GlobalLogic®

The Basics of Bitcoin Transfers

Authored by: Deenus Akkara, Senior Solution Architect

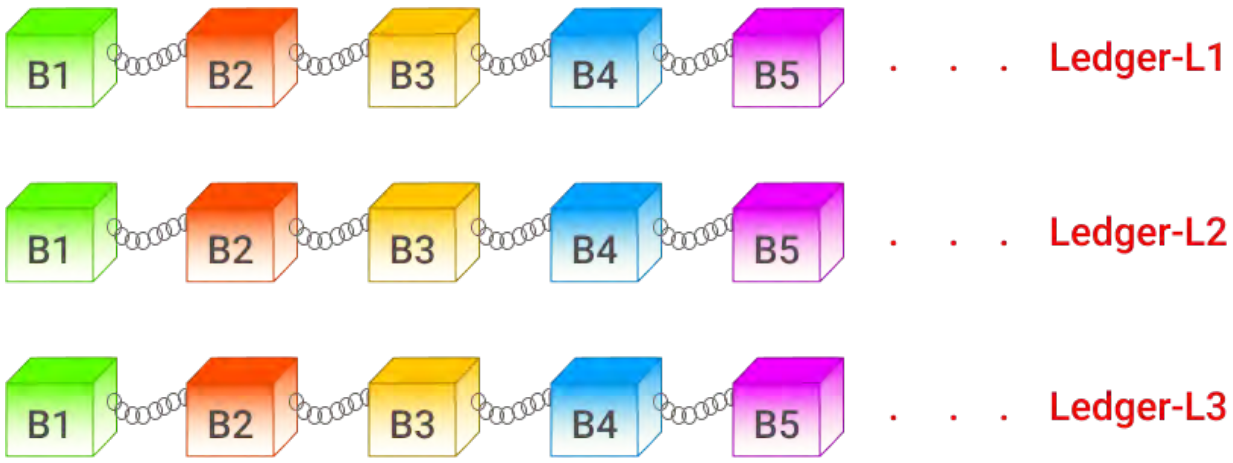
Contents

Disclaimer

This paper is based on information gathered in documents and online research. I am open to any corrections in this paper as and when required.

An Introduction to Blockchain

A blockchain is a decentralized, distributed, and public digital ledger that is used to record transactions across many computers. Records cannot be altered retroactively without the alteration of all subsequent blocks and the consensus of the network.



L1, L2, and L3 exist in different locations of the world and all are the same.

Bitcoin is a digital currency built on top of this blockchain technology. Since it is a currency, we can transact it; we can transfer this currency between two accounts or addresses, with a transaction fee paid to Miners.

Miners are the nodes or computers that insert our transactions into the blockchain by running a hashcash proof of work (PoW) function.

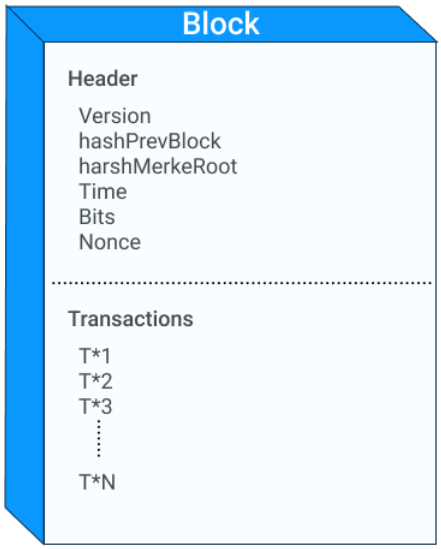
Each block in a Bitcoin contains multiple transactions and each block is linked to the previous block based on its hash.

Bitcoin Block Structure

A Bitcoin block contains a Header section, block header, and a Transaction section. The size of the block is limited to 1MB to propagate it fast in the network.

Header

- Version - The Bitcoin version (eg; 0x20600004).
- hashPrevBlock - Double SHA256 Hash of the previous block.
- hashMerkleRoot - Merkle root hash of all transactions in the block.
- Time - Timestamp.
- Bits - Target hash. (Miners should create a hash less than or equal to this value.)
- Nonce - A numeric value used by the Miners to create the target hash.
- LockTime - The earliest time the transaction is added to the blockchain, eg.: as in a post-dated cheque. If this value is <500 million, then it is block height. Otherwise, it is the timestamp when the transaction is to be added.

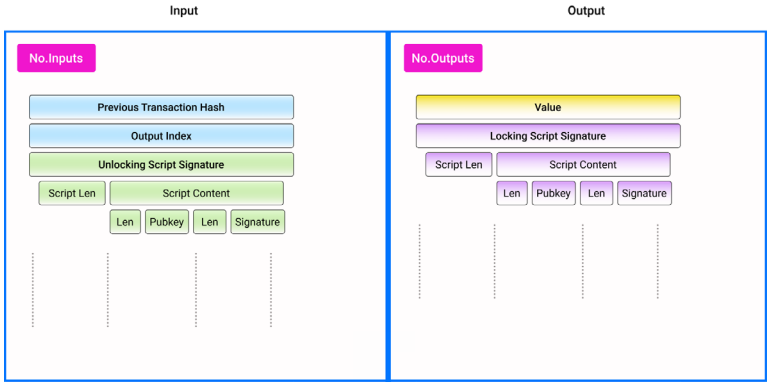


Transaction

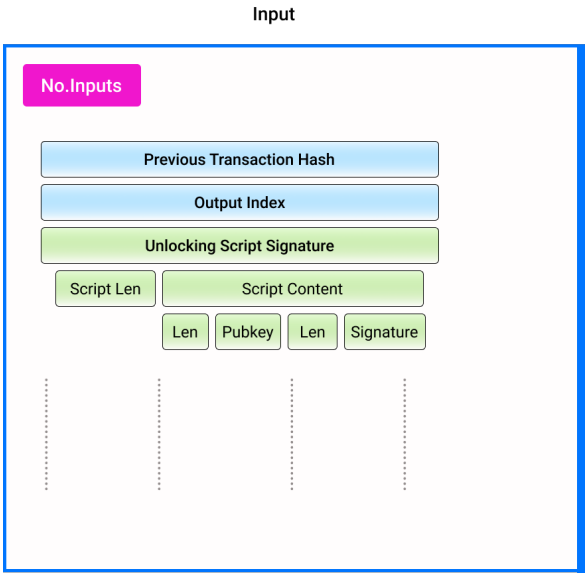
A transaction is used to transfer Bitcoin between addresses or accounts. When a transaction is created, it is signed by the owner's private key and sent to the network for validation. Once validated by the network, it is added to the blockchain.

UTXO, Unspent Transaction Output, is the model used by Bitcoin to record the transaction, and therefore whenever a user receives Bitcoin it is recorded in the blockchain as UTXO. As a result, a user's Bitcoin transaction is scattered across multiple blocks. There is no account balance in the UTXO model. The majority of the transactions processed in Bitcoin are in Pay-to-Public-Key-Hash (P2PKH).

The transaction consists of the following fields: input and output.

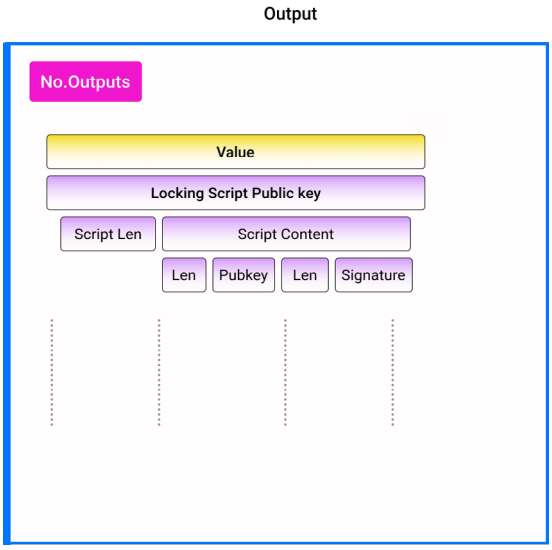


Input Fields



1.	No. Inputs	The total number of the previous transaction hashes going to be embedded in the current transaction.
2.	Previous Transaction Hash	Transaction hash of the previous transaction.
3.	Output Index	Index position in the Output of the previous transaction.
4.	Unlocking Script Signature	Unlocking script containing the public key of the sender; contains variable-length signature and public key.

Output Fields

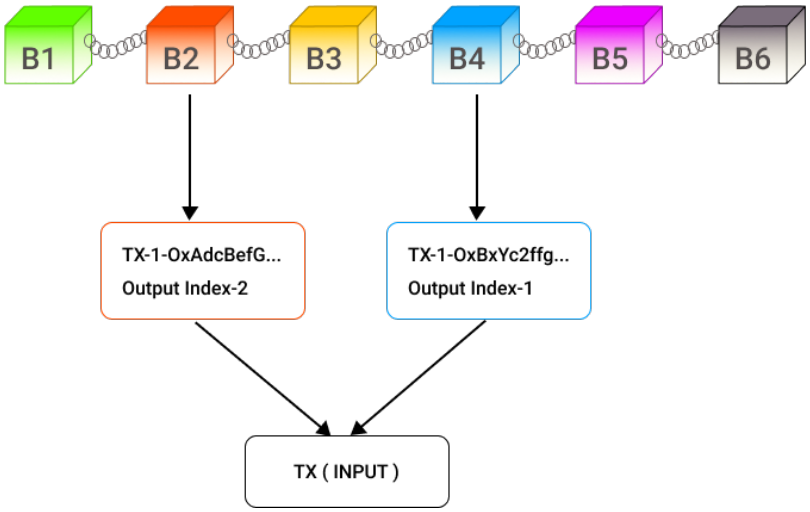


1.	No. Outputs	Total number of outputs
2.	Value	Bitcoin value, satoshi, to be transferred to the receiver.
3.	Locking Script Public Key	Contains the condition that must be met to spend this amount of satoshi. Hence, only the Private Key of this user can decrypt this transaction. 1.Public Key of the receiver (variable length of the public key and signature)

Bitcoin Use Case

1. Alice sent 0.75 BTC to Bob

Let's assume that Alice wants to transfer 0.75 BTC to Bob. In order for this to happen, Alice must have all unspent Bitcoin transactions worth greater than 0.75 BTC from her past transactions put in as Input transactions. Eg.: with 1BTC, assume 0.05 BTC as the fee for the Miner.



TX-1 and TX-2 are the two transactions from which Alice got 1 BTC in total. Output Index is the position of these transactions in those output sections of the specific blocks, B2 and B4. When Alice transfers 1 BTC, both the TX-1 and TX-2 will be added to the Input part of the transaction to show that these are the unspent transactions of Alice.

The unlocking script contains a digital signature produced by the sender, Alice, Private Key. Along with this is Alice's Public Key.

Output

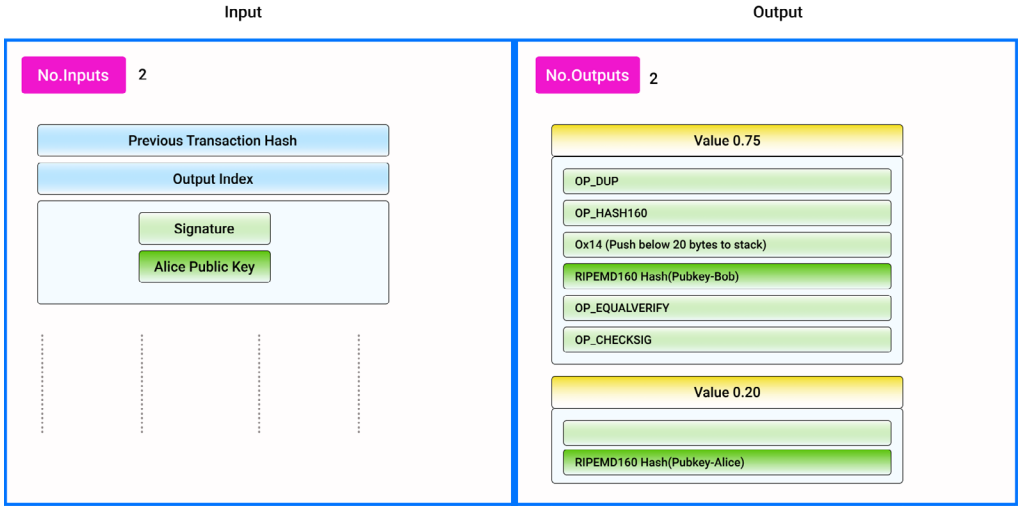
This part contains how much Bitcoin Alice needs to send to Bob. As per the above use case, it is 0.75. Since Alice has input 1BTC, she needs to get back 0.20 BTC to her address. The value field contains 1 BTC.

Hence, Alice's output transaction contains two entries:

- 0.75 BTC to Bob
- 0.20 BTC to Alice itself as the balance.

The balance of the above, 0.05, will be transferred to the Miner as transaction fees. A transaction fee is not mandatory and if it is added, there is a chance of the transaction to be added into the block at a faster pace.

Here is a representation of transactions for transferring 0.75 BTC from Alice to Bob:

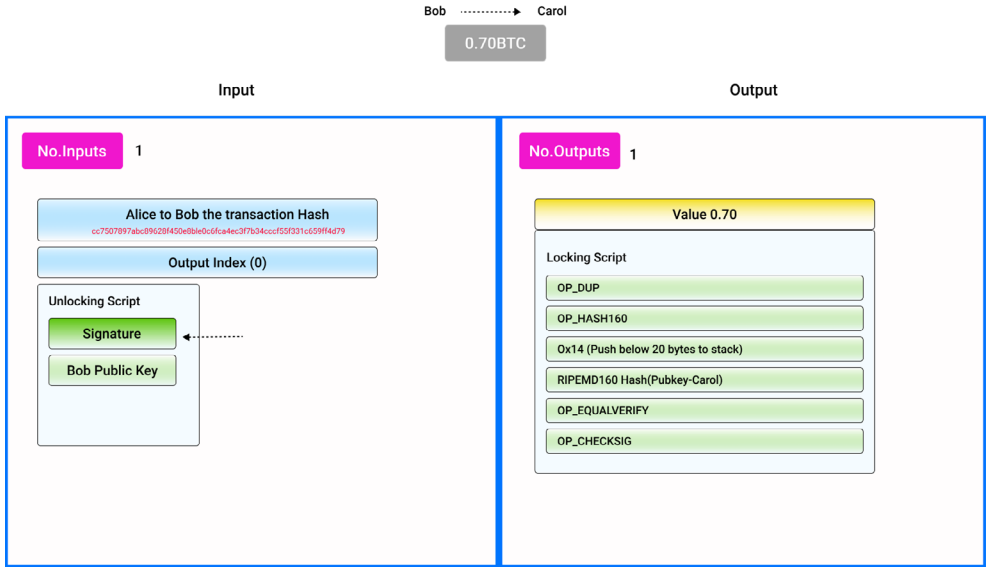


Input contains the sender's public key along with the digital signature.

Output contains two values:

- 0.75 BTC transferred to Bob.
- 0.20 BTC as a change back to Alice. (Since 0.05 BTC is not mentioned here, it will go to the Miner as a transaction fee.)

2. Bob then transferred 0.70 BTC to Carol

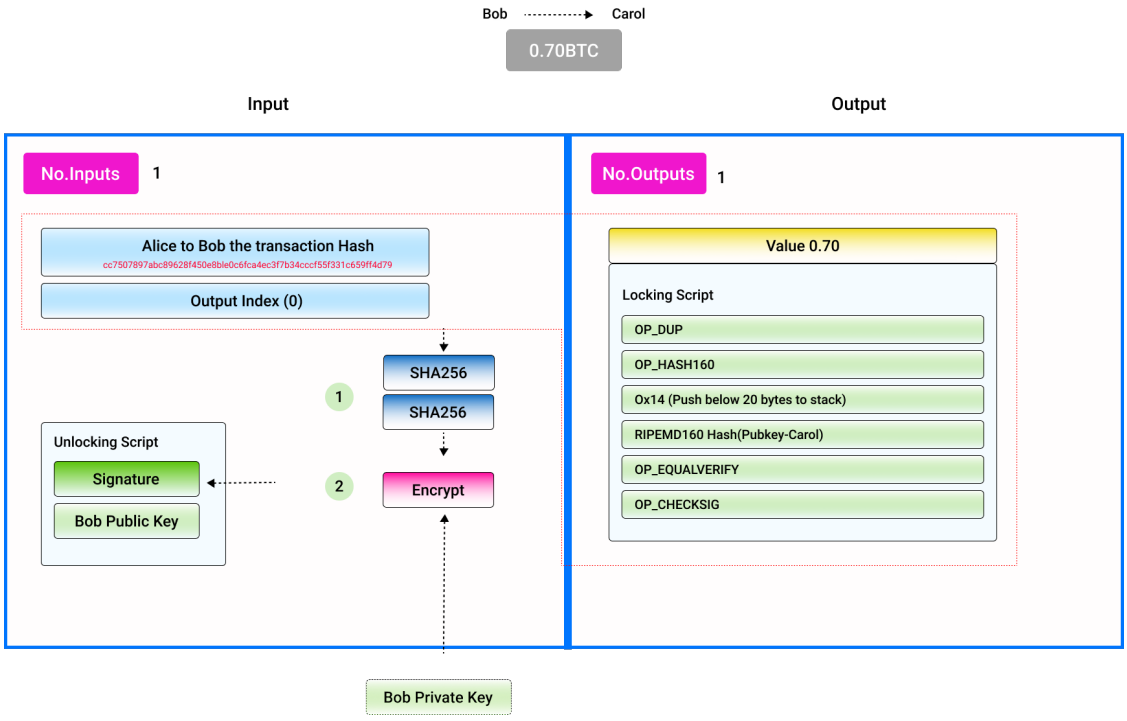


The preceding diagram shows how the Input and Output section of a transaction exists.

Let's see how the Signature of the Unlocking script is generated.

- Create a SHA256 hash of the information embedded in the dotted lines of the below diagram. Generate another SHA256 hash of the output of the above step.
- Take the output of the above step and encrypt using the private key of the sender, eg; Bob Private Key.

The output of the above is written as the Signature of the Unlocking script. This next diagram explains it in more detail.

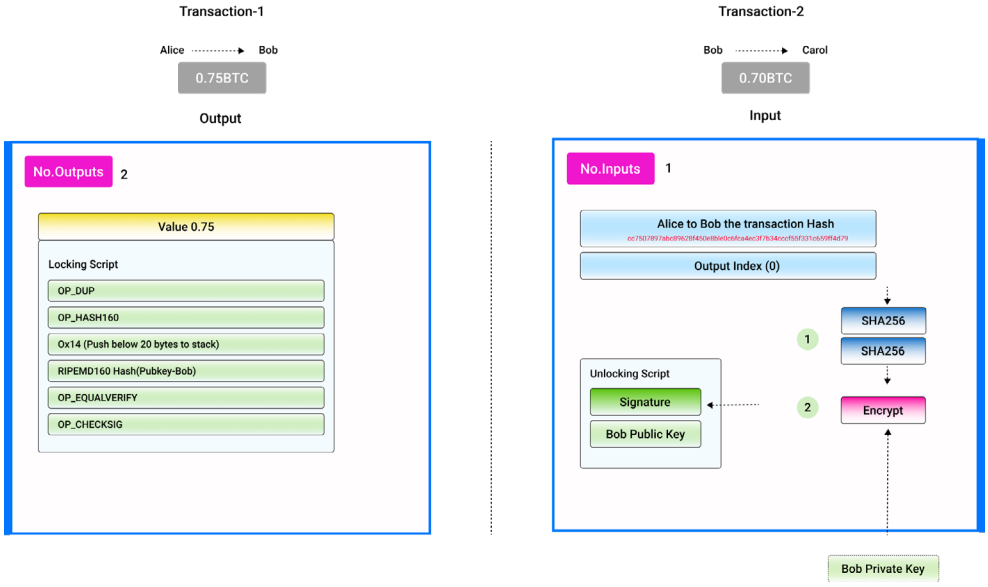


Transaction Validation

Let's see how bitcoin transactions are validated by each node. For validating a transaction, we need information from two blocks:

- The transaction where Bob received 0.75 BTC from Alice.
- The transaction in which Bob transferred 0.70 BTC to Carol.

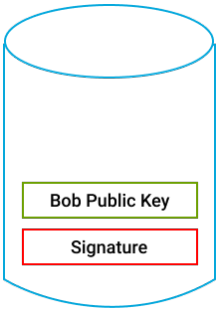
We need to use the locking script of the Alice to Bob transaction, Transaction-1, and the unlocking script of the Bob to Carol transaction, Transaction-2.



Validation of the transaction is done by executing the opcodes from locking and unlocking scripts. Opcodes are executed on the stack for a transaction and if the top result is TRUE, then it is valid.

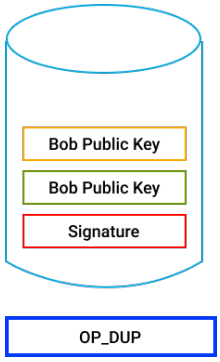
Validating Bitcoin Transactions, Step by Step

Step 1. Copy the Public Key and Signature to Stack.



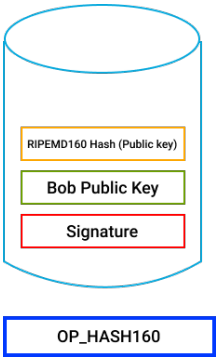
Signature and Bob's Public Key from Unlocking Script of Transaction-2 are pushed to the stack, as shown here.

Step 2. Execute OP_DUP.



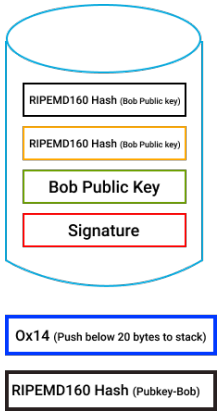
OP_DUP opcode, from the Transaction -1 locking script, is executed. The value on the top of the stack, Bob Public Key, is duplicated and put on the top of the stack.

Step 3. Execute OP_HASH160.



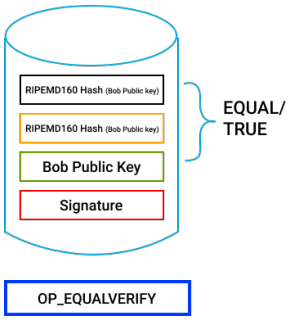
OP_HASH160 opcode, from the Transaction-1 locking script, is executed. The duplicated Bob Public Key from Step 2 is replaced with its HASH160 value.

Step 4. Copying RIPEMD160 Hash.



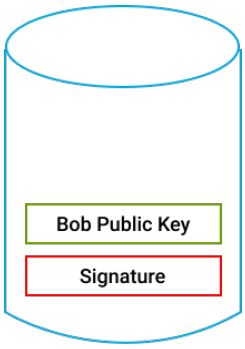
0x14, the hexadecimal value of how many bytes are to be copied from the top of the stack, 0x14 = 20, is executed. We can see the same copy of Bob Public Key (which is HASH160) on the top of the stack.

Step 5. Execute OP_EQUALVERIFY.



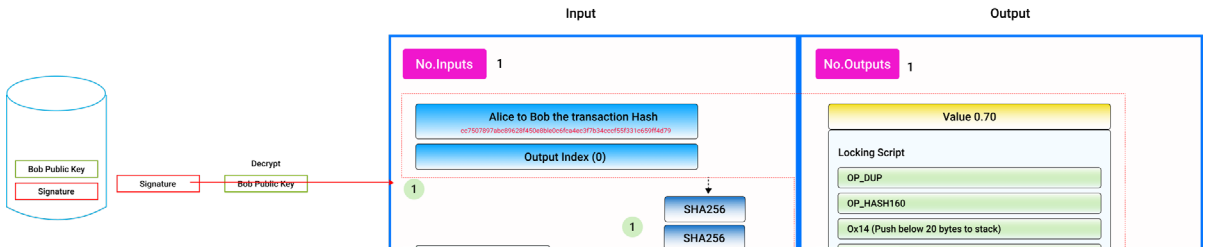
OP_EQUALVERIFY opcode verifies the two values on the top of the stack. If the resultant value is not TRUE, then the validation is failed. If true, it continues the execution of the rest of the opcodes.

Step 6. Current values in the Stack.



In this state, the stack contains only the Bob Public Key and the Signature from the unlocking script of Transaction-2. Further verification is done as below.

Step 7. Validating Signature from Input Script.



The signature value is decrypted using Bob Public Key. If the resultant value is EQUAL to the value of DOUBLE SHA256, then the transaction of Bob to Carol is valid.

Likewise, all the transactions are validated for a block and each Node in the network will add this validated block into their chain. Once this block is added to its own chain, it will be transmitted to the nearest Nodes.

Summary

Bitcoin transactions are highly secured based on the Private/Public key encryption and the script execution methods built on the bitcoin engine to validate each transaction. Since all transactions are validated by distributed nodes in the network, it is impossible to break them.

A few other notes of interest include the different types of Pay-To-Public-Key-Hash (P2PKH) and transactions.

Different Types of P2PKH

Other different types of bitcoin payments like pay-to-public-key-hash are:

- Pay to Public Key (Store Public Key as it is in the locking script instead of hash).
- Multi-Signature (Store N public keys are stored in the script).

Different Types of Transactions

Coinbase is a special type of transaction that doesn't have an unlocking script field. This field contains coinbase data, which is used by miners to put in arbitrary data. This is the first transaction in every block of the blockchain as a reward transaction for the Miner who created this block. This transaction generates new Bitcoin in the network from the air and therefore, there are no UTXO details. Satoshi Nakamoto, the inventor of Bitcoin, put the text below in this space:

"The Times 03/Jan/2009 Chancellor on brink of second bailout for banks."

Acronyms

UTXO	Unspent Transaction Output
P2PKH	Pay-to-Public-Key-Hash
RIPEMD-160	Is a cryptographic hash function based on Merkle–Damgård construction

References

https://en.bitcoin.it/wiki/Block_hashing_algorithm
https://en.bitcoin.it/wiki/Transaction
https://www.oreilly.com/library/view/mastering-bitcoin/9781491902639/ch05.html
https://www.youtube.com/watch?v=aaTchHpdplo

About the Author

Deenus Akkara has more than 20 years of experience in the IT field and has worked as Sr Solution Architect in GlobalLogic for almost 6 years. Akkara has worked on different domains like Telecom, AdNetwork, Blockchain, etc.

GlobalLogic®

GlobalLogic, a Hitachi Group Company, is a leader in digital product engineering. We help our clients design and build innovative products, platforms, and digital experiences for the modern world. By integrating our strategic design, complex engineering, and vertical industry expertise with Hitachi's Operating Technology and Information Technology capabilities, we help our clients imagine what's possible and accelerate their transition into tomorrow's digital businesses. Headquartered in Silicon Valley, GlobalLogic operates design studios and engineering centers around the world, extending our deep expertise to customers in the automotive, communications, financial services, healthcare & life sciences, media and entertainment, manufacturing, semiconductor, and technology industries.



www.globallogic.com