Blockchain Technology (BIS613A)

MODULE-2
Bitcoin Blockchain-1

Syllabus

Module-2

History of Bitcoin, Transactions, Blocks, Mining and the Blockchain, Bitcoin Transactions, Constructing a Transaction, Bitcoin Mining, Mining Transaction in Blocks, Spending the Transaction, Bitcoin Addresses, Wallets, Transactions: Transaction Lifecycle, Transaction Structure, Transaction Outputs and Inputs, Transaction Chaining and Orphan Transactions, Transaction Scripts and Script Language, Standard Transactions- Pay to Public Key Hash (P2PKH)

Double Spending Problem

- Double-spending is a potential flaw in a digital cash scheme in which the same single digital token can be spent more than once.
- Unlike physical cash, a digital token consists of a digital file that can be duplicated or falsified.
- As with counterfeit money, such double-spending leads to inflation by creating a new amount of copied currency that did not previously exist.

Double Spending Problem - Solutio

- Two types of solutions to the Double Spend Problem have arisen to maintain trust in digital money.
- The first relies on trusted central authorities to prevent double spends and other types of fraud.
- The second rejects central authorities and allows any individual to check all transactions for double spends

Double Spending Problem - Solutio

Trusted Third Parties

- In order to prevent fraudulent transactions such as double spends, certain institutions are entrusted to verify all transactions privately.
- These institutions include payment processors, banks, Automated Clearing Houses, and ultimately central banks.
- Each institution maintains its own private ledger and applies its own rules to verifying the transactions

Double Spending Problem - Solutio

A Distributed Ledger

- Bitcoin solves the double spend problem by using a decentralized ledger, which all users can access.
- Because all members of the Bitcoin network can examine the full history of transactions, they can be sure that neither their coins nor any other coins have been double spent.

Double Spending Problem - Solution

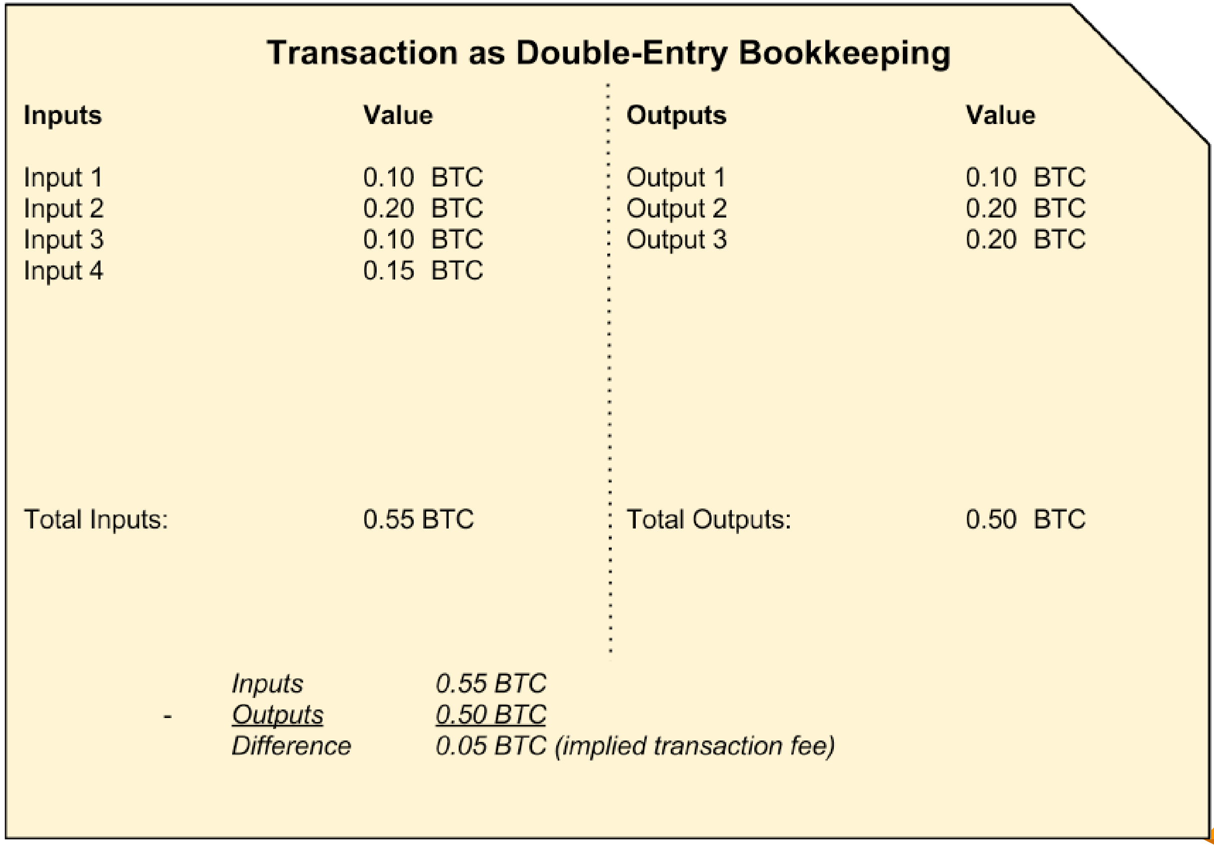
A Distributed Ledger

- Because all members of the Bitcoin network can examine the full history of transactions, they can be sure that neither their coins nor any other coins have been double spent.

The Bitcoin

Bitcoin (BTC) is a decentralized currency that eliminates the need for central authorities such as banks or governments by using a peer-to-peer internet network to confirm transactions directly between users

BTC Transactions



Transaction as Double-Entry Bookkeeping

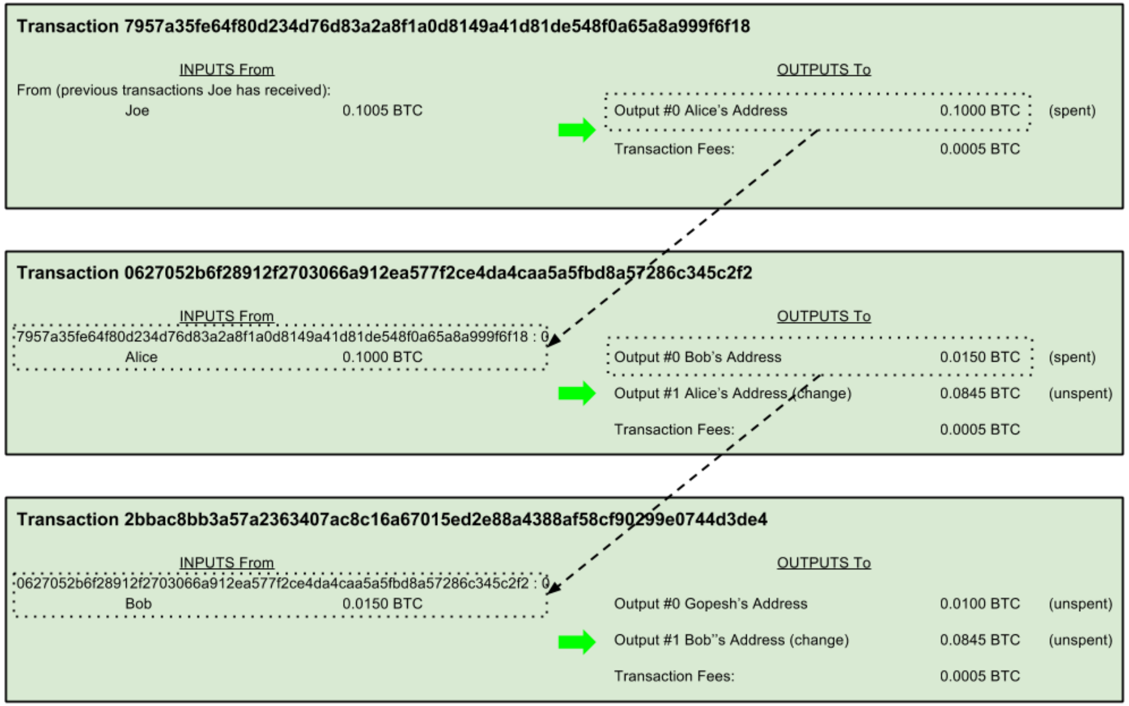
Inputs	**Value**	**Outputs**	**Value**
Input 1	0.10 BTC	Output 1	0.10 BTC
Input 2	0.20 BTC	Output 2	0.20 BTC
Input 3	0.10 BTC	Output 3	0.20 BTC
Input 4	0.15 BTC		
Total Inputs:	0.55 BTC	Total Outputs:	0.50 BTC

	Inputs	*0.55 BTC*
-	*<u>Outputs</u>*	*<u>0.50 BTC</u>*
	Difference	*0.05 BTC (implied transaction fee)*

BTC Transactions

- The transaction also contains proof of ownership for each amount of bitcoin (inputs) whose value is transferred, in the form of a digital signature from the owner, which can be independently validated by anyone.
- An input is where the coin value is coming from, usually a previous transaction's output.

BTC Transactions



Transaction 7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18

INPUTS From		OUTPUTS To		
From (previous transactions Joe has received):				
Joe	0.1005 BTC	Output #0 Alice's Address	0.1000 BTC	(spent)
		Transaction Fees:	0.0005 BTC	

Transaction 0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

INPUTS From		OUTPUTS To		
7957a35fe64f80d234d76d83a2a8f1a0d8149a41d81de548f0a65a8a999f6f18 : 0				
Alice	0.1000 BTC	Output #0 Bob's Address	0.0150 BTC	(spent)
		Output #1 Alice's Address (change)	0.0845 BTC	(unspent)
		Transaction Fees:	0.0005 BTC	

Transaction 2bbac8bb3a57a2363407ac8c16a67015ed2e88a4388af58cf90299e0744d3de4

INPUTS From		OUTPUTS To		
0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2 : 0				
Bob	0.0150 BTC	Output #0 Gopesh's Address	0.0100 BTC	(unspent)
		Output #1 Bob''s Address (change)	0.0845 BTC	(unspent)
		Transaction Fees:	0.0005 BTC	

BTC Transactions

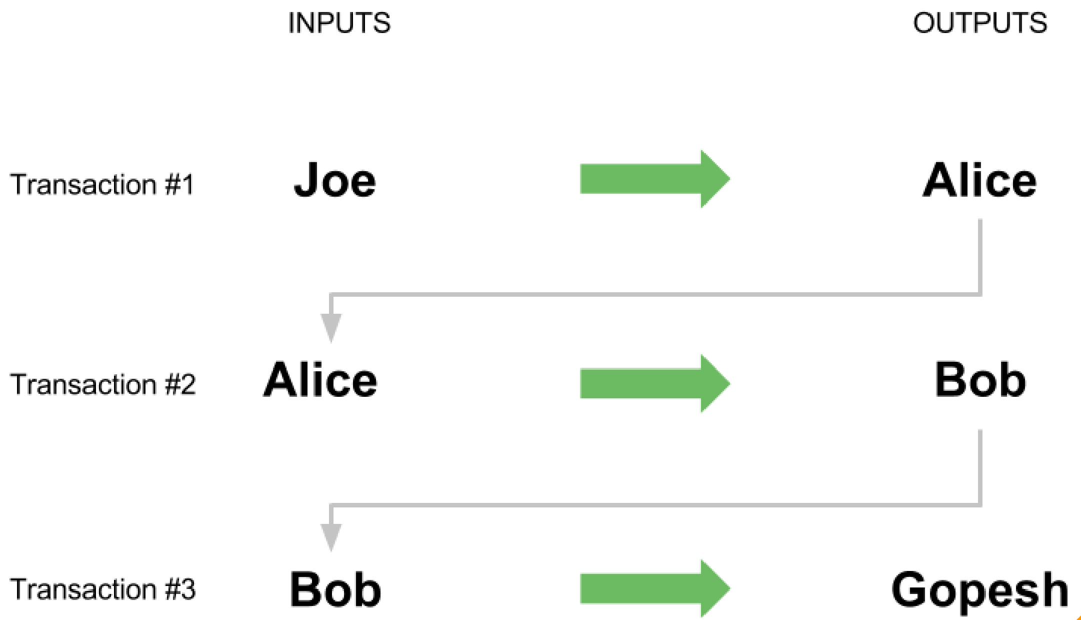
- The transaction also contains proof of ownership for each amount of bitcoin (inputs) whose value is transferred, in the form of a digital signature from the owner, which can be independently validated by anyone.
- An input is where the coin value is coming from, usually a previous transaction's output.

Bitcoin Transactions- Types

- The common transactions
- Aggregating transactions
- Distributing transactions

Adding transactions to the Ledger

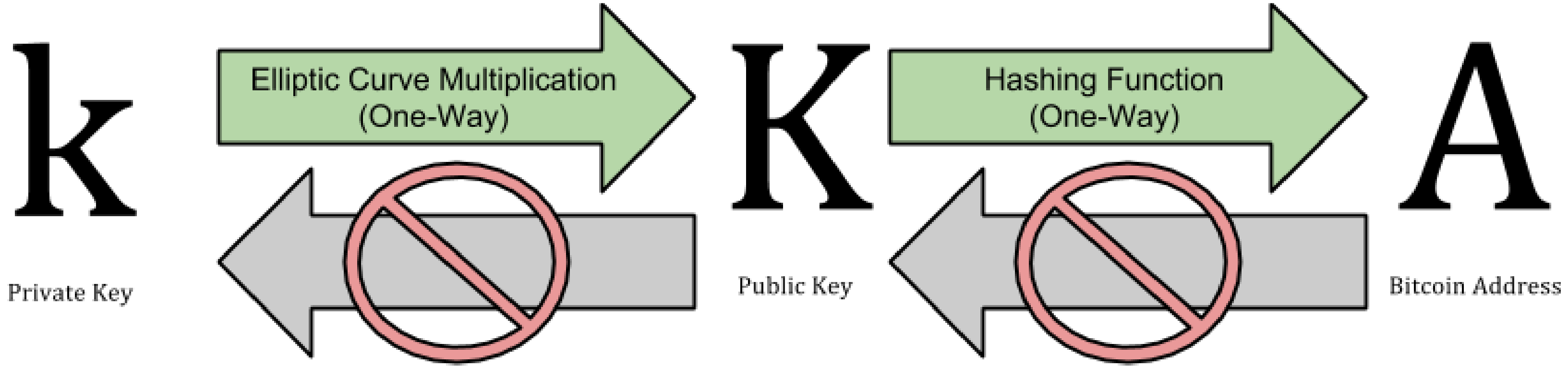
- Transmitting the transaction



Keys, Addresses and Wallets

- Ownership of bitcoin is established through digital keys, bitcoin addresses and digital signatures.
- The digital keys are not actually stored in the network but are instead created and stored by end-users in a file, or simple database, called a wallet.
- The digital keys in a user's wallet are completely independent of the bitcoin protocol and can be generated and managed by the user's wallet software without reference to the blockchain or access to the Internet

Addresses


k
Private Key
Elliptic Curve Multiplication
(One-Way)
K
Public Key
Hashing Function
(One-Way)
A
Bitcoin Address

Private Keys

Generation

- The first and most 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”.
- More accurately, the private key can be any number between 1 and n - 1, where n is a constant (n = 1.158 * 1077, slightly less than 2^256) defined as the order of the elliptic curve used in bitcoin.

Private Keys

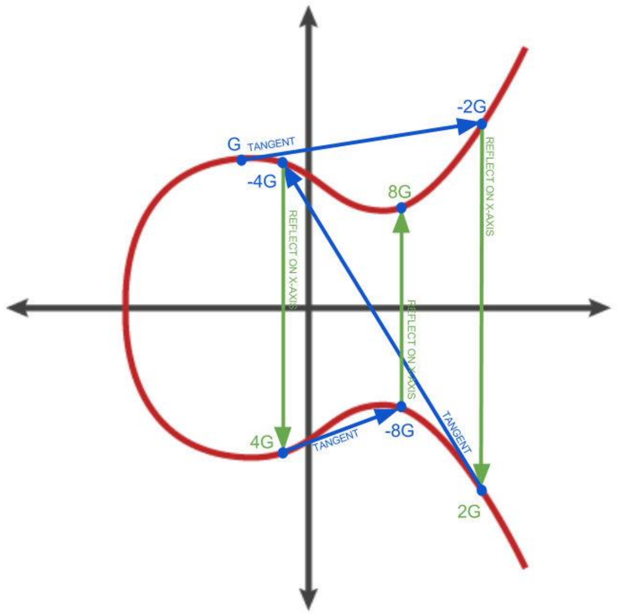
- This is usually achieved by feeding a larger string of random bits, collected from a cryptographically-secure source of randomness, into the SHA-256 hash algorithm which will conveniently produce a 256-bit number.
- If the result is less than n - 1, we have a suitable private key. Otherwise, we simply try again with another random number.

Public keys

- The 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

Elliptic Curve Cryptography

- The 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



Bitcoin Addresses

- A bitcoin address is a string of digits and characters that can be shared with anyone who wants to send you money.
- Addresses produced from public keys consist of a string of numbers and letters, beginning with the digit “1”.

Bitcoin Addresses

- The bitcoin address is derived from the public key through the use of one-way cryptographic hashing; a “hashing algorithm” or simply “hash algorithm” is a one-way function that produces a fingerprint or “hash” of an arbitrary sized input.
- Cryptographic hash functions are used extensively in bitcoin: in bitcoin addresses, in script addresses and in the mining “Proof-of-Work” algorithm.

Bitcoin Addresses

- The algorithms used to make a bitcoin address from a public key are the Secure Hash Algorithm (SHA) and the RACE Integrity Primitives Evaluation Message Digest (RIPEMD), specifically SHA256 and RIPEMD160.
- Base-64 representation uses 26 lower case letters, 26 capital letters, 10 numerals and two more characters such as "+" and "/" to transmit binary data over text-based media such as email.
- Base-58 is Base-64 without the 0 (number zero), O (capital o), l (lower L), I (capital i) and the symbols "\+" and "/".

Bitcoin Addresses

- Starting with the public key K, we compute the SHA256 hash and then compute the RIPEMD160 hash of the result, producing a 160 bit (20 byte) number:

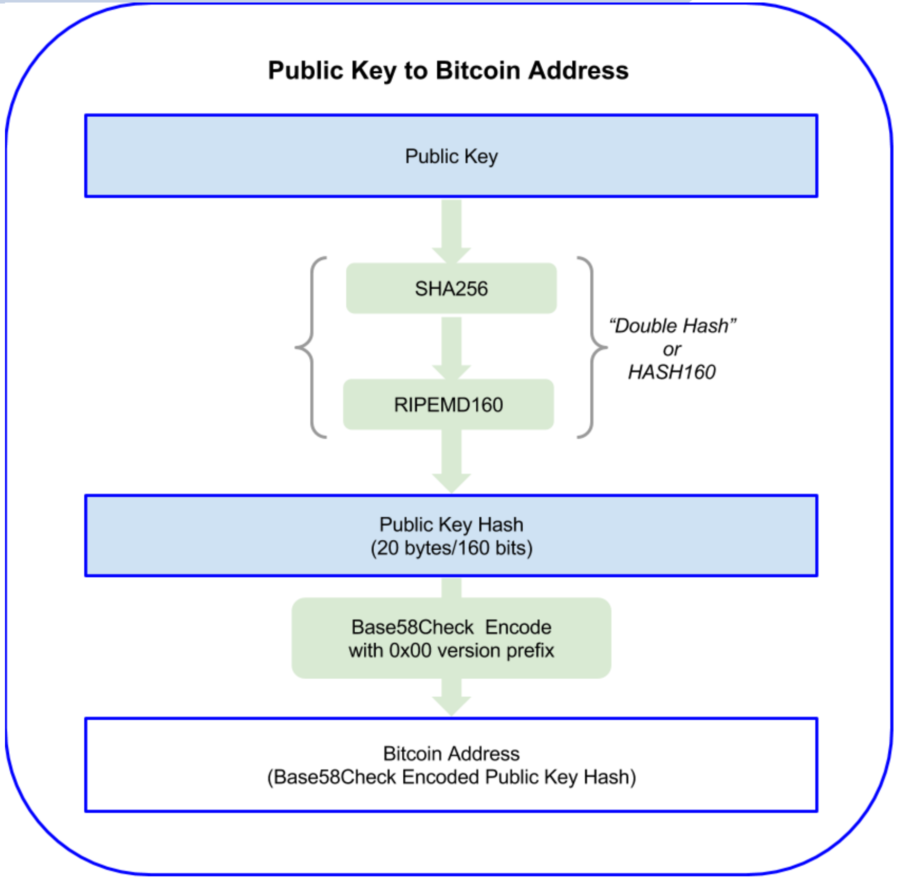
$$A = \text{RIPEMD160}(\text{SHA256}(K))$$

- where K is the public key and A is the resulting bitcoin address.

Bitcoin Addresses

- Bitcoin addresses are almost always presented to users in an encoding called “Base58Check” which uses 58 characters (a base-58 number system) and a checksum to help human readability, avoid ambiguity and protect against errors in address transcription and entry.

Bitcoin Addresses


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)

Bitcoin Addresses

Base58Check Encoding

- To add extra security against typos or transcription errors, Base58Check is a Base-58 encoding format, frequently used in bitcoin, which has a built-in error-checking code.
- The checksum is an additional four bytes added to the end of the data that is being encoded.
- The checksum is derived from the hash of the encoded data and can thereforebe used to detect and prevent transcription and typing errors

Bitcoin Addresses

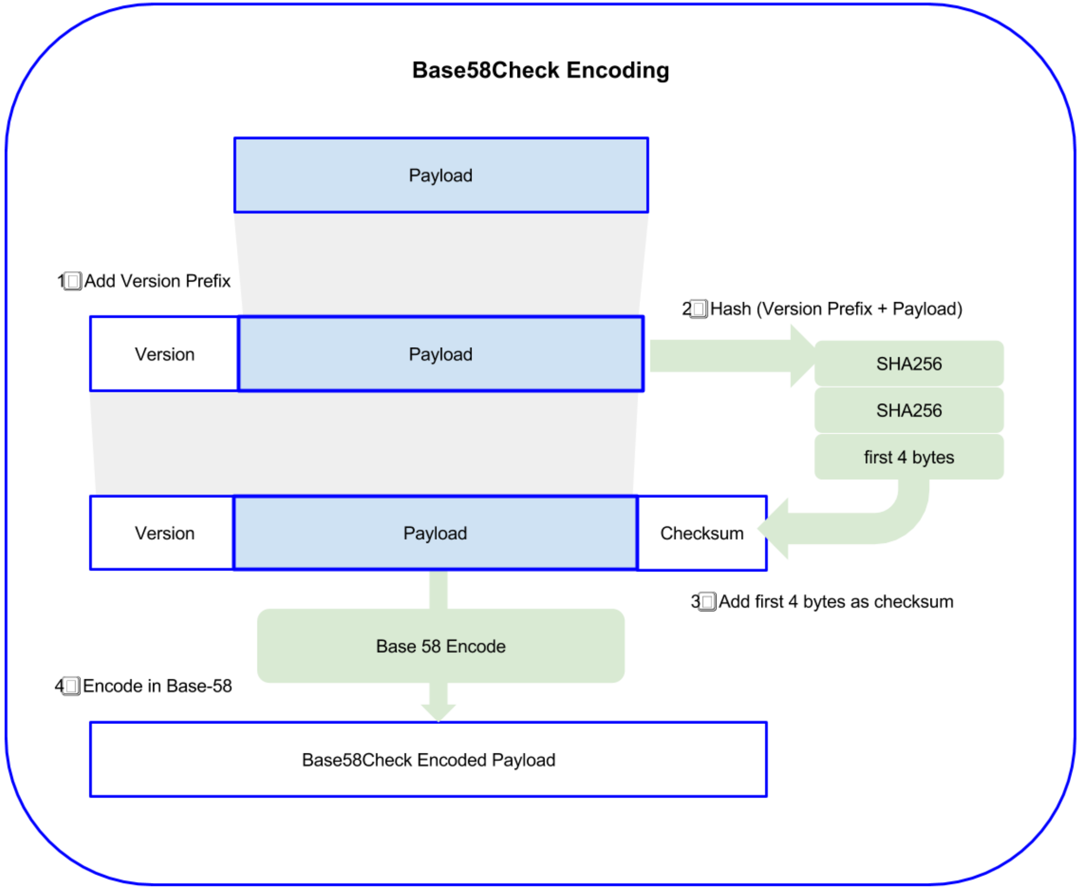
- To convert data (a number) into a Base58Check format, we first add a prefix to the data, called the “version byte”, which serves to easily identify the type of data that is encoded.
- For example, in the case of a bitcoin address the prefix is zero (0x00 in hex), whereas the prefix used when encoding a private key is 128 (0x80 in hex).

Bitcoin Addresses

- Next compute the “double-SHA” checksum, meaning we apply the SHA256 hash algorithm twice on the previous result

checksum = SHA256(SHA256(prefix+data))

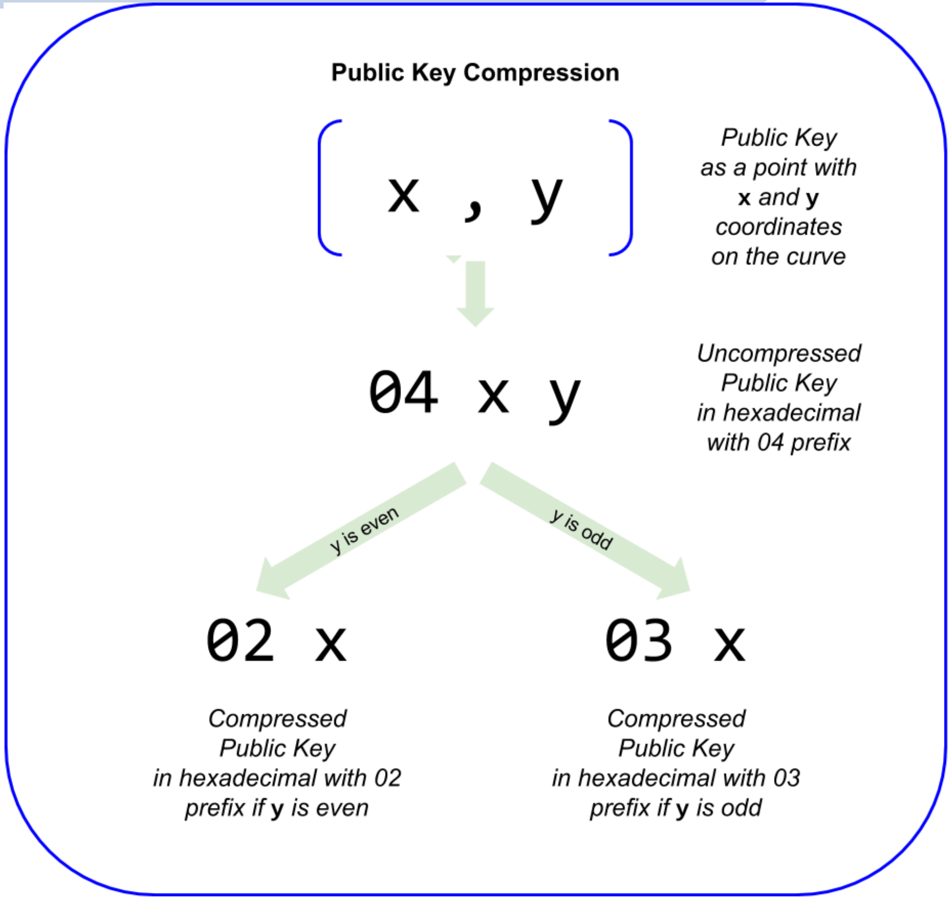
Bitcoin Addresses



Compressed Public Keys

- Compressed public keys were introduced to bitcoin to reduce the size of transactions and conserve disk space on nodes that store the bitcoin blockchain database.
- Uncompressed public keys have a prefix of 04, compressed public keys start with either a 02 or a 03 prefix.
- To distinguish between the two possible values of y, we store a compressed public key with the prefix 02 if the y is even, and 03 if it is odd.

Compressed Public Keys


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

Compressed Public Keys

- Here’s the same public key generated previously, shown as a compressed public key stored in 264-bits (66 hex digits) with the prefix 03 indicating the y coordinate is odd
- Compressed Public Key K shown in hex (66 hex digits) as K = {02 or 03} x.

K = 03F028892BAD...DC341A

Wallets

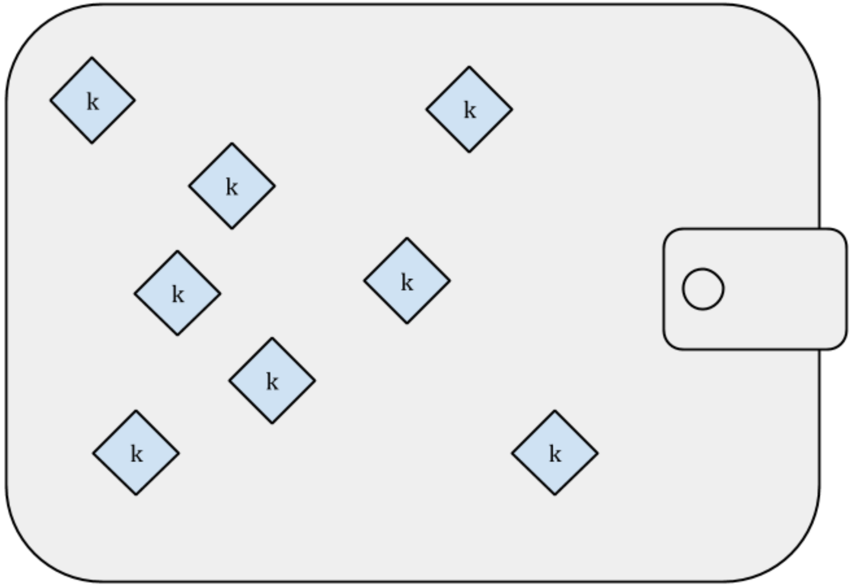
- Wallets are containers for private keys, usually implemented as structured files or simple databases.
- Here you derive each new private key, using a one-way hash function from a previous private key, linking them in a sequence.
- As long as you can re-create that sequence, you only need the first key (known as a seed or master key) to generate them all

Non-Deterministic (Random) Wallets

- In the first implementations of bitcoin clients, wallets were simply collections of randomly generated private keys.
- Bitcoin Core Client pre-generates 100 random private keys when first started and generates more keys as needed, using each key only once.
- This type of wallet is nicknamed “Just a Bunch Of Keys”, or JBOK, and such wallets are being replaced with deterministic wallets because they are cumbersome to manage, backup and import.

Non-Deterministic (Random) Wallets

- The disadvantage of random keys is that if you generate many of them you must keep copies of all of them, meaning that the wallet must be backed-up frequently



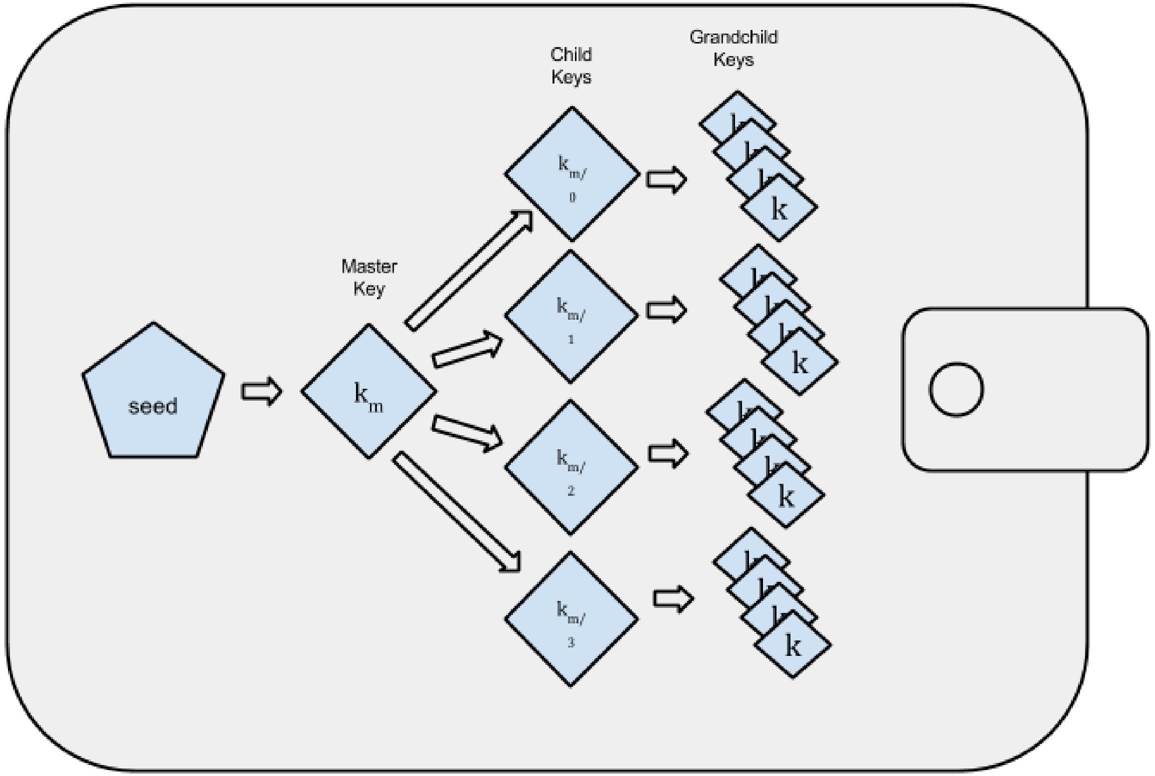
Deterministic (Seeded) Wallets

- Deterministic, or “seeded” wallets are wallets that contain private keys which are all derived from a common seed, through the use of a one-way hash function.
- In a deterministic wallet, the seed is sufficient to recover all the derived keys and therefore a single backup at creation time is sufficient

Hierarchical Deterministic Wallets

- Deterministic wallets were developed to make it easy to derive many keys from a single “seed”.
- The most advanced form of deterministic wallets is the Hierarchical Deterministic Wallet or HD Wallet defined by the BIP0032 standard.
- Hierarchical deterministic wallets contain keys derived in a tree structure, such that a parent key can derive a sequence of children keys, each of which can derive a sequence of grandchildren keys and so on to an infinite depth

Hierarchical Deterministic Wallets



Hierarchical Deterministic Wallets

Advantage 1:

- The tree structure can be used to express additional organizational meaning, such as when a specific branch of sub-keys is used to receive incoming payments and a different branch is used to receive change from outgoing payments. Branches of keys can also be used in a corporate setting, allocating different branches to departments, subsidiaries, specific functions or accounting categories.

Hierarchical Deterministic Wallets

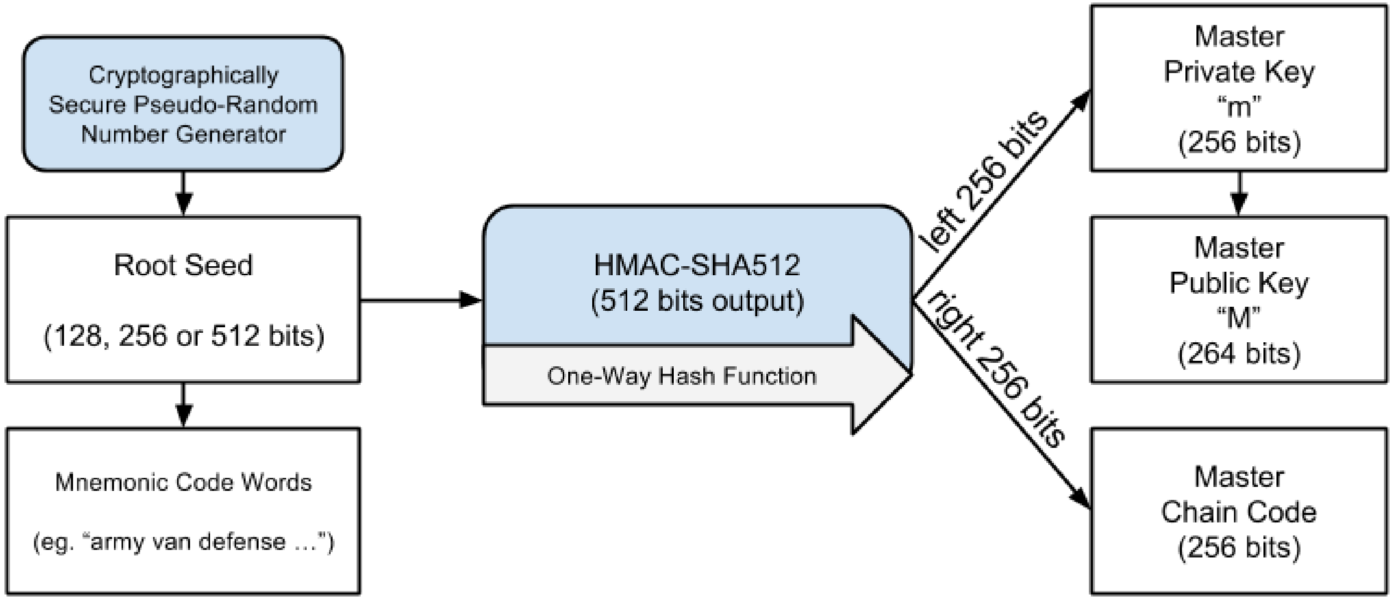
Advantage 2:

- users can create a sequence of public keys without having access to the corresponding private keys.
- This allows HD wallets to be used on an insecure server or in a receive-only capacity, issuing a different public key for each transaction

HD wallet creation from a seed

- HD wallets are created from a single root seed, which is a 128, 256 or 512 bit random number.
- Everything else in the HD wallet is deterministically derived from this root seed, which makes it possible to re-create the entire HD wallet from that seed in any compatible HD wallet.
- This makes it easy to backup, restore, export and import HD wallets containing thousands or even millions of keys by simply transferring only the root seed.

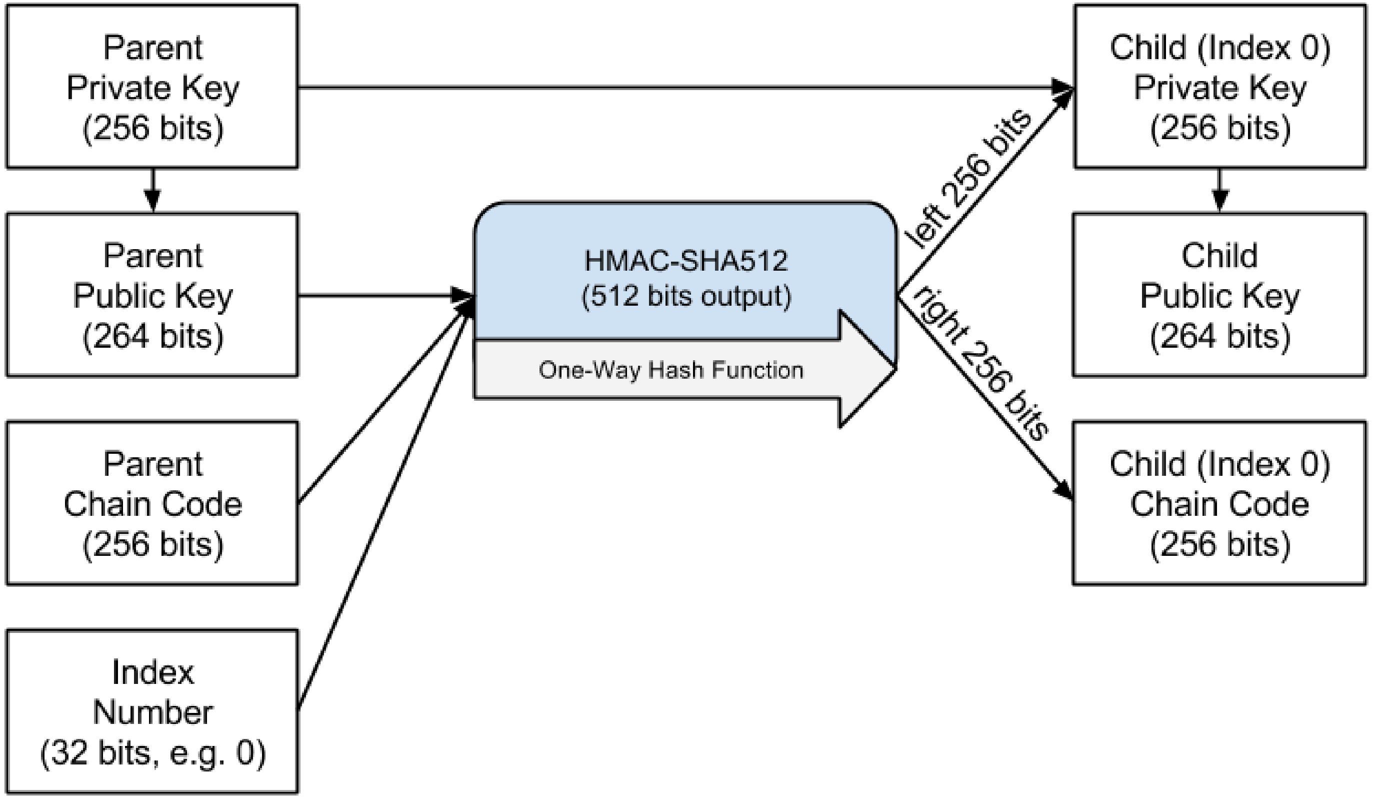
HD wallet creation from a seed


Cryptographically Secure Pseudo-Random Number Generator
Root Seed
(128, 256 or 512 bits)
Mnemonic Code Words
(eg. "army van defense ...")
HMAC-SHA512
(512 bits output)
One-Way Hash Function
left 256 bits
right 256 bits
Master Private Key "m" (256 bits)
Master Public Key "M" (264 bits)
Master Chain Code (256 bits)

HD wallet creation from a seed

- The root seed is input into the HMAC-SHA512 algorithm and the resulting hash is used to create a master private key (m) and a master chain code.
- The master private key (m) then generates a corresponding master public key (M), using the normal elliptic curve multiplication process m * G that we saw previously

Private child key derivation


Parent Private Key (256 bits)
Parent Public Key (264 bits)
Parent Chain Code (256 bits)
Index Number (32 bits, e.g. 0)
HMAC-SHA512 (512 bits output)
One-Way Hash Function
left 256 bits
right 256 bits
Child (Index 0) Private Key (256 bits)
Child Public Key (264 bits)
Child (Index 0) Chain Code (256 bits)

Private child key derivation

- Child private keys are indistinguishable from non-deterministic (random) keys.
- Because the derivation function is a one way function, the child key cannot be used to find the parent key.
- The child key can also not be used to find any siblings.
- If you have the nth child, you cannot find its siblings, such as the n-1 child or the n+1 child, or any other children that are part of the sequence.

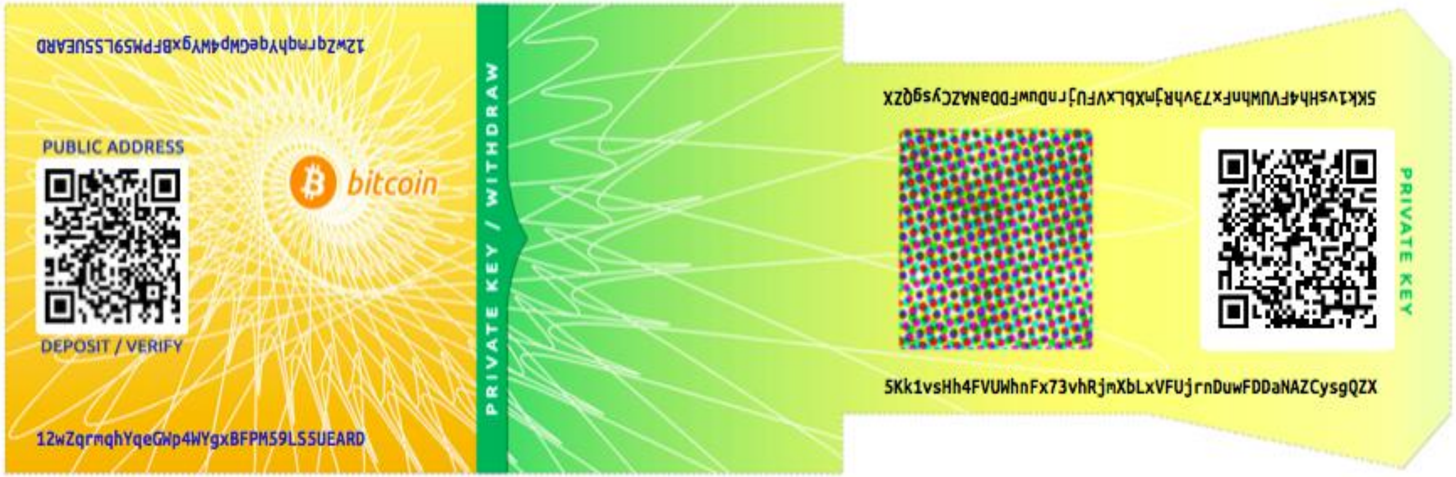
Private child key derivation

- Only the parent key and chain code can derive all the children.
- Without the child chain code, the child key cannot be used to derive any grandchildren either.
- You need both the child private key and the child chain code to start a new branch and derive grandchildren
- So what can the child private key be used for on its own? It can be used to make a public key and a bitcoin address.
- Then, it can be used to sign transactions to spend anything paid to that address

Paper Wallets

- Paper wallets are bitcoin private keys printed on paper. Often the paper wallet also includes the corresponding bitcoin address, for convenience, but this is not necessary since 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

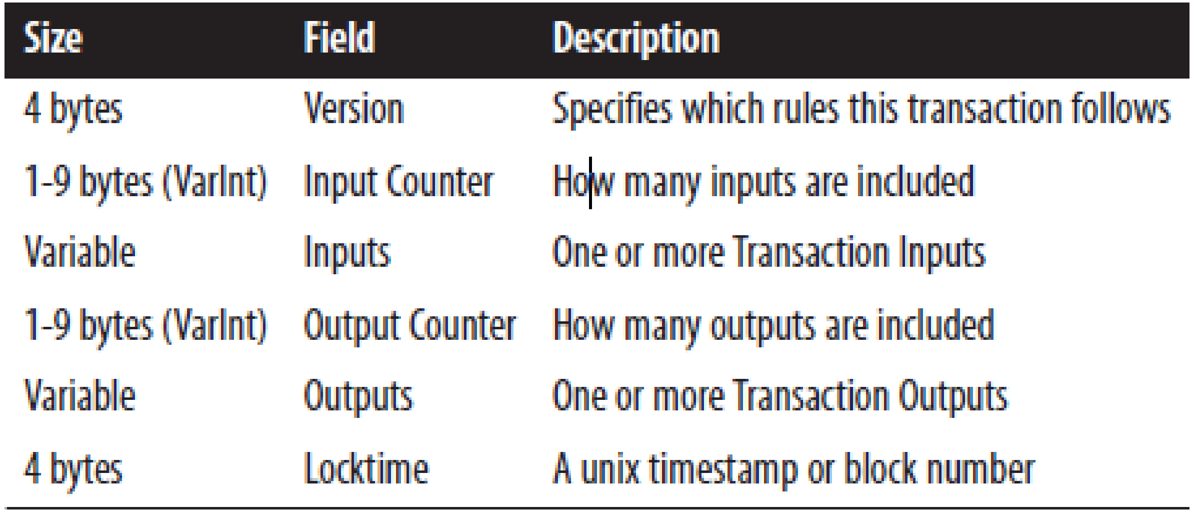
Paper Wallets


12wZqrmqhYqeGWp4WYgxBFPM59LSSUEARD
PUBLIC ADDRESS
bitcoin
DEPOSIT / VERIFY
12wZqrmqhYqeGWp4WYgxBFPM59LSSUEARD
PRIVATE KEY / WITHDRAW
5Kk1vsHh4FVUWhnFx73vhRjmXbLxVFUjrnDuwFDDaNAZCysgQZX
PRIVATE KEY
5Kk1vsHh4FVUWhnFx73vhRjmXbLxVFUjrnDuwFDDaNAZCysgQZX

Transactions

- A transaction is a data structure that encodes a transfer of value from a source of funds, called an input, to a destination, called an output.
- Transaction inputs and outputs are not related to accounts or identities. Instead you should think of them as bitcoin amounts, chunks of bitcoin, being locked with a specific secret which only the owner, or person who knows the secret, can unlock

Transactions – Structure



Size	Field	Description
4 bytes	Version	Specifies which rules this transaction follows
1-9 bytes (VarInt)	Input Counter	How many inputs are included
Variable	Inputs	One or more Transaction Inputs
1-9 bytes (VarInt)	Output Counter	How many outputs are included
Variable	Outputs	One or more Transaction Outputs
4 bytes	Locktime	A unix timestamp or block number

Transactions

Create 25 coins and credit to Alice$_{\text{ASSERTED BY MINERS}}$
Transfer 17 coins from Alice to Bob$_{\text{SIGNED(Alice)}}$
Transfer 8 coins from Bob to Carol$_{\text{SIGNED(Bob)}}$
Transfer 5 coins from Carol to Alice$_{\text{SIGNED(Carol)}}$
Transfer 15 coins from Alice to David$_{\text{SIGNED(Alice)}}$

Transaction Outputs and Inputs

- The fundamental building block of a bitcoin transaction is an unspent transaction output or UTXO.
- UTXO are indivisible chunks of bitcoin currency locked to a specific
- owner, recorded on the blockchain, and recognized as currency units by the entire network.
- The bitcoin network tracks all available (unspent) UTXO currently numbering in the millions

Transaction Outputs and Inputs

- Whenever a user receives bitcoin, that amount is recorded within the blockchain as a UTXO.
- Thus, a user's bitcoin may be scattered as UTXO amongst hundreds of transactions and hundreds of blocks.
- The wallet calculates the user's balance by scanning the blockchain
- and aggregating all UTXO belonging to that user
- There are no accounts or balances in bitcoin, there are only unspent transaction outputs (UTXO) scattered in the blockchain

Transaction Outputs and Inputs

- A UTXO can have an arbitrary value denominated as a multiple of satoshis. Just like dollars can be divided down to two decimal places as cents, bitcoins can be divided down to eight decimal places as satoshis.

Transaction Outputs and Inputs

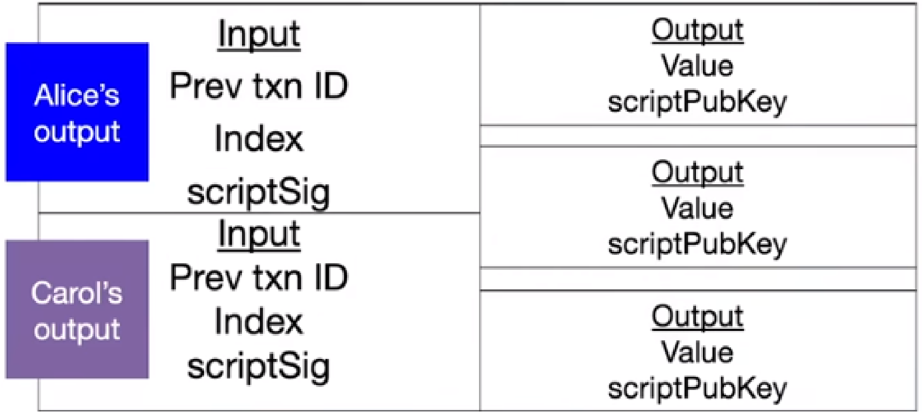
- The UTXO consumed by a transaction are called transaction inputs, while the UTXO created by a transaction are called transaction outputs.
- This way, chunks of bitcoin value move forward from owner to owner in a chain of transactions consuming and creating UTXO.
- Transactions consume UTXO unlocking it with the signature of the current owner and create UTXO locking it to the bitcoin address of the new owner

UTXO

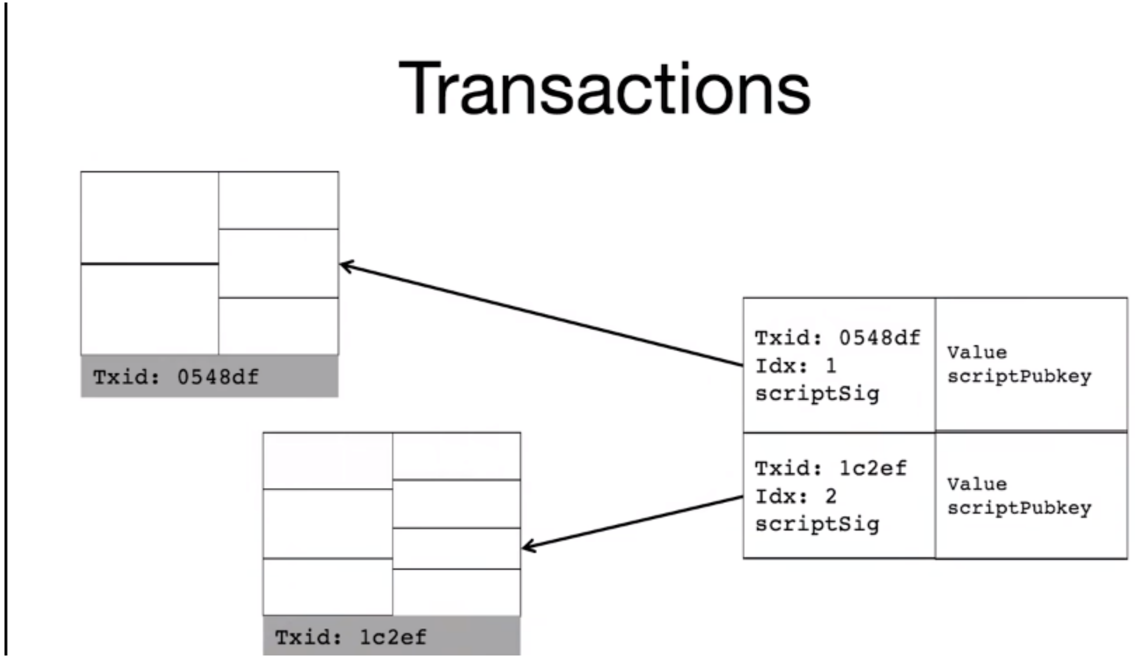
Transaction format

Input	Output
Prev txn ID	Value
Index	scriptPubKey
scriptSig	

Inputs and outputs are independent



UTXO


Transactions
Txid: 0548df
Txid: 1c2ef
Txid: 0548df
Idx: 1
scriptSig
Value
scriptPubkey
Txid: 1c2ef
Idx: 2
scriptSig
Value
scriptPubkey

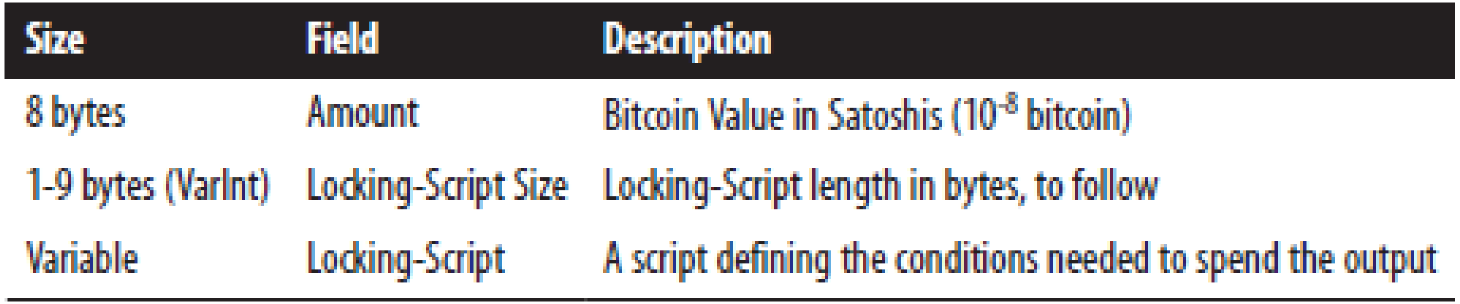
Transaction Outputs

- Every bitcoin transaction creates outputs, which are recorded on the bitcoin ledger.
- Almost all of these outputs create spendable chunks of bitcoin called unspent transaction outputs or UTXO, which are then recognized by the whole network and available for the owner to spend in a future transaction.
- Sending someone bitcoin is creating an unspent transaction output (UTXO) registered to their address and available for them to spend

Transaction Outputs

- UTXO are tracked by every full node bitcoin client in a database held in memory, called the UTXO set or UTXO pool.
- New transactions consume (spend) one or more of these outputs from the UTXO set.
- Transaction outputs consist of two parts:
  - An amount of bitcoin, denominated in satoshis, the smallest bitcoin unit
  - A locking script, also known as an “encumbrance” that “locks” this amount by specifying the conditions that must be met to spend the output

Structure of a Transaction Output



Size	Field	Description
8 bytes	Amount	Bitcoin Value in Satoshis (10^{-8} bitcoin)
1-9 bytes (VarInt)	Locking-Script Size	Locking-Script length in bytes, to follow
Variable	Locking-Script	A script defining the conditions needed to spend the output

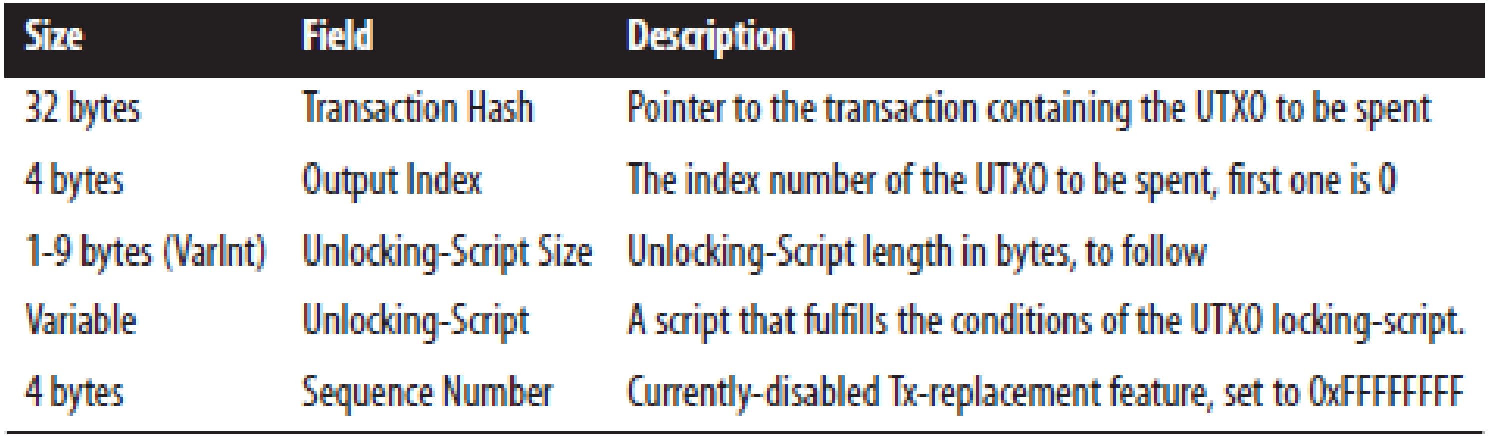
Transaction Inputs

- In simple terms, transaction inputs are pointers to UTXO. They point to a specific UTXO by reference to the transaction hash and sequence number where the UTXO is recorded in the blockchain.
- To spend UTXO, a transaction input also includes unlocking-scripts that satisfy the spending conditions set by the UTXO.
- The unlocking script is usually a signature proving ownership of the bitcoin address that is in the locking script

Transaction Inputs

- When a user makes a payment, their wallet constructs a transaction by selecting from the available UTXO.
- For example, to make a 0.015 bitcoin payment, the wallet app may select a 0.01 UTXO and a 0.005 UTXO, using them both to add up to the desired payment amount

Structure of a Transaction Input



Size	Field	Description
32 bytes	Transaction Hash	Pointer to the transaction containing the UTXO to be spent
4 bytes	Output Index	The index number of the UTXO to be spent, first one is 0
1-9 bytes (VarInt)	Unlocking-Script Size	Unlocking-Script length in bytes, to follow
Variable	Unlocking-Script	A script that fulfills the conditions of the UTXO locking-script.
4 bytes	Sequence Number	Currently-disabled Tx-replacement feature, set to 0xFFFFFFFF

Structure of a Transaction Input

- When a user makes a payment, their wallet constructs a transaction by selecting from the available UTXO.
- For example, to make a 0.015 bitcoin payment, the wallet app may select a 0.01 UTXO and a 0.005 UTXO, using them both to add up to the desired payment amount

Transaction Fees

- Transaction fees serve as an incentive to include (mine) a transaction into the next block and also as a disincentive against “spam” transactions or any kind of abuse of the system, by imposing a small cost on every transaction.
- Transaction fees are calculated based on the size of the transaction in kilobytes, not the value of the transaction in bitcoin.

Transaction Fees

- Transaction fees affect the processing priority, meaning that a transaction with sufficient fees is likely to be included in the next-most mined block, while a transaction with insufficient or no fees may be delayed.
- If you consume a 20 bitcoin UTXO to make a 1 bitcoin payment, you must include a 19 bitcoin change output back to your wallet. Otherwise, the 19 bitcoin “leftover” will be counted as a transaction fee and will be collected by the miner who mines your transaction in a block.

Adding Fees to Transactions

- The data structure of transactions does not have a field for fees. Instead, fees are implied as the difference between the sum of inputs and the sum of outputs.
- Any excess amount that remains after all outputs have been deducted from all inputs is the fee that is collected by the miners.
- Transaction fees are implied, as the excess of inputs minus outputs.

$$\text{Fees} = \text{Sum}(\text{Inputs}) - \text{Sum}(\text{Outputs})$$

Transaction Chaining and Orphan Trans

- Parent and Child transactions
- Sometimes an entire chain of transactions depending on each other, say a parent, child and grandchild transaction are created at the same time, to fulfill a complex transactional workflow that requires valid children be signed before the parent is signed.

Transaction Chaining and Orphan Trans

- Sometimes, the child might arrive before the parent.
- In that case, the nodes which see a child first can see that it references a parent transaction that is not yet known.
- Rather than reject the child, they put it in a temporary pool to await the arrival of its parent and propagate it to every other node.
- The pool of transactions without parents is known as the orphan transaction pool.

Script Construction (Lock + Unlock)

- Bitcoin’s transaction validation engine relies on two types of scripts to validate transactions
- – a locking script and an unlocking script

- A locking script is an encumbrance placed on an output, and it specifies the conditions that must be met to spend the output in the future. Historically, the locking script was called a *scriptPubKey*, because it usually contained a public key or bitcoin address

Script Construction (Lock + Unlock)

- 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 and most of the time they contain a digital signature produced by the user’s wallet from their private key

Script Construction (Lock + Unlock)

Unlocking Script
(scriptSig)

+

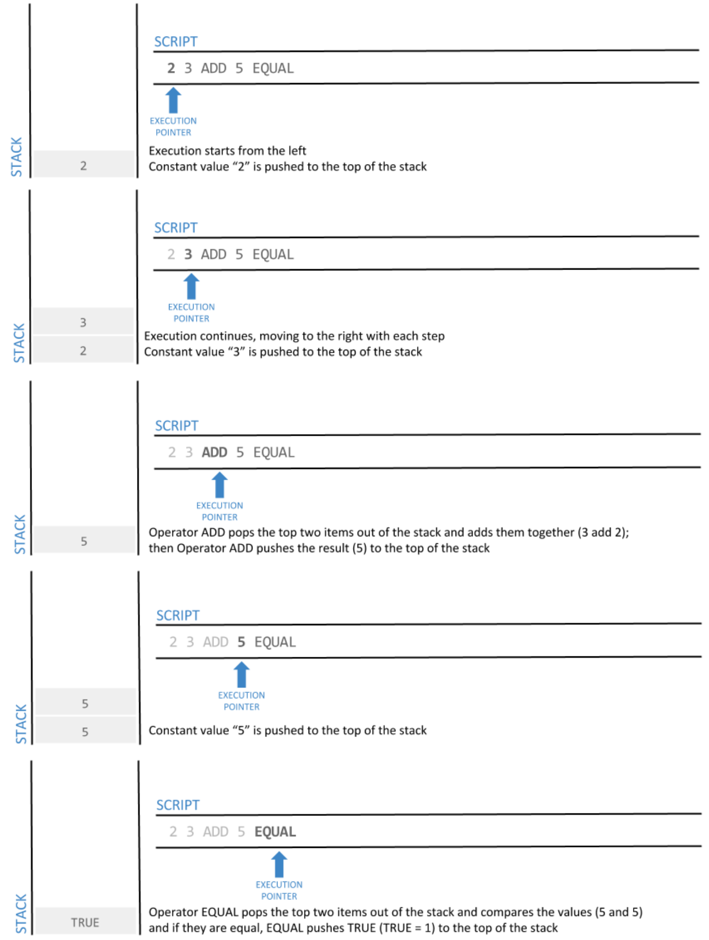
Locking Script
(scriptPubKey)

```
<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

Script Execution



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

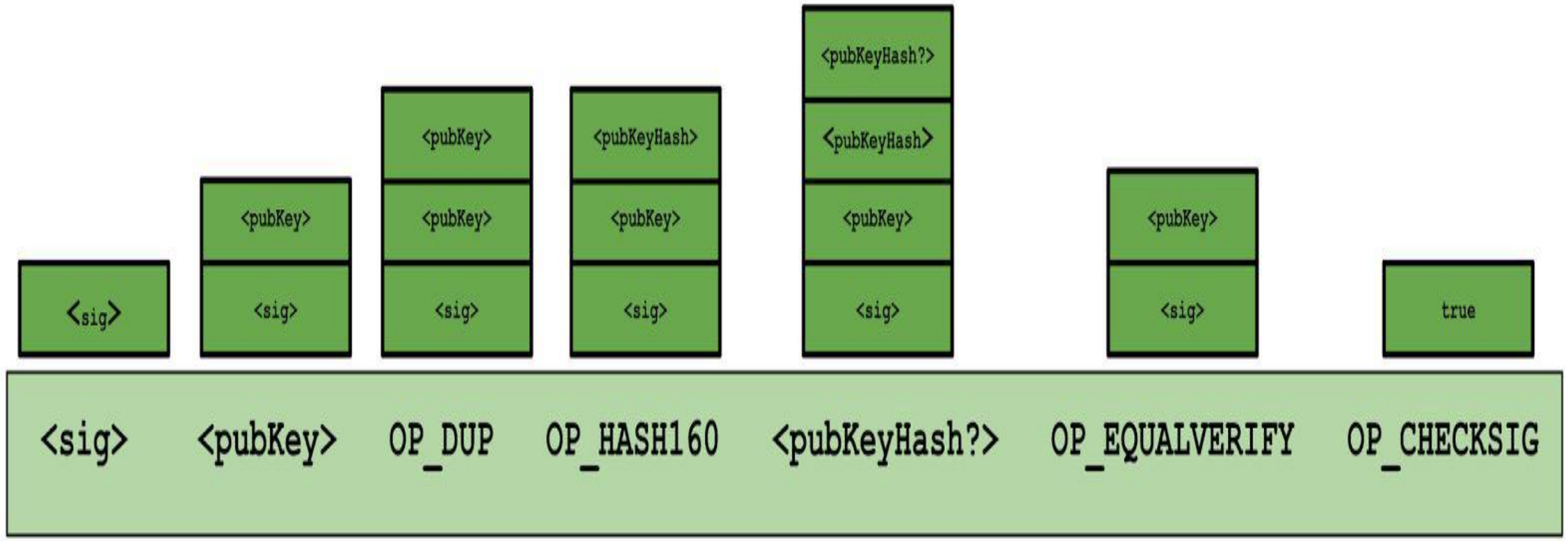
Execution continues, moving to the right with each step
Constant value “3” is pushed to the top of the stack

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

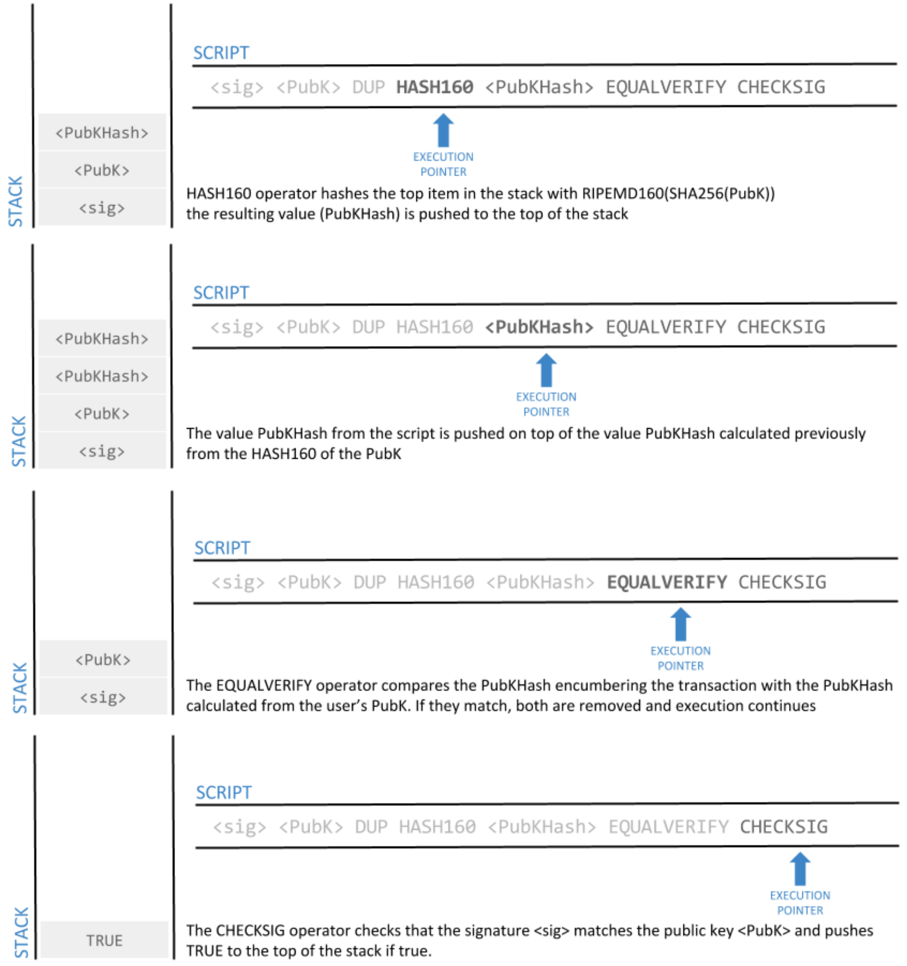
Constant value “5” is pushed to the top of the stack

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

Script Execution



Script Construction


STACK
<PubKHash>
<PubK>
<sig>
SCRIPT
<sig> <PubK> DUP **HASH160** <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubKHash>
<PubKHash>
<PubK>
<sig>
SCRIPT
<sig> <PubK> DUP HASH160 **<PubKHash>** EQUALVERIFY CHECKSIG
EXECUTION POINTER
STACK
<PubK>
<sig>
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> **EQUALVERIFY** CHECKSIG
EXECUTION POINTER
STACK
TRUE
SCRIPT
<sig> <PubK> DUP HASH160 <PubKHash> EQUALVERIFY CHECKSIG
EXECUTION POINTER

HASH160 operator hashes the top item in the stack with RIPEMD160(SHA256(PubK)) the resulting value (PubKHash) is pushed to the top of the stack

The value PubKHash from the script is pushed on top of the value PubKHash calculated previously from the HASH160 of the PubK

The EQUALVERIFY operator compares the PubKHash encumbering the transaction with the PubKHash calculated from the user's PubK. If they match, both are removed and execution continues

The CHECKSIG operator checks that the signature <sig> matches the public key <PubK> and pushes TRUE to the top of the stack if true.

End