reference implementation that began with a **Genesis Block of 50 coins**
- **2008**
  - **August 18** Domain name "bitcoin.org" registered[1].
  - **October 31** Bitcoin design paper published
  - **November 09** Bitcoin project registered at SourceForge.net
- **2009**
  - **January 3** Genesis block established at 18:15:05 GMT
  - **January 9** Bitcoin v0.1 released and announced on the cryptography mailing list
  - **January 12** First Bitcoin transaction, in block 170 from Satoshi to Hal Finney

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



PRICE $58,021.58

24 HOUR % CHANGE ▲0.73%

MARKET CAP $1.08T

VOLUME (24H) $39.07B

USD

EXPORT DATA

Linear ● Log

05/03/2020 to 05/03/2021

Bitcoin

- Regulation of bitcoin is a controversial subject
  - Law enforcement agencies and governments are proposing various regulations to control it
- BitLicense issued by New York's state department of financial services.
  - This is a license issued to businesses that perform activities related to virtual currencies.
- Growth of Bitcoin is also due to so-called Network Effect.
  - Also called demand-side economies of scale, it is a concept that basically means more users who use the network, the more valuable it becomes.
- Over time, an exponential increase has been seen in the Bitcoin network growth.
  - **1 Bitcoin = 43,00,858.79 Indian Rupee**
  - **1 Bitcoin= 58033.80 United States Dollar**

Bitcoin definition

- Bitcoin can be defined in various ways;
  - it's a protocol, a digital currency, and a platform.
  - It is a combination of peer-to-peer network, protocols, and software that facilitate the creation and usage of the digital currency named bitcoin.
- Bitcoin with a capital *B* is used to refer to the Bitcoin protocol
- bitcoin with a lowercase *b* is used to refer to bitcoin, the currency.
- Nodes in this peer-to-peer network talk to each other using the Bitcoin protocol.

Bitcoin vs. bitcoins

- **Bitcoin** is the system
- **bitcoins** are the units

bitcoin
ACCEPTED HERE

Bitcoin definition

- Decentralization of currency was made possible for the first time with the invention of bitcoin
- Double spending problem was solved in an elegant and ingenious way in bitcoin.

Keys and addresses

- Elliptic curve cryptography is used to generate public and private key pairs in the Bitcoin network.
- Bitcoin address is created by taking the corresponding public key of a private key and hashing it twice,
  - First with the SHA256 algorithm
  - Then with RIPEMD160.

Keys and addresses

- Resultant 160-bit hash is then prefixed with a version number and finally encoded with a Base58Check encoding scheme.
  - **Base58** is a group of binary-to-text encoding schemes used to represent large integers as alphanumeric text, introduced by Satoshi Nakamoto for use with Bitcoin.
  - It is similar to Base64 but has been modified to avoid both non-alphanumeric characters and letters which might look ambiguous when printed.
- The bitcoin addresses are 26-35 characters long and begin with digit 1 or 3.

Keys and addresses

- Typical bitcoin address looks like

  1ANAguGG8bikEv2fYsTBnRUmx7QUcK58wt

- This is also commonly encoded in a QR code for easy sharing

Keys and addresses

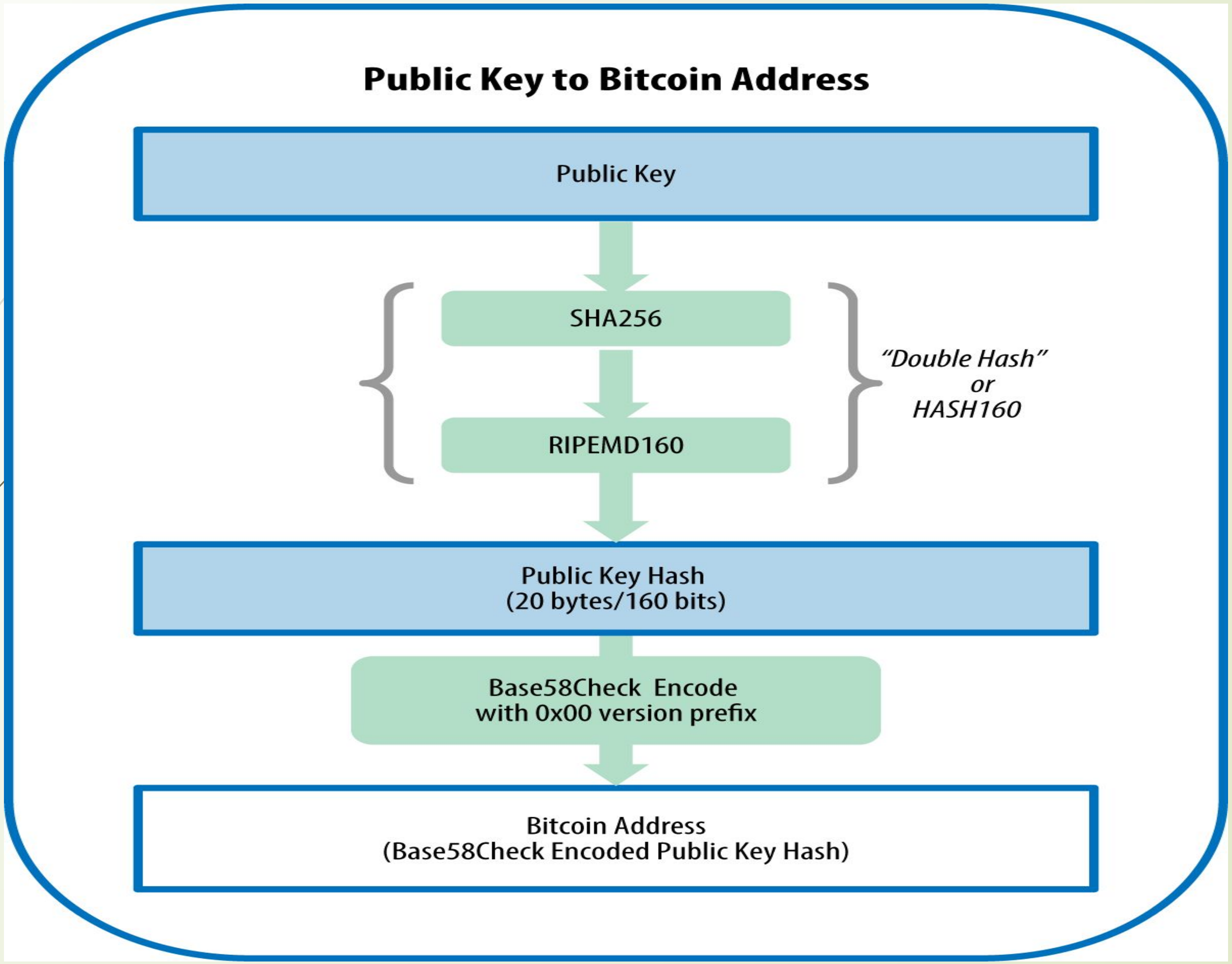
- Currently, there are two types of addresses, the commonly used
  - Pay-to-PubKey-Hash (Pay-to-Public-Key-Hash, P2PKH)
  - Pay to script hash (P2SH)
- In the early days, bitcoin used direct Pay-to- Pubkey, which is now superseded by P2PKH.
  - direct Pay-to-Pubkey is still used in bitcoin for coinbase addresses.
- Addresses should not be used more than once; otherwise, privacy and security issues can arise.
- Avoiding address reuse bypasses
  - anonymity issues to an extent,
  - bitcoin has some other security issues as well, such as transaction malleability, which requires different approaches to resolve.

Pay-to-PubKey-Hash (Pay-to-Public-Key-Hash, P2PKH)

- P2PKH is the basic form of making a transaction and is the most common form of transaction on the Bitcoin network.
- Transactions that pay to a Bitcoin address contain P2PKH scripts that are resolved by sending the public key and a digital signature created by the corresponding private key.
- Bitcoin payment address comprising a hashed public key, allowing the spender to create a standard pubkey script that Pays To PubKey Hash (P2PKH).

Pay to script hash (P2SH)

- Pay to script hash (P2SH) transactions were standardised in BIP 16.
- They allow transactions to be sent to a script hash (address starting with 3) instead of a public key hash (addresses starting with 1).
- To spend bitcoins sent via P2SH, the recipient must provide a script matching the script hash and data which makes the script evaluate to true.
- Using P2SH, you can send bitcoins to an address that is secured in various unusual ways without knowing anything about the details of how the security is set up.
- You just send bitcoins to the ~34-character P2SH address.
- Recipient might need the signatures of several people to spend these bitcoins, or a password might be required, or the requirements could be completely unique.


Public Key to Bitcoin Address
Public Key
SHA256
RIPEMD160
"Double Hash" or HASH160
Public Key Hash (20 bytes/160 bits)
Base58Check Encode with 0x00 version prefix
Bitcoin Address (Base58Check Encoded Public Key Hash)

Private key

- Private key is simply a number, picked at random.
- Ownership and control over the private key is the root of user control over all funds associated with the corresponding bitcoin address.
- Private key is used to create signatures that are required to spend bitcoin by proving ownership of funds used in a transaction.
- Private key must remain secret at all times, because revealing it to third parties is equivalent to giving them control over the bitcoin
- Private key must also be backed up and protected from accidental loss
  - because if it’s lost it cannot be recovered

Generating a private key from a random number

- Important step in generating keys is to find a secure source of entropy, or randomness.
- Creating a bitcoin key is essentially the same as "Pick a number between 1 and 2^{256}."
  - The exact method you use to pick that number does not matter as long as it is not predictable or repeatable.
- Bitcoin software uses the underlying operating system’s random number generators to produce 256 bits of entropy
- More precisely, the private key can be any number between 0 and n - 1 inclusive, where n is a constant ($n = 1.1578 * 10^{77}$, slightly less than 2^{256}) defined as the order of the elliptic curve used in bitcoin

Generating a private key from a random number

- To create such a key, we randomly pick a 256-bit number and check that it is less than n.
- In programming terms, this is usually achieved by feeding a larger string of random bits, collected from a cryptographically secure source of randomness, into the SHA256 hash algorithm, which will conveniently produce a 256-bit number.
- If the result is less than n, we have a suitable private key. Otherwise, we simply try again with another random number.

Private keys in bitcoin

- Private keys are basically 256-bit numbers chosen in the range specified by the SECP256K1 ECDSA recommendation.
- Any randomly chosen 256-bit number from 0x1 to 0xFFFF FFFF
- FFFF FFFF FFFF FFFF FFFF FFFE BAAE DCE6 AF48 A03B BFD2 5E8C D036 4140 is a valid private key.
- Private keys(ECDSA Key) are usually encoded using **Wallet Import Format** (**WIF**) in order to make them easier to copy and use.
- WIF can be converted into private key and vice versa.
- **Mini Private Key Format** is sometimes used to encode the key in under 30 characters in order to allow storage where physical space is limited,
- Bitcoin core client also allows the encryption of the wallet that contains the private keys.

Public keys in bitcoin

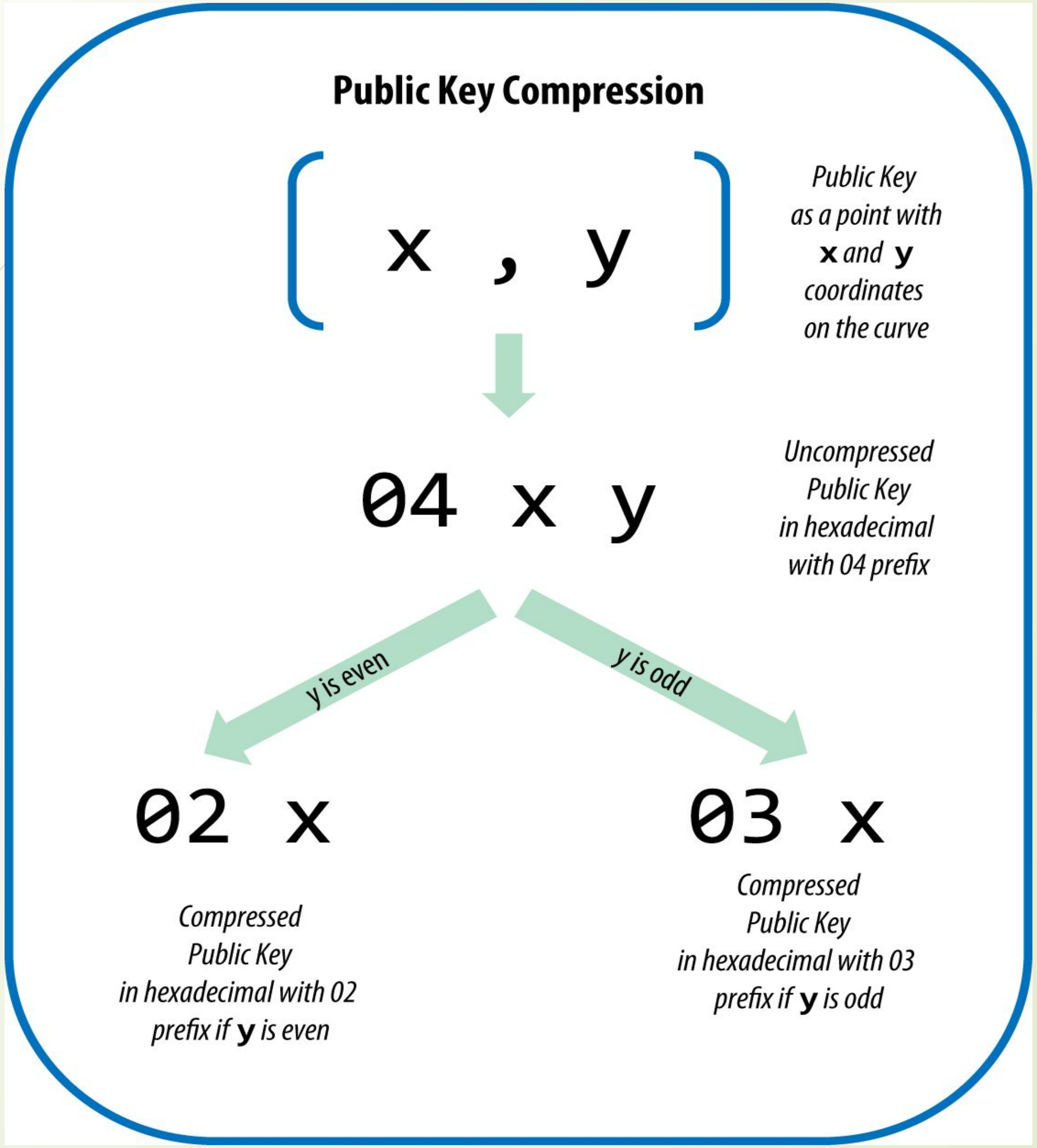
- Public key is calculated from the private key using elliptic curve multiplication, which is irreversible:
- $K = k * G$, where k is the private key, G is a constant point called the *generator point*, and K is the resulting public key.
- The reverse operation, known as "finding the discrete logarithm"—calculating k if you know K—is as difficult as trying all possible values of k, i.e., a brute-force search.

Public keys in bitcoin

- A private key is randomly selected and is 256-bit in length.
- Public keys can be presented in an uncompressed or compressed format.
- Public keys are basically *x* and *y* coordinates on an elliptic curve and in an uncompressed format and are presented with a prefix of 04 in a hexadecimal format.
  - Public key K is defined as a point K = (x,y):
  - *X* and *Y* coordinates are both 32- bit in length.
- In total, the compressed public key is 33 bytes long as compared to 65 bytes in the uncompressed format.
- The compressed version of public keys basically includes only the *X* part, since the *Y* part can be derived from it.

Public keys in bitcoin

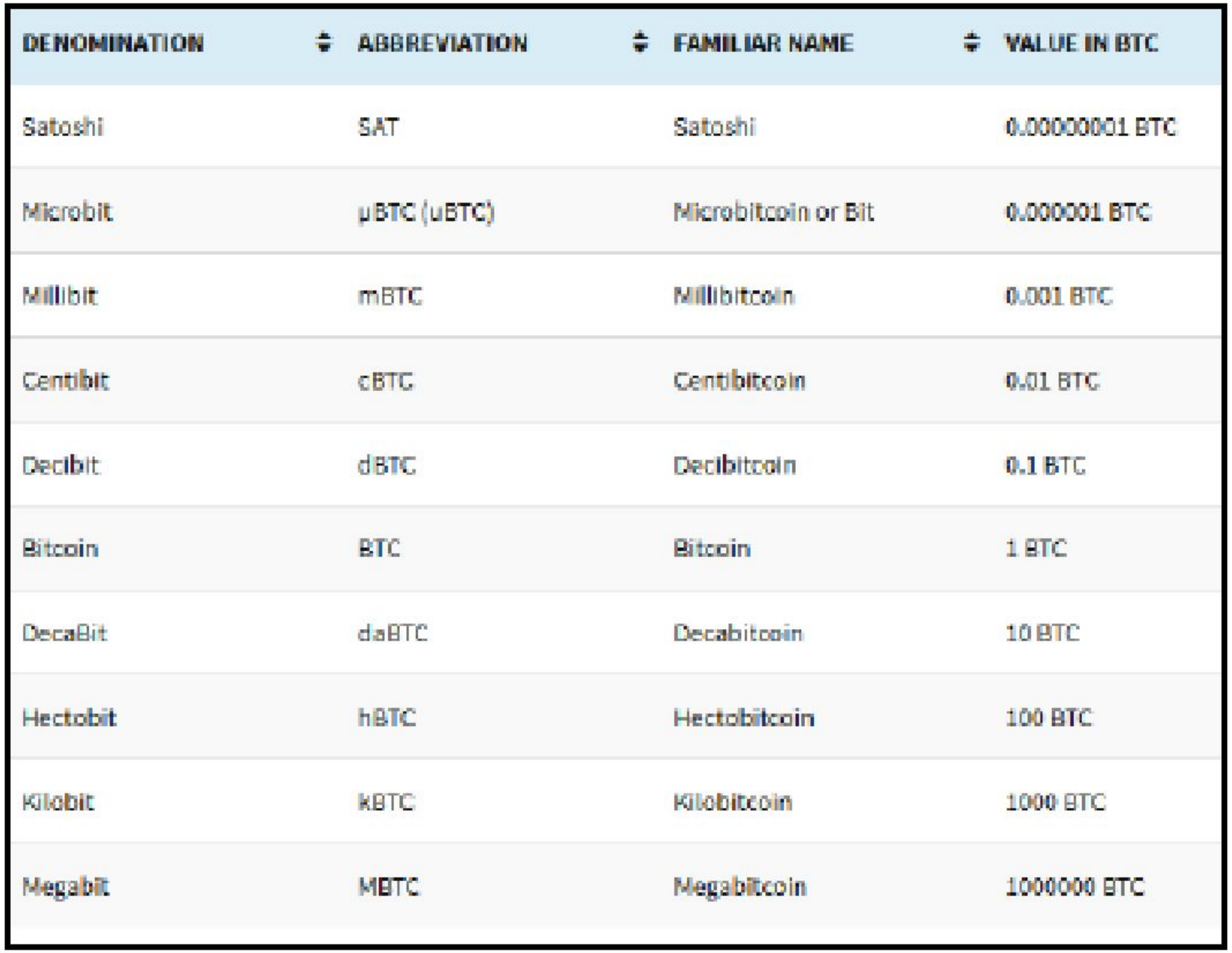
- Keys are identified by various prefixes, described as follows:
  - Uncompressed public keys used 0x04 as the prefix
  - Compressed public key starts with 0x03 if the y 32-bit part of the public key is odd
  - Compressed public key starts with 0x02 if the y 32-bit part of the public key is even


Public Key Compression
x , y
Public Key as a point with x and y coordinates on the curve
04 x y
Uncompressed Public Key in hexadecimal with 04 prefix
y is even
y is odd
02 x
03 x
Compressed Public Key in hexadecimal with 02 prefix if y is even
Compressed Public Key in hexadecimal with 03 prefix if y is odd

Public keys in bitcoin

- Compressed public keys are gradually becoming the default across bitcoin clients, which is having a significant impact on reducing the size of transactions and therefore the blockchain.
  - However, not all clients support compressed public keys yet.
- Newer clients that support compressed public keys have to account for transactions from older clients that do not support compressed public keys.
- This is especially important when a wallet application is importing private keys from another bitcoin wallet application,
  - because the new wallet needs to scan the blockchain to find transactions corresponding to these imported keys.

Bitcoin currency units



DENOMINATION	ABBREVIATION	FAMILIAR NAME	VALUE IN BTC
Satoshi	SAT	Satoshi	0.00000001 BTC
Microbit	µBTC (uBTC)	Microbitcoin or Bit	0.000001 BTC
Millibit	mBTC	Millibitcoin	0.001 BTC
Centibit	cBTC	Centibitcoin	0.01 BTC
Decibit	dBTC	Decibitcoin	0.1 BTC
Bitcoin	BTC	Bitcoin	1 BTC
DecaBit	daBTC	Decabitcoin	10 BTC
Hectobit	hBTC	Hectobitcoin	100 BTC
Kilobit	kBTC	Kilobitcoin	1000 BTC
Megabit	MBTC	Megabitcoin	1000000 BTC

Base58Check encoding

- In order to represent long numbers in a compact way, using fewer symbols, many computer systems use mixed-alphanumeric representations with a base (or radix) higher than 10.
  - For example, whereas the traditional decimal system uses the 10 numerals 0 through 9, the hexadecimal system uses 16, with the letters A through F as the six additional symbols.
- A number represented in hexadecimal format is shorter than the equivalent decimal representation.
- Base64 representation uses 26 lowercase letters, 26 capital letters, 10 numerals, and 2 more characters such as “``&#x201d; and "/" to transmit binary data over text-based media such as email.
- Base64 is most commonly used to add binary attachments to email.
- Base58 is a text-based binary-encoding format developed for use in bitcoin and used in many other cryptocurrencies

Base58Check encoding

- Base58 offers a balance between compact representation, readability, and error detection and prevention.
- Base58 is a subset of Base64, using upper- and lowercase letters and numbers, but omitting some characters that are frequently mistaken for one another and can appear identical when displayed in certain fonts.
- Specifically, Base58 is Base64 without the 0 (number zero), O (capital o), l (lower L), I (capital i), and the symbols &#x201c;``” and "/". Or, more simply, it is a set of lowercase and capital letters and numbers without the four (0, O, l, I) just mentioned.
- Bitcoin’s Base58 alphabet shows the full Base58 alphabet.

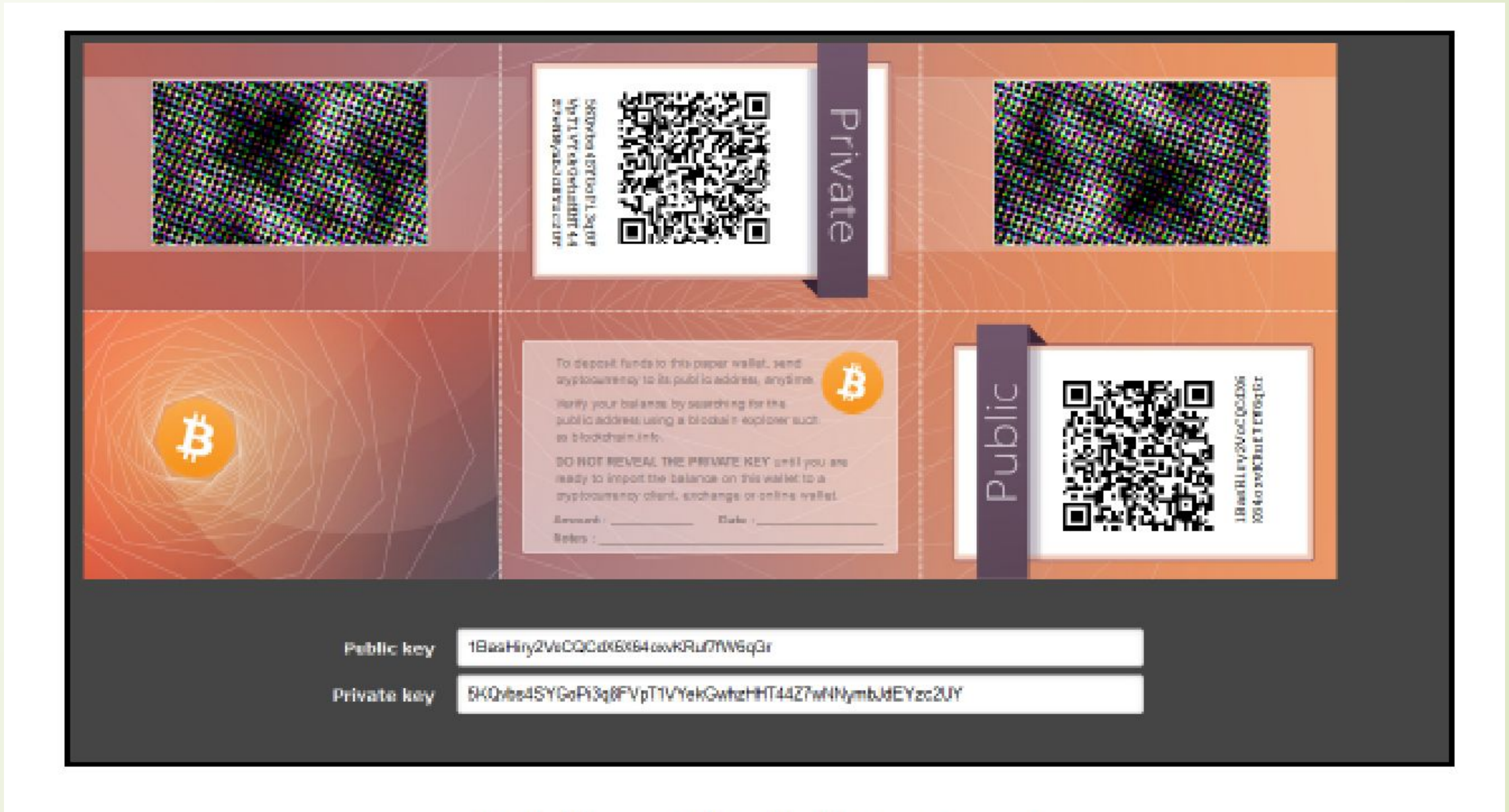
Vanity addresses

- As bitcoin addresses are based on base 58 encoding, it is possible to generate addresses that contain human-readable messages.

1BasHiry2VoCQCdX6X
64oxvKRuf7fW6qGr

Public address encoded in QR

- Vanity addresses are generated using a purely brute-force method.
- Example



To deposit funds to this paper wallet, send cryptocurrency to its public address, anytime.

Verify your balance by searching for the public address using a blockchain explorer such as blockchain.info.

DO NOT REVEAL THE PRIVATE KEY until you are ready to import the balance on this wallet to a cryptocurrency client, exchange or online wallet.

Amount : ____________ Date : ____________

Notes : ____________

Public key 1BasHiry2VsCQCdX6X64cwKRuf7fW6qGr

Private key 5KQvbs4SYGoPi3q8FVpT1VYekGwhzHHT44Z7wNNymbJdEYzc2UY

Vanity address generated from https://bitcoinvanitygen.com/

Vanity Addresses

- Vanity addresses are valid bitcoin addresses that contain human-readable messages.
- For example, 1LoveBPzzD72PUXLzCkYAtGFYmK5vYNR33 is a valid address that contains the letters forming the word "Love" as the first four Base-58 letters.
- Vanity addresses require generating and testing billions of candidate private keys, until a bitcoin address with the desired pattern is found.
- Vanity generation algorithm,
    - the process essentially involves picking a private key at random,
    - deriving the public key,
    - deriving the bitcoin address
    - checking to see if it matches the desired vanity pattern,
    - repeating billions of times until a match is found.

Vanity Addresses

- Once a vanity address matching the desired pattern is found,
  - Private key from which it was derived can be used by the owner to spend bitcoin in exactly the same way as any other address.
- Vanity addresses are no less or more secure than any other address.
- They depend on the same Elliptic Curve Cryptography (ECC) and SHA as any other address.
  - You can no more easily find the private key of an address starting with a vanity pattern than you can any other address.

Paper Wallets

- Paper wallets are bitcoin private keys printed on paper.
- Paper wallet also includes the corresponding bitcoin address for convenience
  - but this is not necessary because it can be derived from the private key.
- Paper wallets are a very effective way to create backups or offline bitcoin storage, also known as "cold storage.“
- As a backup mechanism, a paper wallet can provide security against the loss of key due to a computer mishap such as a hard-drive failure, theft, or accidental deletion.
- If the paper wallet keys are generated offline and never stored on a computer system, they are much more secure against hackers, keyloggers, and other online computer threats.

$1-\sum_{k=0}^{z}\frac{\lambda^k e^{-\lambda}}{k!}(1-(q/p)^{(z-k)})$
Load & Verify
Bitcoin Address
1Ar6e15vfE5696ms8UBPuAcDckWVnmUGWv
bitcoin bitcoin bitcoin bitcoin bitcoin bitcoin bitcoin bitcoin
Strength in Numbers
bitcoin
Amount:
Private Key
5K1TaGDVJD3XETBA2GT7GzmS4Lna24XQ3MpSRsZTRwfnwjYLn
Spend

Transactions

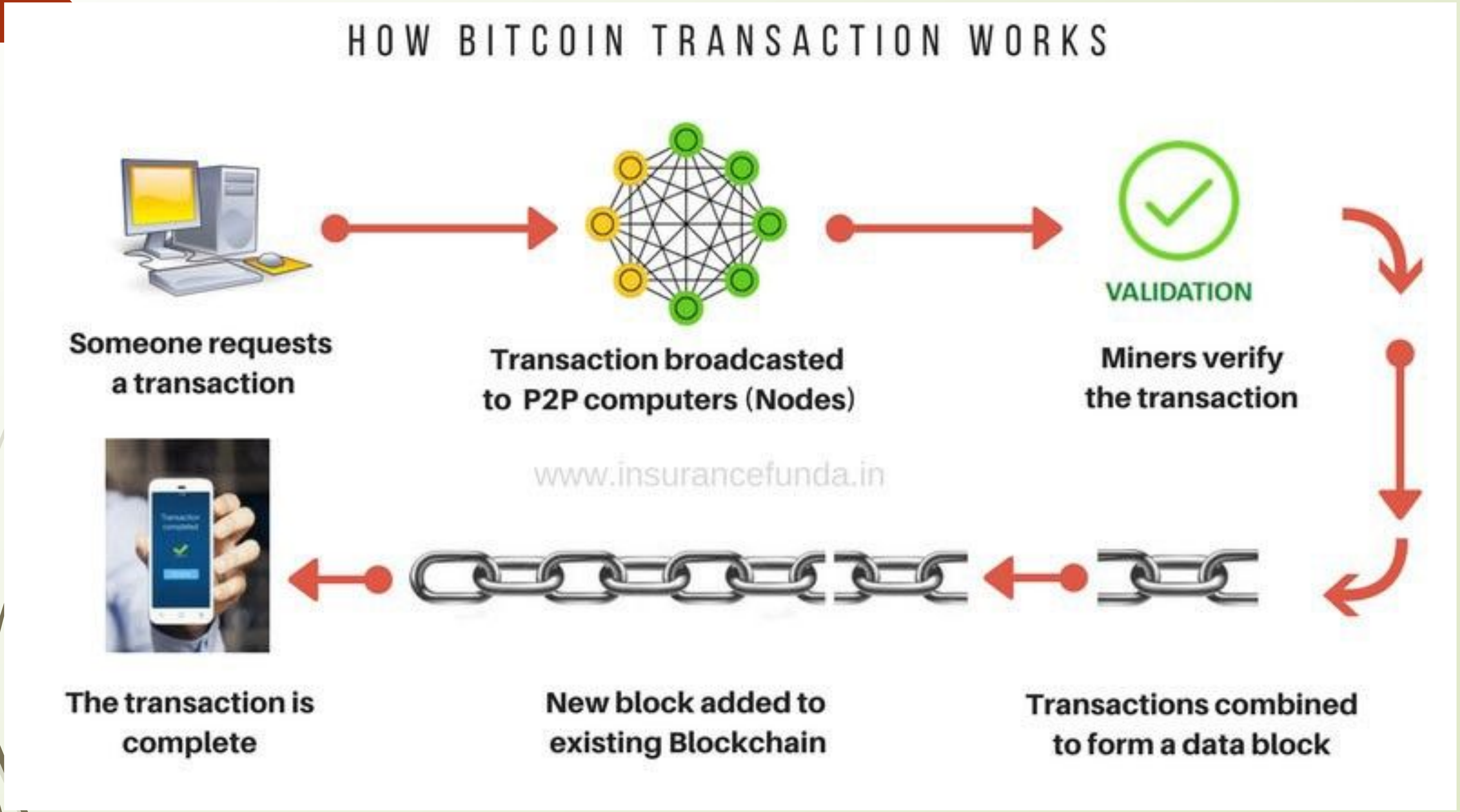
- Transactions are the most important part of the bitcoin system.
- Everything else in bitcoin is designed to ensure that transactions can be
  - created
  - propagated on the network,
  - validated,
  - Finally added to the global ledger of transactions (the blockchain).
- Transactions are data structures that encode the transfer of value between participants in the bitcoin system.
- Each transaction is a public entry in bitcoin's blockchain, the global double-entry bookkeeping ledger.

Transactions

- Transactions are at the core of the bitcoin ecosystem.
- Transactions can be as simple as just sending some bitcoins to a bitcoin address, or it can be quite complex depending on the requirements.
- Each transaction is composed of at least one input and output.
- Inputs can be thought of as coins being spent that have been created in a previous transaction and outputs as coins being created.
- If a transaction is minting new coins, then there is no input and therefore no signature is needed.
- If a transaction is to sends coins to some other user (a bitcoin address), then it needs to be signed by the sender with their private key and a reference is also required to the previous transaction in order to show the origin of the coins.
- Coins are, in fact, unspent transaction outputs represented in Satoshis.

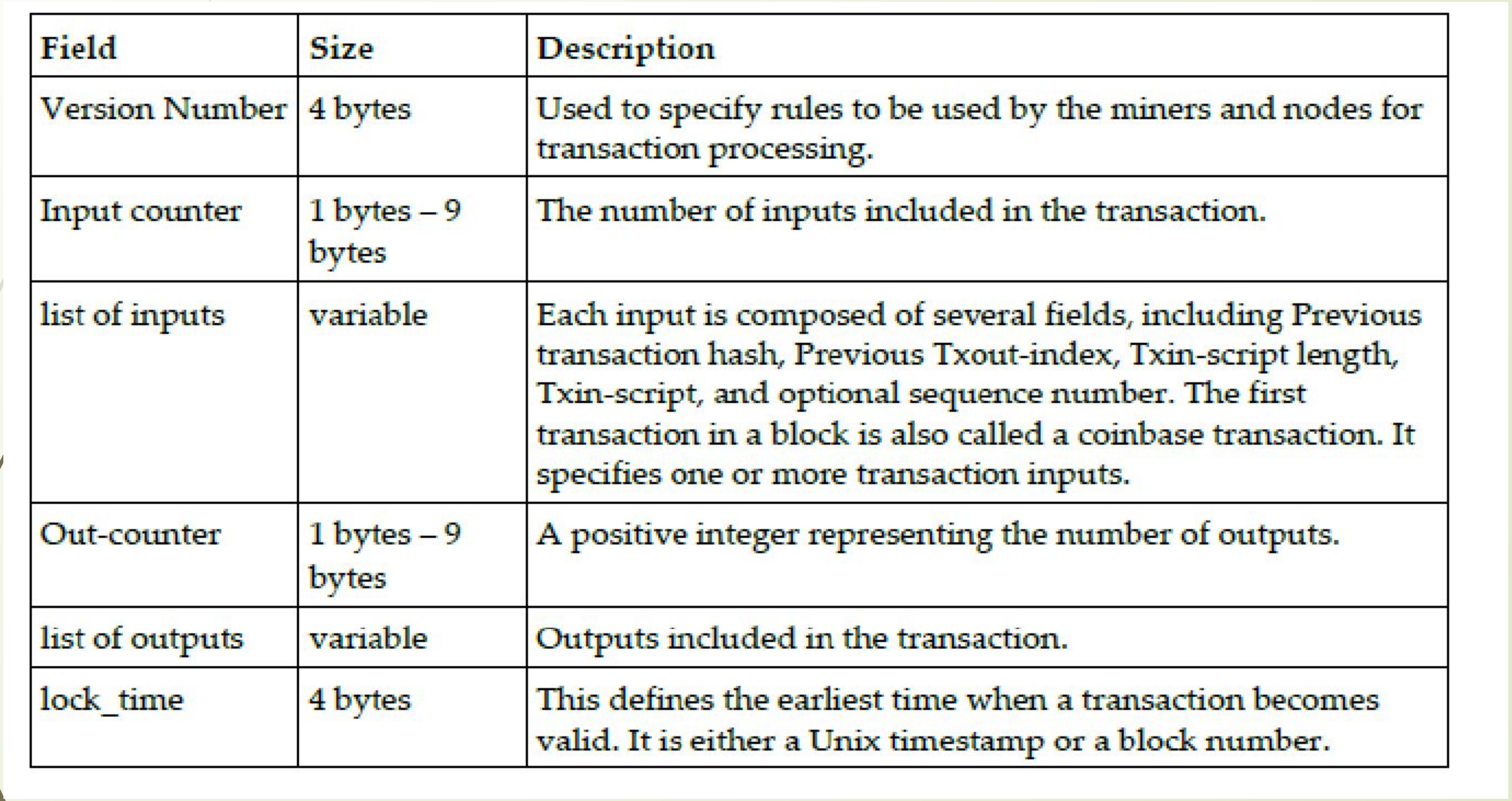
The transaction life cycle

- A user/sender **sends a transaction** using wallet software or some other interface.
- The **wallet software signs** the transaction using the sender's private key.
- The **transaction is broadcasted** to the Bitcoin network using a flooding algorithm.
- Mining nodes **include this transaction** in the next block to be mined.
- Mining starts once a miner who solves the **Proof of Work** problem broadcasts the **newly mined block** to the network.
- The nodes **verify the block** and propagate the block further, and confirmation starts to generate.
- Finally, the confirmations start to appear in the receiver's wallet and after approximately **six confirmations**, the transaction is considered finalized and confirmed.


HOW BITCOIN TRANSACTION WORKS
Someone requests a transaction
Transaction broadcasted to P2P computers (Nodes)
VALIDATION
Miners verify the transaction
www.insurancefunda.in
Transactions combined to form a data block
New block added to existing Blockchain
The transaction is complete

The transaction structure

- A transaction at a high level contains metadata, inputs, and outputs.
- Transactions are combined to create a block



Field	Size	Description
Version Number	4 bytes	Used to specify rules to be used by the miners and nodes for transaction processing.
Input counter	1 bytes – 9 bytes	The number of inputs included in the transaction.
list of inputs	variable	Each input is composed of several fields, including Previous transaction hash, Previous Txout-index, Txin-script length, Txin-script, and optional sequence number. The first transaction in a block is also called a coinbase transaction. It specifies one or more transaction inputs.
Out-counter	1 bytes – 9 bytes	A positive integer representing the number of outputs.
list of outputs	variable	Outputs included in the transaction.
lock_time	4 bytes	This defines the earliest time when a transaction becomes valid. It is either a Unix timestamp or a block number.

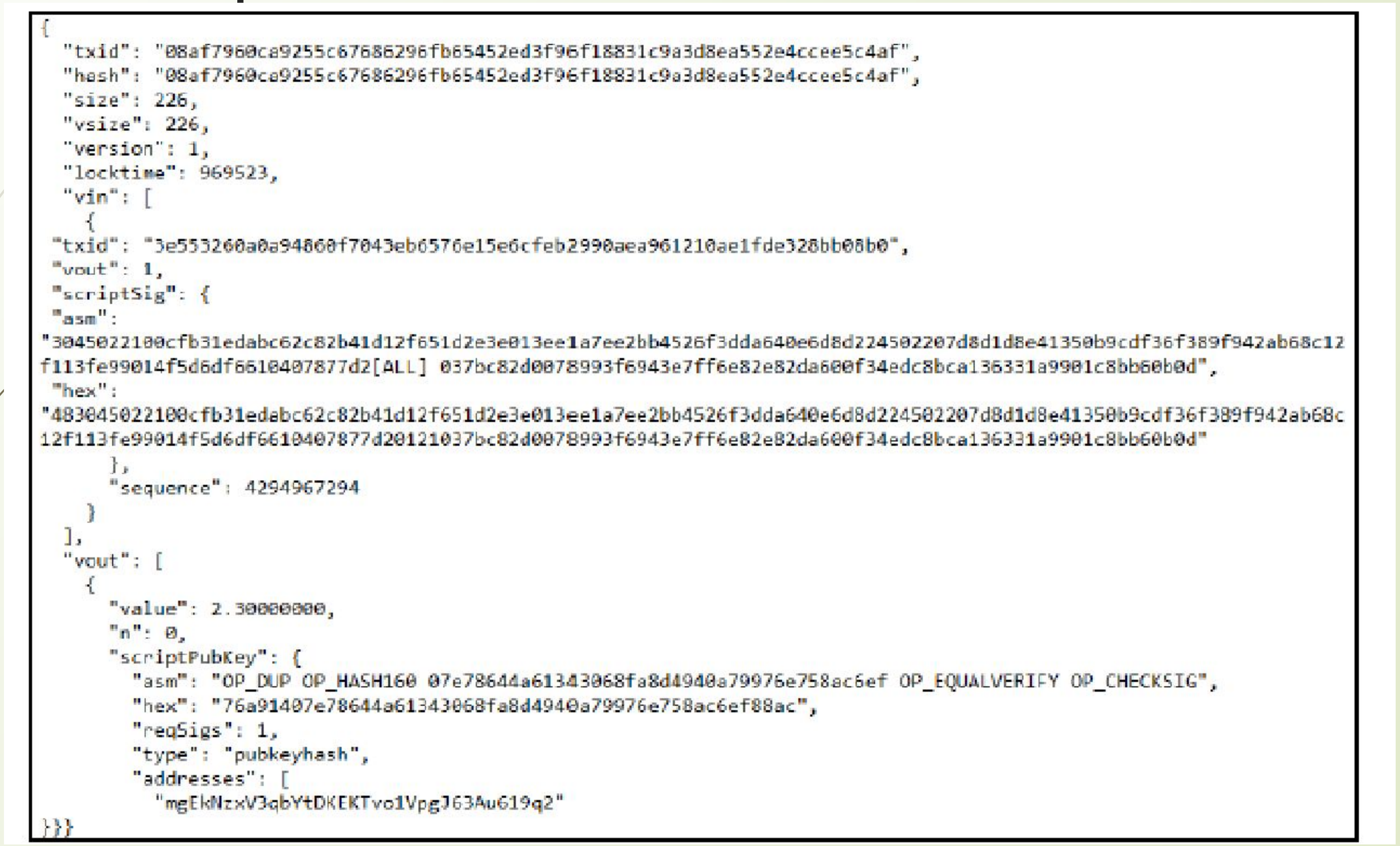
The transaction structure

- MetaData:
  - This part of the transaction contains some values such as the size of the transaction, the number of inputs and outputs, the hash of the transaction, and a lock_time field.
  - Every transaction has a prefix specifying the version number.
- Inputs:
  - Generally, each input spends a previous output.
  - Each output is considered an Unspent Transaction Output (UTXO) until an input consumes it.

The transaction structure

- Outputs:
  - Outputs have only two fields, and they contain instructions for the sending of bitcoins.
  - The first field contains the **amount of Satoshis**,
  - Second field is a **locking script** that contains the conditions that need to be met in order for the output to be spent.
- Verification:
  - Verification is performed using bitcoin's scripting language.

sample transaction



```
{
  "txid": "08af7960ca9255c67686296fb65452ed3f96f18831c9a3d8ea552e4ccee5c4af",
  "hash": "08af7960ca9255c67686296fb65452ed3f96f18831c9a3d8ea552e4ccee5c4af",
  "size": 226,
  "vsize": 226,
  "version": 1,
  "locktime": 969523,
  "vin": [
    {
 "txid": "3e553260a0a94860f7043eb6576e15e6cfeb2990aea961210ae1fde328bb08b0",
 "vout": 1,
 "scriptSig": {
 "asm":
"3045022100cfb31edabc62c82b41d12f651d2e3e013ee1a7ee2bb4526f3dda640e6d8d224502207d8d1d8e41350b9cdf36f389f942ab68c12
f113fe99014f5d6df6610407877d2[ALL] 037bc82d0078993f6943e7ff6e82e82da600f34edc8bca136331a9901c8bb60b0d",
 "hex":
"483045022100cfb31edabc62c82b41d12f651d2e3e013ee1a7ee2bb4526f3dda640e6d8d224502207d8d1d8e41350b9cdf36f389f942ab68c
12f113fe99014f5d6df6610407877d20121037bc82d0078993f6943e7ff6e82e82da600f34edc8bca136331a9901c8bb60b0d"
      },
      "sequence": 4294967294
    }
  ],
  "vout": [
    {
      "value": 2.30000000,
      "n": 0,
      "scriptPubKey": {
        "asm": "OP_DUP OP_HASH160 07e78644a61343068fa8d4940a79976e758ac6ef OP_EQUALVERIFY OP_CHECKSIG",
        "hex": "76a91407e78644a61343068fa8d4940a79976e758ac6ef88ac",
        "reqSigs": 1,
        "type": "pubkeyhash",
        "addresses": [
          "mgEkNzxV3qbYtDKEKTvo1VpgJ63Au619q2"
}}}
```

Bitcoin Scripts

- In Bitcoin transactions, there is necessary use of digital signatures and validation with the help private and public key pairs.
- This is implemented through **Bitcoin scripts**.
  - Input of transaction consists of scripts instead of signatures.
- To validate that a transaction is in sync with the previous blocks of the chain correctly,
  - we combine the new transaction's input script and the earlier transaction's output script by concatenating them.

Bitcoin Scripting Language

- It is based on an old, simple, **stack-based programming** language called **Forth**.
- Main goals of designing such a script for Bitcoins was simple, compact and yet supporting key cryptographic operations.
- key features of this language are:
  - **Stack based:**
    - **Every instruction** specified in the Bitcoin script **is executed exactly once** and in a linear fashion.
    - **No implementation of loops** in the Bitcoin scripting language.

key features of this language

- **Not Turing-complete**
  - This results to the language's inability to execute arbitrarily powerful methods based on recursion and loops.
  - Miners of the Bitcoin network run these scripts appended by the sender of a transaction and this is the reason for dis-allowance of infinite loops in Bitcoin scripts.
- **Finite Size**
  - Bitcoin scripting language is finitely small in size.
  - It can only accommodate 256 instructions as each instruction is represented by one byte.

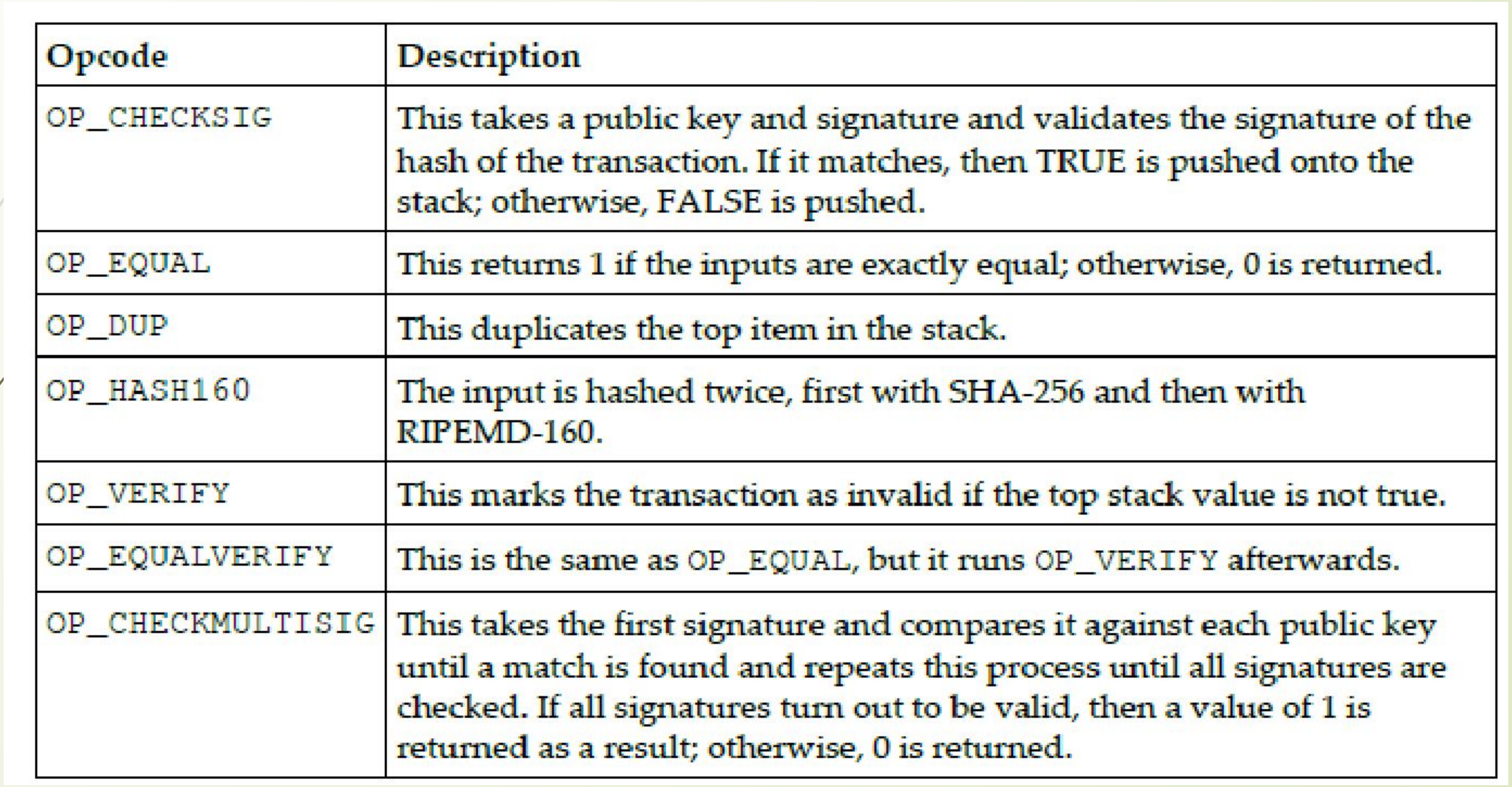
Script Instructions

- Large part of the instruction set in Bitcoin scripts is same as any of those in common programming languages.
  - **For examples**: arithmetic operations, logic blocks such as if-else, throwing and not throwing of errors and function returns
- There are two types of instructions:
  - **Data instructions:**
    - If a data instructions is encountered in a script, the data corresponding to the instruction is simply pushed onto the top of the stack data structure.
  - **Opcodes:**
    - Whenever an opcode is encountered in a script, it executes some particular function which is mostly taken as input data residing on the top of the stack.

Most common instructions

- **OP DUP**: Duplicates the top item of the stack
- **OP HASH160**: Hashes twice, first using SHA-256 and then RIPEMD-160
- **OP EQUALVERIFY**: Returns true if the inputs are equal. Returns false and marks the transaction as invalid if they are unequal
- **OP CHECKSIG**: Checks that the input signature is a valid signature using the input public key for the hash of the current transaction
- **OP CHECKMULTISIG**: Checks that the k signatures on the transaction are valid signatures from k of the specified public keys

Commonly used Opcodes



Opcode	Description
OP_CHECKSIG	This takes a public key and signature and validates the signature of the hash of the transaction. If it matches, then TRUE is pushed onto the stack; otherwise, FALSE is pushed.
OP_EQUAL	This returns 1 if the inputs are exactly equal; otherwise, 0 is returned.
OP_DUP	This duplicates the top item in the stack.
OP_HASH160	The input is hashed twice, first with SHA-256 and then with RIPEMD-160.
OP_VERIFY	This marks the transaction as invalid if the top stack value is not true.
OP_EQUALVERIFY	This is the same as OP_EQUAL, but it runs OP_VERIFY afterwards.
OP_CHECKMULTISIG	This takes the first signature and compares it against each public key until a match is found and repeats this process until all signatures are checked. If all signatures turn out to be valid, then a value of 1 is returned as a result; otherwise, 0 is returned.

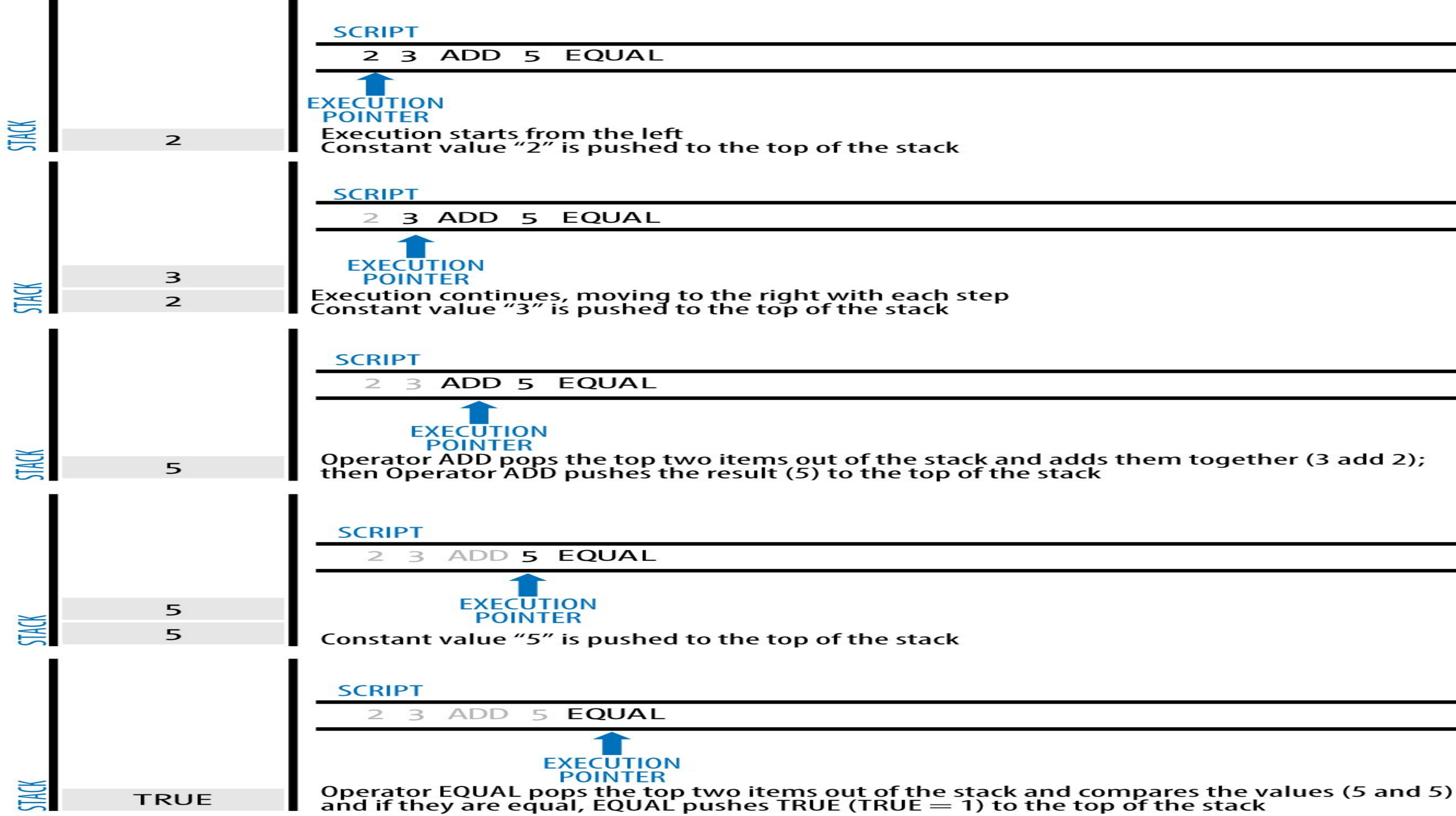
Execution of the script

- All that we will need to execute is a stack data structures which supports push and pop methods.
- Any other form of memory or variables are not required for the script’s execution
- example of a Bitcoin script:

```
<sig>
<pubKey>
OP DUP
OP HASH160
<pubKeyHash?>
OP EQUALVERIFY
OP CHECKSIG
```

Execution of the script

- Data instructions are denoted with surrounding angle brackets, whereas opcodes begin with ”OP ”
- In this example, the first two instructions of the script are the signature and public key used to verify that signature.
- Then we duplicate the instruction with OP DUP where we simply push a copy of the public key onto the stack data structure.
- This instructions is followed by OP HASH160 which basically pops the top value and computes the cryptographic hash and then pushes the result onto the stack.



Execution starts from the left
Constant value “2” is pushed to the top of the stack

Execution continues, moving to the right with each step
Constant value “3” is pushed to the top of the stack

Operator ADD pops the top two items out of the stack and adds them together (3 add 2);
then Operator ADD pushes the result (5) to the top of the stack

Constant value “5” is pushed to the top of the stack

Operator EQUAL pops the top two items out of the stack and compares the values (5 and 5)
and if they are equal, EQUAL pushes TRUE (TRUE = 1) to the top of the stack

Stateless Verification

- Bitcoin transaction script language is **stateless**, in that there is no state prior to execution of the script, or state saved after execution of the script.
  - All the information needed to execute a script is contained within the script.
- If your system verifies a script, you can be sure that every other system in the bitcoin network will also verify the script
  - Valid transaction is valid for everyone and everyone knows this.

Script Construction (Lock + Unlock)

- Bitcoin’s transaction validation engine relies on two types of scripts to validate transactions:
  - locking script
  - unlocking script.

A locking script

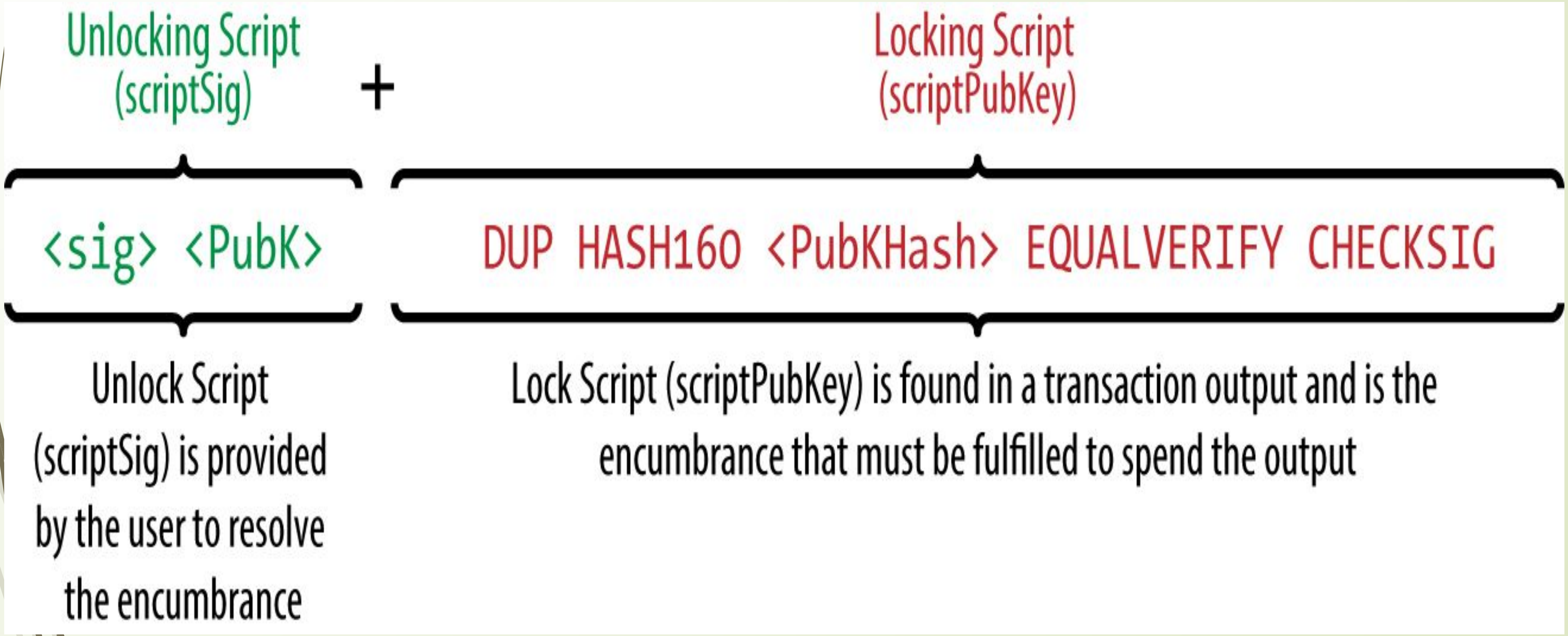
- Locking script is a spending condition placed on an output
- It specifies the **conditions** that must be met to spend the output in the future.
- Locking script was called a ***scriptPubKey***, because it usually contained a public key or bitcoin address (public key hash).
  - locking script also referred as a ***witness script*** or more generally as a ***cryptographic puzzle*.**

unlocking script

- An unlocking script is a script that "solves," or satisfies, the conditions placed on an output by a locking script and allows the output to be spent.
- Unlocking scripts are part of every transaction input.
- It contain a digital signature produced by the user’s wallet from his or her private key.
- Unlocking script was called ***scriptSig***, because it usually contained a digital signature.

Script Construction (Lock + Unlock)

- Every bitcoin validating node will validate transactions by executing the locking and unlocking scripts together.
- Each input contains an unlocking script and refers to a previously existing UTXO.
- The validation software will
  - Copy the unlocking script
  - Retrieve the UTXO referenced by the input,
  - Copy the locking script from that UTXO.
- The unlocking and locking script are then executed in sequence.
- The **input is valid if the unlocking script satisfies the locking script conditions**
- All the inputs are validated independently, as part of the overall validation of the transaction.



Unlocking Script (scriptSig) + Locking Script (scriptPubKey)

```
<sig> <PubK>    DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
```

Unlock Script (scriptSig) is provided by the user to resolve the encumbrance

Lock Script (scriptPubKey) is found in a transaction output and is the encumbrance that must be fulfilled to spend the output

Types of transaction

- There are **various scripts** available in bitcoin **to handle the value transfer from the source to the destination**
  - These scripts range from very **simple to quite complex** depending upon the requirements of the transaction.
- Standard transactions are evaluated using **IsStandard()** and **IsStandardTx()** tests
  - Only standard transactions that pass the test are **generally allowed** to be mined or broadcasted on the bitcoin network.
  - However, Nonstandard transactions are valid and allowed on the network.

Types of transaction

- **Pay to Public Key Hash (P2PKH):**
  - P2PKH is the most commonly used transaction type and is used to send transactions to the bitcoin addresses.
  - The format of the transaction is shown as folows:
    - ScriptPubKey: OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
    - ScriptSig: <sig> <pubKey>
  - The ScriptPubKey and ScriptSig parameters are concatenated together and executed.

Types of transaction

- **Pay to Script Hash (P2SH)**:
  - P2SH is used in order to send transactions to a script hash and was standardized in BIP16.
  - In addition to passing the script, the redeem script is also evaluated and must be valid.
  - The template is shown as follows:
    - ScriptPubKey: OP_HASH160 <redeemScriptHash> OP_EQUAL
    - ScriptSig: [<sig>…<sign>] <redeemScript>

Types of transaction

- **MultiSig (Pay to MultiSig)**:
  - M of n multisignature transaction script is a complex type of script where it is possible to construct a script that required multiple signatures to be valid in order to redeem a transaction.
  - Various complex transactions such as escrow and deposits can be built using this script.
  - The template is shown here:
    - ScriptPubKey: <m> <pubKey> [<pubKey> . . .] <n> OP_CHECKMULTISIG
    - ScriptSig: 0 [<sig > . . . <sign>]
  - Raw multisig is obsolete, and multisig is usually part of the P2SH redeem script

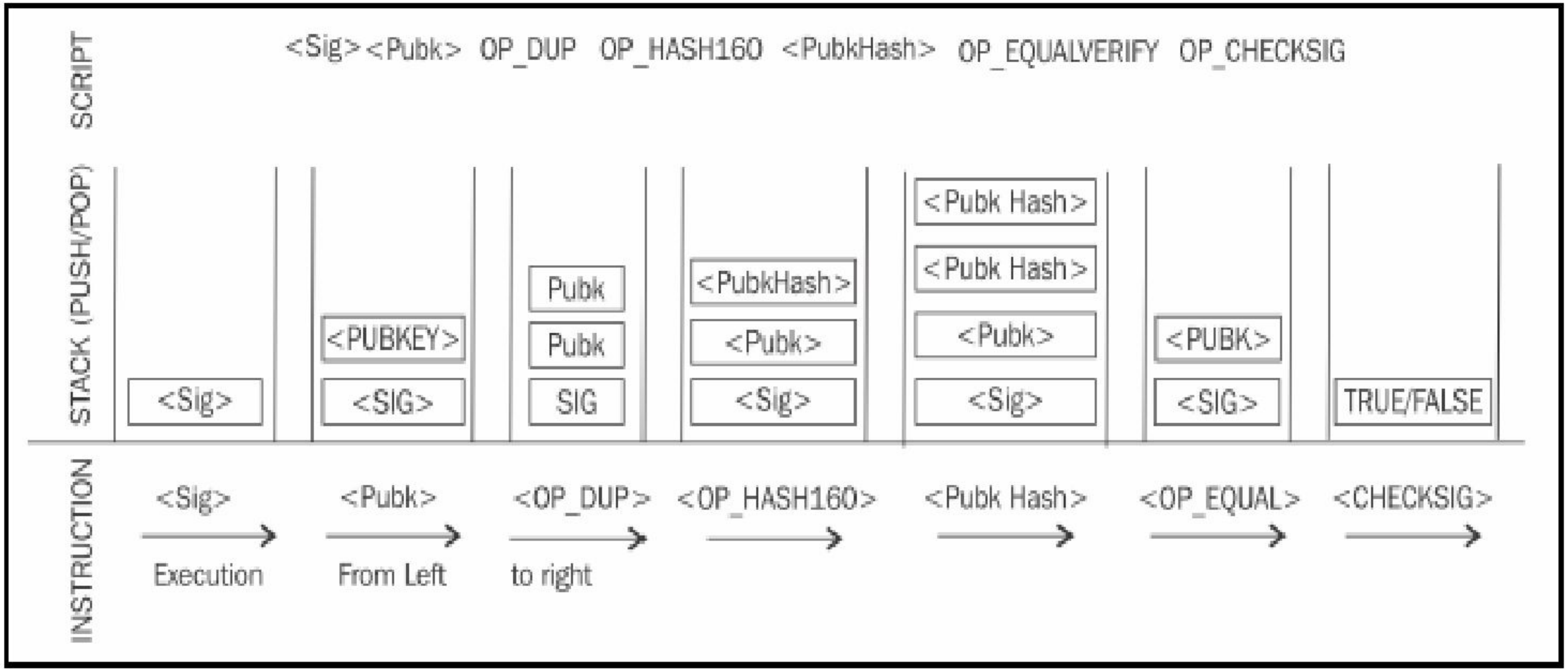
Types of transaction

- **Pay to Pubkey**:
  - This script is a very simple script that is commonly used in coinbase transactions.
  - It is now obsolete and was used in an old version of bitcoin.
  - The public key is stored within the script in this case, and the unlocking script is required to sign the transaction with the private key.
  - The template is shown as follows:
    - <PubKey> OP_CHECKSIG

Types of transaction

- **Null data/OP_RETURN**:
  - This script is used to store arbitrary data on the blockchain for a fee.
  - The limit of the message is 40 bytes.
  - The output of this script is unredeemable because OP_RETURN will fail the validation in any case.
  - ScriptSig is not required in this case.
  - The template is very simple and is shown as follows:
    - OP_RETURN <data>

A P2PKH script execution



Coinbase transactions

- Coinbase transaction or generation transaction is always created by a miner and is the first transaction in a block.
  - It is used to create new coins.
- It includes a special field, also called *coinbase*, which acts as an input to the coinbase transaction.
- This transaction also allows up to 100 bytes of arbitrary data that can be used to store arbitrary data.
- In the genesis block, this included the most famous comment taken from The Times newspaper:
  - *“The Times 03/Jan/2009 Chancellor on brink of second bailout for banks”*
  - This message is proof that the genesis block was not mined earlier than January 3, 2009.

What is UTXO?

- **Unspent Transaction Output (UTXO)** is an unspent transaction output that can be spent as an input to a new transaction

Transaction fee

- Transaction fees are charged by the miners.
- Fee charged is dependent upon the size of the transaction.
- Transaction fees are calculated by subtracting the sum of the inputs and the sum of the outputs.
- Fees are used as an **incentive for miners to encourage them to include a user transaction in the block the miners are creating.**
- All transactions end up in the memory pool, from where miners pick up transactions based on their priority to include them in the proposed block.

Transaction fee

- From a transaction fee point of view, a transaction with a higher fee will be picked up sooner by the miners.
- Different rules based on which fee is calculated for various types of actions, such as
  - Sending transactions
  - Inclusion in blocks
  - Relaying by nodes.
- Fees are not fixed by the Bitcoin protocol and are not mandatory;
  - Even a transaction with no fee will be processed in due course but may take a very long time.

Contracts

- Contracts are basically transactions that use the bitcoin system to enforce a financial agreement.
- Allows users to design complex contracts that can be used in many real-world scenarios.
- Contracts allow the development of a completely decentralized, independent, and reduced risk platform.
- Various contracts, such as escrow, arbitration, and micropayment channels, can be built using the bitcoin scripting language

Contracts

- Various types of contracts are possible to develop.
- For example
  - Release of funds only if multiple parties sign the transaction
  - Release of funds only after a certain time has elapsed.
- Both of these scenarios can be realized using multiSig and transaction lock time options.

Transaction malleability

- Transaction malleability in bitcoin was introduced due to a bug in the bitcoin implementation.
- Due to this bug, it becomes possible for an adversary to change the Transaction ID of a transaction
- If the ID is changed before confirmation, it would seem that the transaction did not happen at all,
  - which allow double deposits or withdrawal attacks.
- In other words, this bug allows the changing of the unique ID of a bitcoin transaction before it is confirmed.

Transaction pools

- Also known as ***memory pools***
- These pools are basically created in local memory by nodes in order to maintain a temporary list of transactions that are not yet confirmed in a block.
- Transactions are included in a block after passing verification and based on their priority.

Transaction verification

- Check the syntax and ensure that the syntax of the transaction is correct.
- Verify that inputs and outputs are not empty.
- Check whether the size in bytes is less than the maximum block size
  - Bitcoin blocks now have a theoretical maximum size of 4 megabytes and a more realistic maximum size of 2 megabytes.
  - Exact size depends on the types of transactions included.
- Output value must be in the allowed money range (0 to 21 million BTC).
- All inputs must have a specified previous output, except for coinbase transactions, which should not be relayed.

Transaction verification

- Verify that nLockTime must not exceed 31-bits.
  - **nLockTime** is a parameter of a transaction, mandates a minimal time (specified in either unix time or block height), before which the transaction cannot be accepted into a block.
  - For a transaction to be valid, it should not be less than 100 bytes.
- Number of signature operands in a standard signature should be less than or not more than 2.
- Reject *nonstandard* transactions;
  - for example, ScriptSig is allowed to only push numbers on the stack.
  - ScriptPubkey not passing the isStandard() checks.
- Transaction is rejected if there is already a matching transaction in the pool or in a block in the main branch.
- Transaction will be rejected if the referenced output for each input exists in any other transaction in the pool.

Transaction verification

- For each input, there must exist a referenced output transaction.
  - This is searched in the main branch and the transaction pool to find whether the output transaction is missing for any input, and this will be considered an orphan transaction.
  - It will be added to the **orphan transactions pool** if a matching transaction is not in the pool already.
- For each input, if the referenced output transaction is the coinbase, it must have at least **100 confirmations**; otherwise, the transaction will be rejected.
- For each input, if the referenced output does not exist or has been spent already, the transaction will be rejected.

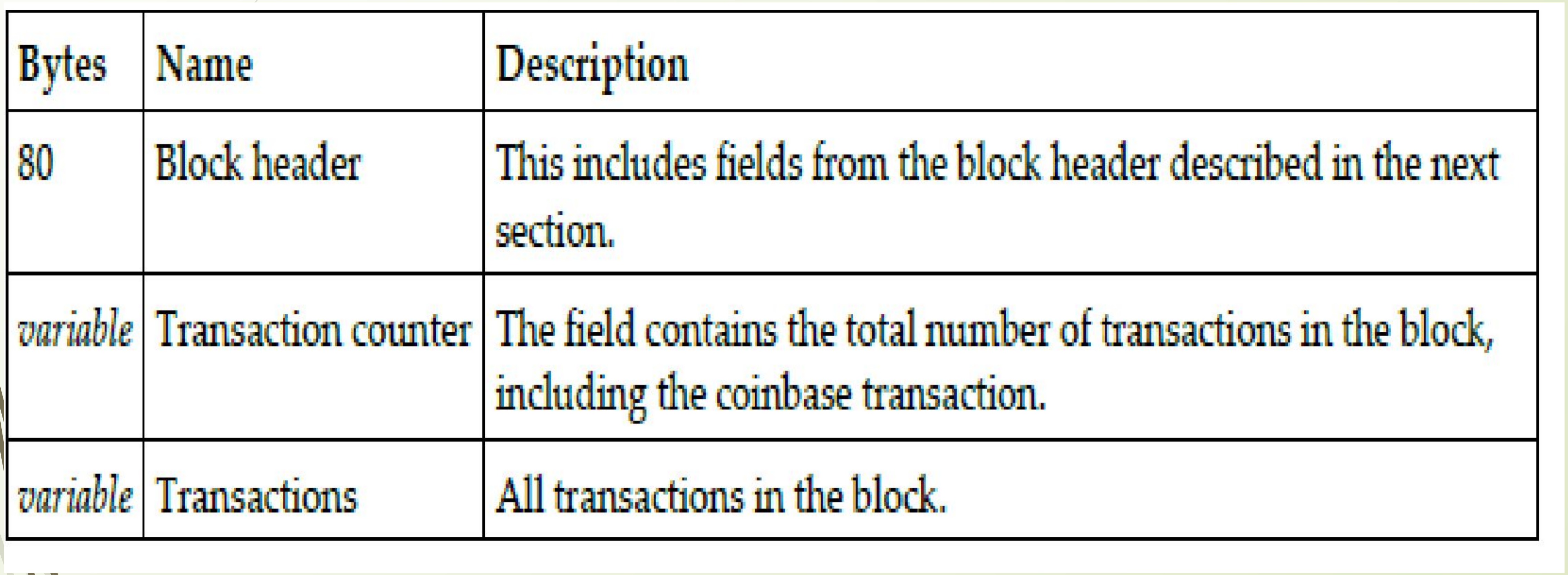
Transaction verification

- Using the referenced output transactions to get input values, verify that each input value, as well as the sum, is in the allowed range of 0-21 million BTC.
- Reject the transaction if the sum of input values is less than the sum of output values.
- Reject the transaction if the transaction fee would be too low to get into an empty block.

Blockchain

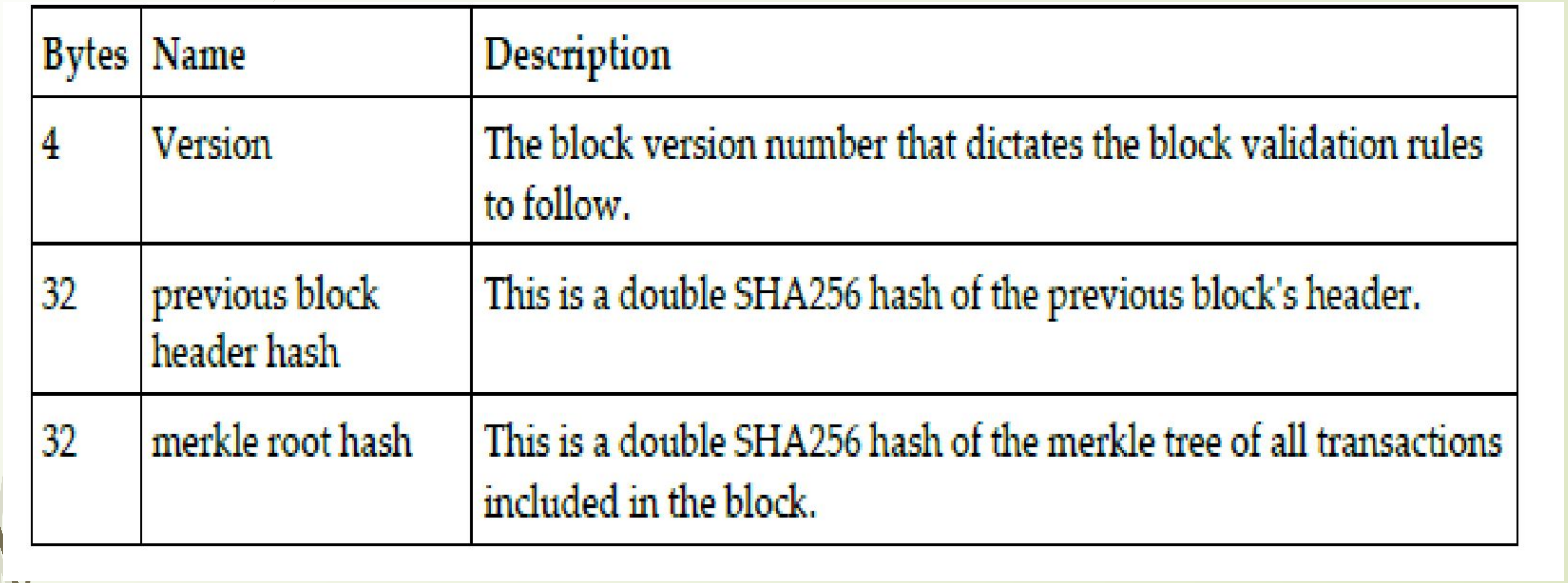
- Blockchain is a public ledger of a timestamped, ordered, and immutable list of all transactions on the bitcoin network.
- Each block is identified by a hash in the chain and is linked to its previous block by referencing the previous block's hash.

The structure of a block

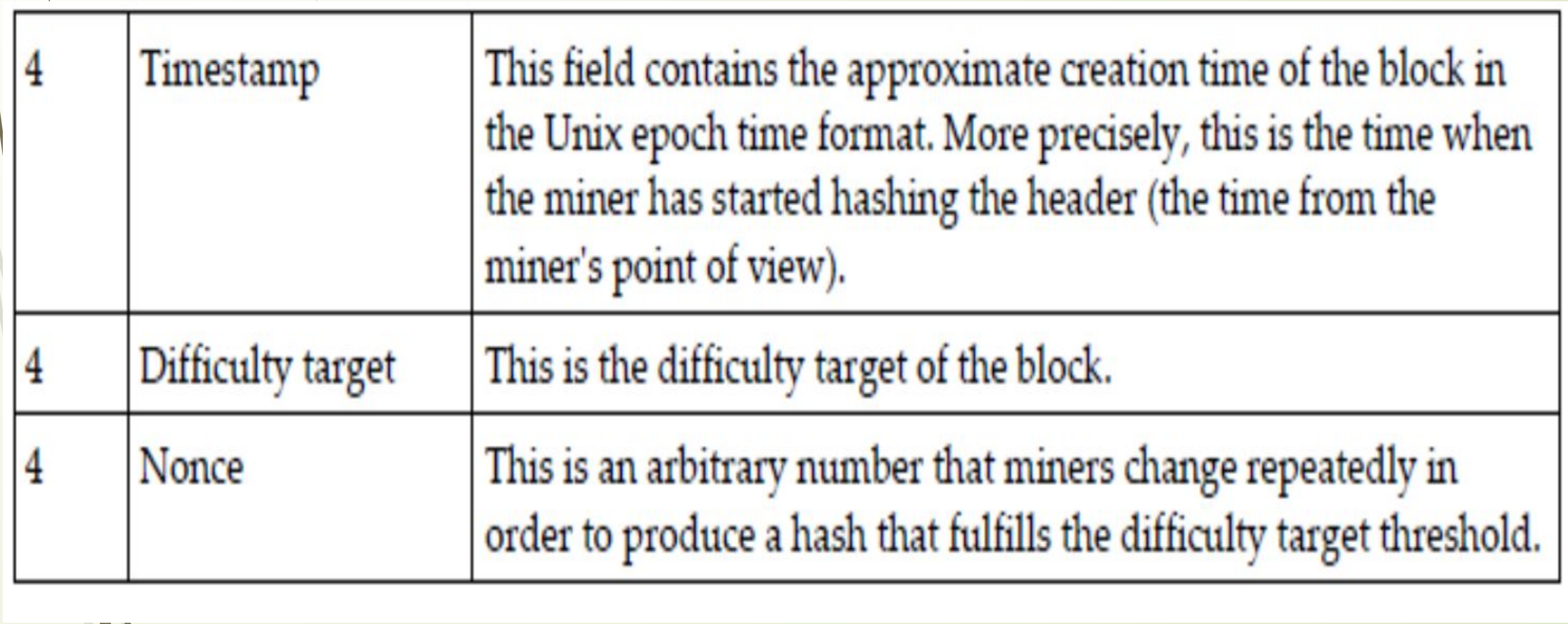


Bytes	Name	Description
80	Block header	This includes fields from the block header described in the next section.
variable	Transaction counter	The field contains the total number of transactions in the block, including the coinbase transaction.
variable	Transactions	All transactions in the block.

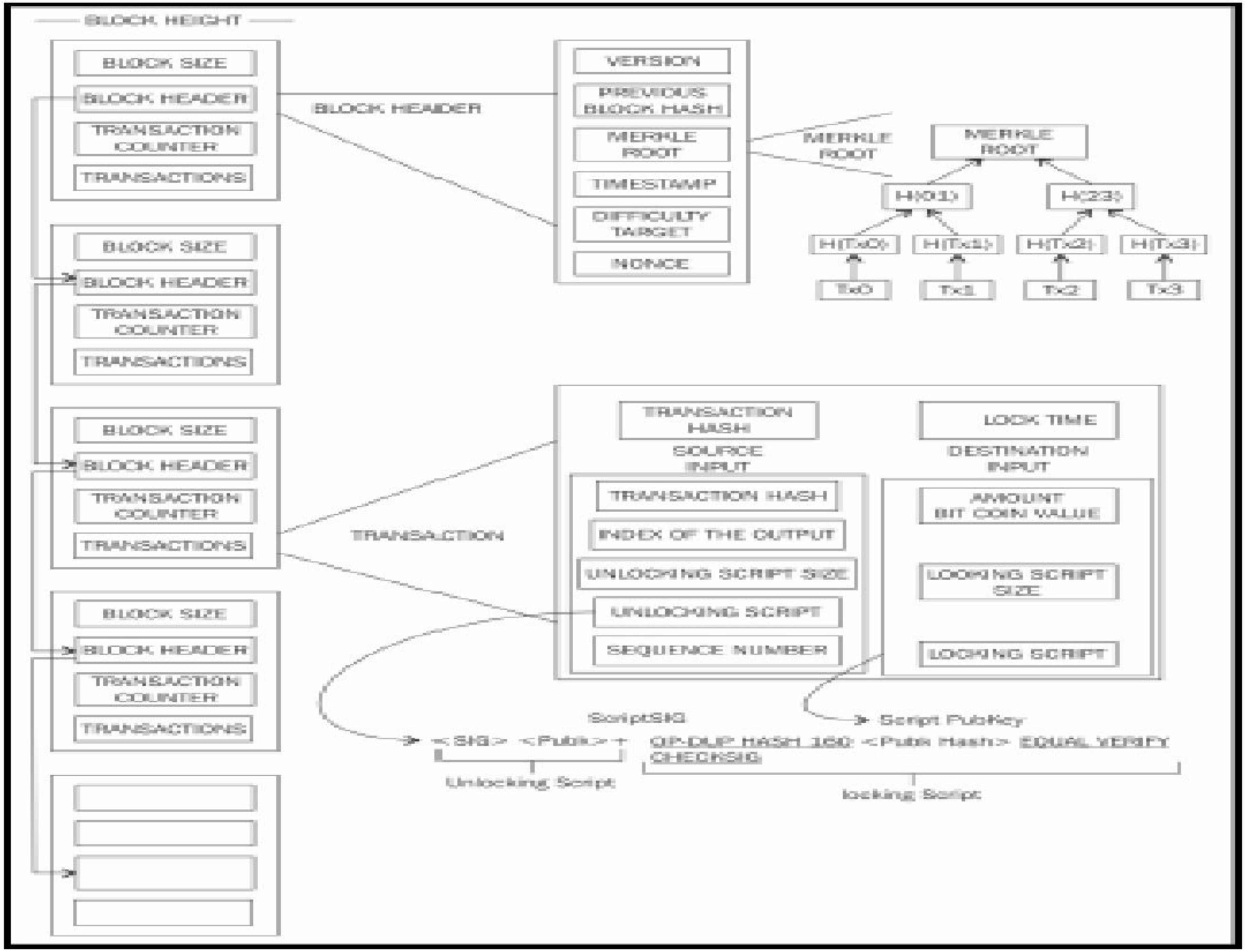
The structure of a block header



Bytes	Name	Description
4	Version	The block version number that dictates the block validation rules to follow.
32	previous block header hash	This is a double SHA256 hash of the previous block's header.
32	merkle root hash	This is a double SHA256 hash of the merkle tree of all transactions included in the block.



4	Timestamp	This field contains the approximate creation time of the block in the Unix epoch time format. More precisely, this is the time when the miner has started hashing the header (the time from the miner's point of view).
4	Difficulty target	This is the difficulty target of the block.
4	Nonce	This is an arbitrary number that miners change repeatedly in order to produce a hash that fulfills the difficulty target threshold.


BLOCK HEIGHT
BLOCK SIZE
BLOCK HEADER
TRANSACTION COUNTER
TRANSACTIONS
BLOCK HEADER
VERSION
PREVIOUS BLOCK HASH
MERKLE ROOT
TIMESTAMP
DIFFICULTY TARGET
NONCE
MERKLE ROOT
MERKLE ROOT
H(01)
H(23)
H(Tx0)
H(Tx1)
H(Tx2)
H(Tx3)
Tx0
Tx1
Tx2
Tx3
TRANSACTION
TRANSACTION HASH
LOCK TIME
SOURCE INPUT
DESTINATION INPUT
TRANSACTION HASH
INDEX OF THE OUTPUT
UNLOCKING SCRIPT SIZE
UNLOCKING SCRIPT
SEQUENCE NUMBER
AMOUNT BIT COIN VALUE
LOCKING SCRIPT SIZE
LOCKING SCRIPT
ScriptSIG
Script PubKey
<SIG> <PubK> +
OP-DUP HASH 160 <PubK Hash> EQUAL VERIFY CHECKSIG
Unlocking Script
locking Script

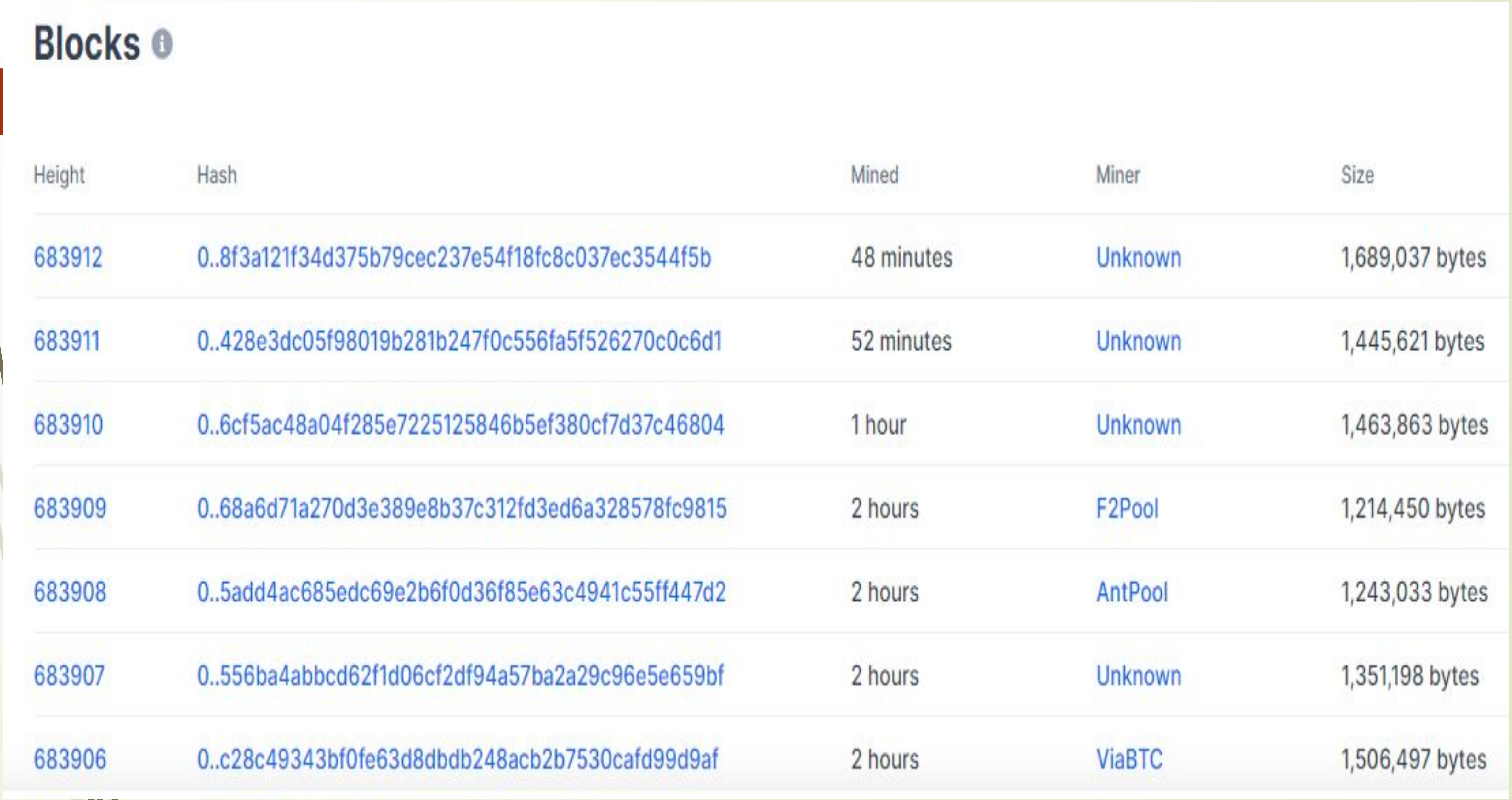
- Blockchain is a chain of blocks where each block is linked to its previous block by referencing the previous block header's hash.
- This linking makes sure that no transaction can be modified unless the block that records it and all blocks that follow it are also modified.
- The first block is not linked to any previous block and is known as the genesis block.

The genesis block

- First block in the bitcoin blockchain.
- Genesis block was hardcoded in the bitcoin core software.
  - chainparams.cpp file
- Every node always starts with a blockchain of at least one block
  - genesis block is statically encoded within the bitcoin client software, such that it cannot be altered.
- Every node always "knows" the genesis block’s hash and structure, the fixed time it was created, and even the single transaction within.
- Every node has the starting point for the blockchain, a secure "root" from which to build a trusted blockchain.

Blockchain

- Bitcoin provides protection against double spending by enforcing strict rules on transaction verification and via mining.
- Blocks are added in the blockchain only after strict rule checking and successful Proof of Work solution.
- Block height is the number of blocks before a particular block in the blockchain.
- The current height of the blockchain is **683912** blocks.
- Proof of Work is used to secure the blockchain.
- Each block contains one or more transactions, out of which the first transaction is a coinbase transaction.
- There is a special condition for coinbase transactions that prevent them to be spent until at least 100 blocks in order to avoid a situation where the block may be declared stale later on



Blocks

Height	Hash	Mined	Miner	Size
683912	0..8f3a121f34d375b79cec237e54f18fc8c037ec3544f5b	48 minutes	Unknown	1,689,037 bytes
683911	0..428e3dc05f98019b281b247f0c556fa5f526270c0c6d1	52 minutes	Unknown	1,445,621 bytes
683910	0..6cf5ac48a04f285e7225125846b5ef380cf7d37c46804	1 hour	Unknown	1,463,863 bytes
683909	0..68a6d71a270d3e389e8b37c312fd3ed6a328578fc9815	2 hours	F2Pool	1,214,450 bytes
683908	0..5add4ac685edc69e2b6f0d36f85e63c4941c55ff447d2	2 hours	AntPool	1,243,033 bytes
683907	0..556ba4abbcd62f1d06cf2df94a57ba2a29c96e5e659bf	2 hours	Unknown	1,351,198 bytes
683906	0..c28c49343bf0fe63d8dbdb248acb2b7530cafd99d9af	2 hours	ViaBTC	1,506,497 bytes

Stale blocks

- **Stale blocks** are created when a block is solved and every other miner who is still working to find a solution to the hash puzzle is working on that block
- As the block is no longer required to be worked on, this is considered a stale block.
- **Orphan blocks** are also called detached blocks and were accepted at one point in time by the network as valid blocks
  - but were rejected when a proven longer chain was created that did not include this initially accepted block.
- They are not part of the main chain and can occur at times when two miners manage to produce the blocks at the same time.

Network difficulty

- New blocks are added to the blockchain approximately every 10 minutes
- Network difficulty is adjusted dynamically every 2016 blocks in order to maintain a steady addition of new blocks to the network.
- Network difficulty is calculated using the following equation:
  - *Target = Previous target * Time/2016 * 10 minutes*
- Difficulty and target are interchangeable and represent the same thing.
- Previous target represents the old target value,
- Time is the time spent to generate previous 2016 blocks.
- Network difficulty basically means how hard it is for miners to find a new block,
  - how difficult the hashing puzzle is now.

Mining

- Mining is a resource-intensive process by which new blocks are added to the blockchain
- Blocks contain transactions that are validated via the mining process by mining nodes and are added to the blockchain.
- Process is resource-intensive in order to ensure that the required resources have been spent by miners in order for a block to be accepted.
- New coins are minted by the miners by spending the required computing resources.
- Secures the system against frauds and double spending attacks while adding more virtual currency to the bitcoin ecosystem.

Mining

- Roughly one new block is created (mined) every 10 minute.
- Miners are rewarded with new coins if and when they create new blocks and are paid transaction fees in return of including transactions in their blocks.
- New blocks are created at an approximate fixed rate.
- Also, the rate of creation of new bitcoins decreases by 50%, every 210,000 blocks, roughly every 4 years.
- When bitcoin was initially introduced, the block reward was 50 bitcoins;
  - Then in 2012, this was reduced to 25 bitcoins. In July 2016, this was further reduced to 12.5 coins (12 coins)
  - Currently, miners get 6.25 BTC per valid block mined.

Mining

- Approximately 144 blocks, that is, 1,728 bitcoins are generated per day.
- The number of actual coins can vary per day; however, the number of blocks remains at 144 per day.
- Bitcoin supply is also limited and in 2140, almost 21 million bitcoins will be finally created and no new bitcoins can be created after that.
- Bitcoin miners, however, will still be able to profit from the ecosystem by charging transaction fees.

Task of miners- Synching up with the network

- Once a new node joins the bitcoin network,
  - It downloads the blockchain by requesting historical blocks from other nodes.
- **Transaction validation**:
  - Transactions broadcasted on the network are validated by full nodes by verifying and validating signatures and outputs.
- **Block validation**:
  - Miners and full nodes can start validating blocks received by them by evaluating them against certain rules.
  - This includes the verification of each transaction in the block along with verification of the nonce value.

Task of miners- Synching up with the network

- **Create a new block**:
  - Miners propose a new block by combining transactions broadcasted on the network after validating them.
- **Perform Proof of Work**:
  - This task is the core of the mining process
  - Miners find a valid block by solving a computational puzzle.
  - Block header contains a 32-bit nonce field
  - Miners are required to repeatedly vary the nonce until the resultant hash is less than a predetermined target.

Task of miners- Synching up with the network

- **Fetch reward**:
  - Once a node solves the hash puzzle,
  - it immediately broadcasts the results, and other nodes verify it and accept the block.
  - There is a slight chance that the newly minted block will not be accepted by other miners due to a clash with another block found at roughly the same time,
  - Once accepted, the miner is rewarded with 6.25 bitcoins (as of 2021) and any associated transaction fees.

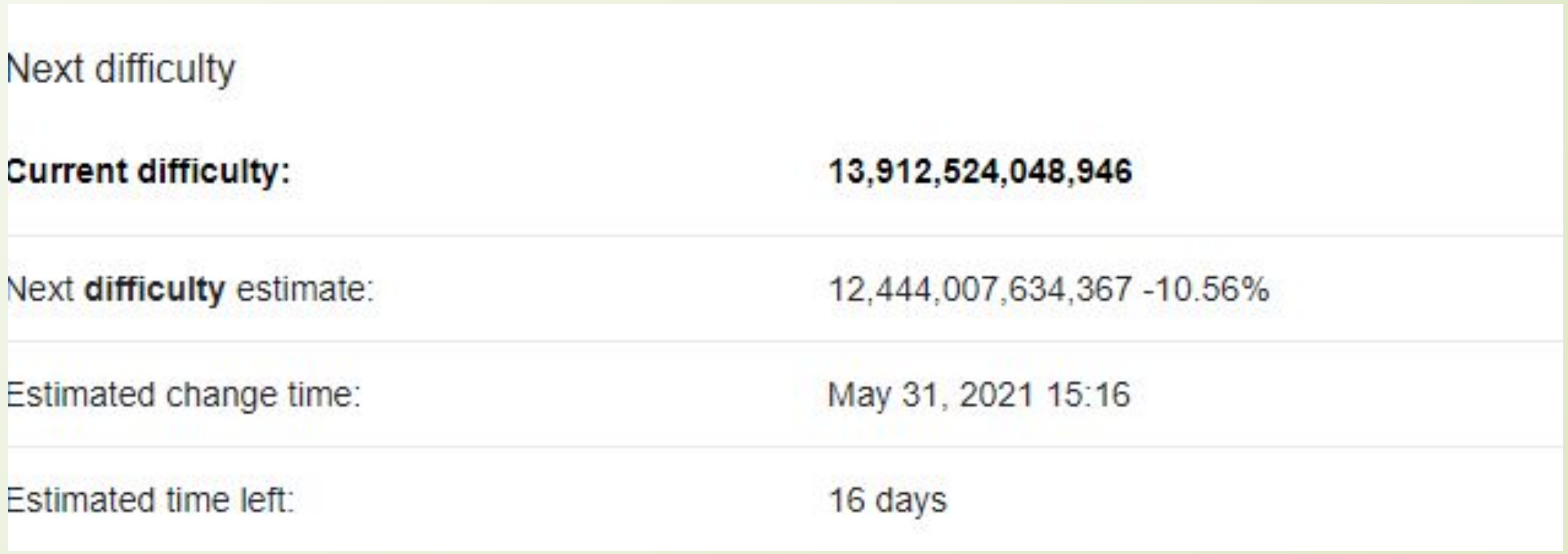
Proof of Work

- Proof that enough computational resources have been spent in order to build a valid block.
- **Proof of Work (PoW)** is based on the idea that a random node is selected every time to create a new block.
- In this model, nodes compete with each other in order to be selected in proportion to their computing capacity.
- The following equation sums up the Proof of Work requirement in bitcoin:
  - $H\ (N\ ||\ P\_hash\ ||\ Tx\ ||\ Tx\ ||\ ...\ Tx) < Target$
  - Where N is a nonce,
  - P_hash is a hash of the previous block
  - Tx represents transactions in the block,
  - Target is the target network difficulty value.
- The only way to find this nonce is the brute force method.

The mining algorithm

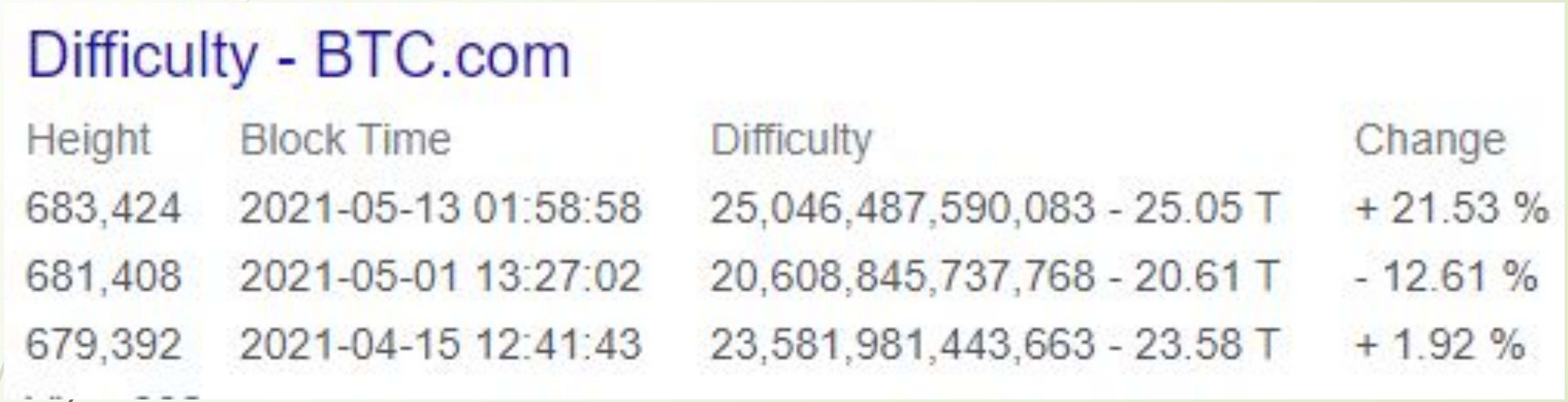
- The previous hash block is retrieved from the bitcoin network.
- Assemble a set of potential transactions broadcasted on the network into a block.
- Compute the double hash of the block header with a nonce and the previous hash using the SHA256 algorithm.
- If the resultant hash is lower than the current difficulty level (target), then stop the process.
- If the resultant hash is greater than the current difficulty level (target), then repeat the process by incrementing the nonce
- Mining difficulty increased over time and bitcoins that could be mined by single CPU laptop computers now require dedicated mining centers to solve the hash puzzle.

- The current difficulty level can be queried using the bitcoin command line interface using the following command:
  - **$ bitcoin-cli getdifficulty**



Next difficulty

Current difficulty:	**13,912,524,048,946**
Next **difficulty** estimate:	12,444,007,634,367 -10.56%
Estimated change time:	May 31, 2021 15:16
Estimated time left:	16 days

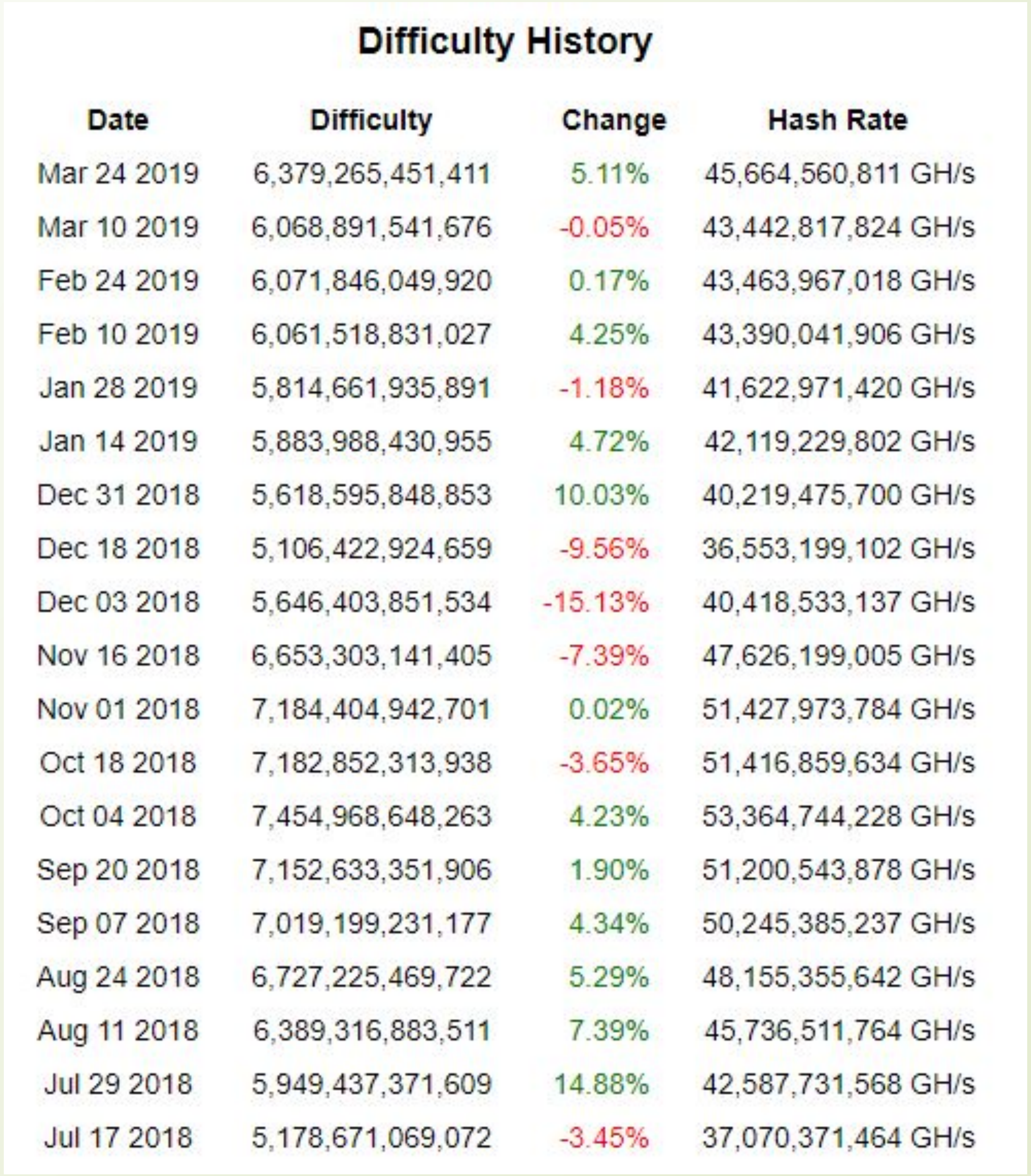


Difficulty - BTC.com

Height	Block Time	Difficulty	Change
683,424	2021-05-13 01:58:58	25,046,487,590,083 - 25.05 T	+ 21.53 %
681,408	2021-05-01 13:27:02	20,608,845,737,768 - 20.61 T	- 12.61 %
679,392	2021-04-15 12:41:43	23,581,981,443,663 - 23.58 T	+ 1.92 %

Hashing rate

- Hashing rate basically represents the rate of calculating hashes per second.
- In early days of bitcoin, it used to be quite small as CPUs were used
- With dedicated mining pools and ASICs now, this has gone up exponentially in the last few years.
  - Application-Specific Integrated Circuit (ASIC)
- This has resulted in increased difficulty.
- Hash rate increase over time and is currently measured in Exa hashes.
  - This means that in 1 second, bitcoin network miners are computing more than 1 000 000 000 000 000 000 hashes per second.

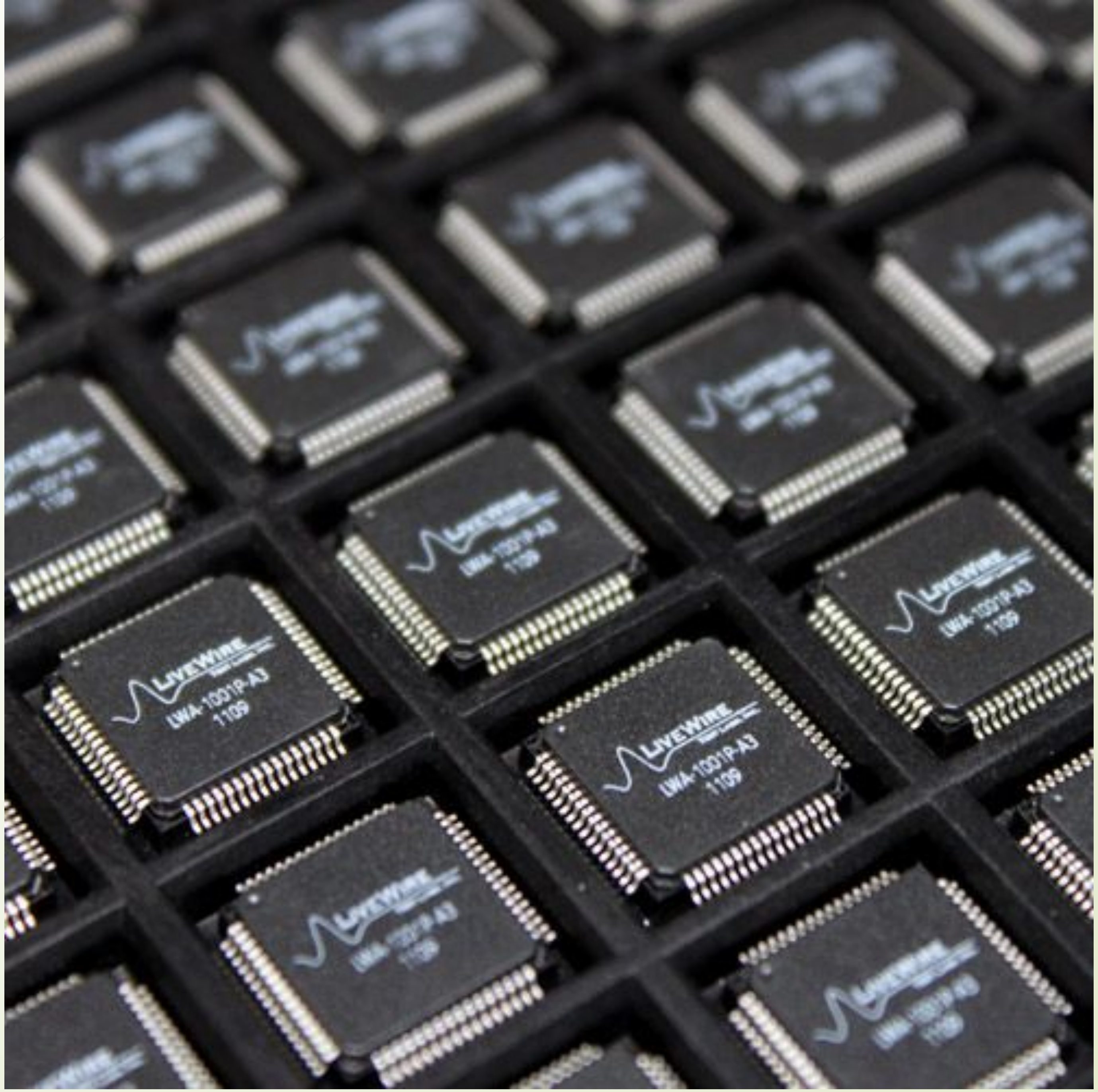


Difficulty History

Date	Difficulty	Change	Hash Rate
Mar 24 2019	6,379,265,451,411	5.11%	45,664,560,811 GH/s
Mar 10 2019	6,068,891,541,676	-0.05%	43,442,817,824 GH/s
Feb 24 2019	6,071,846,049,920	0.17%	43,463,967,018 GH/s
Feb 10 2019	6,061,518,831,027	4.25%	43,390,041,906 GH/s
Jan 28 2019	5,814,661,935,891	-1.18%	41,622,971,420 GH/s
Jan 14 2019	5,883,988,430,955	4.72%	42,119,229,802 GH/s
Dec 31 2018	5,618,595,848,853	10.03%	40,219,475,700 GH/s
Dec 18 2018	5,106,422,924,659	-9.56%	36,553,199,102 GH/s
Dec 03 2018	5,646,403,851,534	-15.13%	40,418,533,137 GH/s
Nov 16 2018	6,653,303,141,405	-7.39%	47,626,199,005 GH/s
Nov 01 2018	7,184,404,942,701	0.02%	51,427,973,784 GH/s
Oct 18 2018	7,182,852,313,938	-3.65%	51,416,859,634 GH/s
Oct 04 2018	7,454,968,648,263	4.23%	53,364,744,228 GH/s
Sep 20 2018	7,152,633,351,906	1.90%	51,200,543,878 GH/s
Sep 07 2018	7,019,199,231,177	4.34%	50,245,385,237 GH/s
Aug 24 2018	6,727,225,469,722	5.29%	48,155,355,642 GH/s
Aug 11 2018	6,389,316,883,511	7.39%	45,736,511,764 GH/s
Jul 29 2018	5,949,437,371,609	14.88%	42,587,731,568 GH/s
Jul 17 2018	5,178,671,069,072	-3.45%	37,070,371,464 GH/s

Mining systems

- **CPU**
- **GPU**
- **Field Programmable Gate Array (FPGA)** is basically an integrated circuit that can be programmed to perform specific operations
  - FPGA offered much better performance as compared to GPUs;
  - However, issues such as accessibility, programming difficulty, and the requirement for specialized knowledge to program and configure FPGAs resulted in a **short life of the FPGA** era for bitcoin mining.
- **Application Specific Integrated Circuit (ASIC)** was designed to perform the SHA-256 operation
- Currently, professional mining centers using thousands of ASIC units in parallel are offering mining contracts to users to perform mining on their behalf.


LIVEWIRE
LWA-1001P-A3
1109

Mining pools

- Mining pool forms when group miners work together to mine a block.
- The *Pool manager* receives the **coinbase transaction** if the block is successfully mined
  - which is then responsible for distributing the reward to the group of miners who invested resources to mine the block.
- Profitable as compared to solo mining,
  - where only one sole miner is trying to solve the partial hash inversion function.

The bitcoin network

- The bitcoin network is a P2P network where nodes exchange transactions and blocks.
- There are two main types of nodes- **full nodes and SPV nodes**.
- **Full nodes**, as the name implies, are implementations of bitcoin core clients performing the **wallet, miner, full blockchain storage, and network routing functions**.
  - it is not necessary to perform all these functions.
- **SPV nodes** or lightweight clients perform only **wallet and network routing functionality**.
  - Simplified Payment Verification(SPV)

The bitcoin network

- Some nodes prefer to be full blockchain nodes only
  - contain complete blockchain
  - perform network routing functions
  - but do not perform mining or store private keys (the wallet function).
- Another type is solo miner nodes
  - that can perform mining
  - store full blockchain
  - act as a bitcoin network routing node.

The bitcoin network

- Few nonstandard but heavily used nodes that are called **pool protocol servers**.
  - These nodes make use of alternative protocols, such as the stratum protocol.
- Some nodes perform only mining functions and are called **mining nodes.**
- Nodes that only compute hashes use the **stratum protocol** to submit their solutions to the mining pool.
- It is possible to run an SPV client runs a wallet and network routing function without a blockchain.
- Most protocols on the Internet are line-based, which means that each line is delimited by a carriage return and newline \r \n character.

The bitcoin network

- Stratum is also a **line-based protocol** that makes use of plain TCP sockets and human-readable JSON-RPC to operate and communicate between nodes.
- When a bitcoin core node starts up, first, it initiates the discovery of all peers.
  - This is achieved by **querying DNS seeds** that are hardcoded into the bitcoin core client and are maintained by bitcoin community members.
  - This lookup returns a number of DNS A records.
- The bitcoin protocol works on TCP port 8333 by default for the main network and TCP 18333 for testnet.

The bitcoin network

- First, the client sends a **protocol message *Version*** that contains various fields, such as version, services, timestamp, network address, nonce, and some other fields.
- The remote node responds with its own version message followed by **verack message** exchange between both nodes, indicating that the connection has been established.
- **Getaddr** and **addr** messages are exchanged to find the peers.
- Either of the nodes can send a ping message to see whether the connection is still live.

The bitcoin network

- Now the block download can begin.
  - If the node already has all blocks fully synchronized, then it listens for new blocks using *the **Inv protocol message***;
  - otherwise, it first checks whether it has a **response to *inv* messages** and have inventories already.
    - If yes, then it requests the blocks using the **Getdata** protocol message;
    - if not, then it requests inventories using the ***GetBlocks*** message

Types of protocol messages

- There are 27 types of protocol messages in total, but they're likely to increase over time as the protocol grows
- **Version**:
  - This is the first message that a node sends out to the network, advertising its version and block count.
  - The remote node then replies with the same information and the connection is then established.
- **Verack :**
  - This is the response of the version message accepting the connection request.
- **Inv:**
  - This is used by nodes to advertise their knowledge of blocks and transactions.

- **Getdata :**
  - This is a response to inv, requesting a single block or transaction identified by its hash.
- **Getblocks:**
  - This returns an *inv* packet containing the list of all blocks starting after the last known hash or 500 blocks.
- **Getheaders :**
  - This is used to request block headers in a specified range.
- **Tx :**
  - This is used to send a transaction as a response to the getdata protocol message.
- **Block:**
  - This sends a block in response to the *getdata* protocol message.

- **Headers:**
  - This packet returns up to 2,000 block headers as a reply to the getheaders request.
- **Getaddr:**
  - This is sent as a request to get information about known peers.
- **Addr:**
  - This provides information about nodes on the network.
  - It contains the number of addresses and address list in the form of IP address and port number.

- **Full client and SPV client**:
  - Full clients are thick clients or full nodes that download the entire blockchain;
  - This is the most secure method of validating the blockchain as a client.
  - Bitcoin network nodes can operate in two fundamental modes:
    - full client or lightweight SPV client.
  - SPV clients are used to verify payments without requiring the download of a full blockchain.
  - SPV nodes only keep a copy of block headers of the current valid longest blockchain

- **Bloom filters**:
  - Bloom filter is basically a data structure (a bit vector with indexes) that is used to test the membership of an element in a probabilistic manner.
  - These filters are mainly used by simple payment verification SPV clients to request transactions and the merkle blocks they are interested in
- **BIP 37** proposed the bitcoin implementation of bloom filters and introduced three new messages to the Bitcoin protocol.
- **Filterload:** This is used to set the bloom filter on the connection.
- **Filteradd:** This adds a new data element to the current filter.
- **FilterClear**: This deletes the currently loaded filter.

Wallets

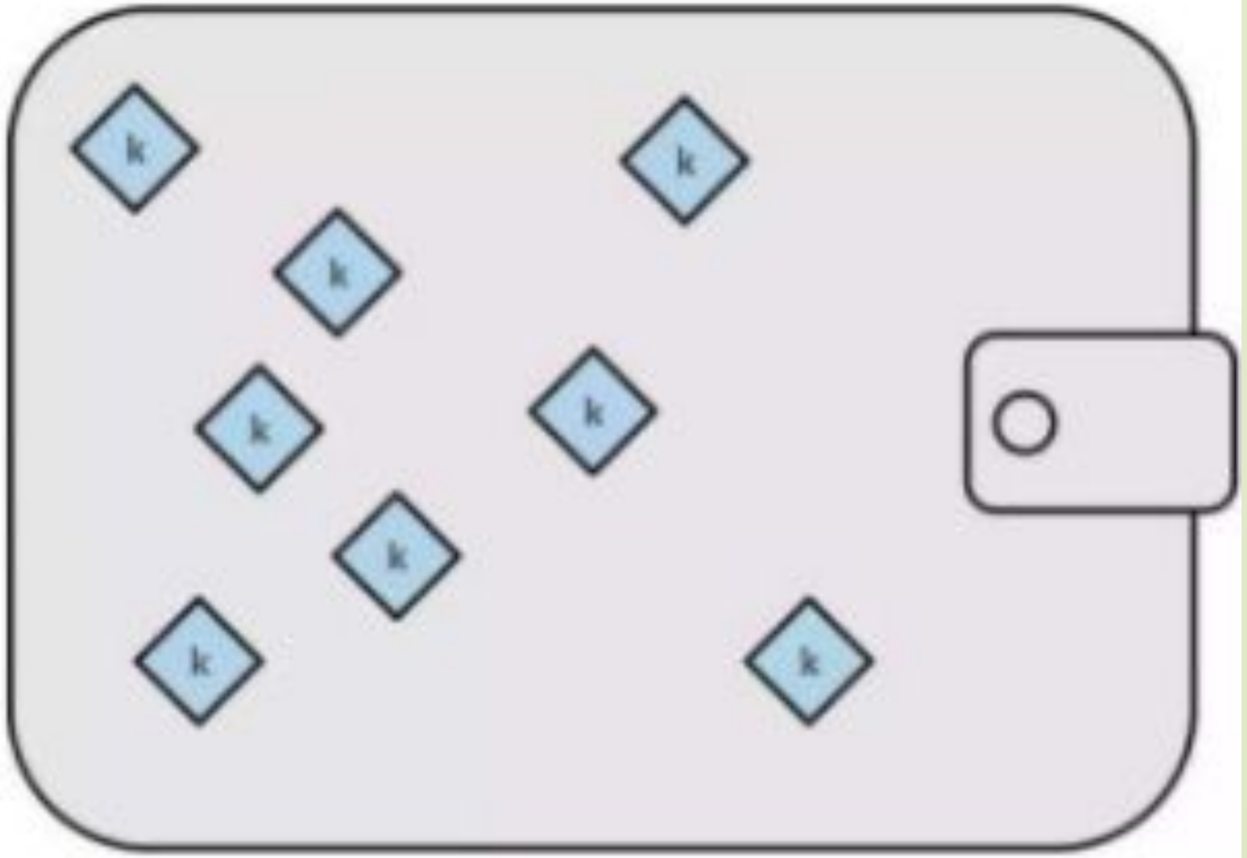
- The wallet software is used to store private or public keys and bitcoin address.
- It performs various functions, such as receiving and sending bitcoins.
- On the disk, the bitcoin core client wallets are stored as the Berkeley DB file
- Private keys can be generated in different ways and are used by different types of wallets.
- Wallets do not store any coins, and there is no concept of wallets storing balance or coins for a user.
- In fact, in the bitcoin network, only transaction information is stored on the blockchain (UTXO, unspent outputs),
  - which are then used to calculate the amount of bitcoins.

Wallet types

- **Non-deterministic wallets**
  - Wallets contain randomly generated private keys and are also called *Just a Bunch of Key wallets*.
  - Bitcoin core client generates some keys when first started and generates keys as and when required.
  - Managing a large number of keys is very difficult and an errorprone process can lead to theft and loss of coins.
  - There is a need to create regular backups of the keys and protect them appropriately in order to prevent theft or loss.

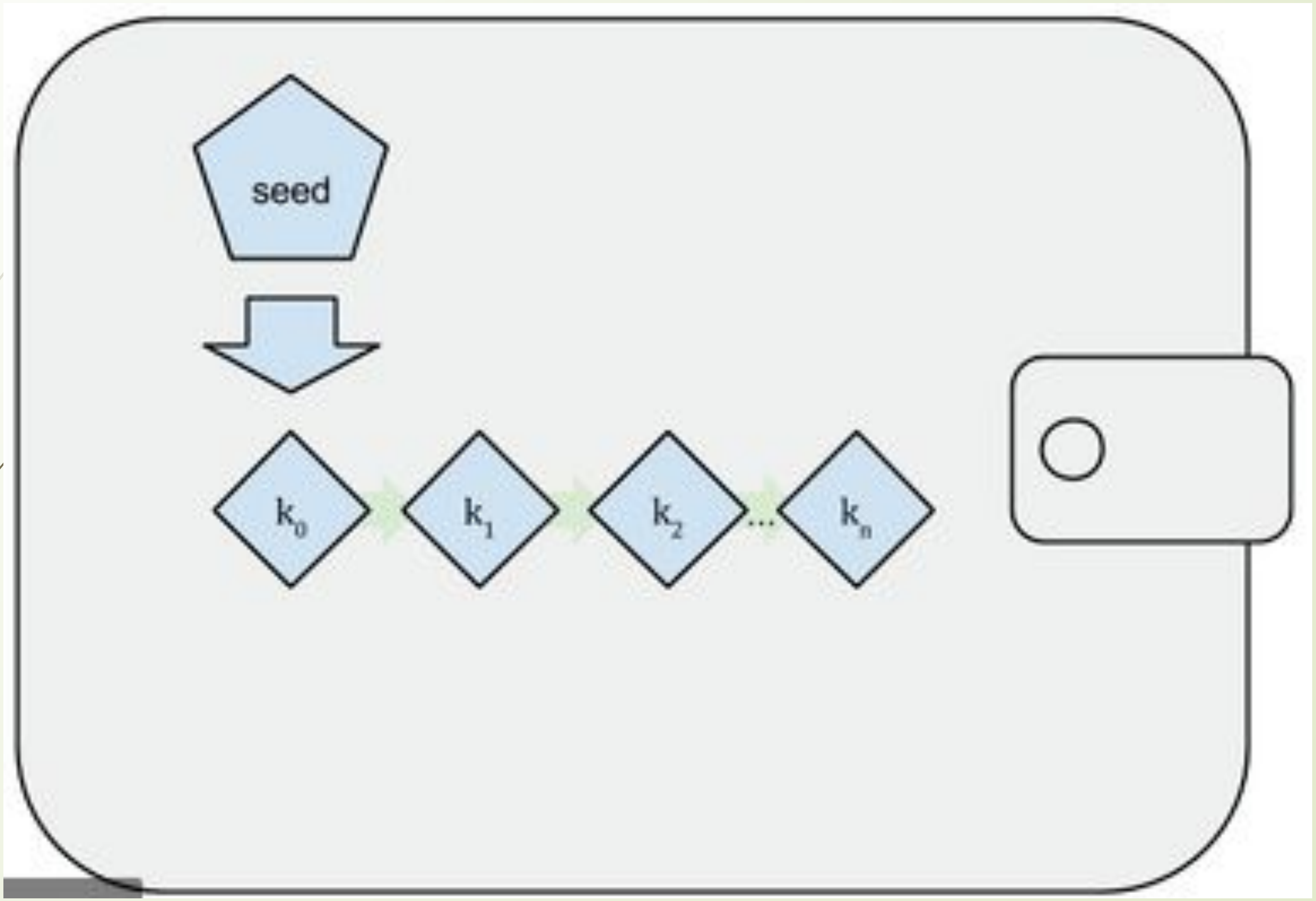
Nondeterministic (Random) Wallets

- Wallets were simply collections of randomly generated private keys.
- This type of wallet is nicknamed "Just a Bunch Of Keys" (JBOK).



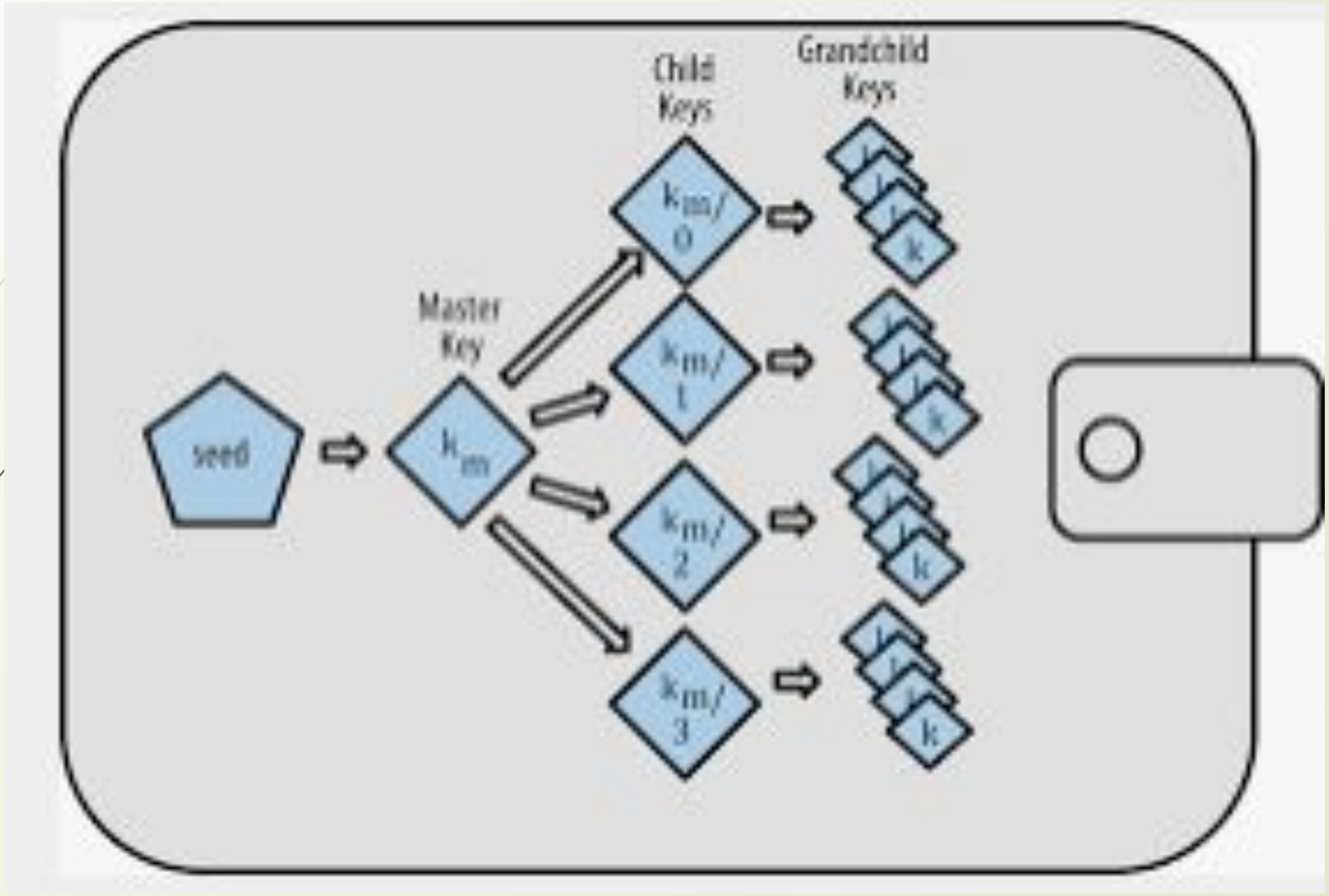
Wallet types

- **Deterministic wallets**
  - In this type of wallet, keys are derived out of a seed value via hash functions.
  - seed number is generated randomly and is commonly represented by human-readable *mnemonic code* words.
  - Mnemonic code words are defined in BIP39.
    - This phrase can be used to recover all keys and makes private key management comparatively easier.


seed
k_0
k_1
k_2
...
k_n

Wallet types

- **Hierarchical deterministic wallets**
  - Defined in BIP32 and BIP44
  - HD wallets store keys in a tree structure derived from a seed.
  - Seed generates the parent key (master key), which is used to generate child keys and, subsequently, grandchild keys.
  - Key generation in HD wallets does not generate keys directly;
    - instead, it produces some information (private key generation information) that can be used to generate a sequence of private keys.
  - Complete hierarchy of private keys in an HD wallet is easily recoverable if the master private key is known.
  - This property that HD wallets are very easy to maintain and are highly portable.


Child Keys
Grandchild Keys
Master Key
seed
k_m
k_m/0
k_m/1
k_m/2
k_m/3
k
k
k
k

Wallet types

- **Brain wallets**
  - Master private key can also be derived from the hash of passwords that are memorized.
  - Key idea is that this passphrase is used to derive the private key and if used in HD wallets, this can result in a full HD wallet that is derived from a single memorized password.
  - Method is prone to password guessing and brute force attacks
    - Techniques such as *key stretching* can be used to slow down the progress made by the attacker.

Wallet types

- **Paper wallets**
  - Paper-based wallet with the required key material printed on it.
  - Requires physical security to be stored.
  - Paper wallets can be generated online from various service providers
- **Hardware wallets**
  - Use a tamper-resistant device to store keys.
  - Tamper-resistant device can be custom-built or with the advent of NFC-enabled phones, this can also be a **secure element** (**SE**) in NFC phones.
  - Trezor and Ledger walletsare the most commonly used bitcoin hardware wallets.

bitcoin
LOAD & VERIFY
SPEND
BITCOIN ADDRESS
PRIVATE KEY

OPENDIME
keep key

Wallet types

- **Online wallets**
  - Stored entirely online and are provided as a service usually via cloud.
  - Provide a web interface to the users to manage their wallets and perform various functions such as making and receiving payments.
  - Easy to use but imply that the user trust the online wallet service provider.
- **Mobile wallets**
  - Mobile wallets, as the name suggests, are installed on mobile devices.
  - Provide various methods to make payments, most notably the ability to use smart phone cameras to scan QR codes quickly and make payments.
  - Mobile wallets are available for the Android platform and iOS, for example, breadwallet, copay, and Jaxx.

References

- Imran Bashir. “Mastring BlockChain”, Packt
- Web Materials