What Is Bitcoin Script?

Bitcoin Script serves as the scripting language behind Bitcoin transactions, allowing users to define the conditions under which funds can be spent. Unlike traditional financial transactions, Bitcoin transactions are not just about transferring value; they are also about executing specific conditions encoded in scripts.

It operates by manipulating items on a stack, with various opcodes representing operations that can be performed. This scripting language is deliberately limited and designed to be non-Turing complete for security reasons, preventing potentially malicious or infinite loops from being executed on the Bitcoin network.

Now, let's cover some of the finer details about Bitcoin Script.

Evolution of Bitcoin Script

Over the years, Bitcoin Script has undergone evolution and refinement. Its initial design was intentionally kept minimalistic to ensure security and avoid potential vulnerabilities. As the Bitcoin ecosystem has grown, developers and the community have explored ways to enhance the functionality of Bitcoin Script without compromising the network's security and stability.

The evolution of Bitcoin Script reflects the ongoing efforts to strike a delicate balance between providing flexibility and maintaining the robustness of the Bitcoin protocol. Innovations in Bitcoin Script aim to enable more sophisticated transaction types and smart contracts while upholding the core principles of security and decentralization that define the Bitcoin network.

In the next sections, we'll delve deeper into the fundamental components of Bitcoin Script, exploring its stack-based architecture, Forth-like structure, and the intriguing concept of Reverse-Polish Notation (RPN). Join us as we unravel the intricacies of Bitcoin Script, bringing clarity to its role in shaping the world of decentralized finance.

Examples of Bitcoin Script

There are several Bitcoin Script types. Here are a few examples.

1. ***Pay-to-PubKey (P2PK)***

- Sends bitcoins to a public key directly.
- ScriptPubKey: OP_CHECKSIG
- Usage: Early Bitcoin transactions.

2. ***Pay-to-PubKey-Hash (P2PKH)***

- Sends bitcoins to a Bitcoin address (hash of a public key).
- ScriptPubKey: OP_DUP OP_HASH160 <PubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
- Usage: Standard transactions, most common script type.

3. ***Pay-to-Script-Hash (P2SH)***

- Sends bitcoins to a script hash, enabling more complex transactions.
- ScriptPubKey: OP_HASH160 <ScriptHash> OP_EQUAL
- Usage: Multisig, escrow, and other complex transaction types.

4. ***Multisignature (Multisig)***

- Requires multiple signatures to authorize spending.
- ScriptPubKey: OP_n <PubKey1> <PubKey2> ... <PubKeyN> OP_m OP_CHECKMULTISIG
- Usage: Joint accounts, corporate accounts.

5. ***Pay-to-Witness-PubKey-Hash (P2WPKH)***

- Part of Segregated Witness (SegWit), sends to a hashed public key in witness data.
- ScriptPubKey: 0 <PubKeyHash>
- Usage: Reduces transaction size, lowers fees, improves scalability.

6. ***Pay-to-Witness-Script-Hash (P2WSH)***

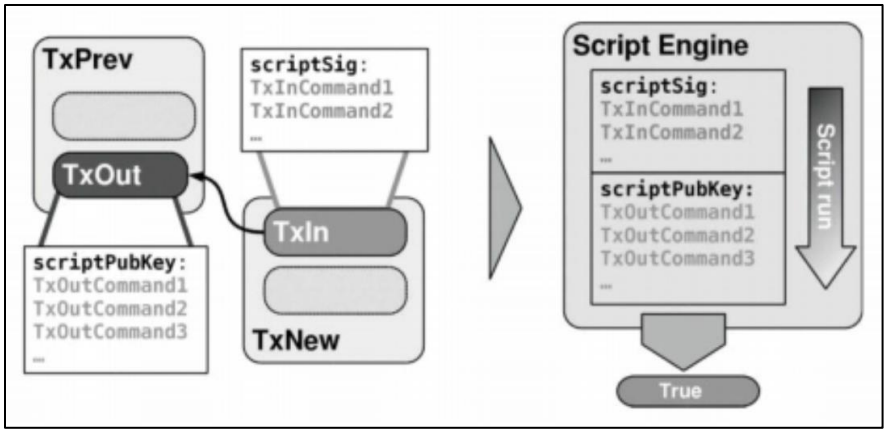
- Part of SegWit, sends to a script hash in witness data.
- ScriptPubKey: 0 <ScriptHash>
- Usage: Complex scripts and multisig transactions in SegWit format.

7. ***Pay-to-Taproot (P2TR)***

- Part of the Taproot upgrade, enabling more privacy and complex scripts.
- ScriptPubKey: OP_1 <TaprootPubKey>
- Usage: Enhanced privacy, efficiency, and flexibility for smart contracts.

8. ***OP_RETURN***

- Allows embedding data in the blockchain; marked as unspendable.
- ScriptPubKey: OP_RETURN <Data>
- Usage: Data storage, proof of existence, colored coins.

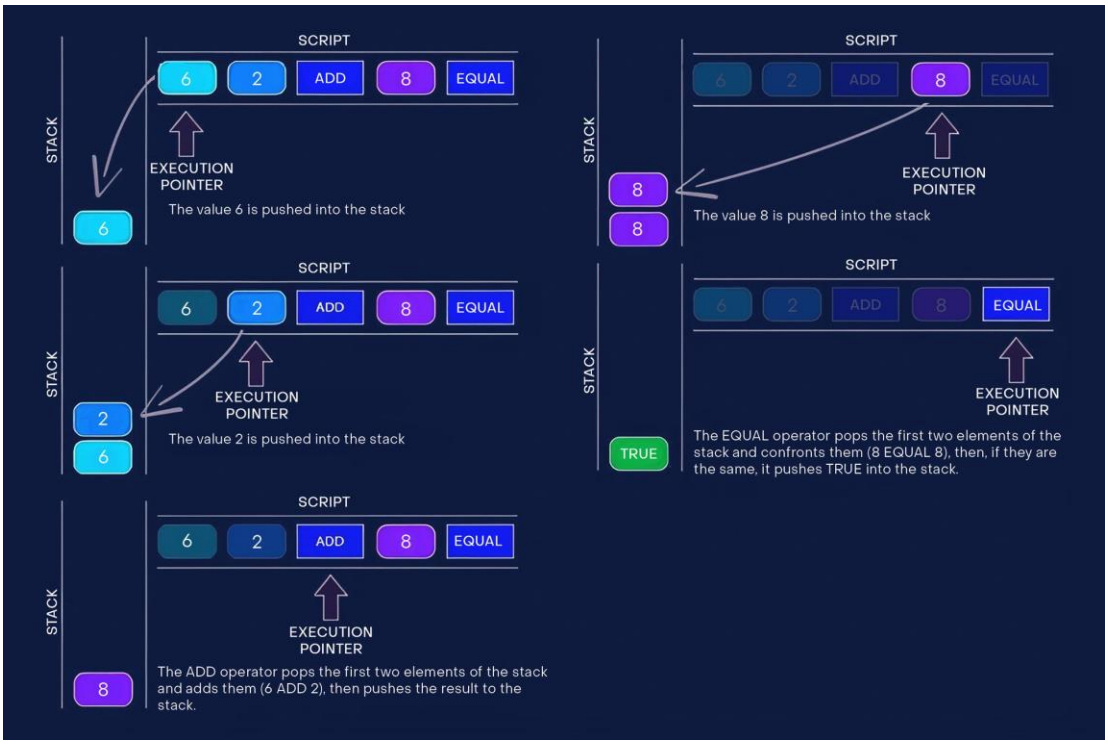


The Fundamentals of Bitcoin Script

Bitcoin Script operates on a set of fundamental principles that shape its functionality. Understanding these fundamentals is key to grasping how scripts are constructed and executed within the Bitcoin network.

Stack-based Architecture

Bitcoin Script employs a stack-based architecture, where data is organized and processed using a Last-In-First-Out (LIFO) structure. The stack serves as a temporary storage mechanism for operands and results of operations, allowing the script to manipulate and operate on these values efficiently. As transactions unfold, the stack dynamically changes, reflecting the state of the script's execution.



Example of how Last In, First Out (LIFO) works with the Stack-based Bitcoin Script

This stack-based approach simplifies the scripting language, making it lightweight and efficient for the specific purpose of enabling conditional spending in Bitcoin transactions. The stack serves as a crucial element in executing various opcodes, each contributing to the overall functionality of the script.

Forth-Like Structure

Bitcoin Script draws inspiration from Forth, a stack-based programming language. This Forth-like structure is evident in the simplicity of Bitcoin Script's design, emphasizing minimalism and efficiency. The script is composed of a series of opcodes, each representing a specific operation that can be performed on the stack.

The Forth-like structure enhances the predictability and security of Bitcoin Script, as each opcode has a well-defined purpose and predictable behavior. This design choice aligns intending to create a secure and robust scripting language for financial transactions on the Bitcoin network.

Reverse-Polish Notation (RPN)

One unique feature of Bitcoin Script is its use of Reverse-Polish Notation (RPN) to express operations. In RPN, operators are placed after their operands, eliminating the need for parentheses and order of operations rules. This notation simplifies the parsing and execution of scripts, contributing to the overall efficiency of Bitcoin Script.

The use of RPN in Bitcoin Script aligns with its commitment to simplicity and clarity. By adopting a notation that reduces ambiguity, Bitcoin Script enhances the readability of scripts and ensures that the execution of transactions follows a clear and predictable path.

Turing Incompleteness

Bitcoin Script is intentionally designed to be Turing incomplete. Turing completeness is a property of a system that can perform any computation that a Turing machine can. While Turing completeness allows for greater flexibility in programming languages, it also introduces potential security risks.

The decision to keep Bitcoin Script Turing incomplete is a deliberate choice to enhance the security and predictability of the Bitcoin network. By limiting the expressive power of scripts, the Bitcoin protocol minimizes the risk of unintended or malicious behavior that could compromise the stability of the entire network.

In the upcoming sections, we'll explore how these fundamental principles of Bitcoin Script come to life in the context of real-world transactions. Join us as we uncover the intricacies of BTC Script in the next installment of our journey into the world of cryptocurrency scripting.

Components of Bitcoin Script

Bitcoin Script is a simple yet powerful scripting language used within the Bitcoin protocol.

- It's designed to process and validate transactions, ensuring the correct transfer of Bitcoin between parties.

- The language consists of a series of opcodes that perform specific operations, two primary scripts that handle the locking and unlocking of funds, and a set of rules that govern their execution.
- These components work together to create a secure and decentralized system that can execute complex conditions for spending bitcoins. Understanding these elements is essential for anyone looking to grasp the functionality and potential of Bitcoin Script.

Now, let's take a closer look at each of these components, starting with the fundamental operation codes, or OP_CODES, that act as the building blocks of every Bitcoin Script.

Understanding OP_CODE

- At the heart of Bitcoin Script are operation codes, commonly known as OP_CODES. These are the basic building blocks of a script, instructing the network on how to process transactions.
- Each OP_CODE represents a specific operation, ranging from mathematical calculations to data manipulation and transaction validation.
- They are the equivalent of verbs in a programming language, defining actions that manipulate the stack during the execution of a script.
- Understanding OP_CODES is crucial because they define the conditions under which a transaction is considered valid and can be added to the blockchain.

scriptSig and scriptPubKey Components

- Two fundamental components in the Bitcoin Script universe are scriptSig and scriptPubKey.
- These two scripts work in tandem across every Bitcoin transaction, ensuring the secure transfer of bitcoins from one party to another.

Role and Function of scriptSig

- The scriptSig, or signature script, is used to provide evidence that the transaction initiator has the necessary permissions to spend the bitcoins.
- It typically contains a digital signature that matches the public key hashed in scriptPubKey of the output being spent.
- This signature serves as proof that the owner of the private key corresponding to the Bitcoin address in question has authorized the transaction.
- It's a critical part of the transaction verification process, as it unlocks the bitcoins held in a previous transaction, allowing them to be spent.

Purpose and Structure of scriptPubKey

- Conversely, the scriptPubKey is the locking script. Placed within an output, this script specifies the conditions that must be met to spend the bitcoins in the future.

- It acts as a safeguard, ensuring that only the intended recipient can unlock and claim the bitcoins.
- This script can encode conditions such as requiring a correct digital signature or even more complex criteria like multisig requirements, time locks, or even the resolution of a cryptographic puzzle.
- The structure of scriptPubKey can vary depending on the type of transaction, but its ultimate purpose is to ensure security and control over Bitcoin transactions.

Overview of Common Opcodes in Bitcoin Script

The following section delves into common OP_CODES in Bitcoin Script, shedding light on their functions and significance in transaction processing.

- **OP_ADD**: Pops two items off the stack, adds them together, and pushes the result back onto the stack.
- **OP_EQUAL**: Pops two items from the stack and compares them to check if they are equal. if they are equal, then it pushes the result TRUE back onto the stack.
- **OP_RETURN**: Can be used to store up to 80 bytes of arbitrary data on the Bitcoin blockchain and also to mark a transaction output as invalid. OP_RETURN transaction outputs are provably unspendable, making this opcode an efficient way to burn BTC. It also makes Komodo's delayed Proof of Work (dPoW) security mechanism possible.
- **OP_CHECKSIG**: Verifies that the signature for a transaction input is valid.
- **OP_CHECKMULTISIG**: Commonly used in Pay To Script Hash (P2SH) transactions. **OP_CHECKMULTISIG** looks at 3 public keys and 2 signatures in the stack and compares them one by one. Funds become spendable only when the order of the signatures matches the order in which the public keys were provided.

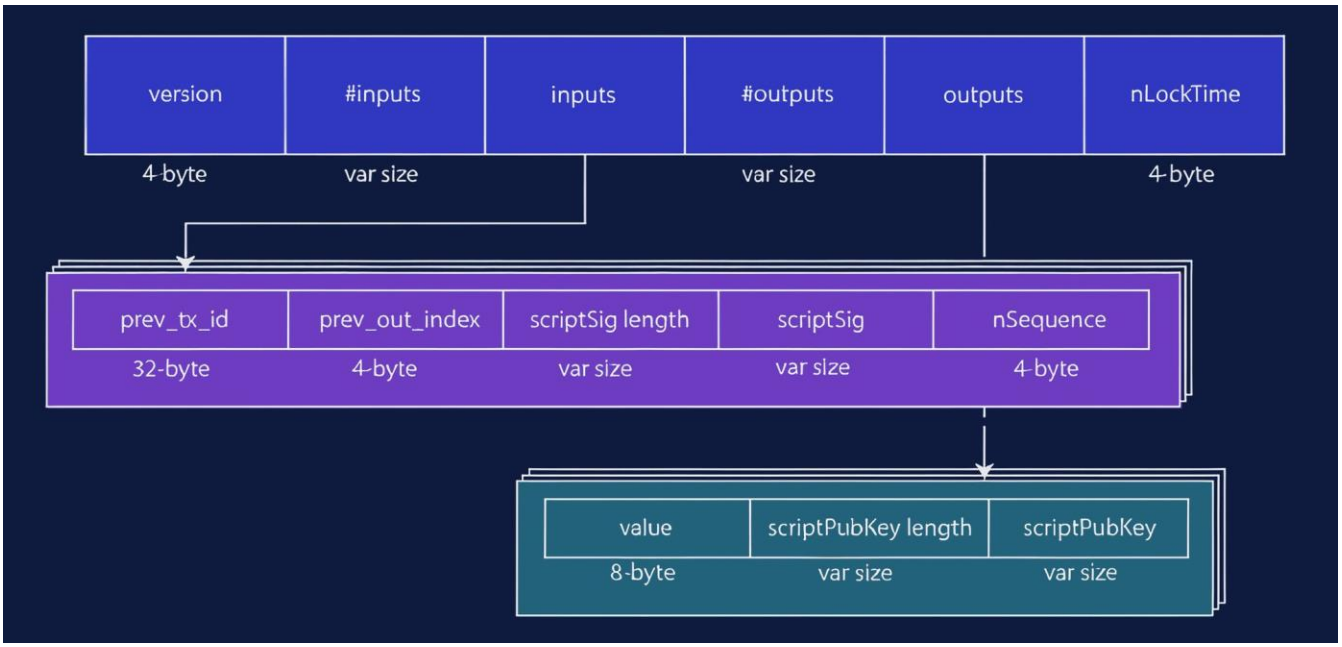
Bitcoin Script in Transactions

Understanding the purpose of Bitcoin Scripts is essential for grasping the basics of Bitcoin transactions.

Within each transaction, Bitcoin Script provides the flexibility to implement various types of payment conditions, from the most straightforward to the exceedingly complex.

It's the foundation of trust in the Bitcoin network, as it allows for the creation of a decentralized system where trust is not placed in a central authority, but in the cryptographic proofs that the scripts provide.

As we explore this section, we will unveil how Bitcoin Script operates within transactions, providing both security and versatility to the network.



Every Bitcoin transaction relies upon a series of inputs and ouputs.

How Bitcoin Script Operates in a BTC Transaction

Bitcoin Script plays a pivotal role in the execution and validation of transactions on the Bitcoin network. It's the underlying mechanism that enforces the rules for spending bitcoins. Every transaction comprises scripts that dictate both the spending conditions and the fulfillment of those conditions. These scripts ensure that bitcoins can only be spent by the rightful owners and that the network reaches a consensus on the state of the blockchain.

Basics of a Bitcoin Transaction

A Bitcoin transaction is a digital exchange of bitcoins between parties. Each transaction consists of inputs and outputs. Inputs refer to where the bitcoins are coming from, typically referencing a previous transaction's output. Each input contains a scriptSig. Outputs designate where the bitcoins are going and contain a scriptPubKey. When you initiate a transaction, you effectively create a cryptographic puzzle that the recipient's scriptSig must solve to spend the bitcoins in the future.

Detailed Look at scriptSig and scriptPubKey in Action

The scriptSig and scriptPubKey are scripts that interact within a Bitcoin transaction to ensure secure transfers. The scriptPubKey lays out the conditions under which the funds can be claimed, and the scriptSig provides the solution to this cryptographic challenge. During the verification process, the scriptSig is executed first, followed by the scriptPubKey. If the final result is true, the transaction is valid, and the funds can be spent; otherwise, the transaction is rejected.

Interacting with Bitcoin Script via Command Line

For those with technical expertise, Bitcoin Script can be interacted with directly via the command line. This interaction involves the creation, analysis, and testing of Bitcoin Script transactions.

Bitcoin Core, the reference implementation of Bitcoin, provides a set of RPC commands that allow users to construct transactions manually, inspect their scripts, and broadcast them to the network.