Table of contents

Ledgers

- Account-based ledger
  - Keep tracking account balance for every individual.

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$_{\text{SIGNED(Alice)}}$

Ledgers

- Tx-based ledger

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$_{\text{SIGNED(Alice)}}$

1	Inputs: Ø Outputs: 25.0→Alice	
2	Inputs: 1[0] Outputs: 17.0→Bob, 8.0→Alice	SIGNED(Alice)
3	Inputs: 2[0] Outputs: 8.0→Carol, 9.0→Bob	SIGNED(Bob)
4	Inputs: 2[1] Outputs: 6.0→David, 2.0→Alice	SIGNED(Alice)

→ here alice is spending total money in the transaction mandatorily. She wants to give 17 to bob, but transaction is of 25. So she is sending remaining 8 to herself.

Ledgers

- Tx-based ledger
  - Change address → here change means, change that we receive after giving a bigger rounded off amount.
    - Sending money to herself in this example?
    - Immutability in Bitcoin, use all or none Tx output
    - Send the change to your (new/old) address

1	Inputs: Ø Outputs: 25.0→Alice	
2	Inputs: 1[0] Outputs: 17.0→Bob, 8.0→Alice	SIGNED(Alice)
3	Inputs: 2[0] Outputs: 8.0→Carol, 9.0→Bob	SIGNED(Bob)
4	Inputs: 2[1] Outputs: 6.0→David, 2.0→Alice	SIGNED(Alice)

Ledgers

- Tx-based ledger
  - Efficient verification
    - Check if input value is sufficient
  - Consolidating funds
    - Multi output, multi input, splitting and merging is easy
  - Joint payments
    - Multi-signature
- Bitcoin is Unspent Transaction Output (UTXO) model
  - Address = identity

Tx syntax

metadata

input(s)

output(s)


```
{
  "hash":"5a42590fbe0a90ee8e8747244d6c84f0db1a3a24e8f1b95b10c9e050990b8b6b",
  "ver":1,
  "vin_sz":2,
  "vout_sz":1,
  "lock_time":0,
  "size":404,
  "in":[
   {
    "prev_out":{
     "hash":"3be4ac9728a0823cf5e2deb2e86fc0bd2aa503a91d307b42ba76117d79280260",
     "n":0
    },
      "scriptSig":"30440..."
   },
   {
    "prev_out":{
     "hash":"7508e6ab259b4df0fd5147bab0c949d81473db4518f81afc5c3f52f91ff6b34e",
     "n":0
    },
    "scriptSig":"3f3a4ce81...."
   }
  ],
  "out":[
   {
    "value":"10.12287097",
    "scriptPubKey":"OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"
   }
  ]
}
```

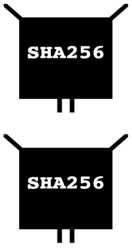
Tx syntax

Field	Description
hash	Hash/TxID
ver	Bitcoin version/rules-set
vin_sz/vout_sz	# of input/output
lock_time	Earliest time when this Tx can be added to blockchain ● 0 = immediate execution ● < 500,000,000 : Unlocked at block height ● ≥ 500,000,000 : Unlocked at specific time (in Unix time)
size	Tx size in bytes
in/out	Array of Tx input
prev_out	Info about previous Tx & output that is being spent ● hash: Previous TxID ● n: Index of the output (being used now) in previous Tx
value	Bitcoin value

Tx syntax

- Transaction ID: A TxID is basically an identification number for a Bitcoin Tx (i.e., hash field)
  - TxID = SHA256(SHA256(Tx_Data)) → double hashing using SHA256.

```
0100000001c997a5e56e104102fa209c6a852dd90660a20b2d9c352423edce25857fcd3704000000004847304402204e45e16932b8af514961a1d3a1a2
5fdf3f4f7732e9d624c6c61548ab5fb8cd410220181522ec8eca07de4860a4acdd12909d831cc56cbbac4622082221a8768d1d0901ffffffff0200ca9a
3b00000000434104ae1a62fe09c5f51b13905f07f06b99a2f7159b2225f374cd378d71302fa28414e7aab37397f554a7df5f142c21c1b7303b8a0626f1
baded5c72a704f7e6cd84cac00286bee0000000043410411db93e1dcdb8a016b49840f8c53bc1eb68a382e97b1482ecad7b148a6909a5cb2e0eaddfb84
ccf9744464f82e160bfa9b8b64f9d4c03f999b8643f656b412a3ac00000000
```



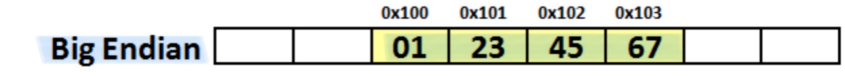

```
169e1e83e930853391bc6f35f605c6754cfead57cf8387639d3b4096c54f18f4
```

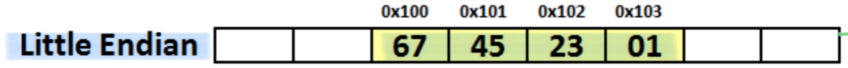
  - Swap endian to find this Tx

```
f4184fc596403b9d638783cf57adfe4c75c605f6356fbc91338530e9831e9e16
```

Tx syntax

- Endianness
  - Big-endian system
    - LSB > largest address (MSB > smallest address)
  - Little-endian system
    - LSB > smallest address
  - A hex number: 0x01234567





→ notice carefully, even though from previous the two here lists which make one byte are 01, 23, ... not 10, 32, ...

- Bitcoin uses little-endian format for some data, so when coding, we often have to get things in to little-endian format

Tx syntax

- Bitcoin uses double hashing almost everywhere
  - Length extension attack, collisions, etc. → double hashing helps in protection against these attacks.
- It hashes in one of two variants:
  - RIPEMD160(SHA256(x)) or Hash160, produces a 160-bit output
    - Hashing public key to generate part of a Bitcoin addresses
  - SHA256(SHA256(x)) or double-SHA256, produces a 256-bit output
    - Creating TxID
    - Hashing Tx in a merkle tree
    - Hash of block header (thus PoW & link to previous block)

Tx syntax

- Length extension attack
  - Algorithms based on Merkle–Damgård construction are susceptible
  - Given *Hash(message1)*, $length_{message1}$
    - Possible to calculate *Hash(message1 || message2)*
      - For attacker-controlled *message2*
    - Without knowing *message1*
  - Construction *H(secret || message)*
    - *message* and $length_{secret}$ is known
    - It allows appending extra info to *message*
    - To produce a valid hash without knowing *secret*

A length extension attack is a type of cryptographic attack on hash functions (like SHA-256) where an attacker, knowing the hash of a message, can add extra data to the original message and calculate a valid hash for the new longer message, without knowing the original secret message. This is possible with certain hash functions based on the Merkle-Damgård construction. Double hashing prevents from this attack.

The Merkle-Damgård construction is a method used to build cryptographic hash functions. It works by breaking an input message into fixed-size blocks, processing each block with a compression function, and passing the output to the next block. It starts with an initialization vector (IV) and produces a fixed-length hash as the final output. This construction is used in hash functions like SHA-256 and MD5. However, it's susceptible to certain attacks like length extension attacks.

Bitcoin script

- When sending a Tx, we say "*this Tx/BTC can be redeemed by a signature from the owner of address X*"
- However, Bitcoin Address = hash of a public key (we'll see it later)
- Only specifying address *X* doesn't tell what the public key is
  - No way to check the signature
- So, we instead say "*this can be redeemed by a public key that hashes to X, along with a signature of owner of that public key*"
  - Pay-to-Public-Key-Hash (P2PKH)

Bitcoin script

- Some sort of logic execution/scripting
  - Who runs it?
  - Who defines the sequence of instructions?
- Who run it?
  - Remember puzzle-solvers, miners?
  - Validating Tx is part of solving the puzzle
- Who defines the sequence of instructions?
  - Both input and output contain scripts

Bitcoin script

- Locking script or scriptPubKey
  - In output
  - When sending, whoever can unlock this lock can take it
  - Specifies a public key (or an address to which this public key hashes)
- Unlocking script or scriptSig
  - In input
  - When spending, to unlock a previous output
  - Specifies a signature with that public key
- If a full script (unlocking || locking) executes successfully, the output is “unlocked” and can be spent

Bitcoin script

- Bitcoin scripting language, Script
  - Basic programming language, consists of two types of thing
    - Data, e.g., public keys, signatures
    - OPCODES are simple functions that operate on data
  - Defines what exactly a Tx does
  - Certain degree of programmability
  - Forth-like language
  - Stack-based
    - Each instruction is executed only once, linearly

Bitcoin script

- Bitcoin scripting language, Script
  - Supports
    - Basic arithmetic operations, Basic logic (e.g., if and then)
    - Error throwing, returning early, etc.
    - Native support for cryptographic operations (e.g., verify signatures)
  - Doesn't support
    - Programming loops
      - "Turing-incomplete" language
      - Intentional
        - Avoid infinite loops
  - Stateless verifications
    - No state prior to execution, no state saved after execution

Bitcoin script

- Bitcoin scripting language, Script
  - The script is valid if after execution
    - The top & only element left on the stack is a ‘1’ (or greater)
  - The script is invalid if after execution
    - Final stack is empty
    - Top & only element on the stack is ‘0’
    - >1 elements left on the stack
    - Exits prematurely

Bitcoin script

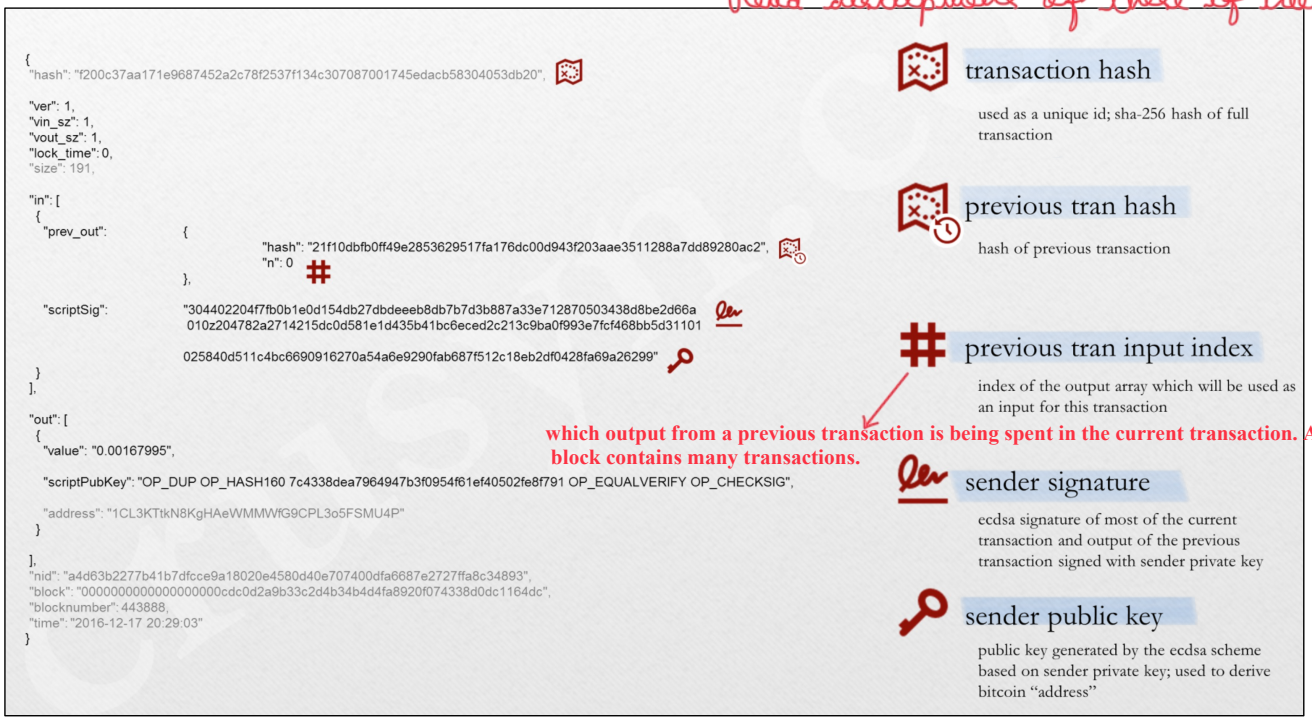
- Bitcoin scripting language, Script
  - Some commonly used instructions

Opcode	Description
OP_DUP	Replicates the top item on the stack
OP_HASH160	Hashes the top item twice (First SHA-256 and then RIPEMD-160)
OP_EQUALVERIFY	Checks if inputs are equal. If not, returns false and invalidates Tx
OP_CHECKSIG	Checks if the input signature is valid for the input public key
OP_CHECKMULTISIG	Validates *<m>* input signatures in *<n>* public keys of locking script
OP_RETURN	Terminates the execution and marks the transaction as invalid

Bitcoin script

- Current Tx

Read descriptions of these if needed



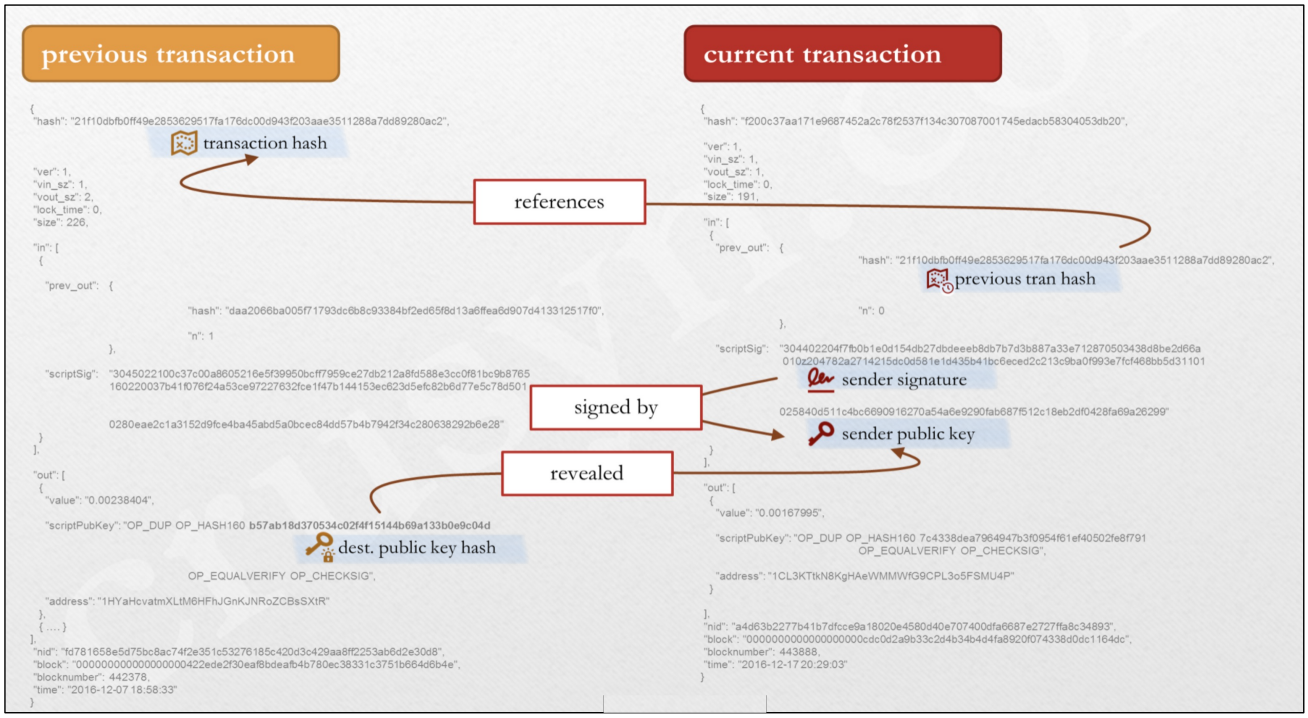
Bitcoin script

- Previous Tx

Read descriptions of these if needed.

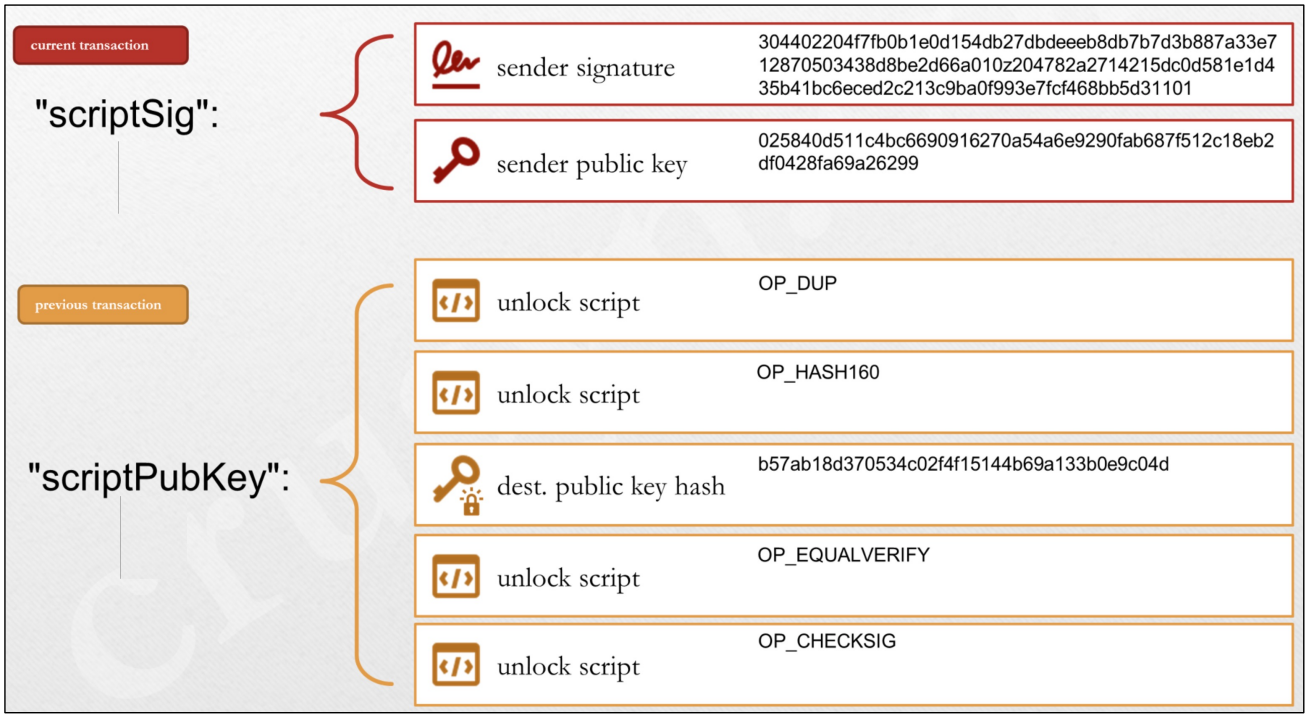
Bitcoin script

- Relationship in current and previous Tx



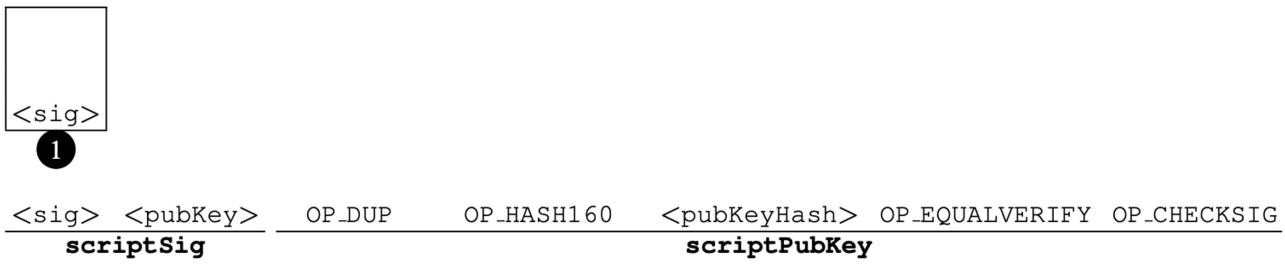
Bitcoin script

- scriptSig and scriptPubKey



Bitcoin script

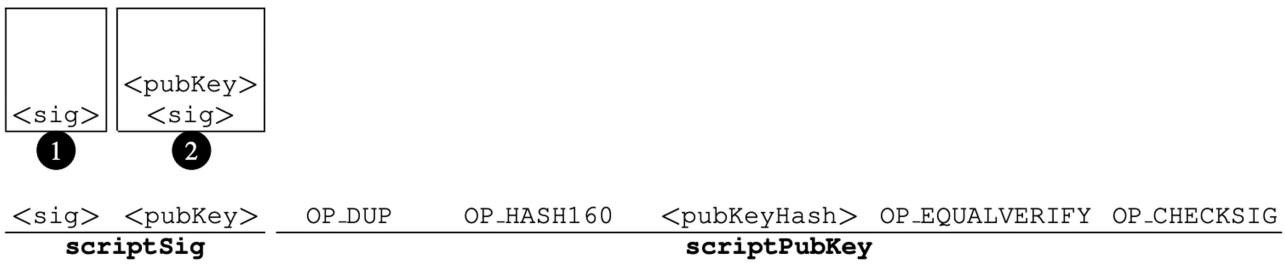
- P2PKH script execution on a stack



- Take scriptSig from **current** Tx & scriptPubKey from **previous** Tx

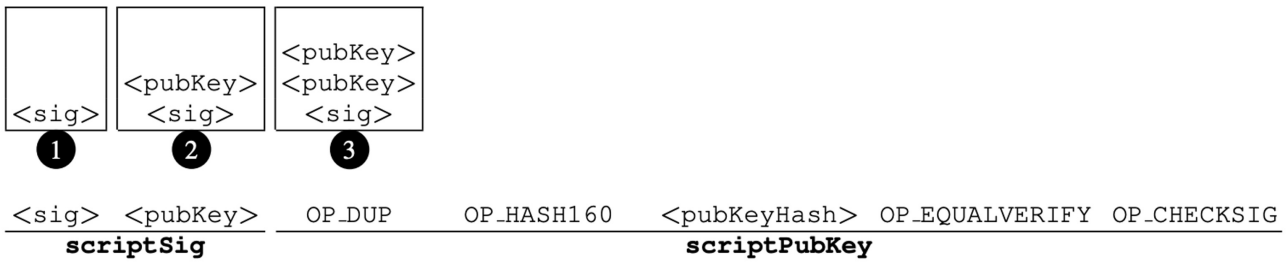
Bitcoin script

- P2PKH script execution on a stack



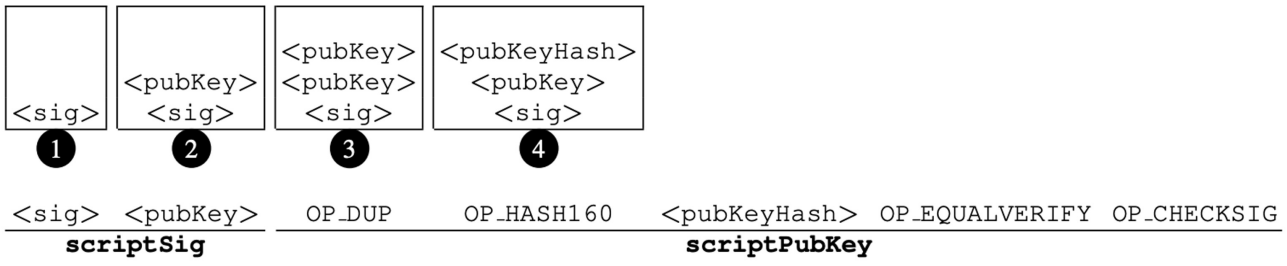
Bitcoin script

- P2PKH script execution on a stack



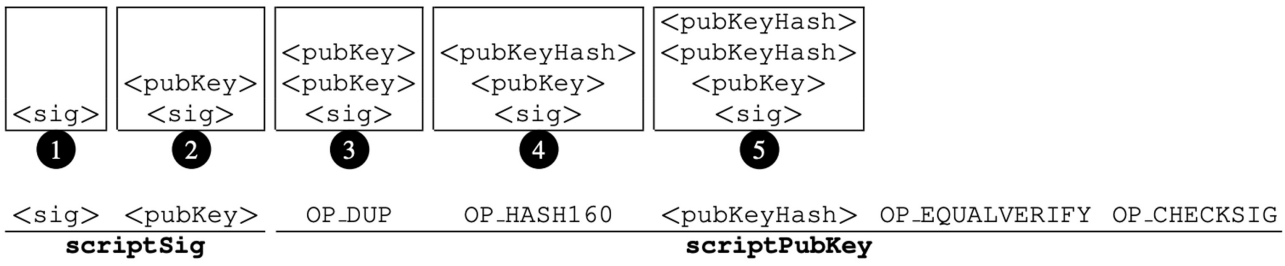
Bitcoin script

- P2PKH script execution on a stack



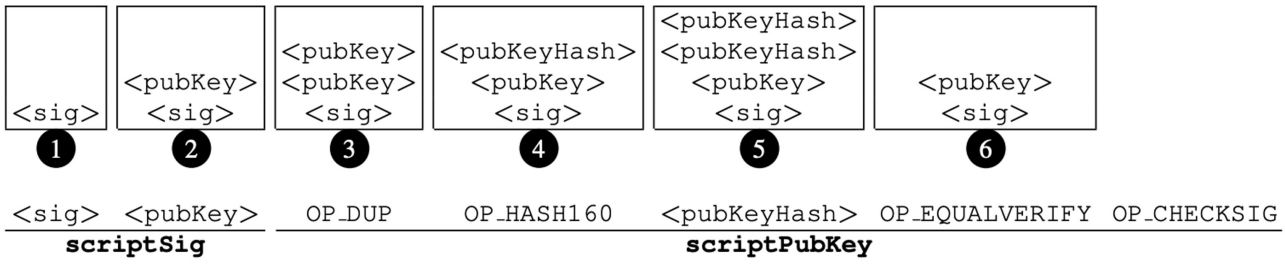
Bitcoin script

- P2PKH script execution on a stack



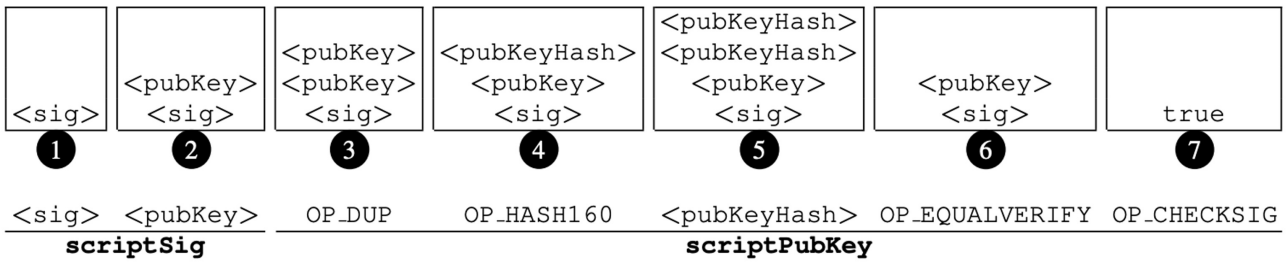
Bitcoin script

- P2PKH script execution on a stack



Bitcoin script

- P2PKH script execution on a stack



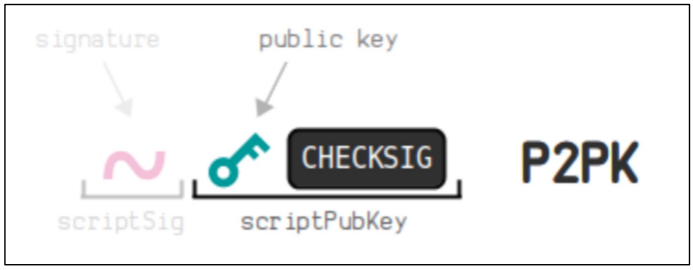
Bitcoin script

- scriptSig and scriptPubKey for standard Bitcoin scripts

Script name	**scriptSig**	**scriptPubKey**
Pay-to-Public-Key (P2PK)	*<sig>*	*<pubKey> OP_CHECKSIG*
Pay-to-Public-Key-Hash (P2PKH)	*<sig> <pubKey>*	*OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG*
Pay-to-MultiSig (P2MS)	*OP_0 [<sig>...]*	*<m> [<pubKey>...] <n> OP_CHECKMULTISIG*
Pay-to-Script-Hash (P2SH)	*[<sig>...] <redeemScript>*	*OP_HASH160 <redeemScriptHash> OP_EQUAL*
NULL DATA		*OP_RETURN <data>*

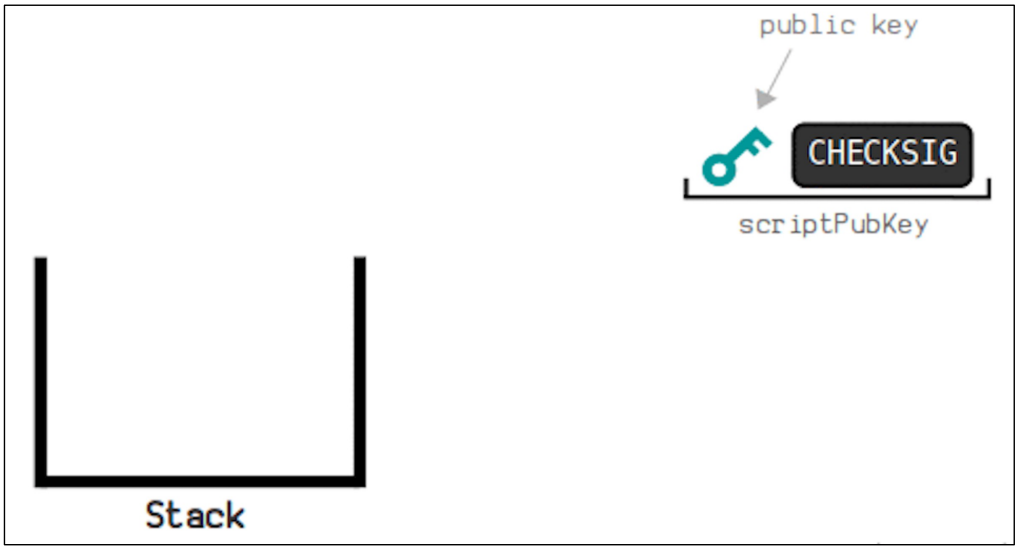
Bitcoin script

- P2PK locks an output to a public key
  - Commonly found in coinbase Tx in earlier blocks (e.g., genesis block)
  - P2PK was soon replaced by P2PKH for users' convenience
    - Public keys are longer than addresses



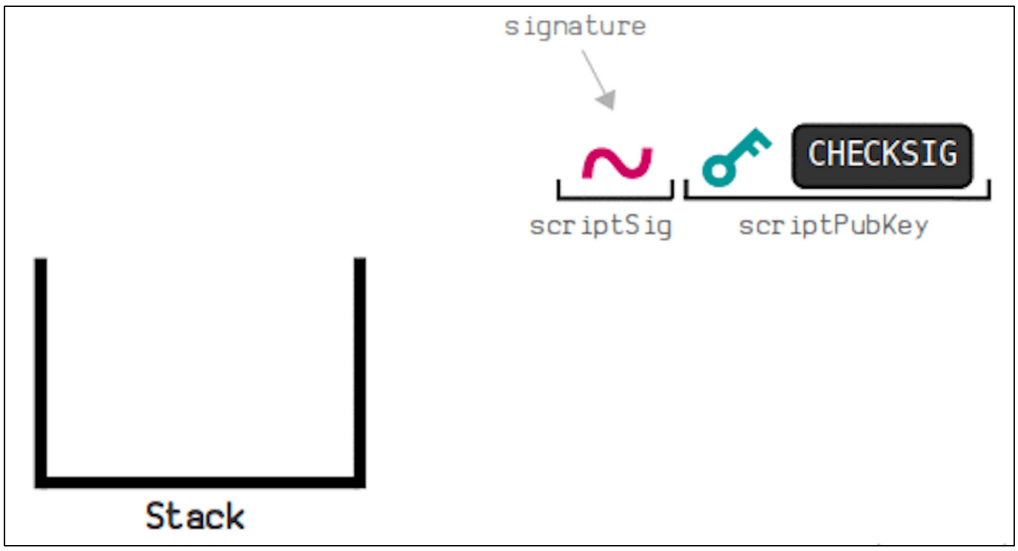
Bitcoin script

- P2PK



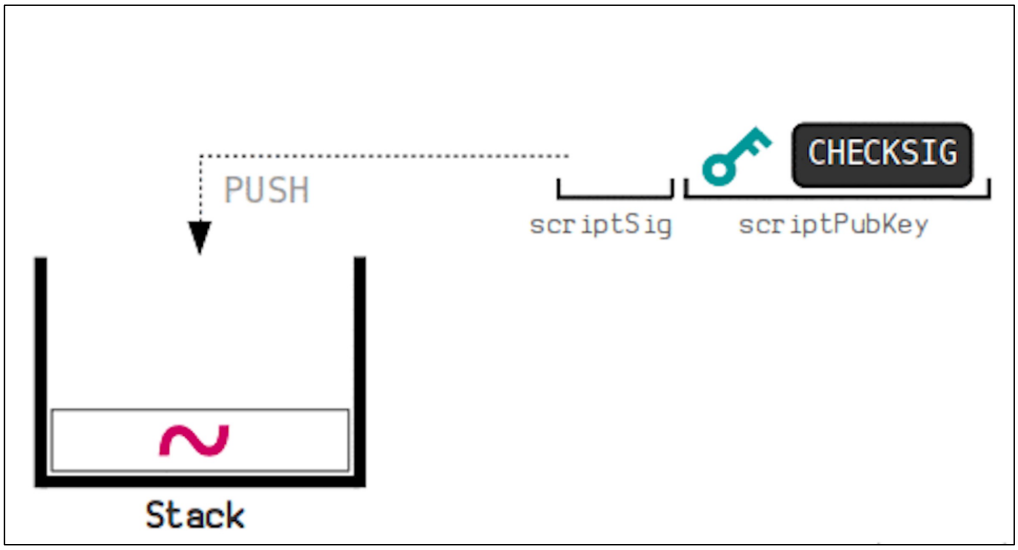
Bitcoin script

- P2PK



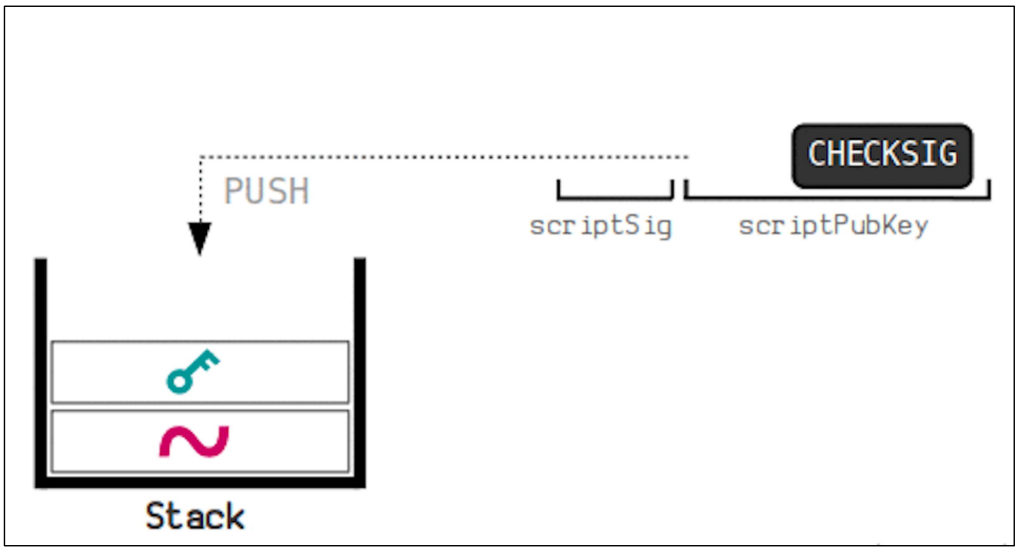
Bitcoin script

- P2PK



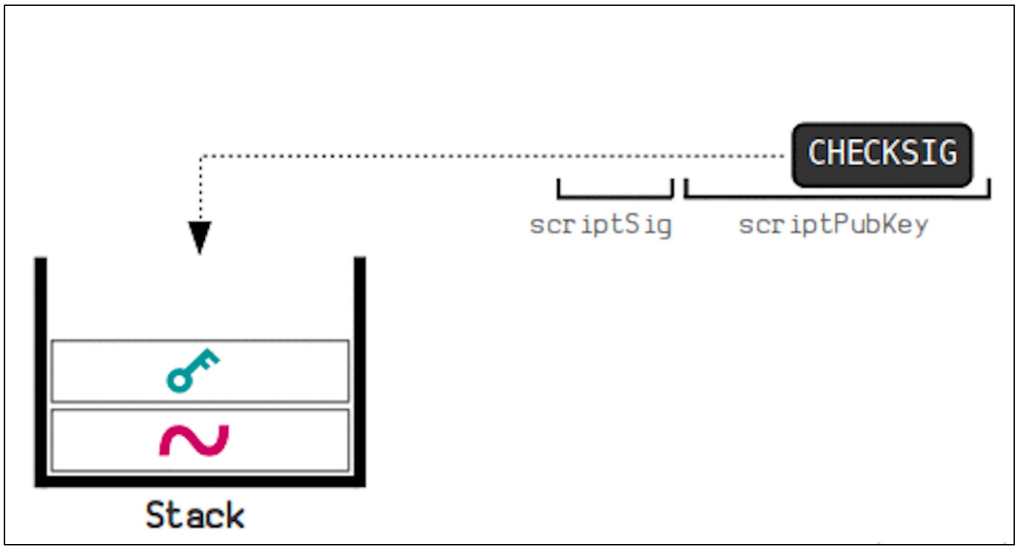
Bitcoin script

- P2PK



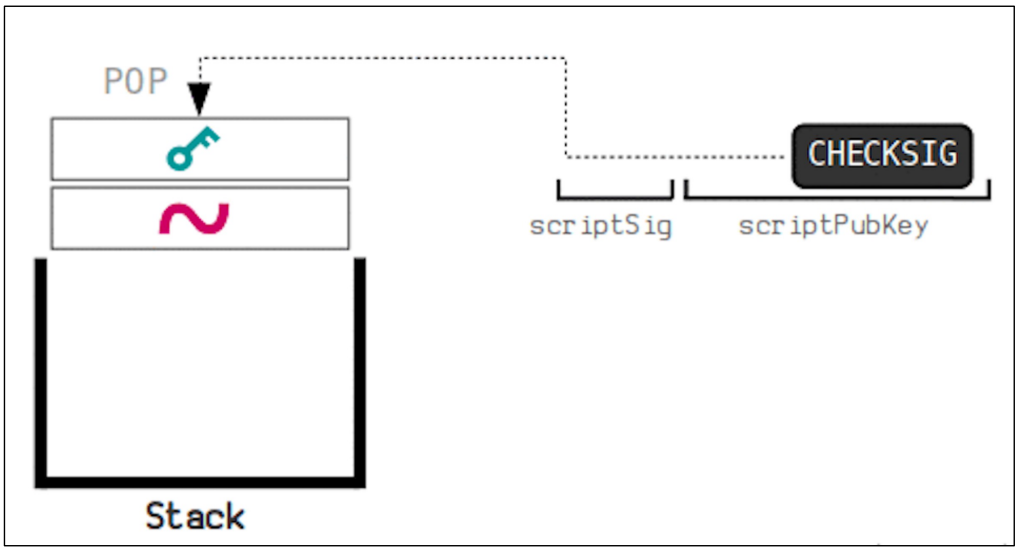
Bitcoin script

- P2PK



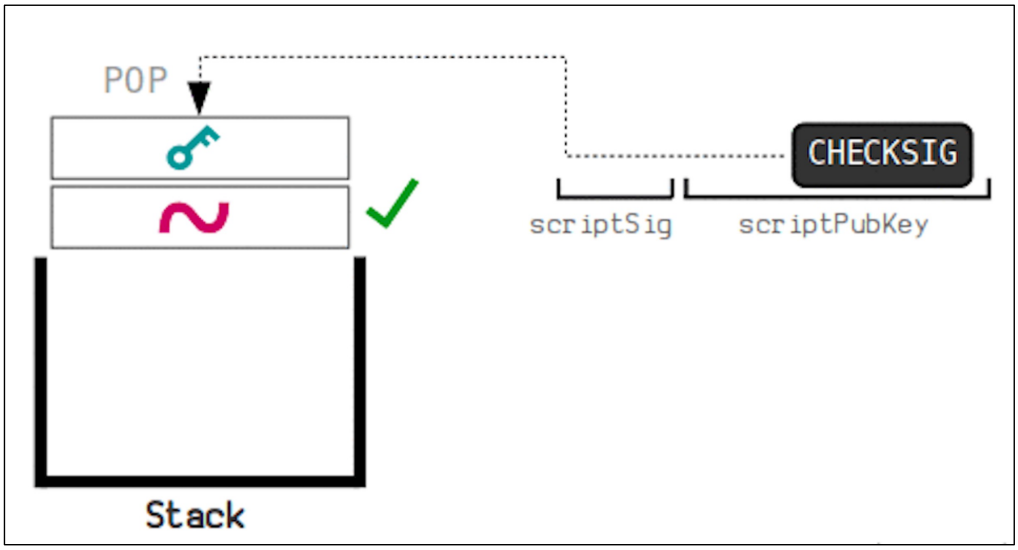
Bitcoin script

- P2PK



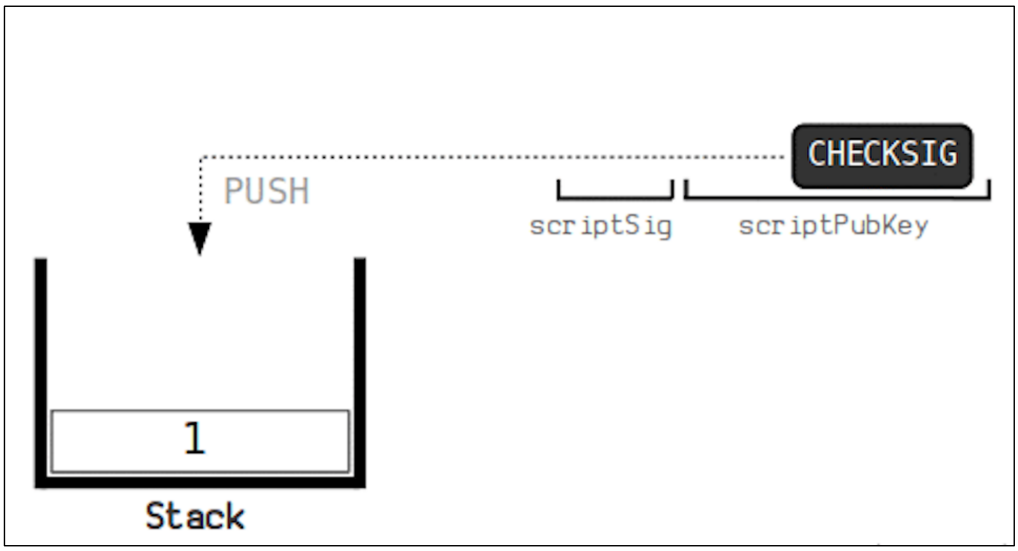
Bitcoin script

- P2PK



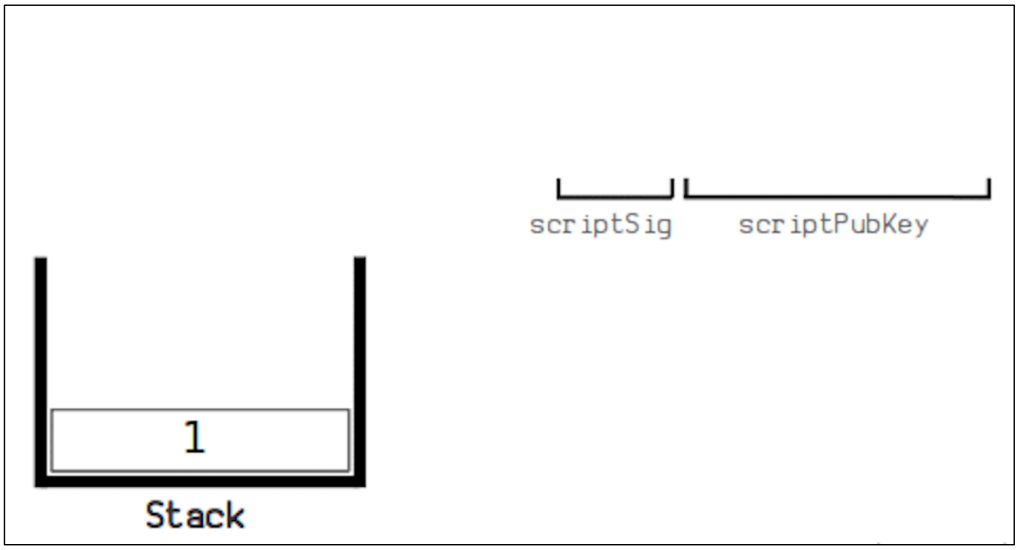
Bitcoin script

- P2PK



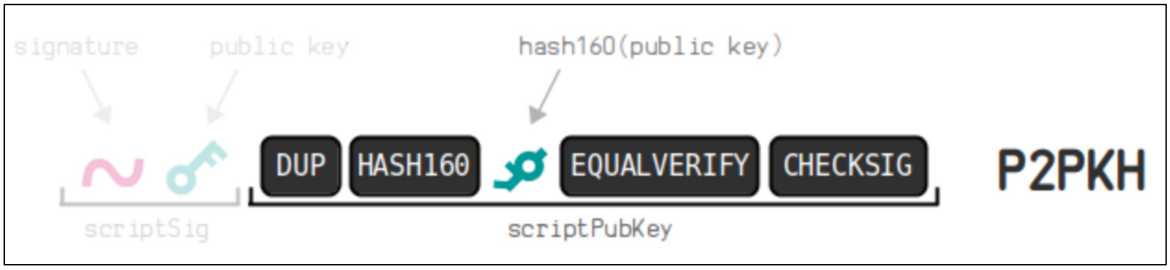
Bitcoin script

- P2PK



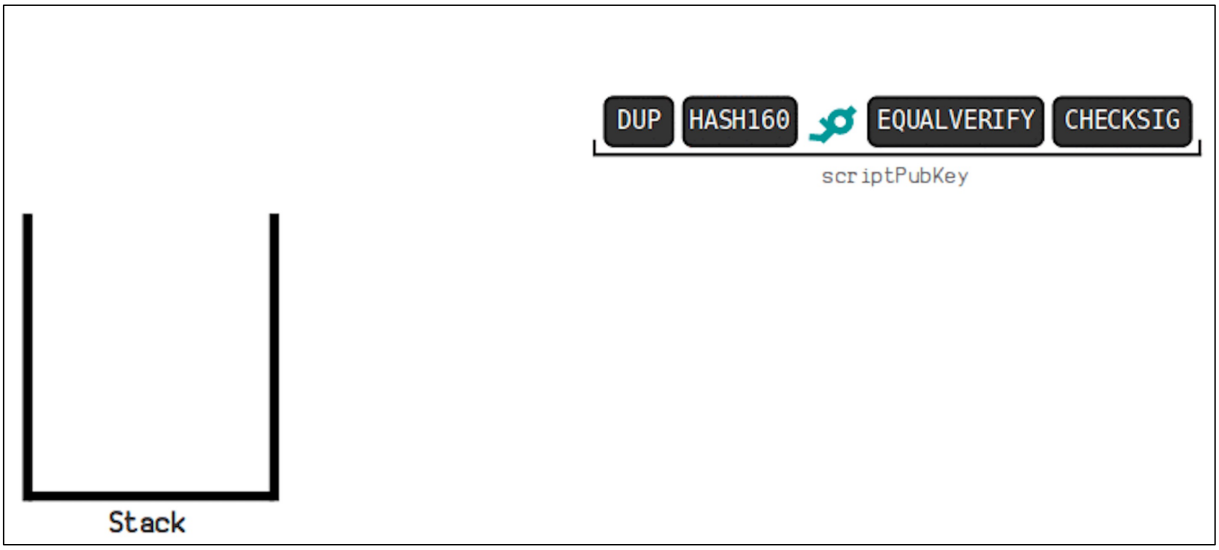
Bitcoin script

- P2PKH lock contains hash of a public key, not the public key itself
  - Uses addresses instead of public keys
  - Default script used by wallets



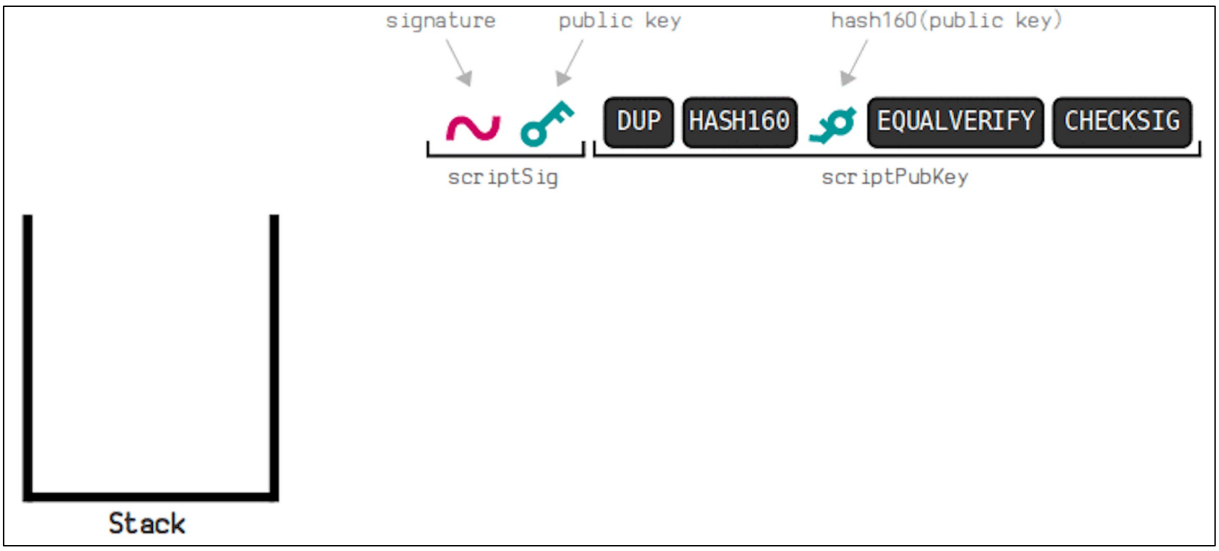
Bitcoin script

- P2PKH



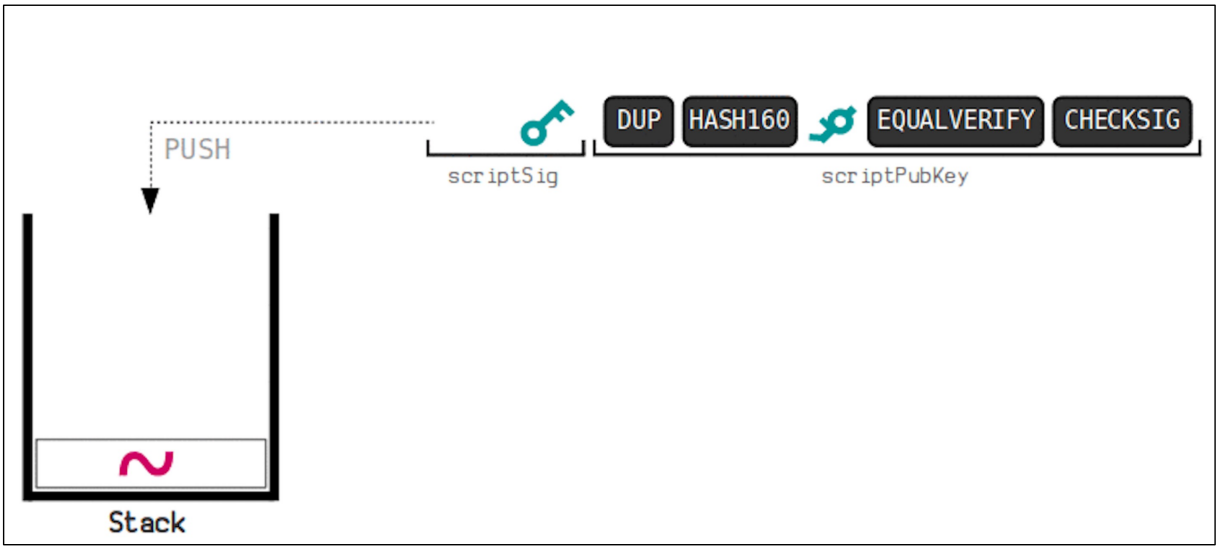
Bitcoin script

- P2PKH



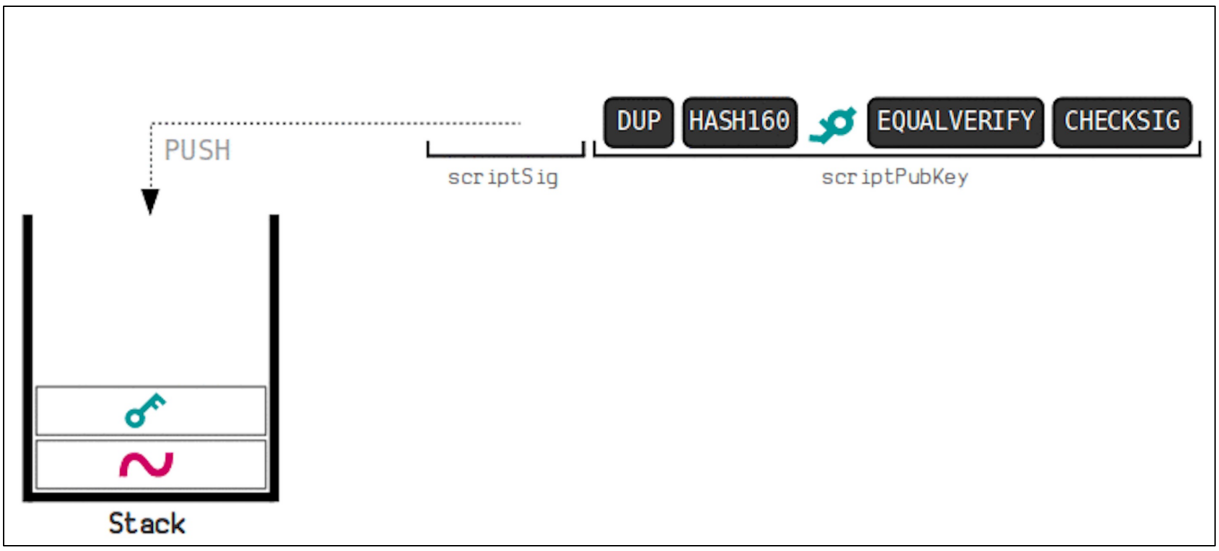
Bitcoin script

- P2PKH



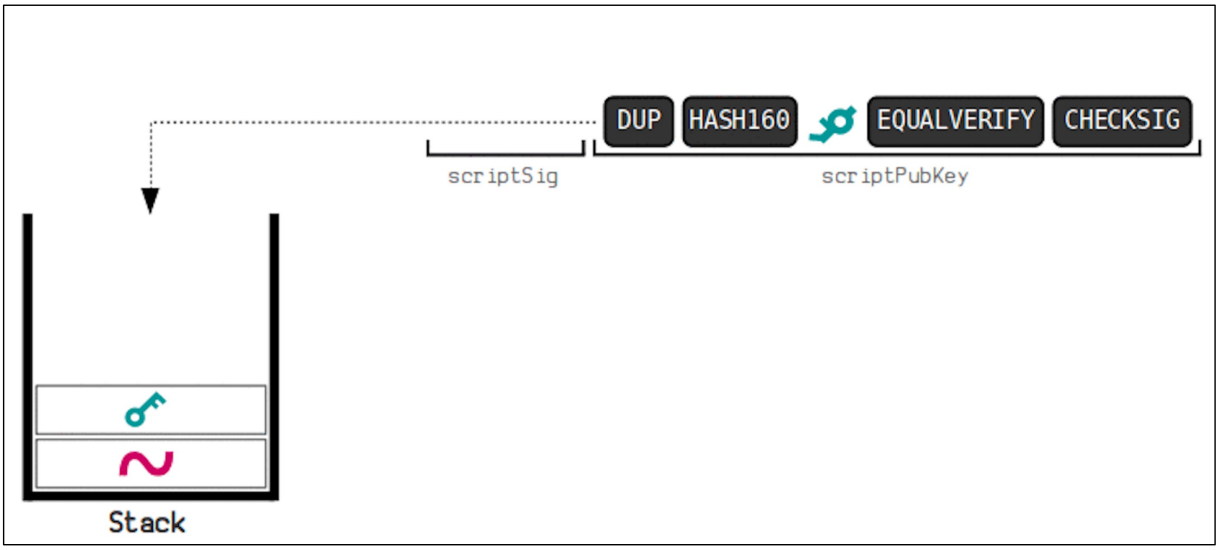
Bitcoin script

- P2PKH



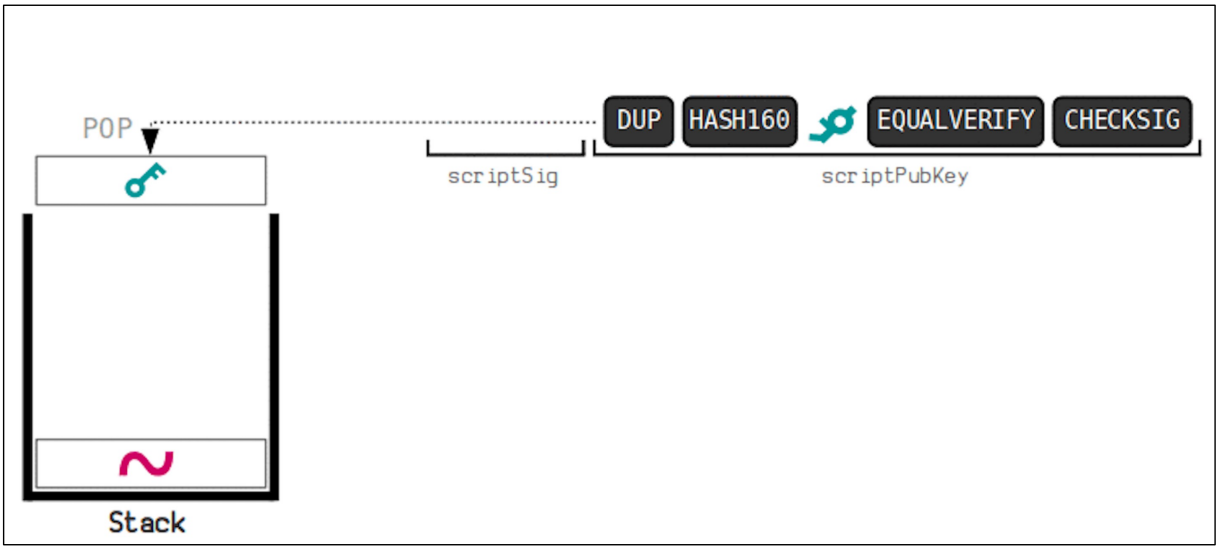
Bitcoin script

- P2PKH



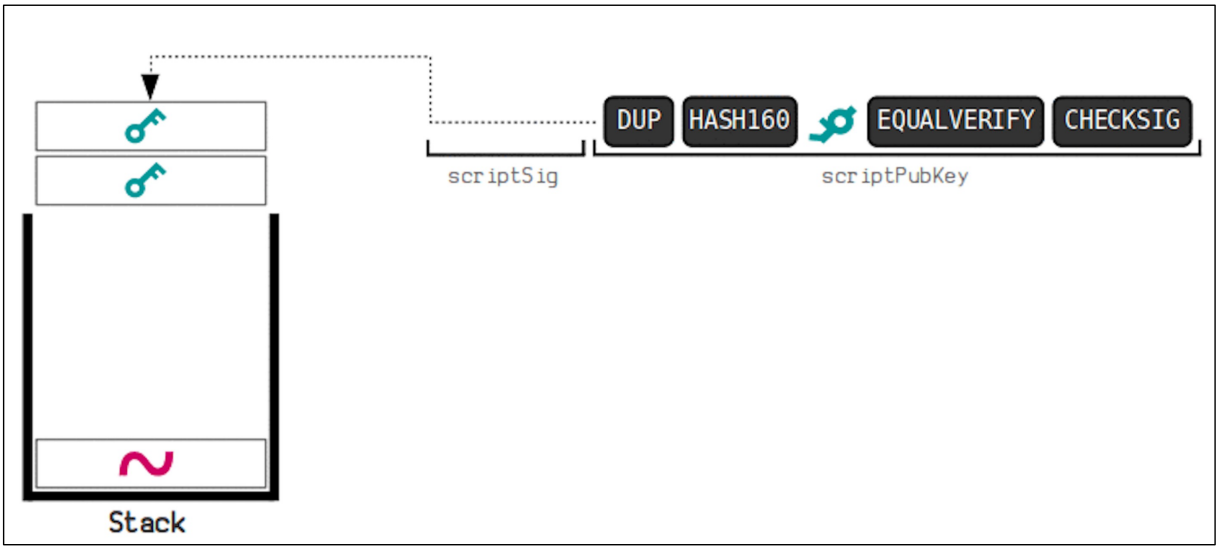
Bitcoin script

- P2PKH



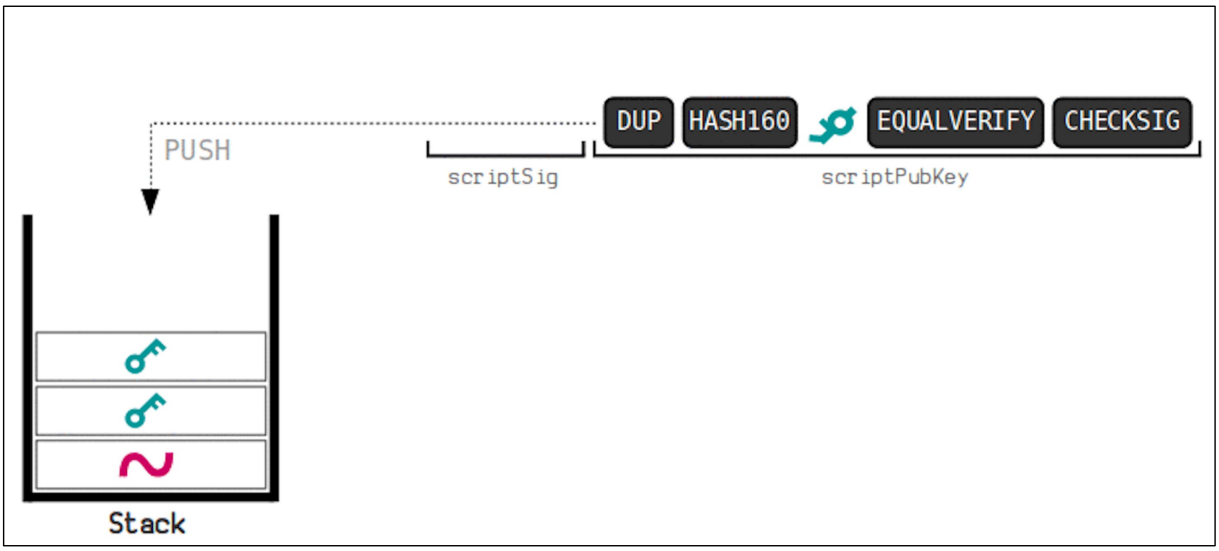
Bitcoin script

- P2PKH



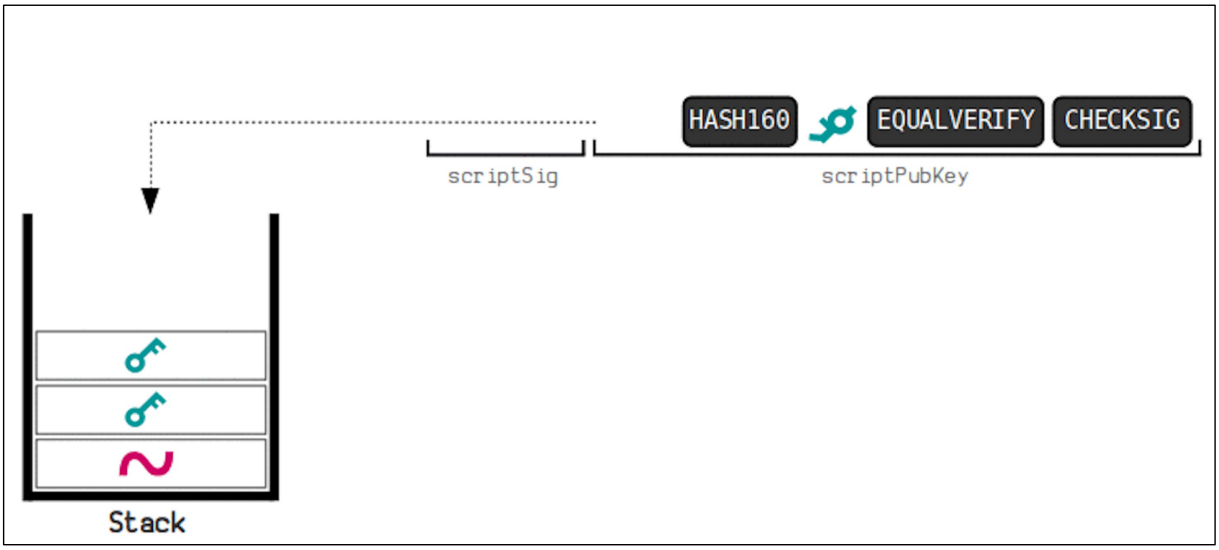
Bitcoin script

- P2PKH



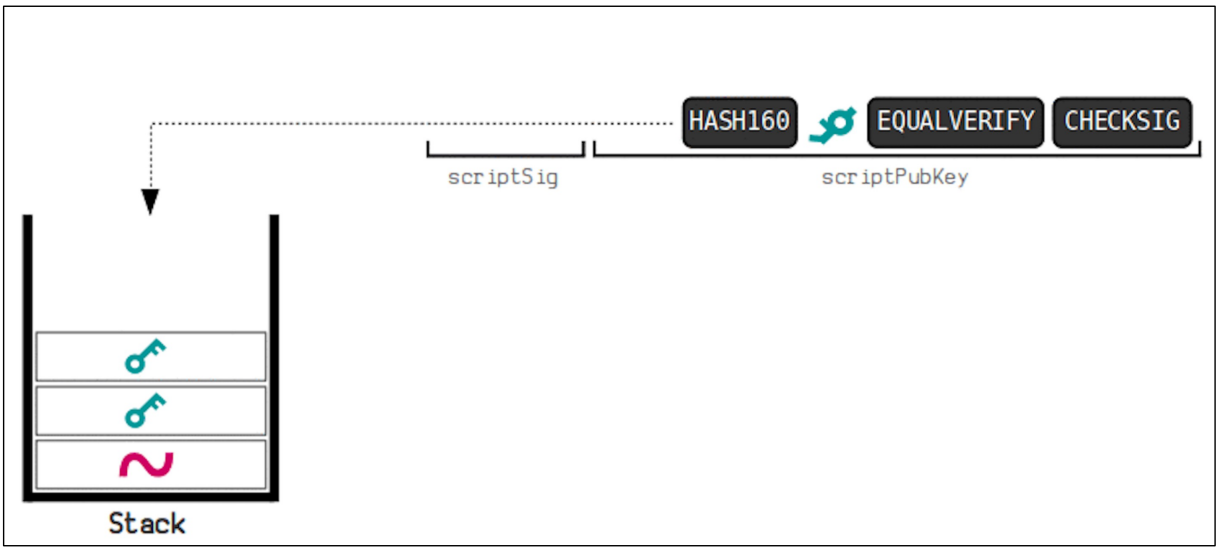
Bitcoin script

- P2PKH



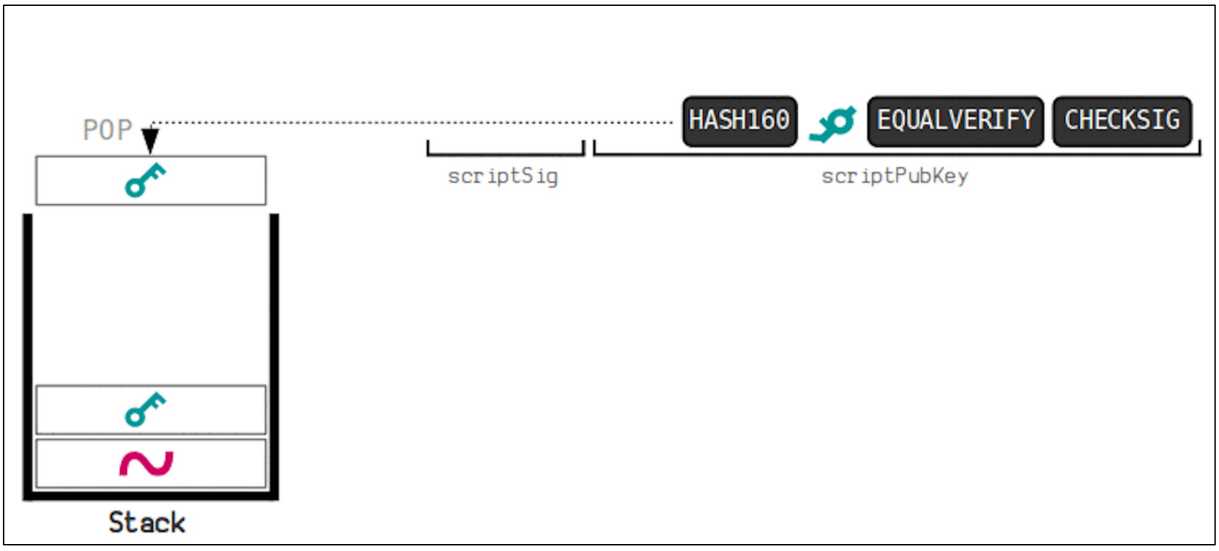
Bitcoin script

- P2PKH



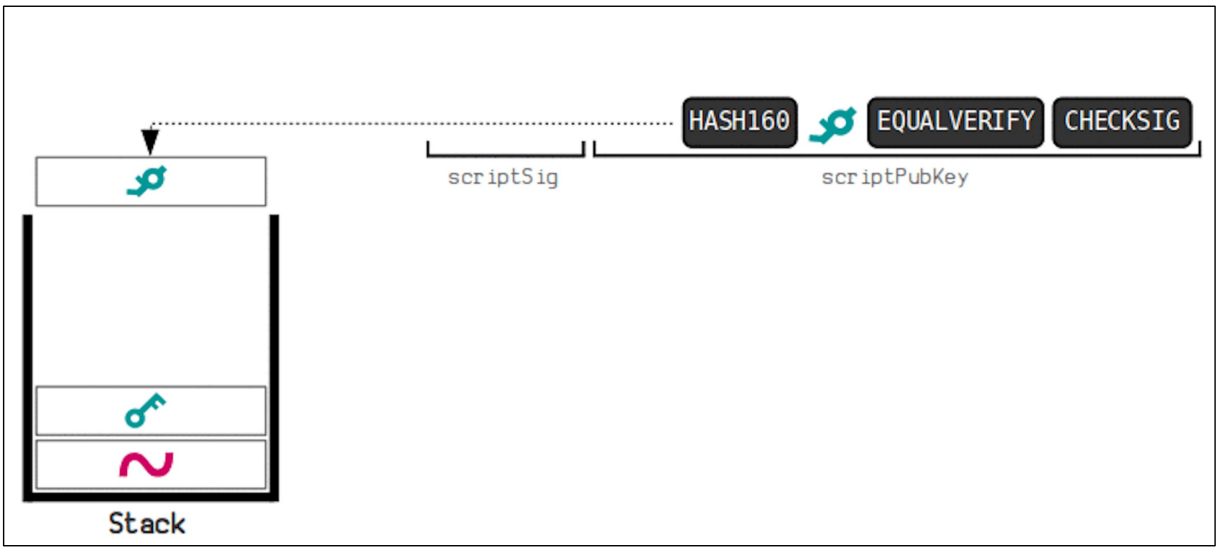
Bitcoin script

- P2PKH



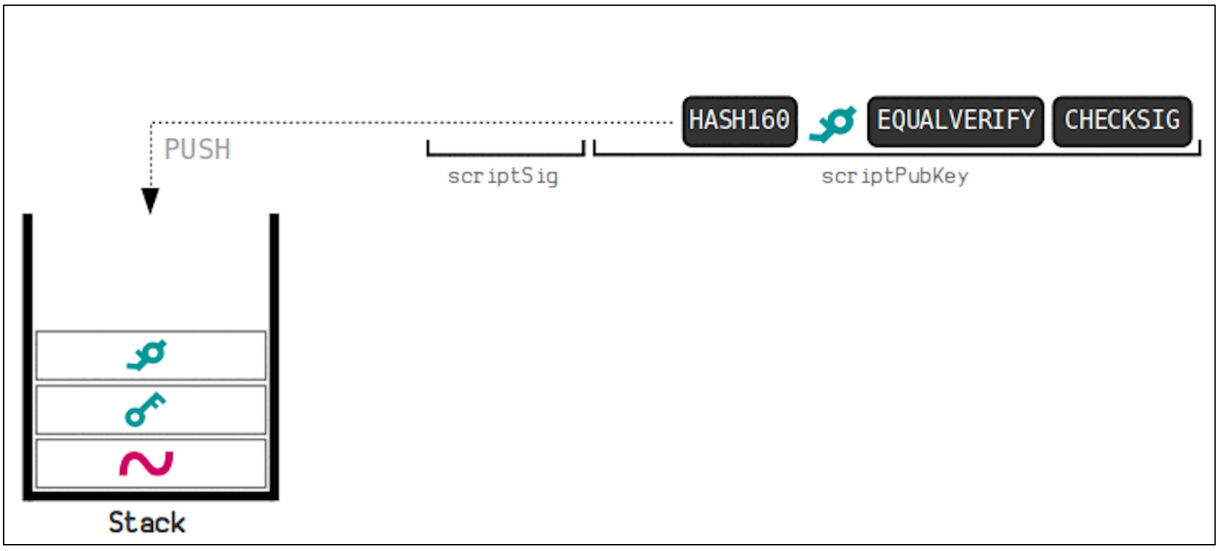
Bitcoin script

- P2PKH



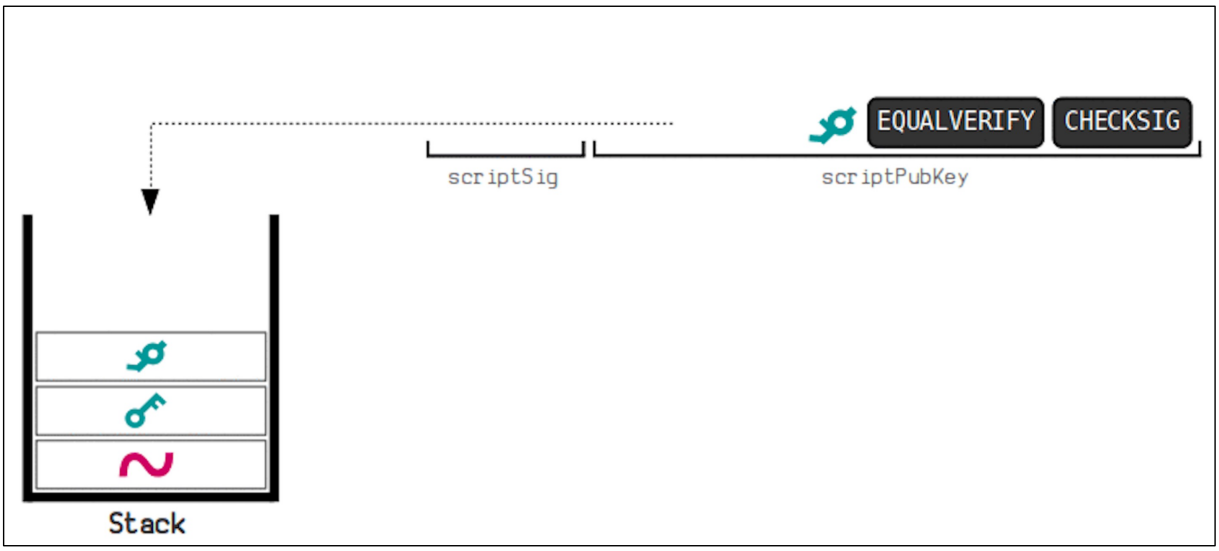
Bitcoin script

- P2PKH



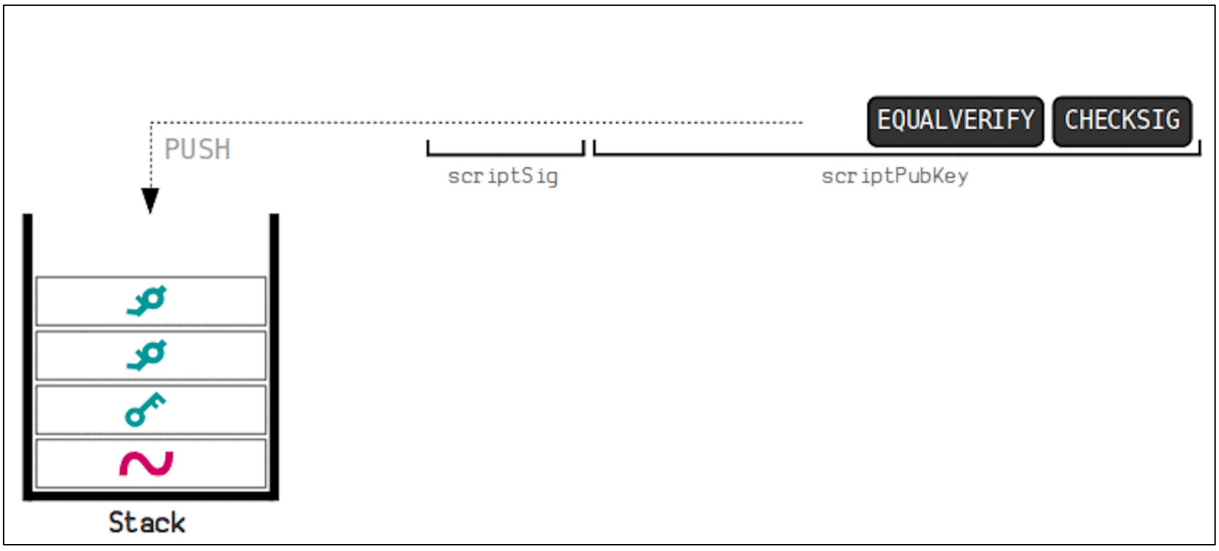
Bitcoin script

- P2PKH



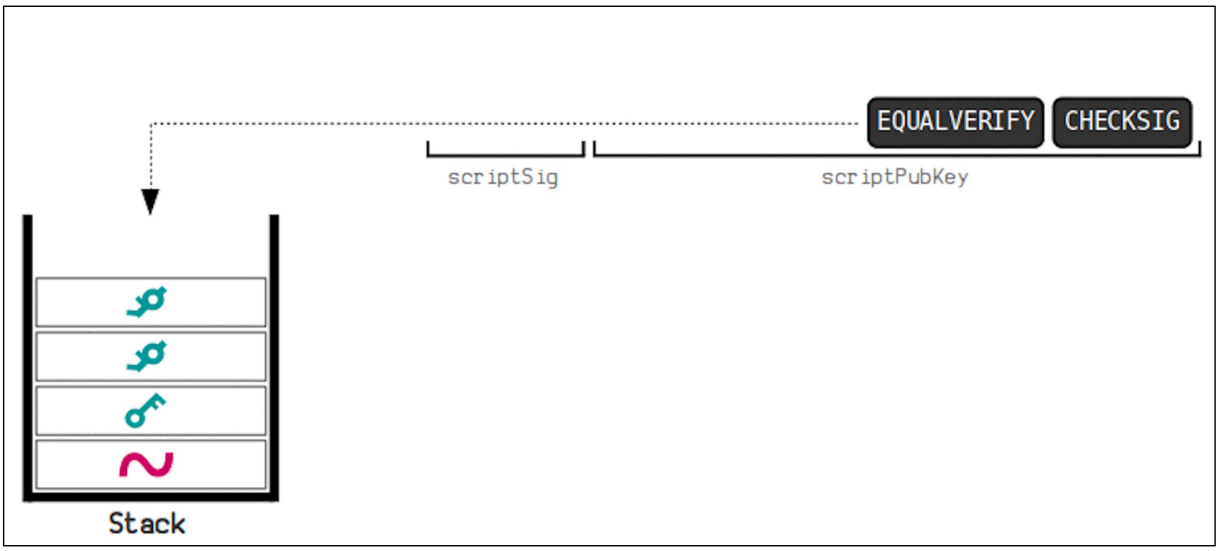
Bitcoin script

- P2PKH



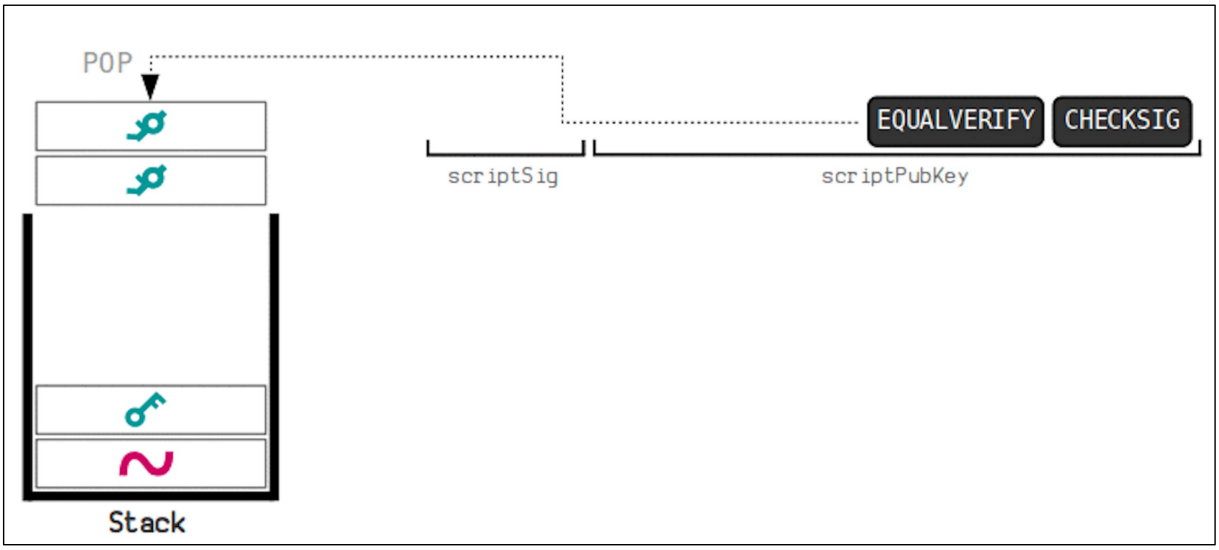
Bitcoin script

- P2PKH



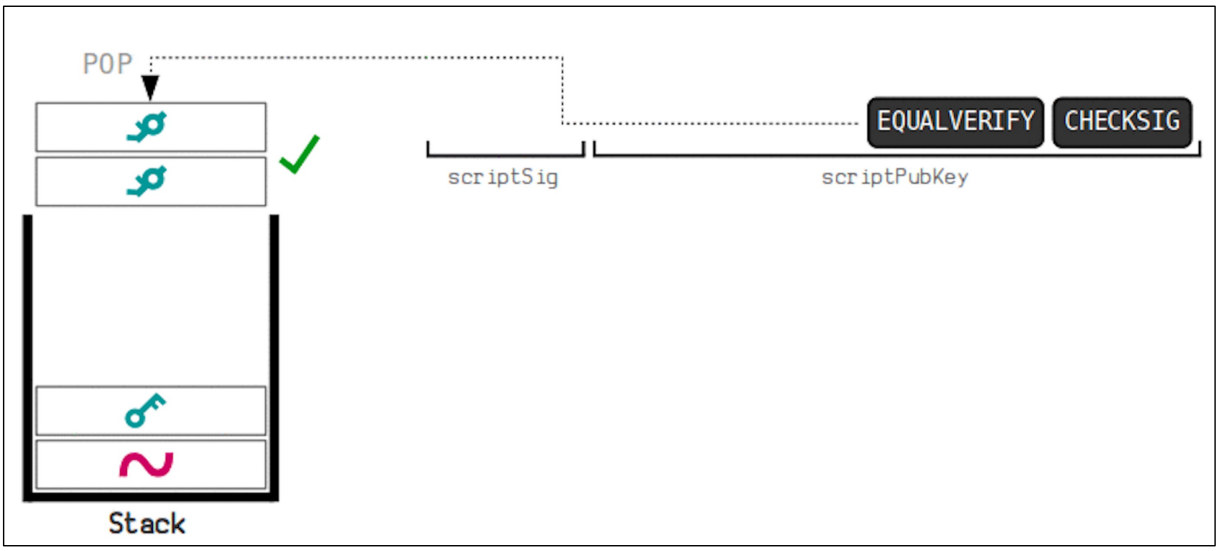
Bitcoin script

- P2PKH



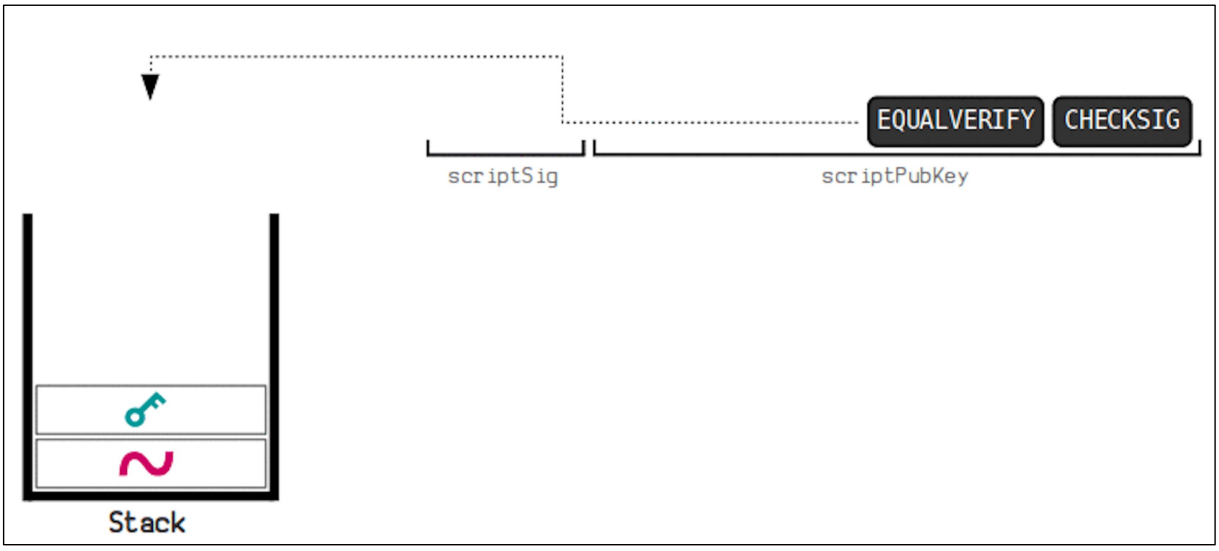
Bitcoin script

- P2PKH



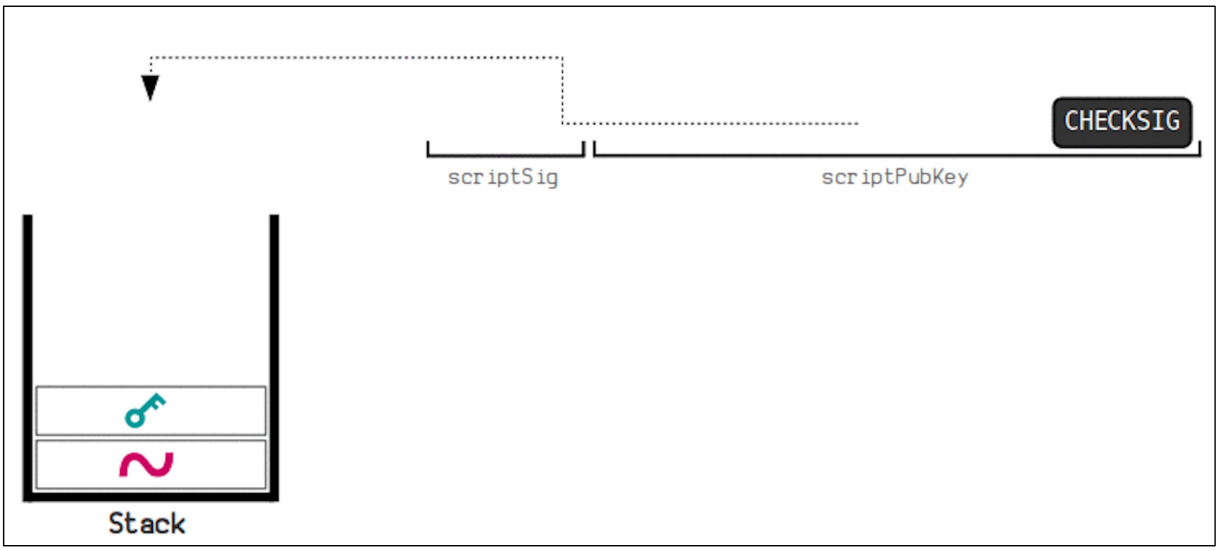
Bitcoin script

- P2PKH



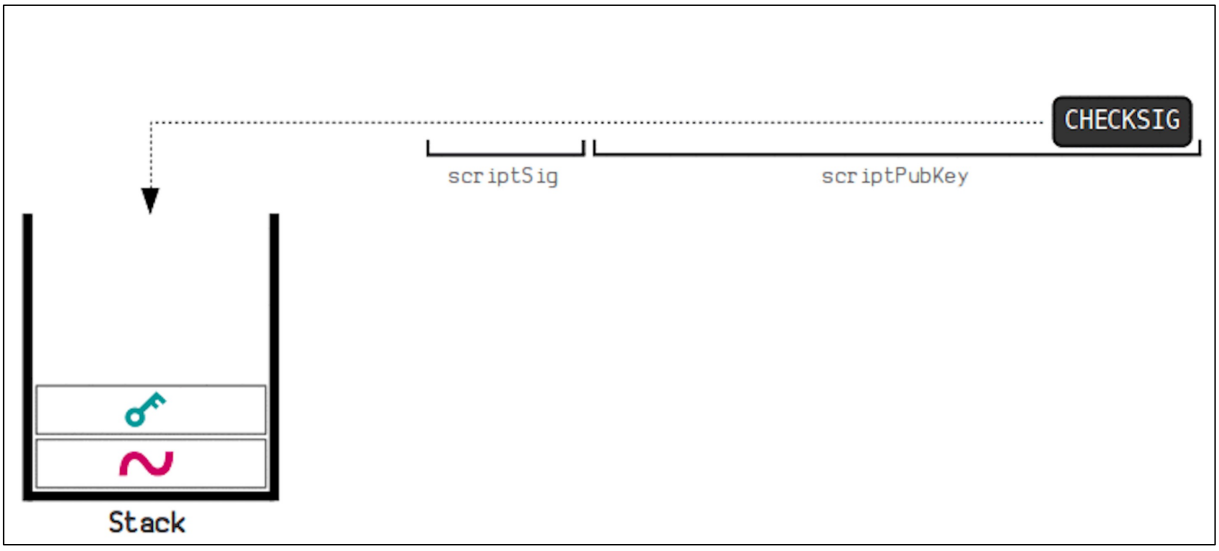
Bitcoin script

- P2PKH



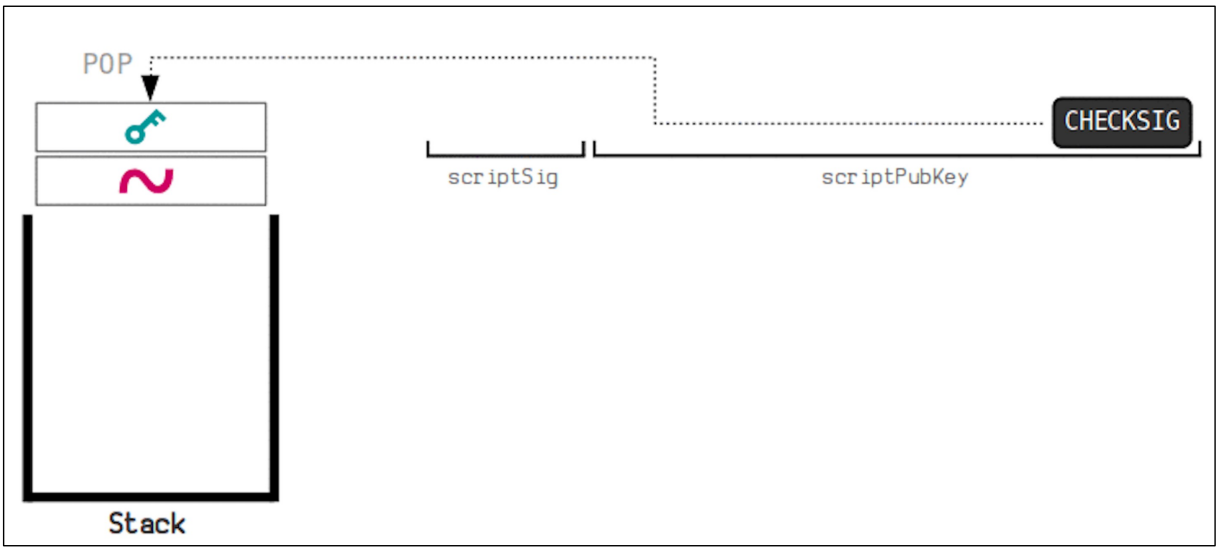
Bitcoin script

- P2PKH



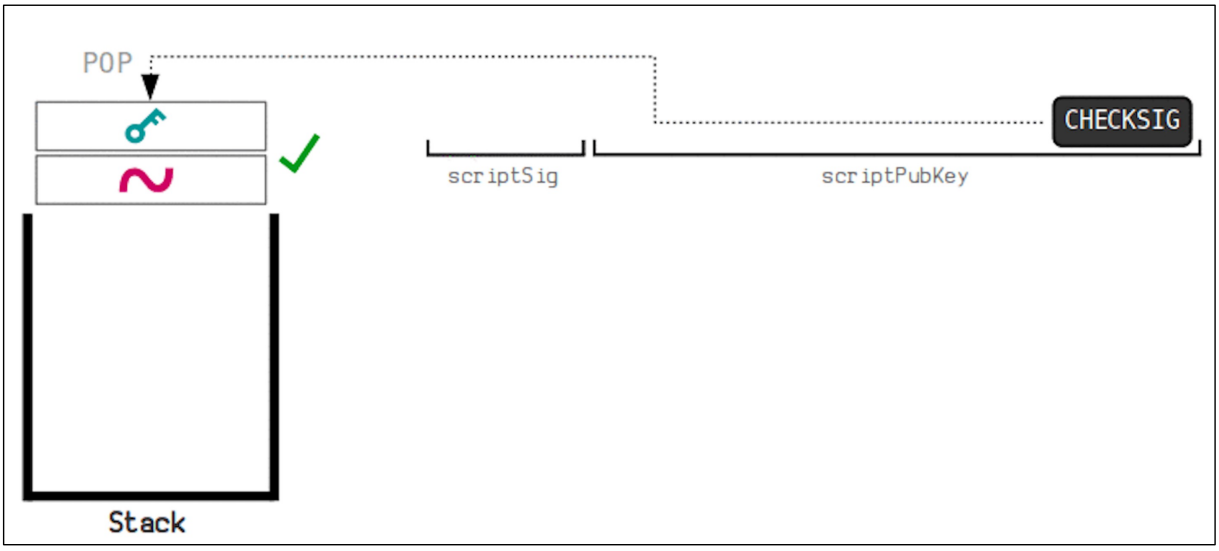
Bitcoin script

- P2PKH



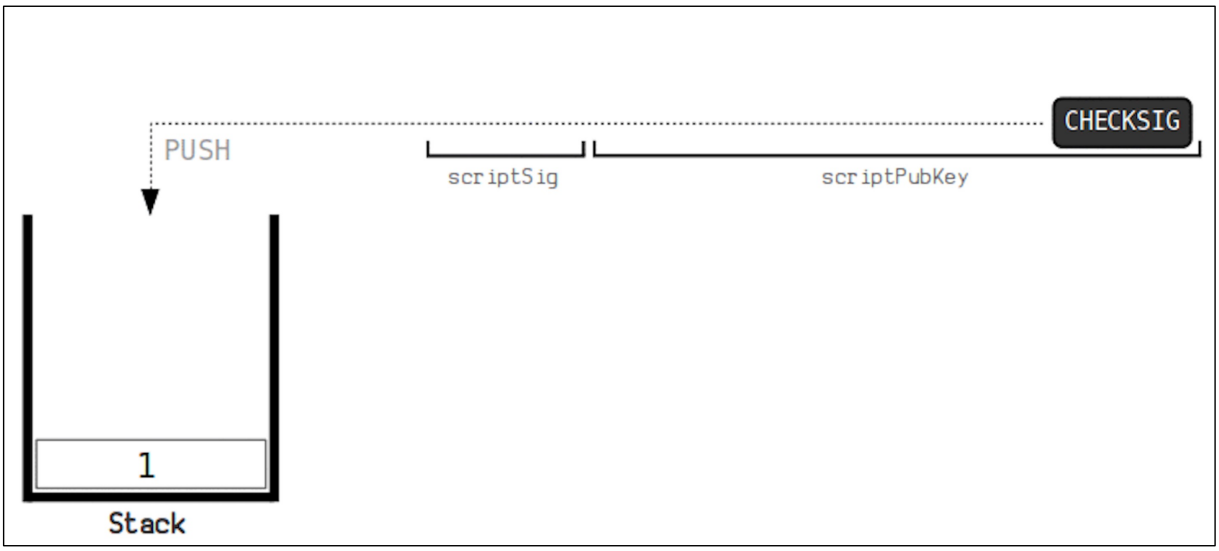
Bitcoin script

- P2PKH



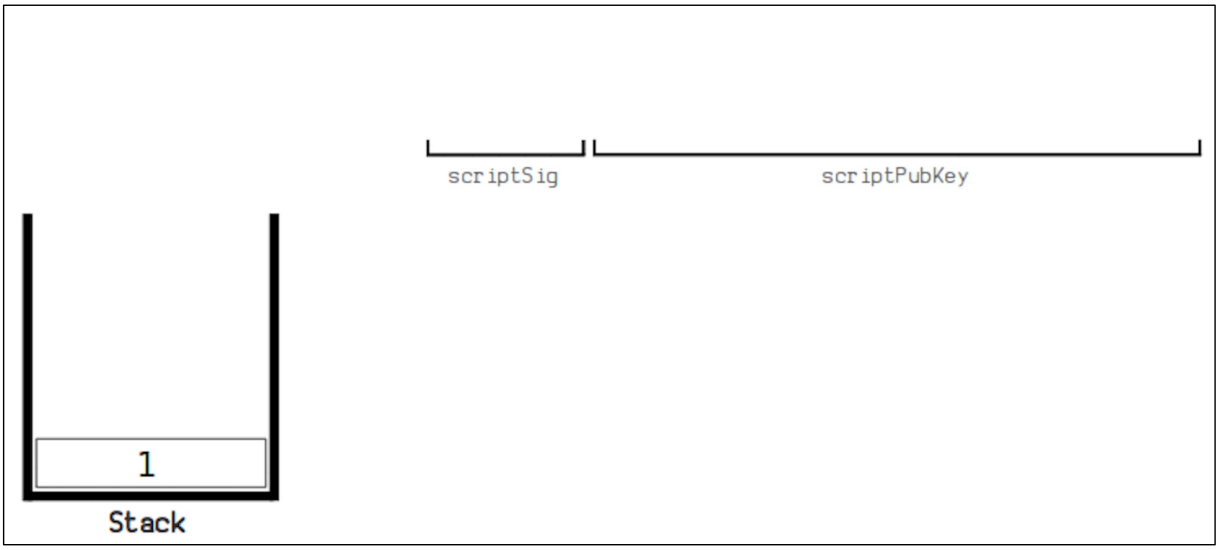
Bitcoin script

- P2PKH



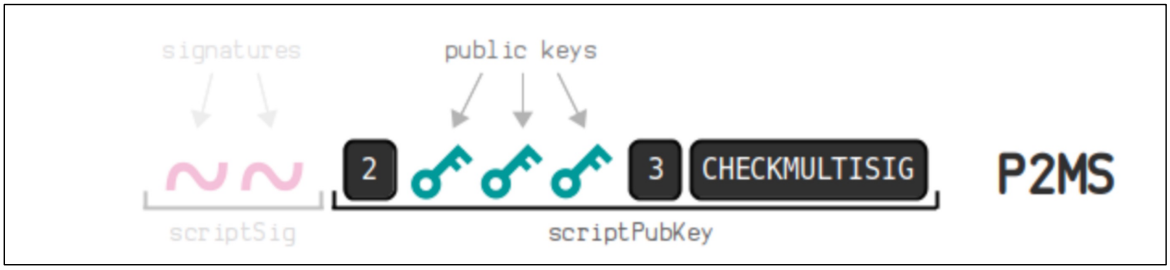
Bitcoin script

- P2PKH



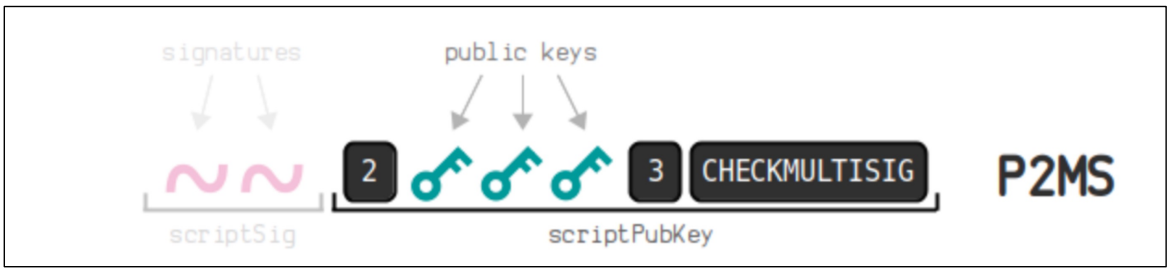
Bitcoin script

- P2MS locks an output to multiple public keys
  - Unlocking requires signatures of some (or all) of those public keys
    - *<m>-of-<n>* signature
  - Larger size locking (with all public keys)
    - So, limited to 3 public keys
      - Avoid storing more data in UTXO
      - Tx cost



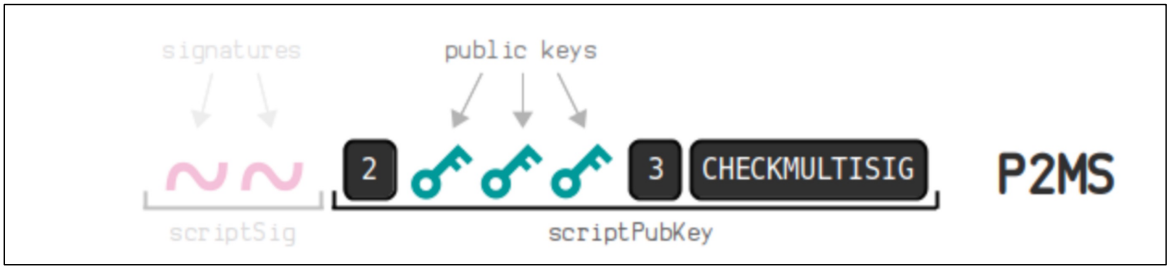
Bitcoin script

- P2MS locks an output to multiple public keys
  - Recipient must send raw locking script to sender
    - As there is no P2MS-specific address format
  - *OP_CHECKMULTISIG* used in P2MS is buggy
    - Pops an extra item from the stack (due to *off-by-one error*)
    - Bug still not fixed (cost, maintain compatibility)
    - Put an extra dummy data (*OP_0*) in scriptSig



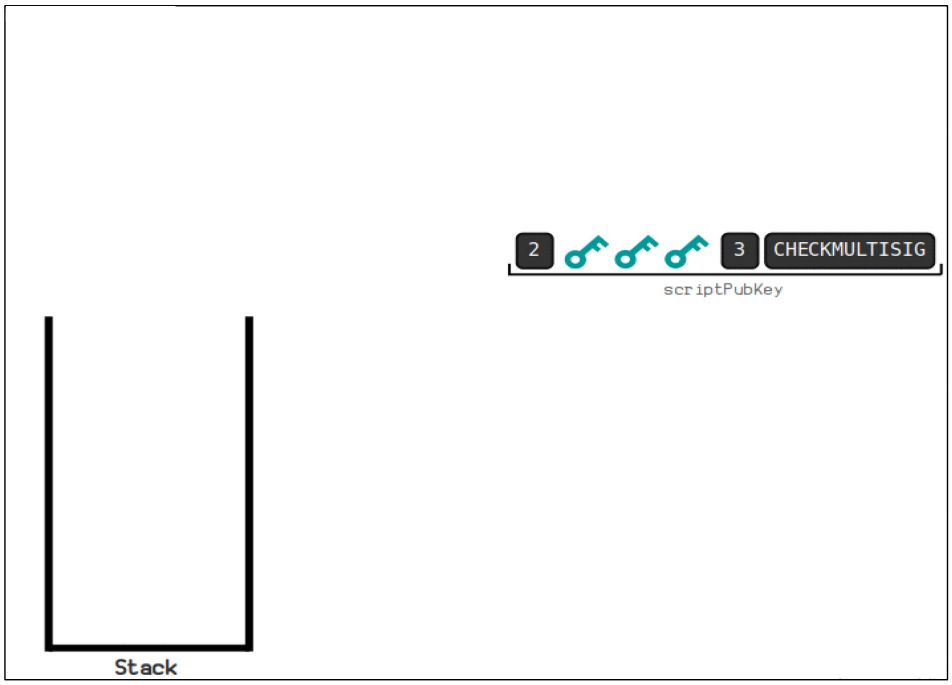
Bitcoin script

- P2MS locks an output to multiple public keys
  - One needs to provide the corresponding signatures in the same order as the order of the public keys in the ScriptPubKey
  - A raw P2MS locking script does not have an address



Bitcoin script

- P2MS

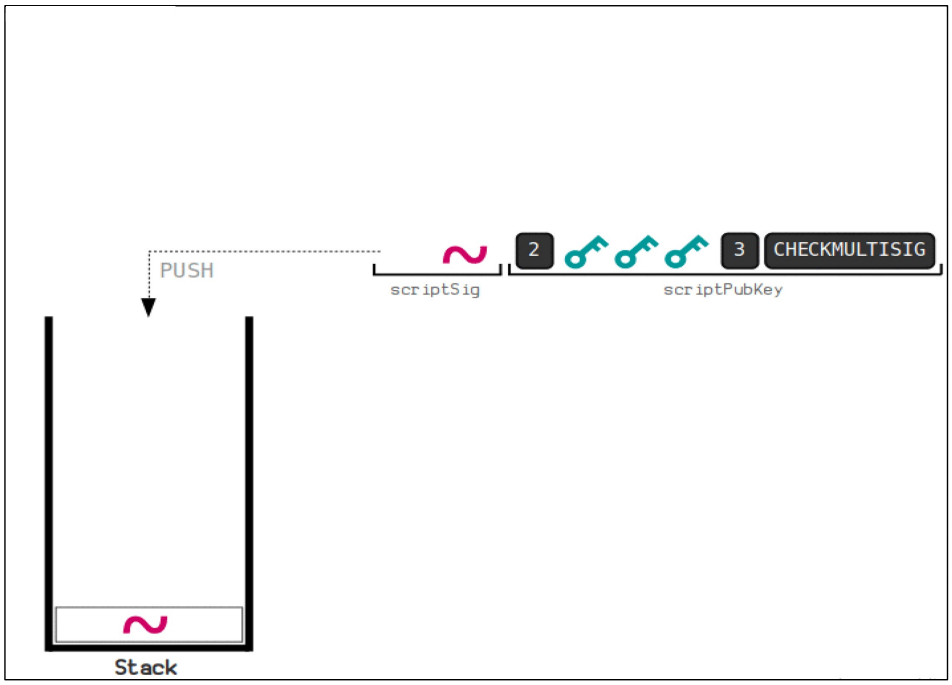


Bitcoin script

- P2MS

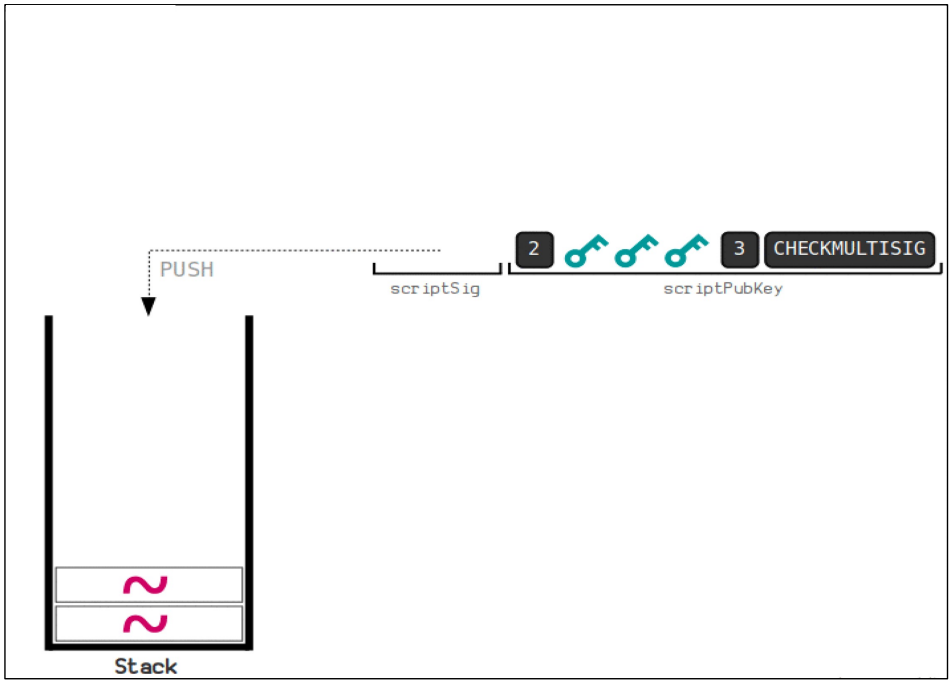
Bitcoin script

- P2MS



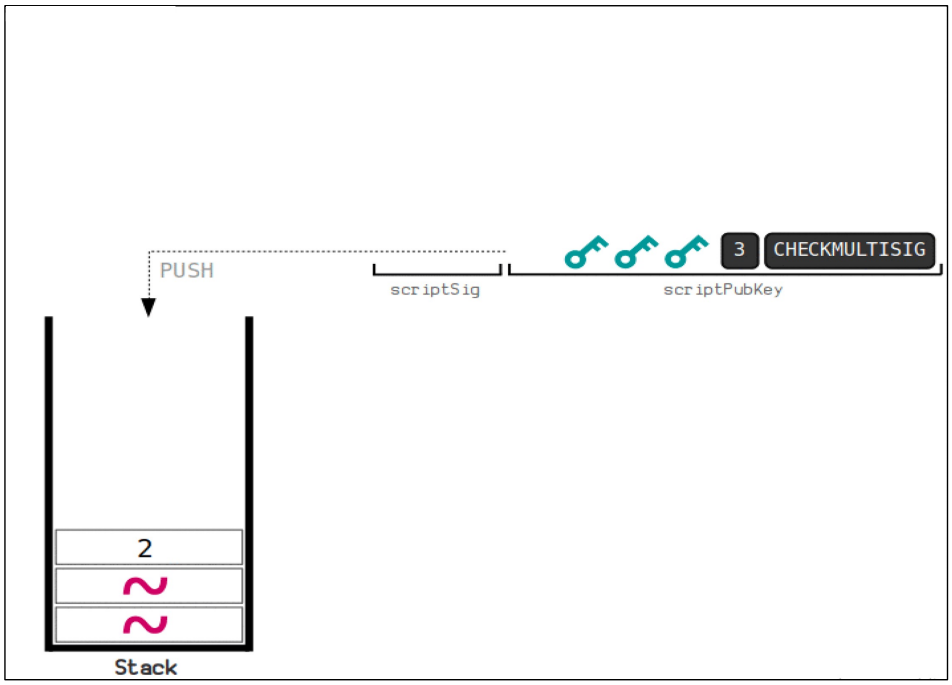
Bitcoin script

- P2MS



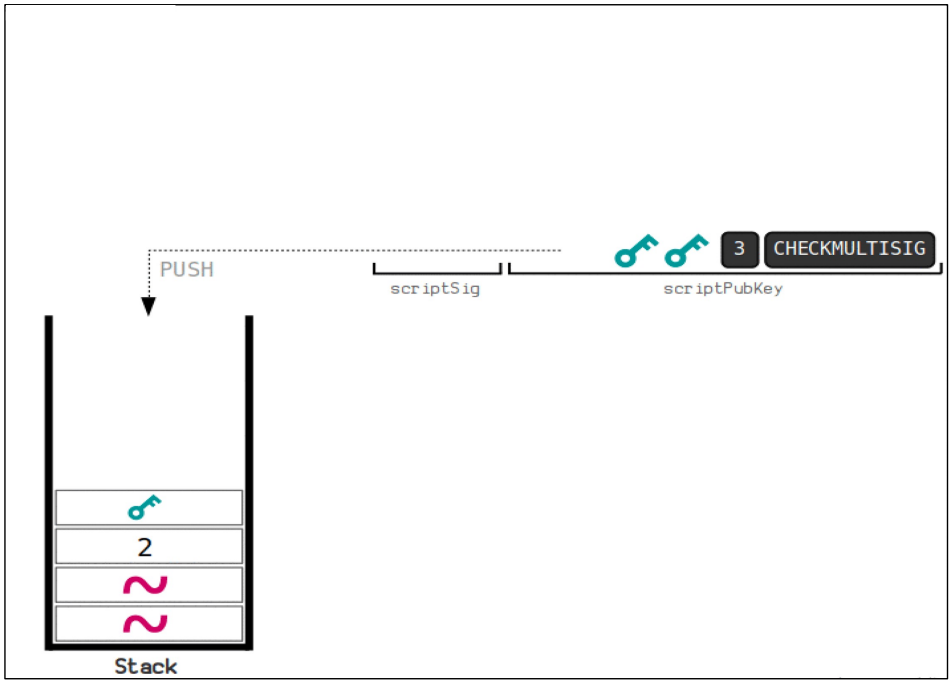
Bitcoin script

- P2MS



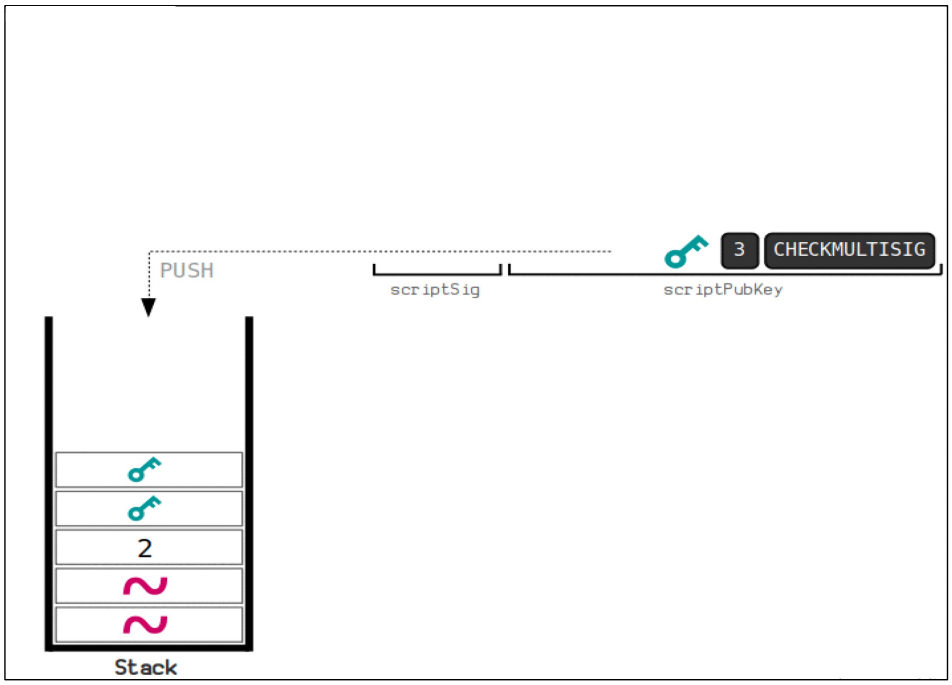
Bitcoin script

- P2MS



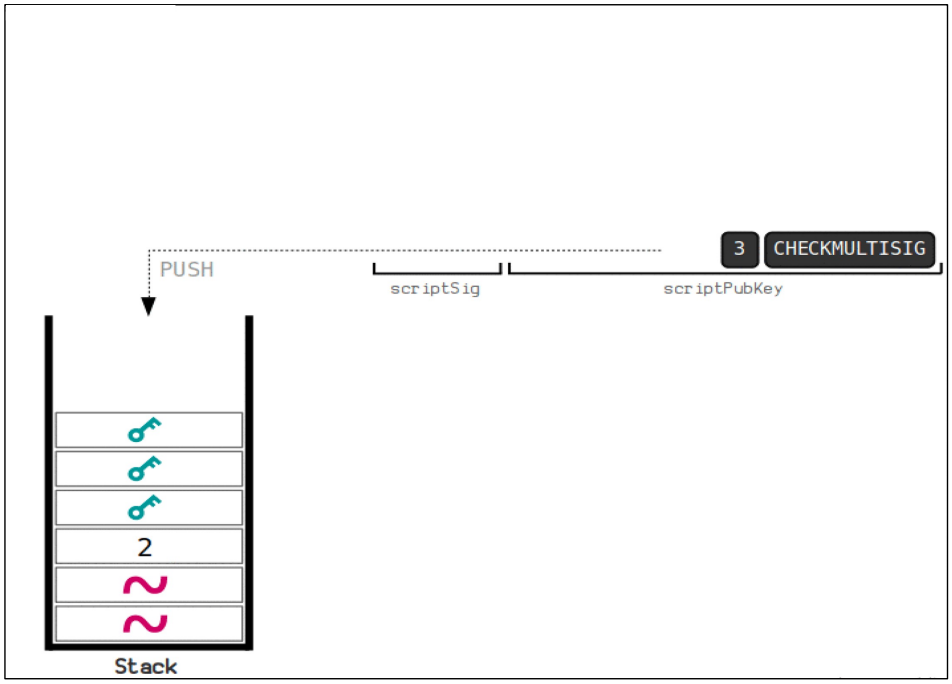
Bitcoin script

- P2MS



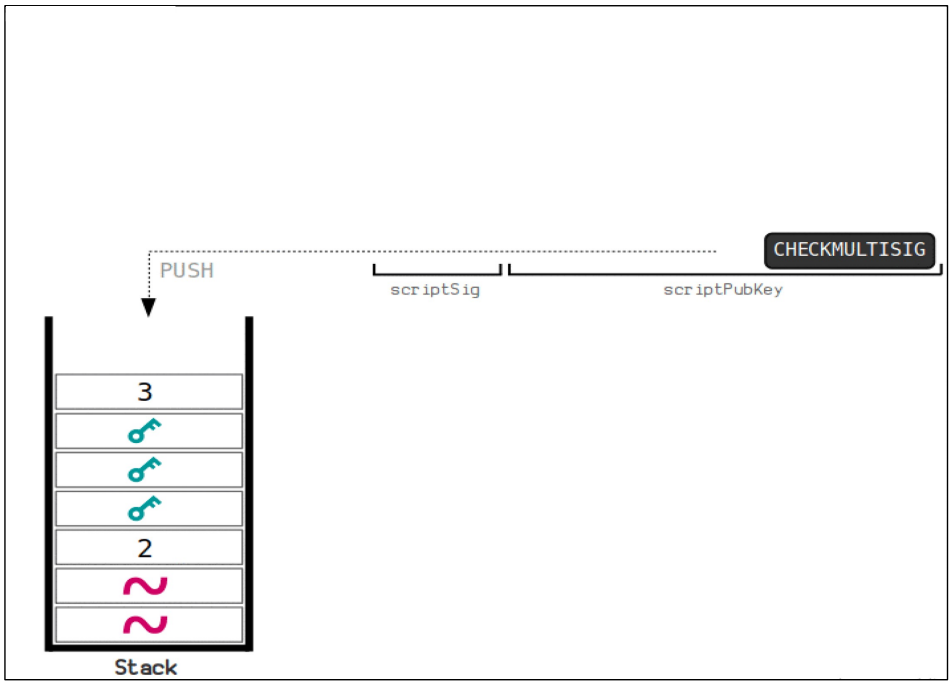
Bitcoin script

- P2MS



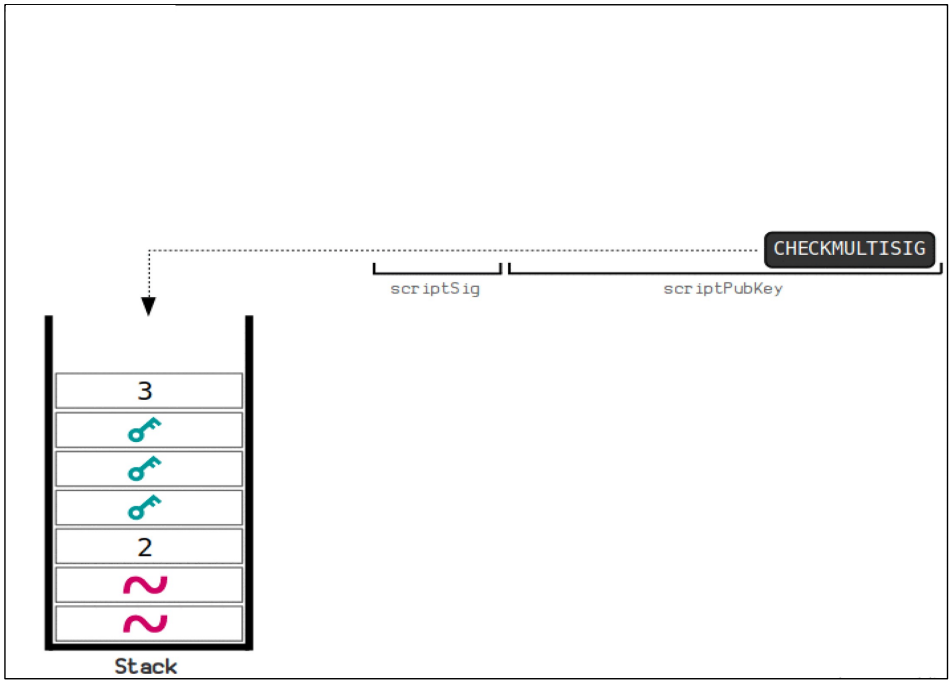
Bitcoin script

- P2MS



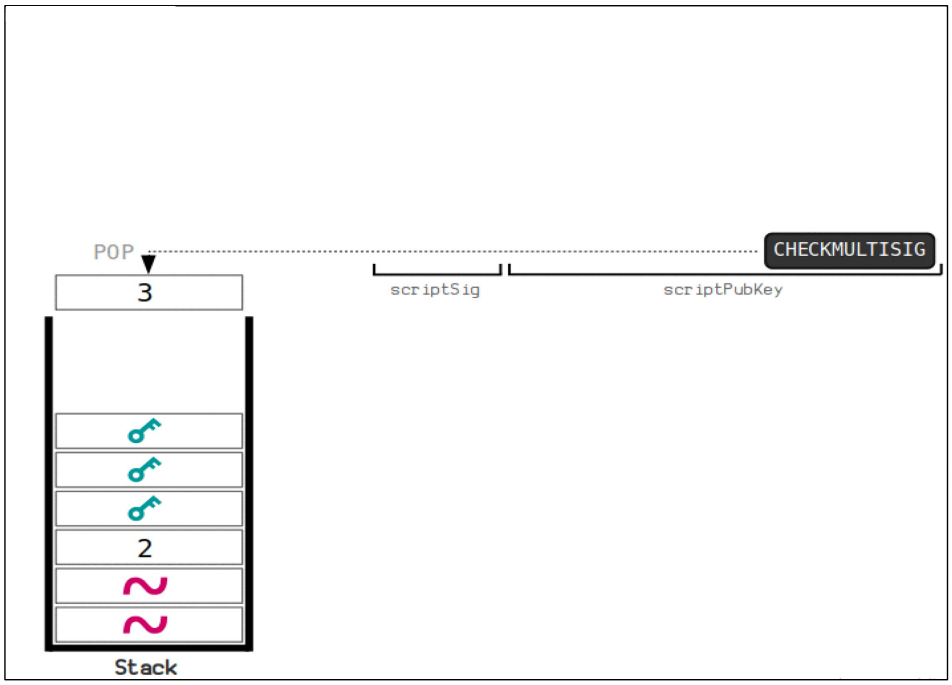
Bitcoin script

- P2MS



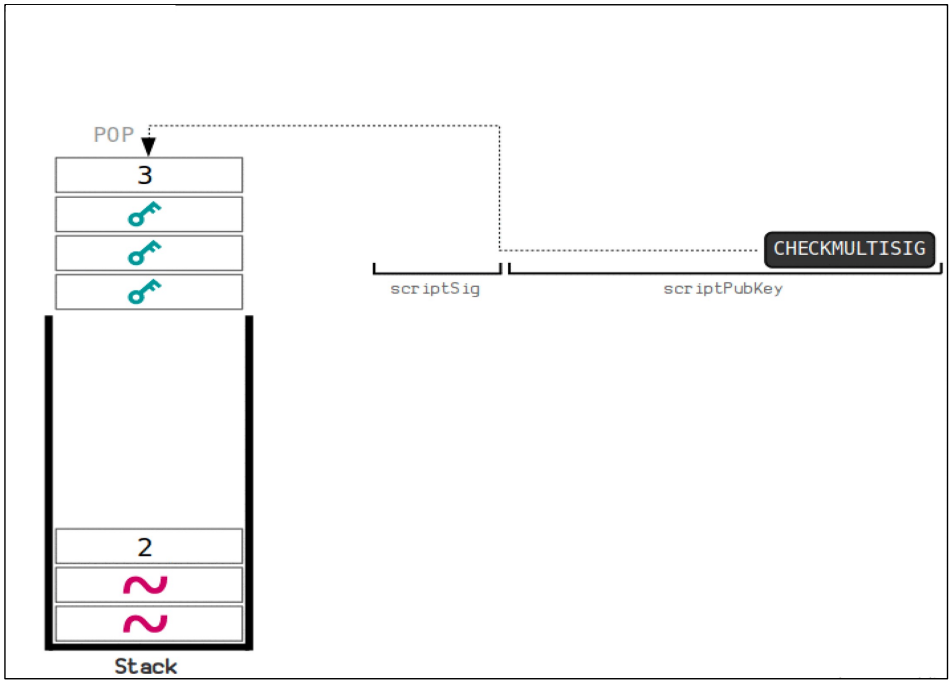
Bitcoin script

- P2MS



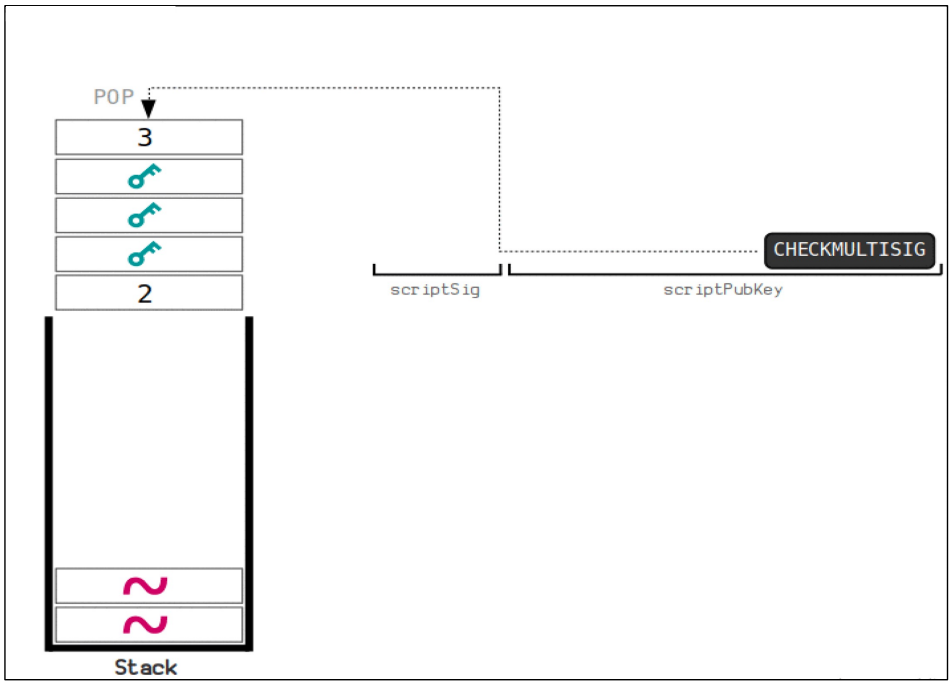
Bitcoin script

- P2MS



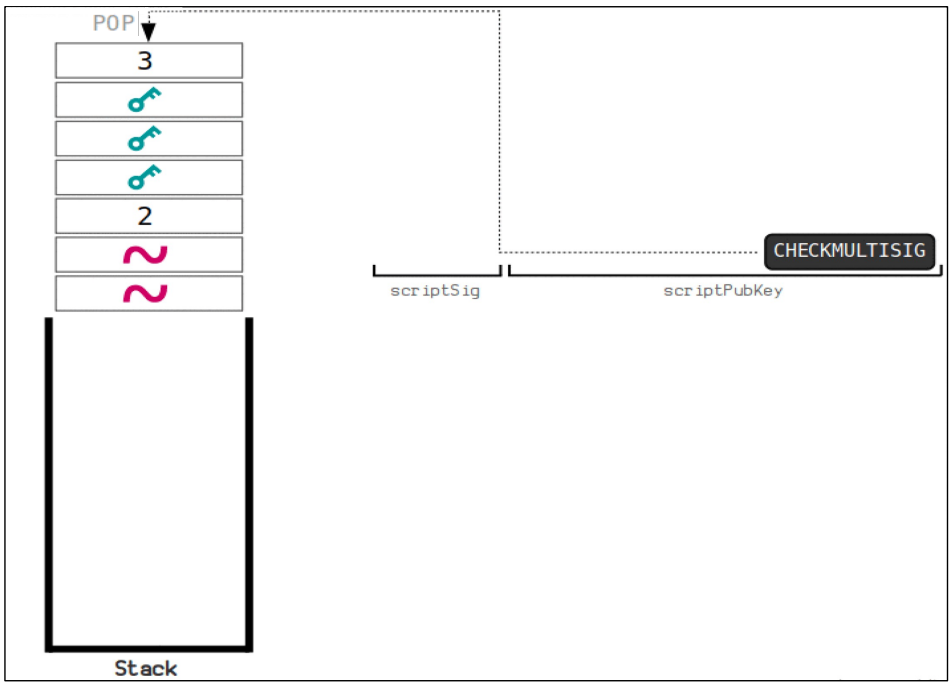
Bitcoin script

- P2MS



Bitcoin script

- P2MS

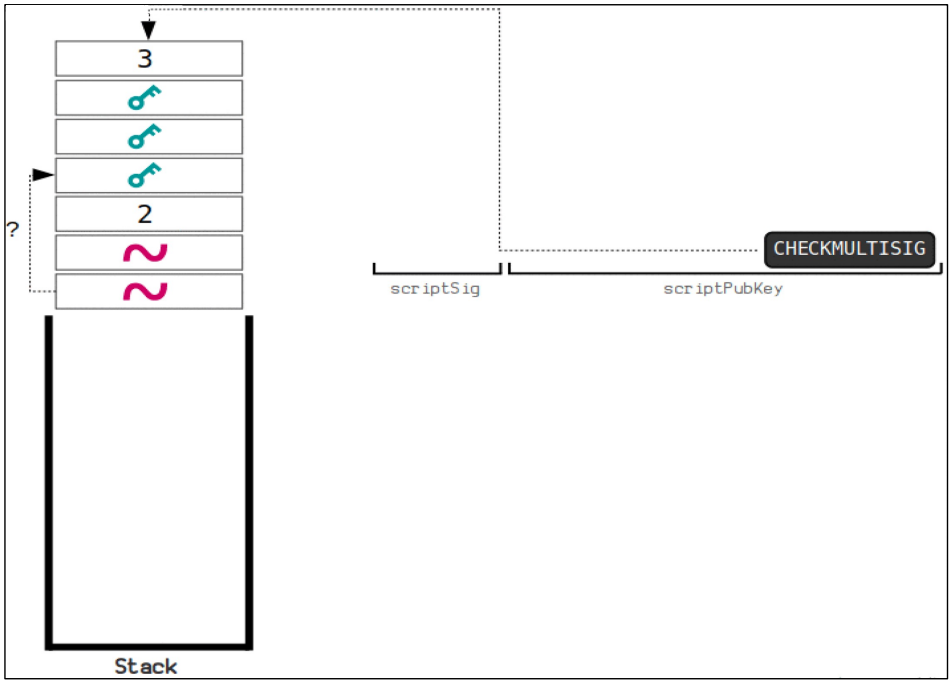


Bitcoin script

- P2MS

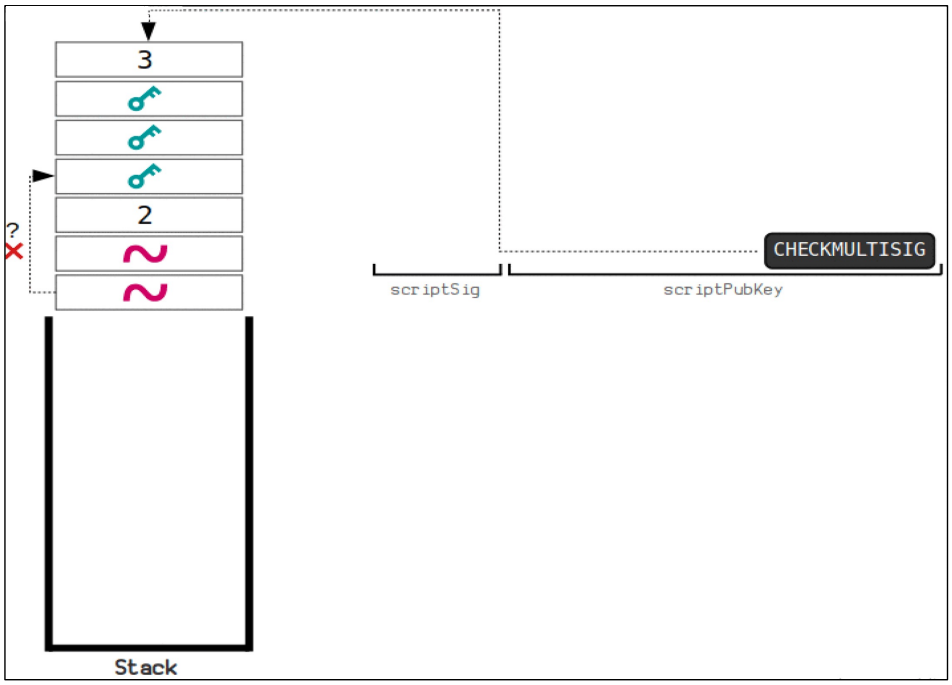
Bitcoin script

- P2MS



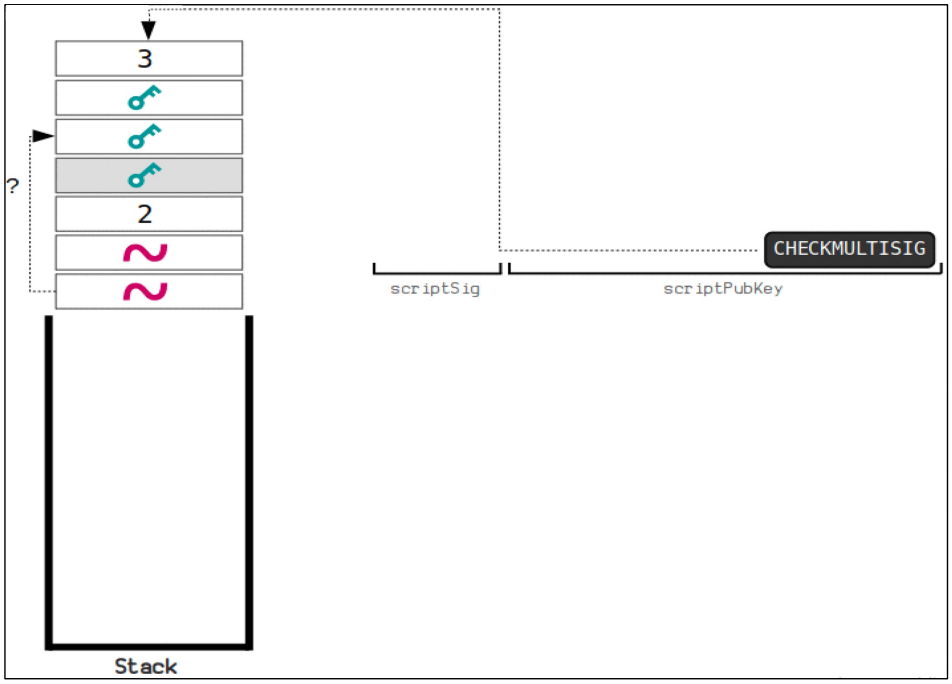
Bitcoin script

- P2MS



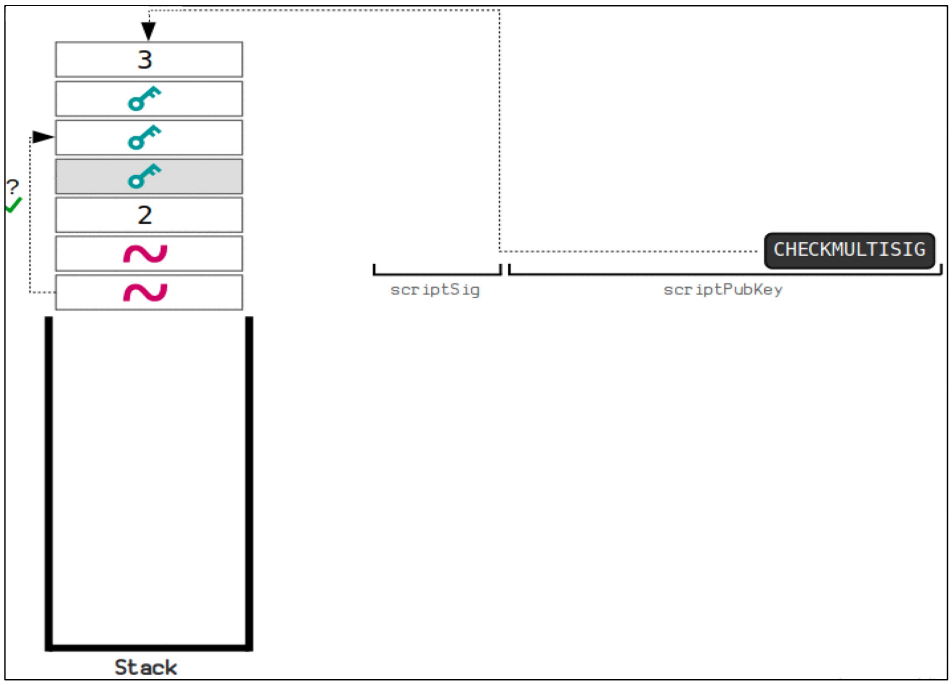
Bitcoin script

- P2MS



Bitcoin script

- P2MS

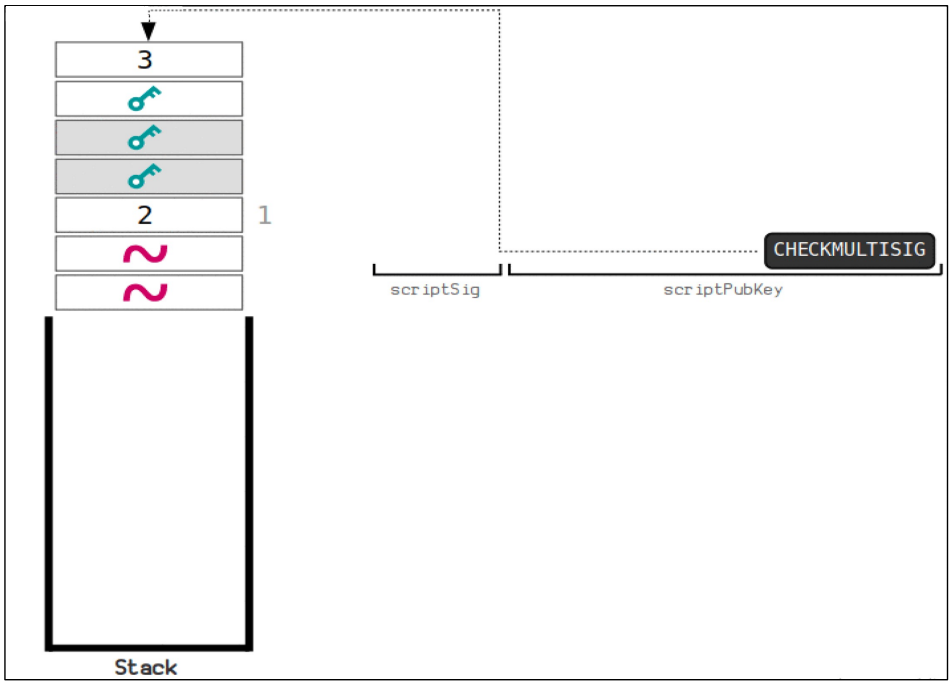


Bitcoin script

- P2MS

Bitcoin script

- P2MS

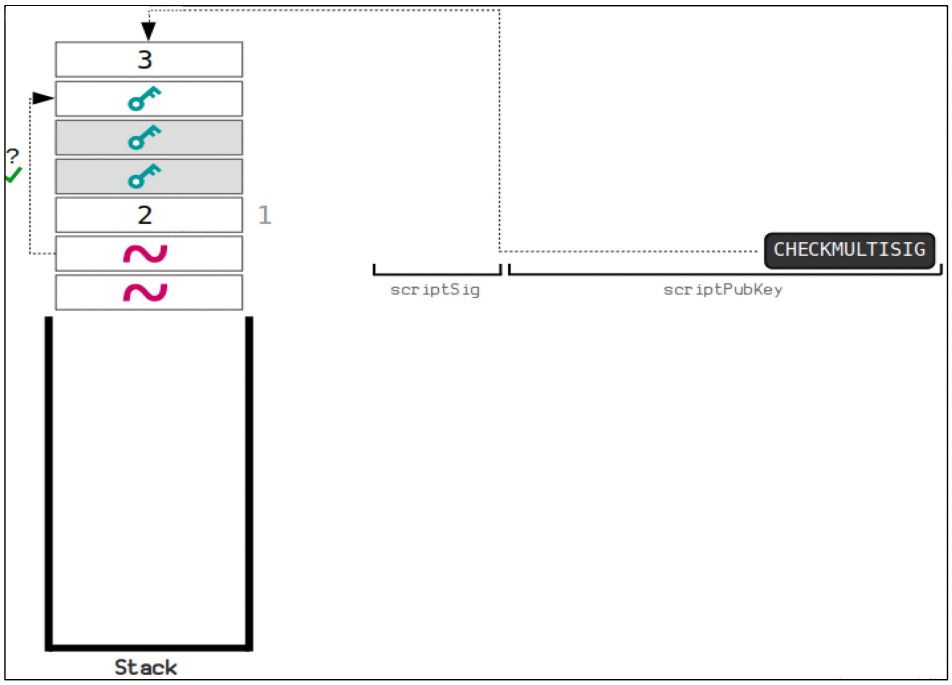


Bitcoin script

- P2MS

Bitcoin script

- P2MS

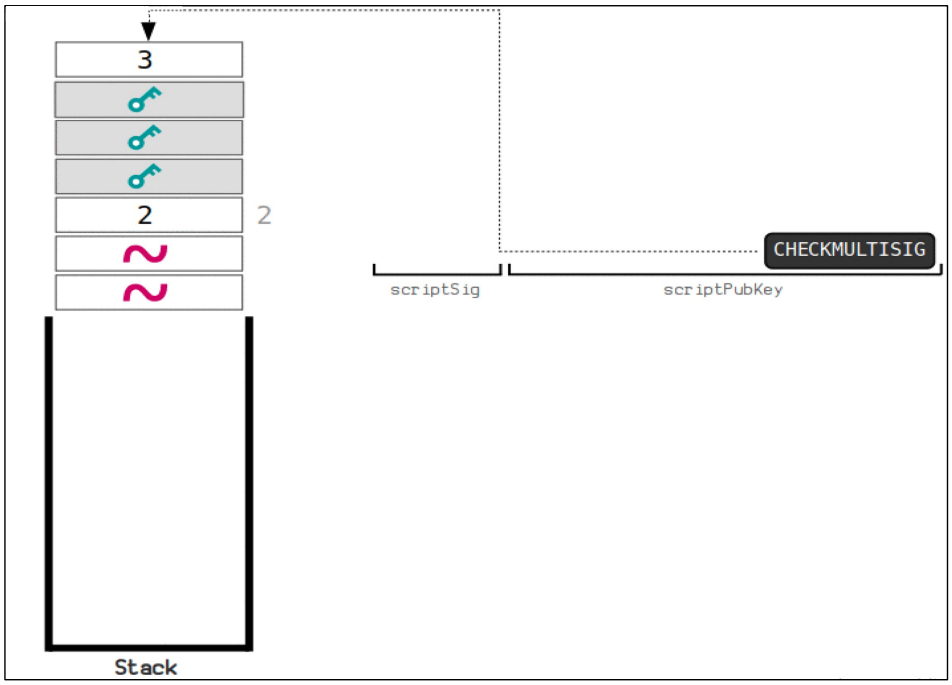


Bitcoin script

- P2MS

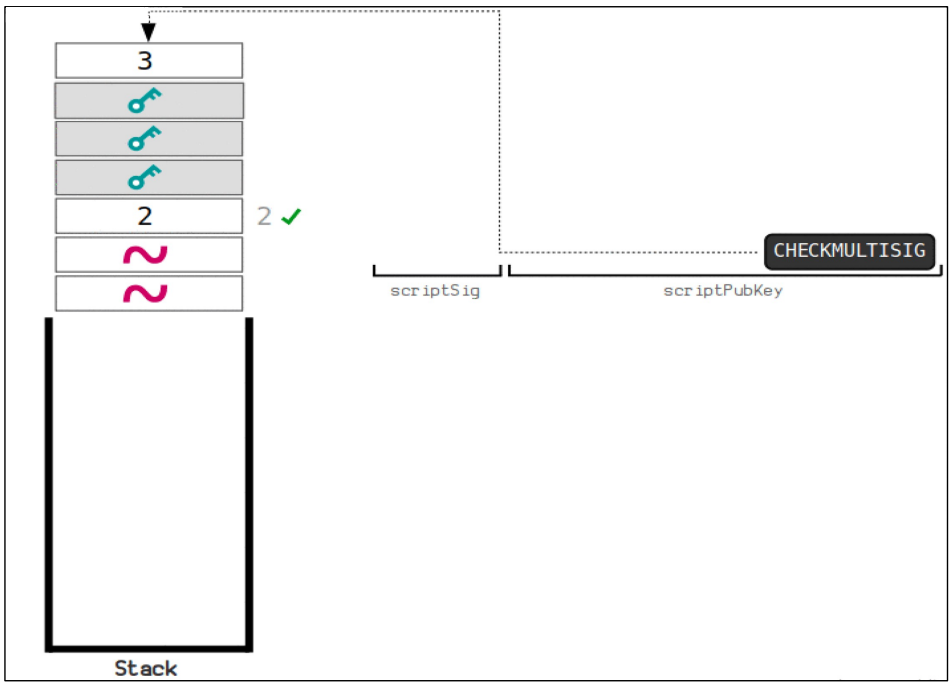
Bitcoin script

- P2MS



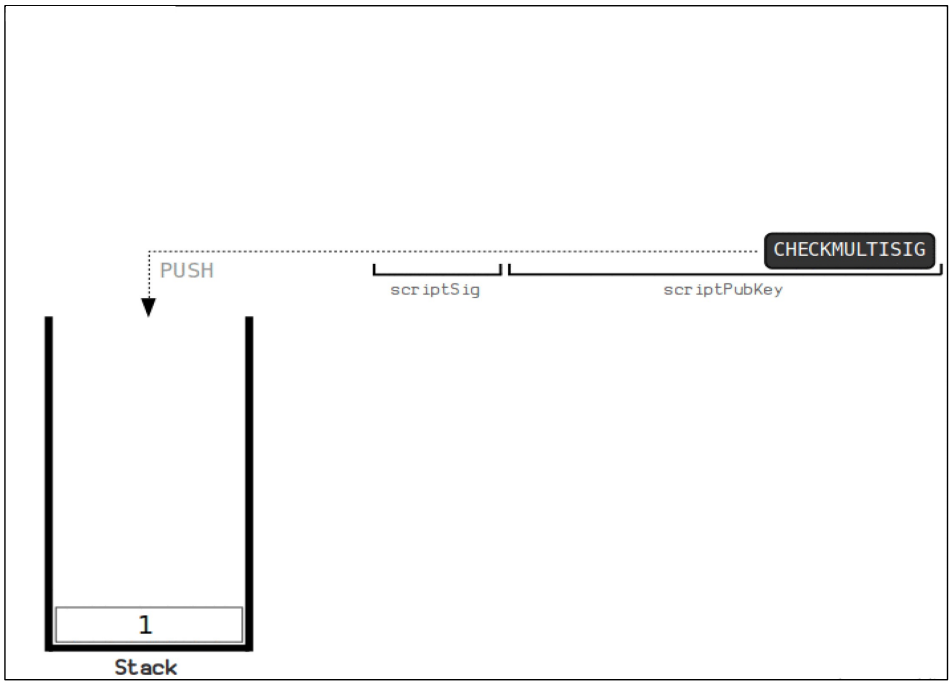
Bitcoin script

- P2MS



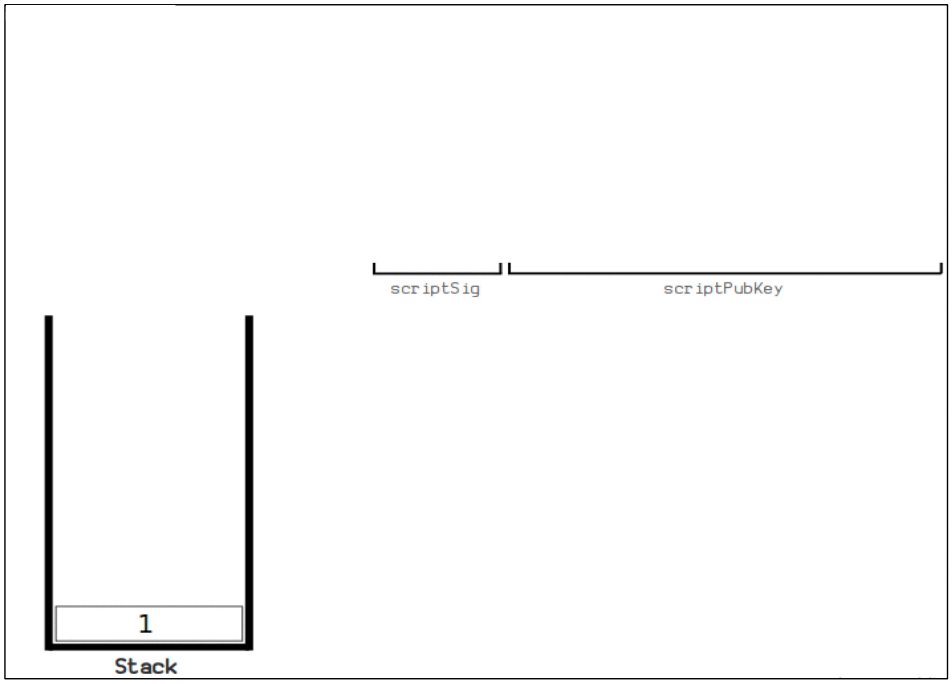
Bitcoin script

- P2MS



Bitcoin script

- P2MS

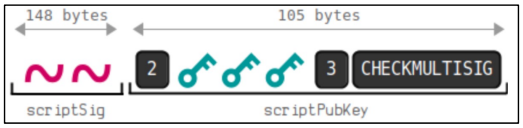


Bitcoin script

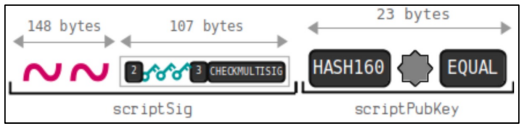
- P2SH locks an output to hash of a script
  - Requires original script at unlocking
  - Creating custom/complex "redeem scripts"
    - Convenient over P2MS

Bitcoin script

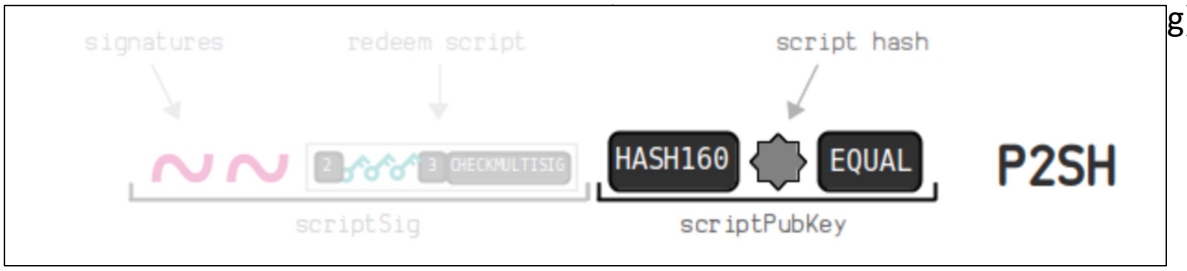
- P2SH locks an output to hash of a script
  - P2SH (vs. P2MS) reduces Tx costs, makes smaller UTXO set?
    - P2MS = 253 bytes



P2SH = 278



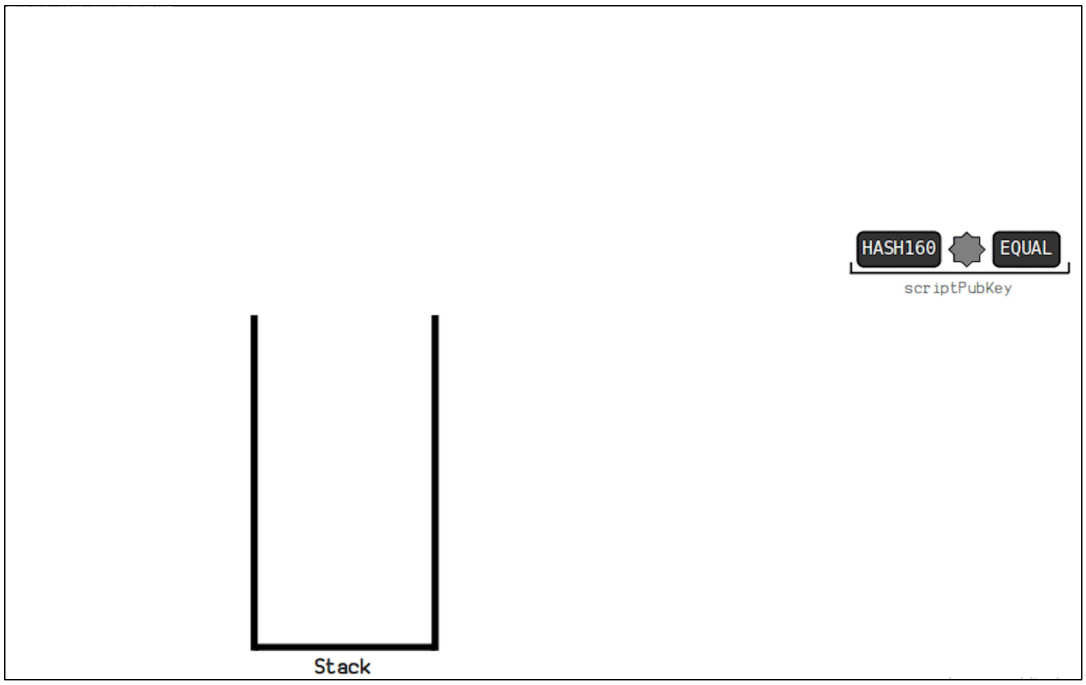
  - P2SH locking pattern has “special” form
    - Executed slightly differently
      - Standard Execution



g)

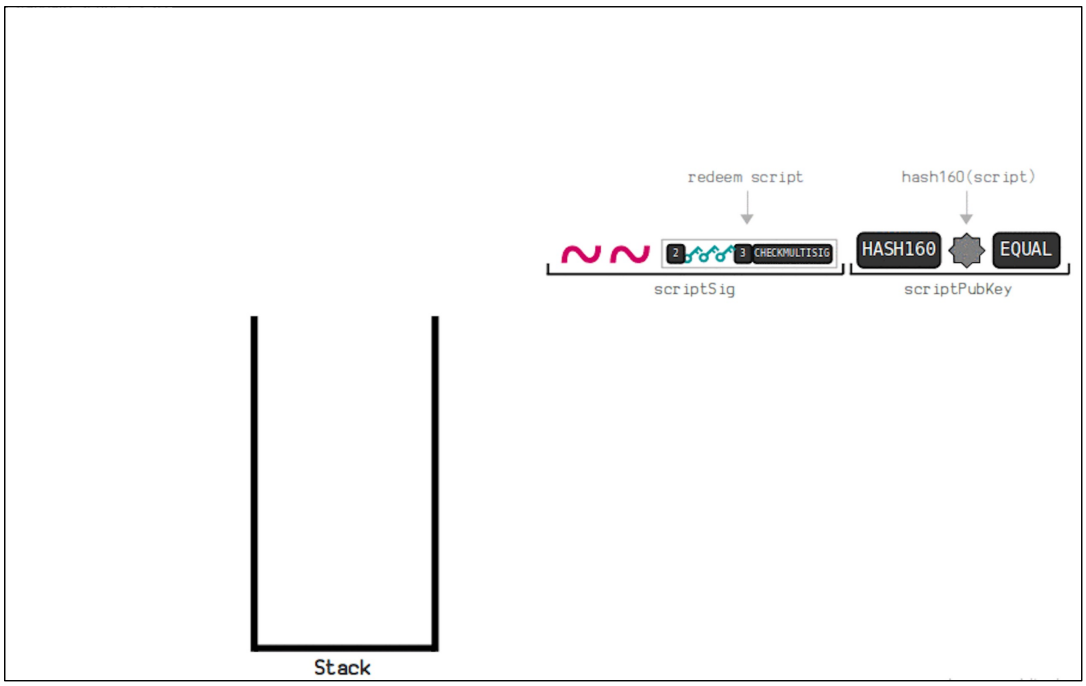
Bitcoin script

- P2SH



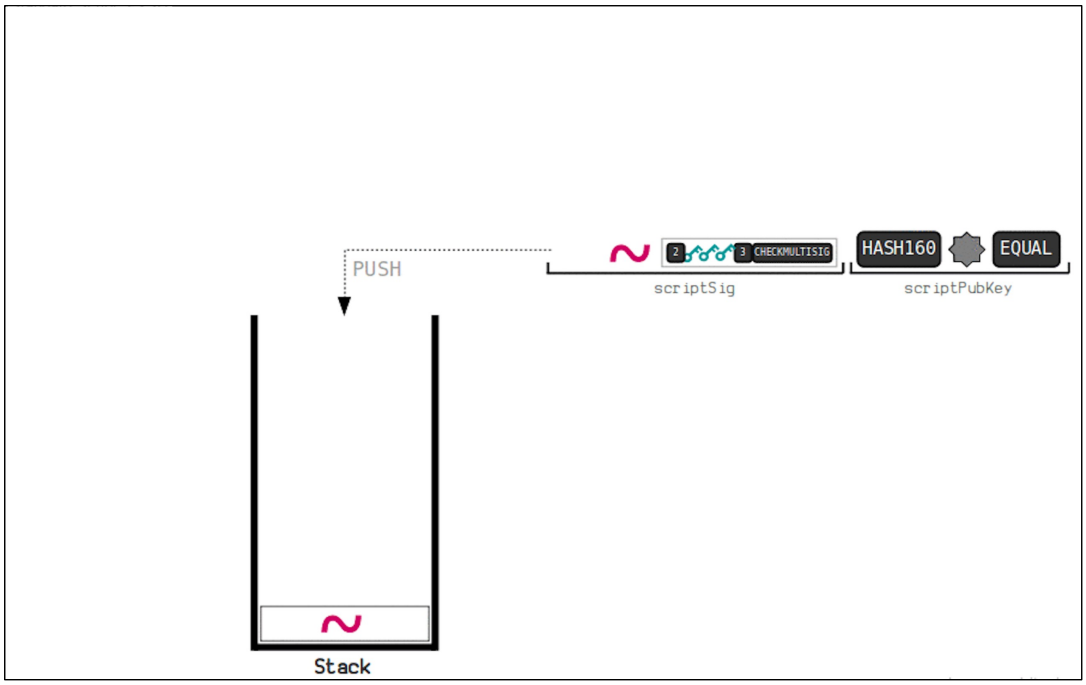
Bitcoin script

- P2SH



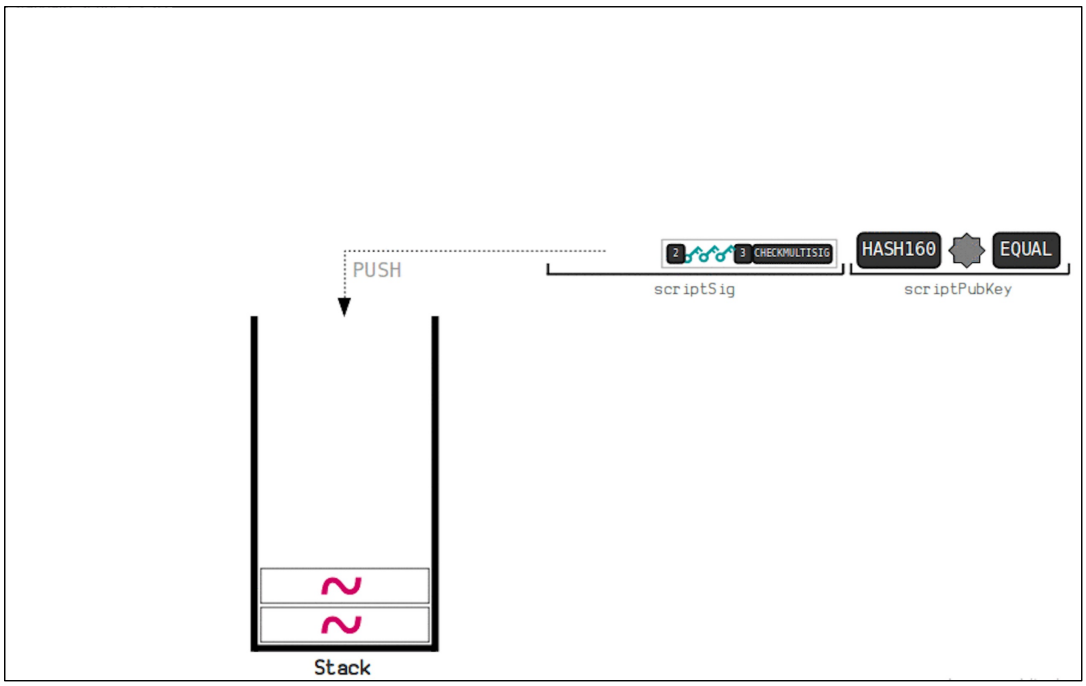
Bitcoin script

- P2SH



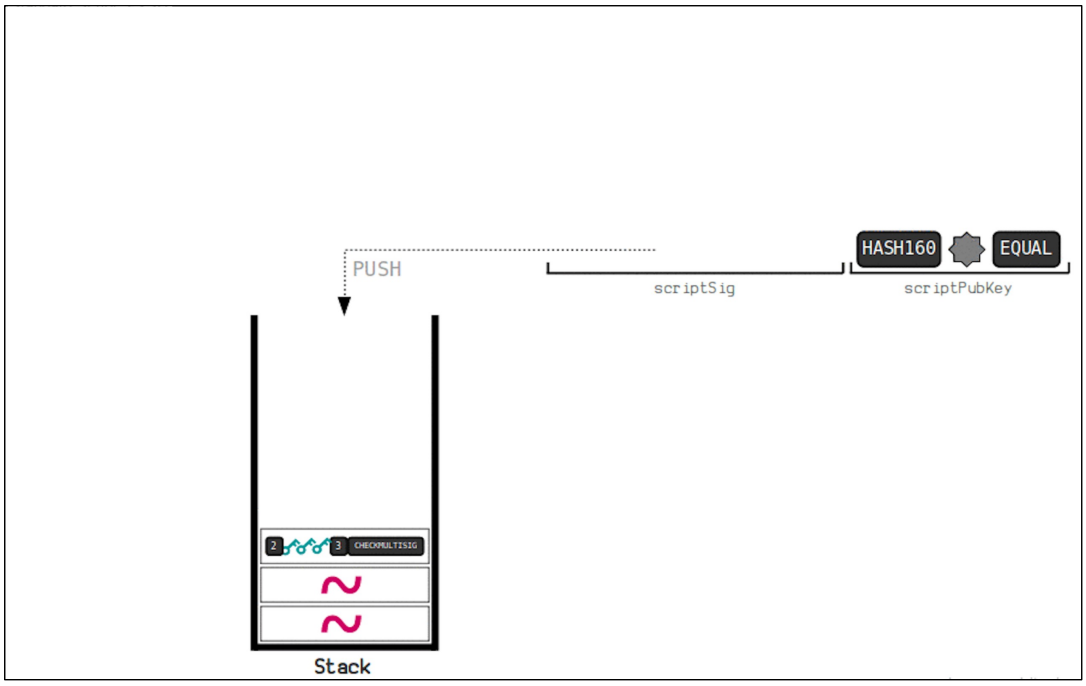
Bitcoin script

- P2SH



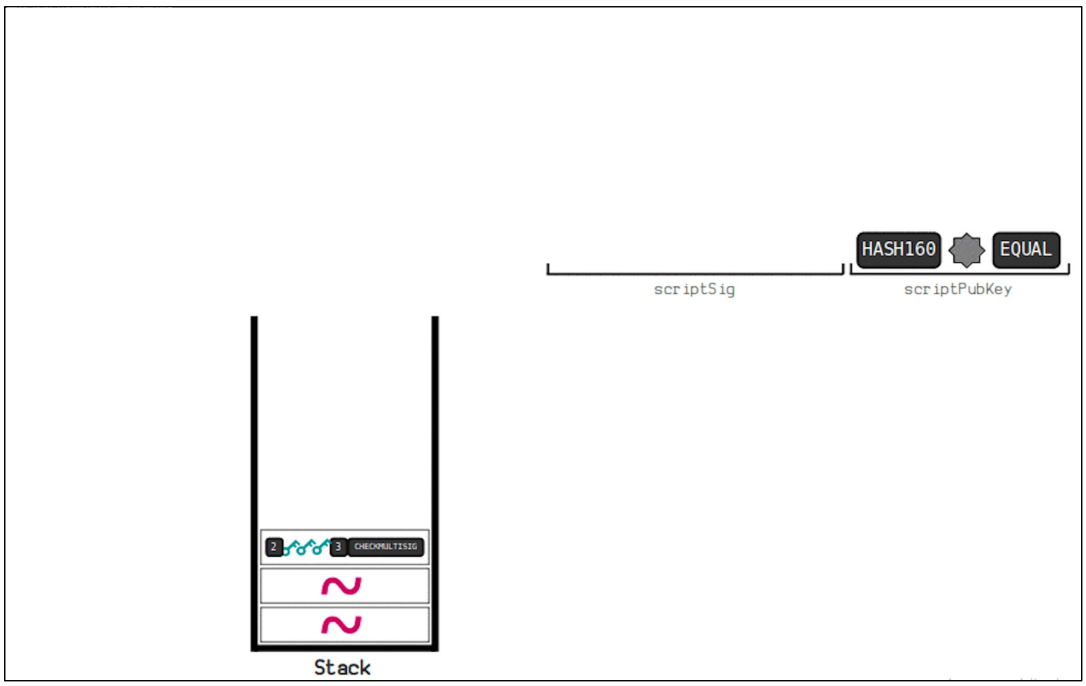
Bitcoin script

- P2SH



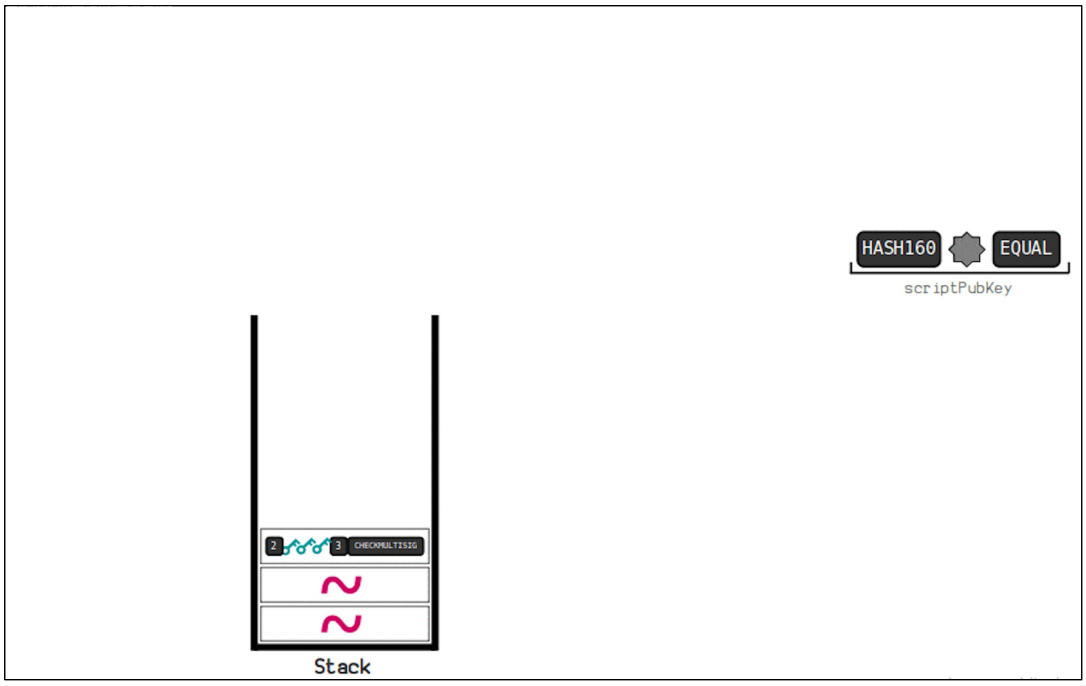
Bitcoin script

- P2SH



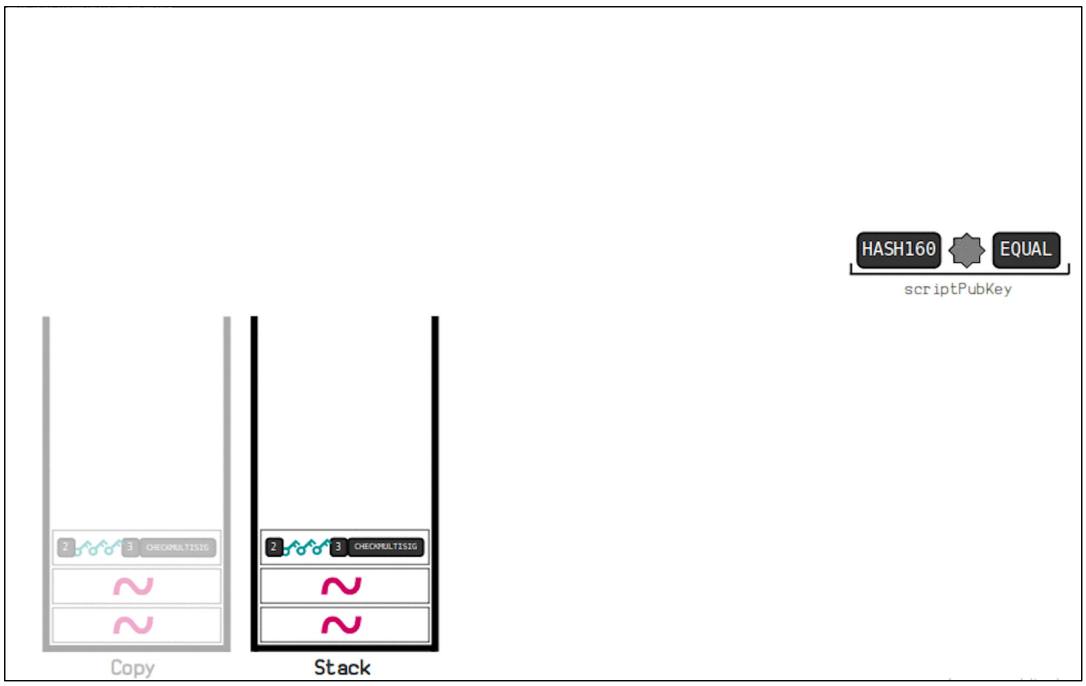
Bitcoin script

- P2SH



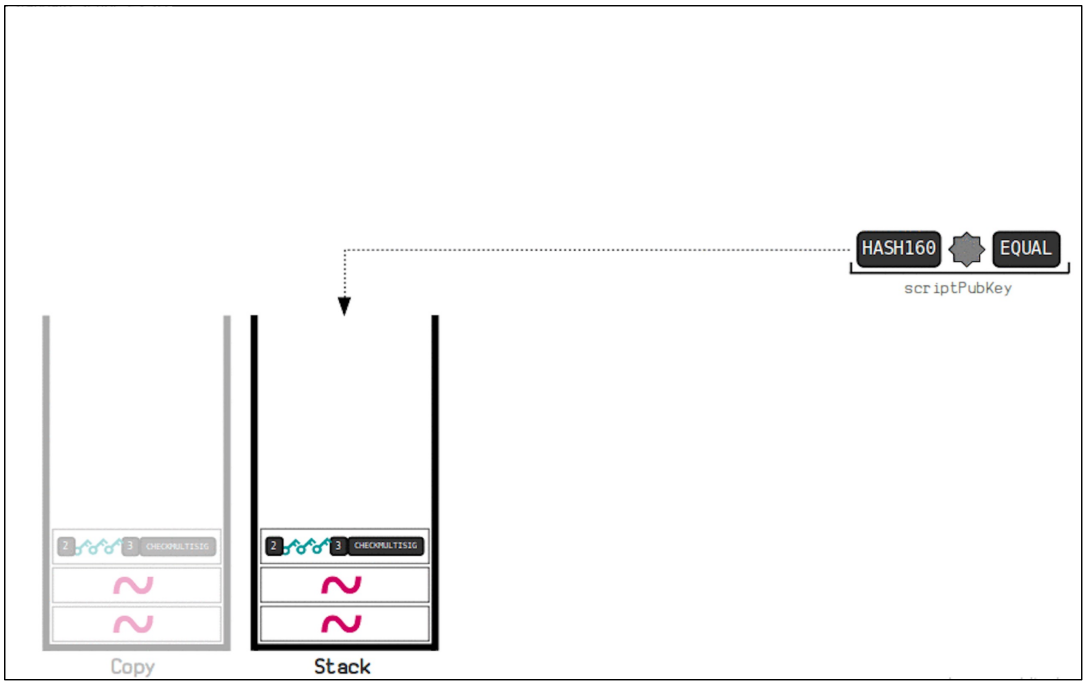
Bitcoin script

- P2SH



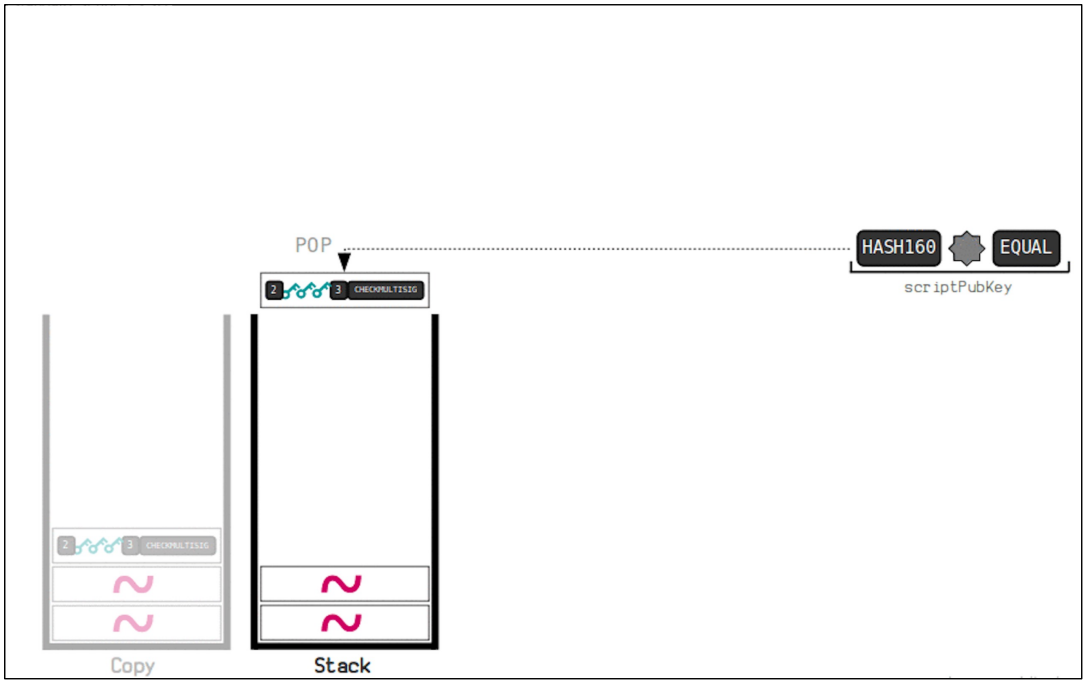
Bitcoin script

- P2SH



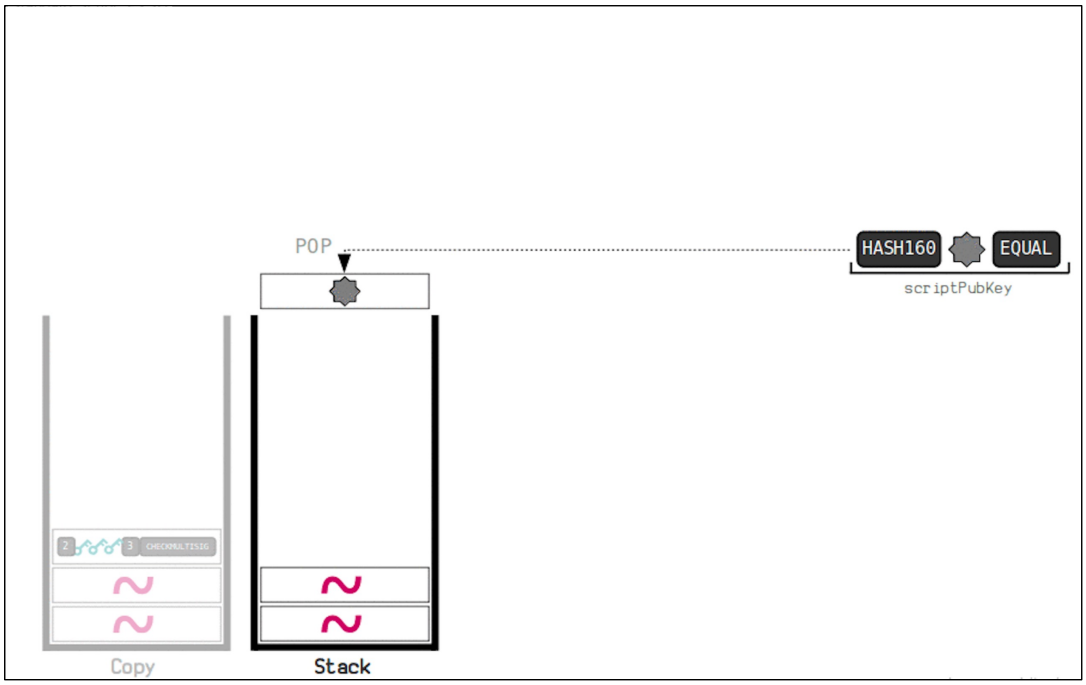
Bitcoin script

- P2SH



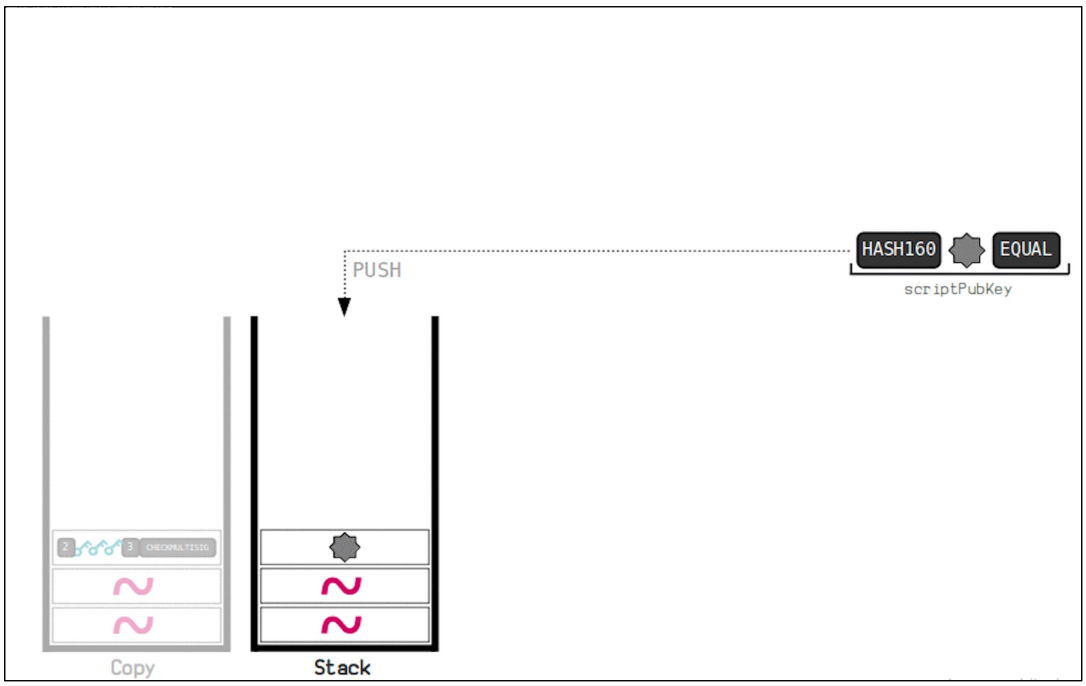
Bitcoin script

- P2SH



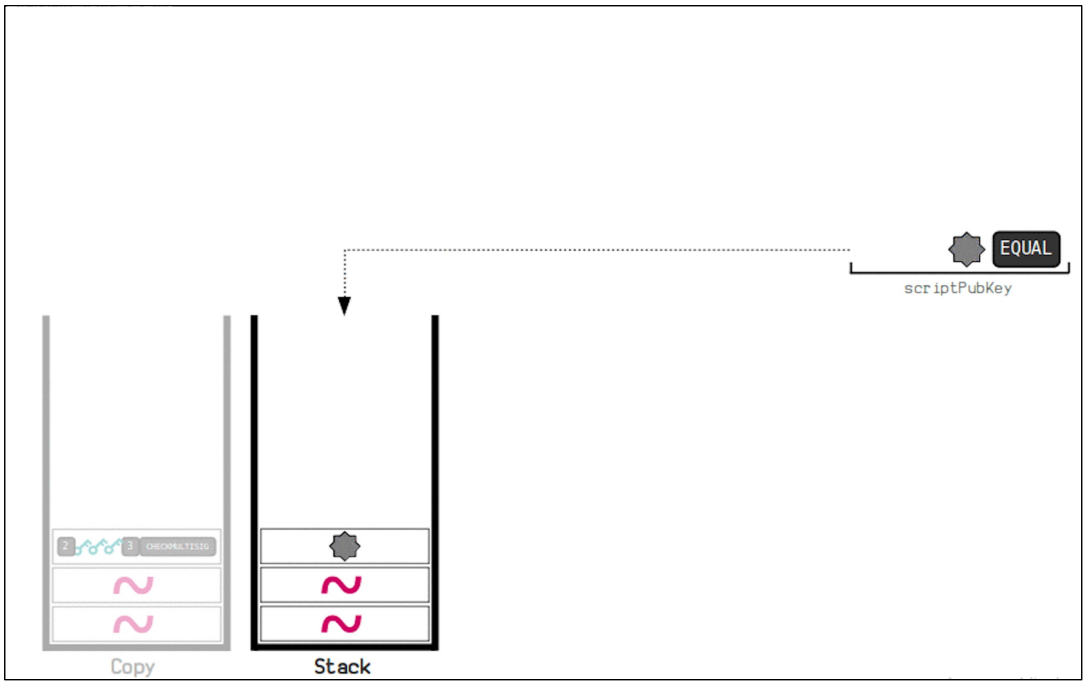
Bitcoin script

- P2SH



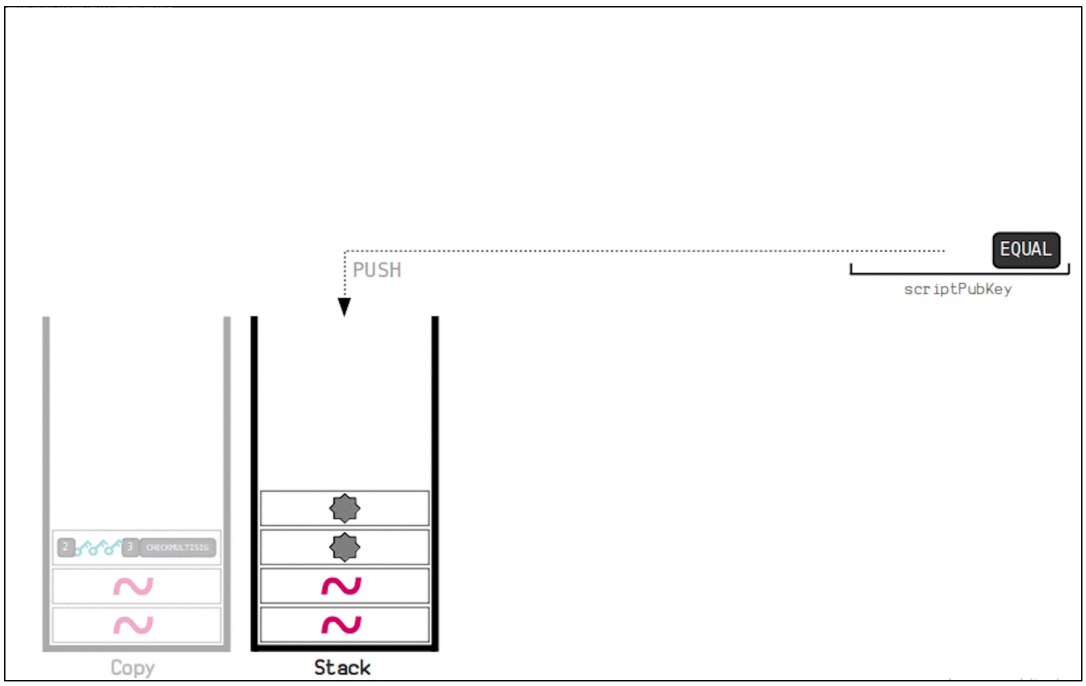
Bitcoin script

- P2SH



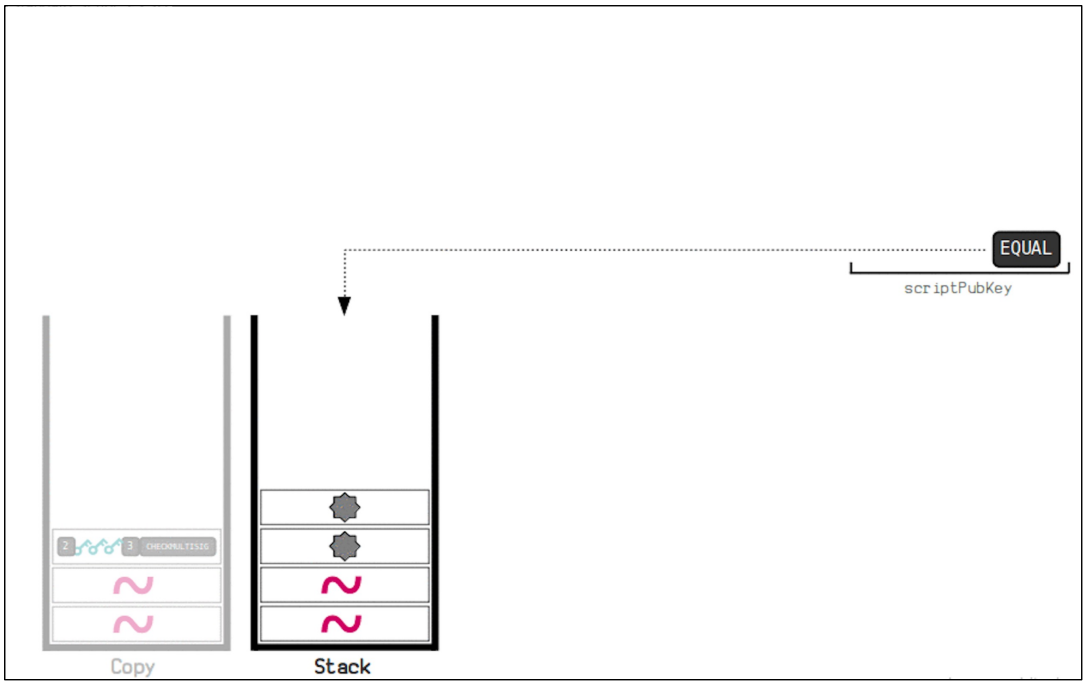
Bitcoin script

- P2SH



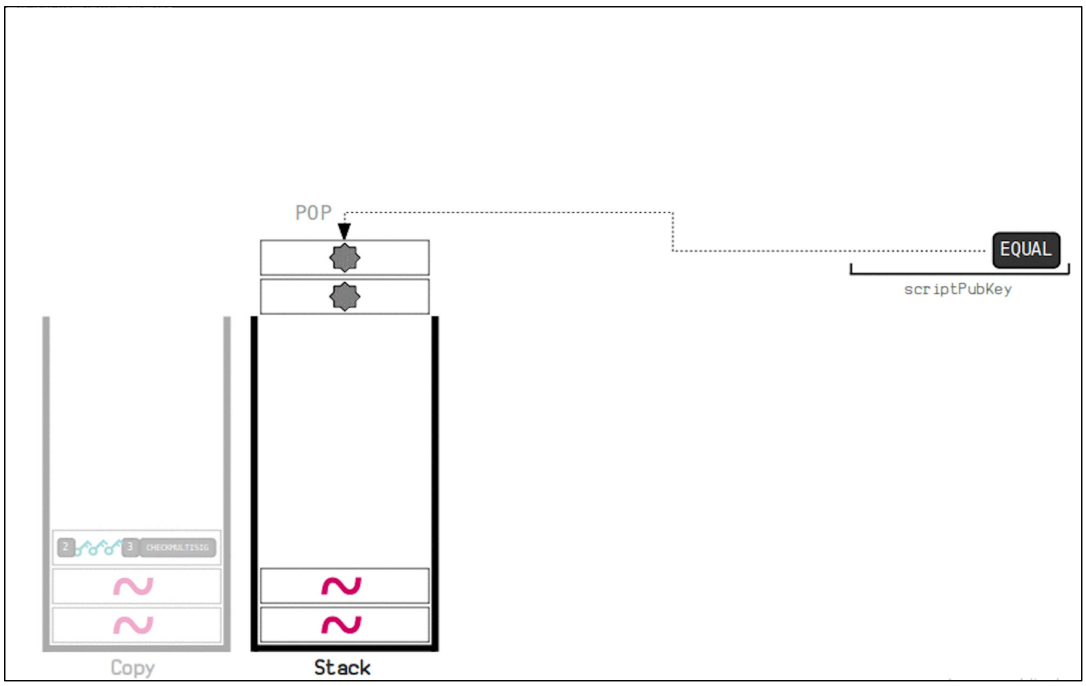
Bitcoin script

- P2SH



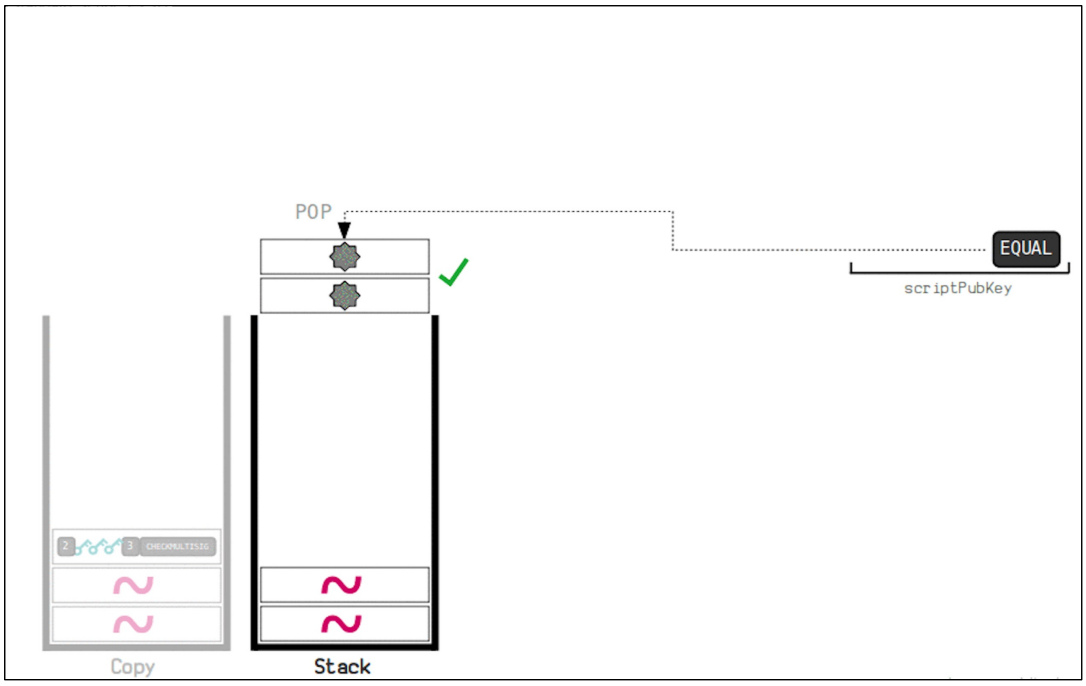
Bitcoin script

- P2SH



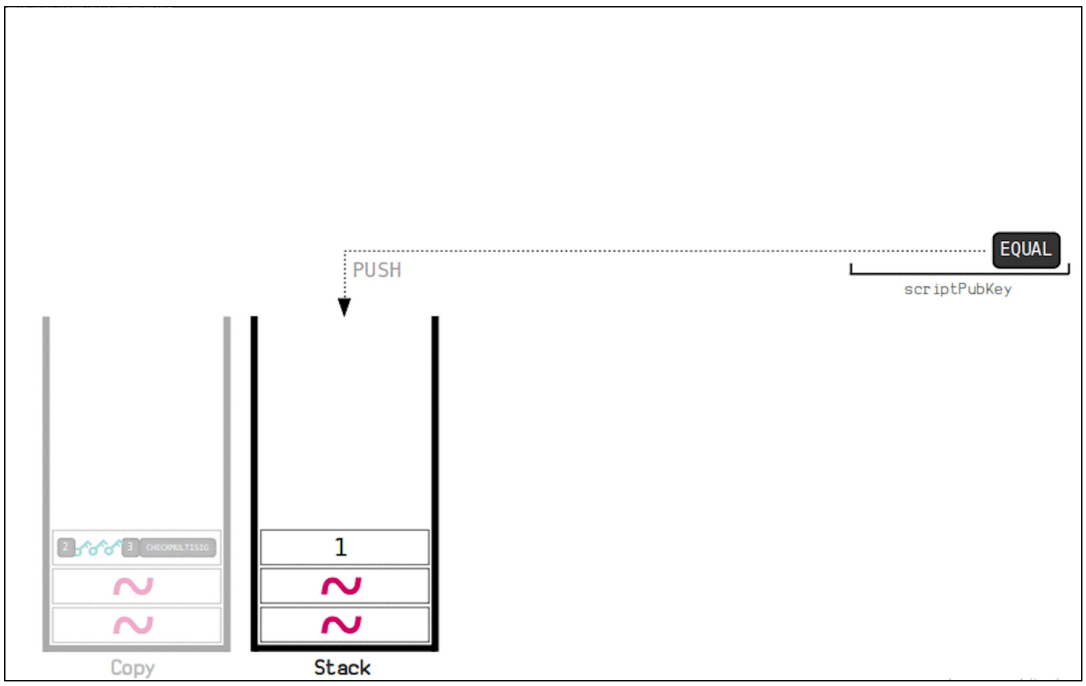
Bitcoin script

- P2SH



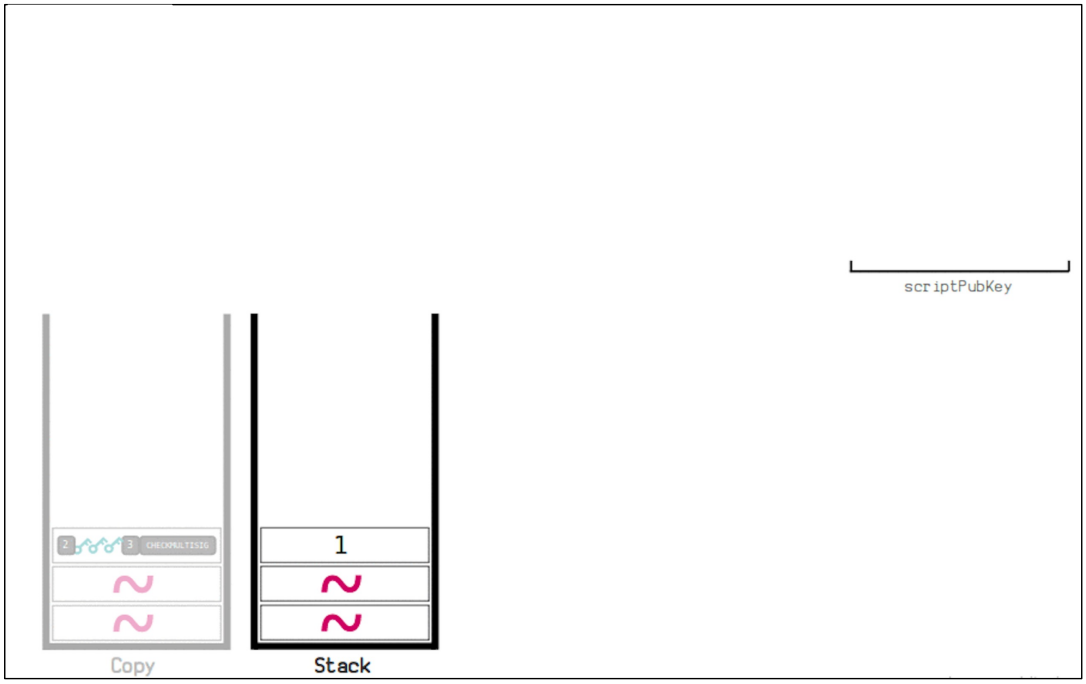
Bitcoin script

- P2SH



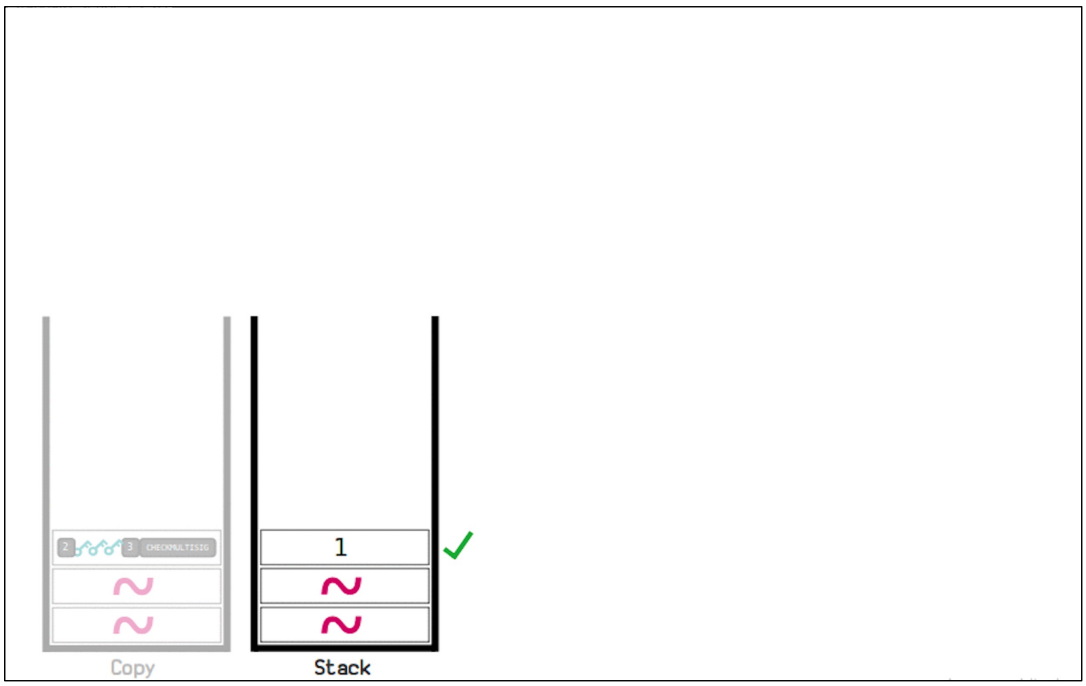
Bitcoin script

- P2SH



Bitcoin script

- P2SH

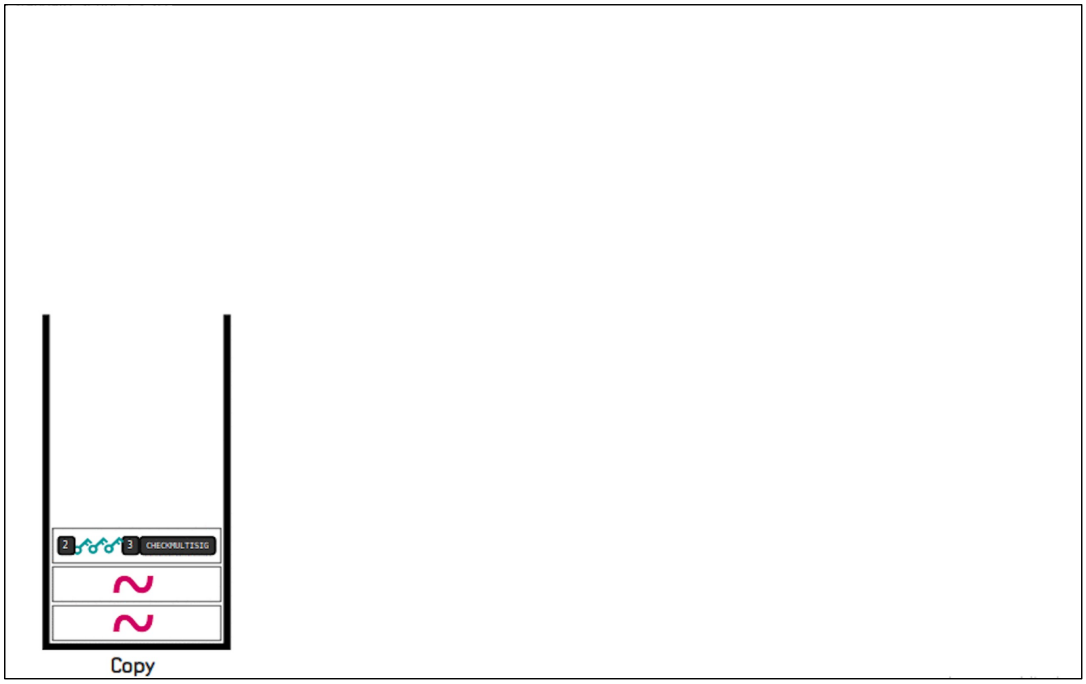


Bitcoin script

- P2SH

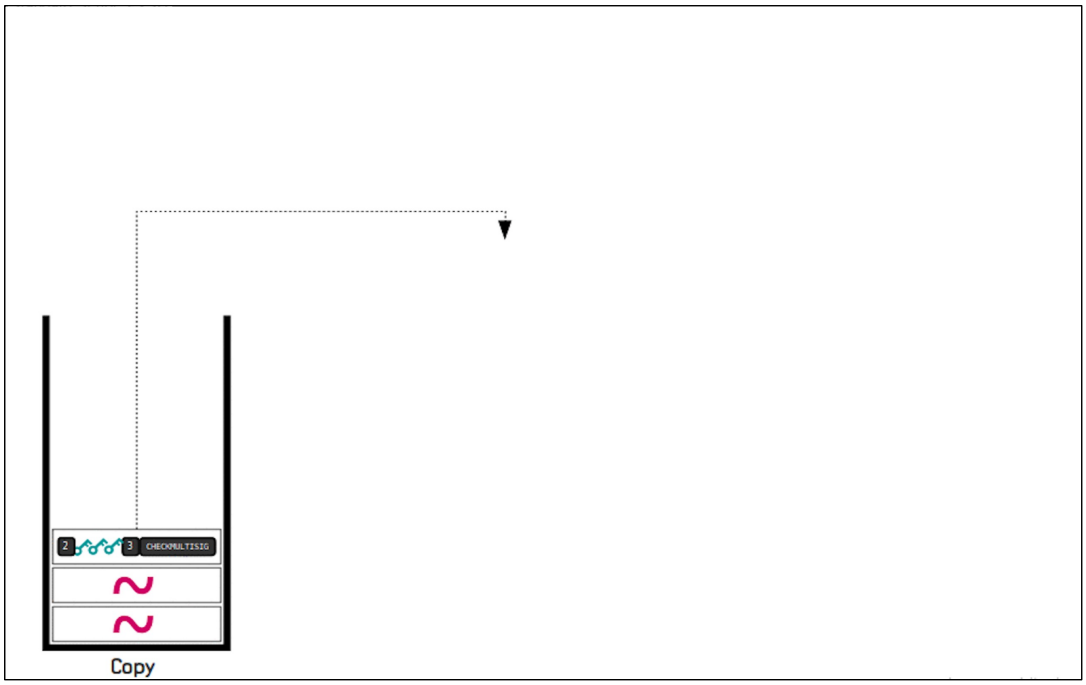
Bitcoin script

- P2SH



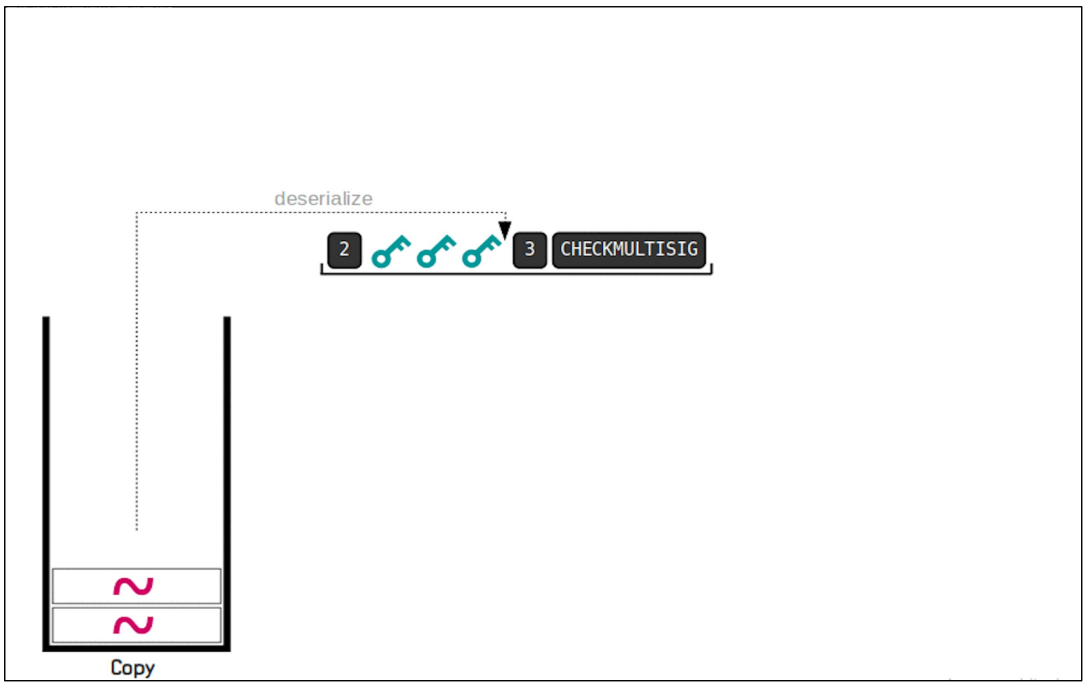
Bitcoin script

- P2SH



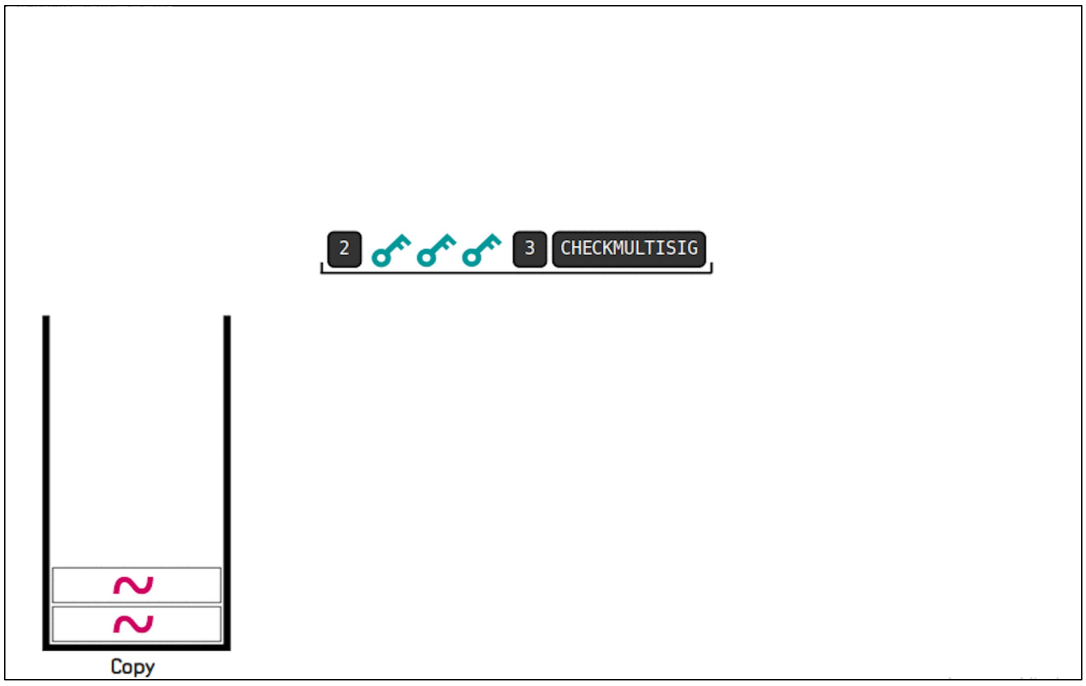
Bitcoin script

- P2SH



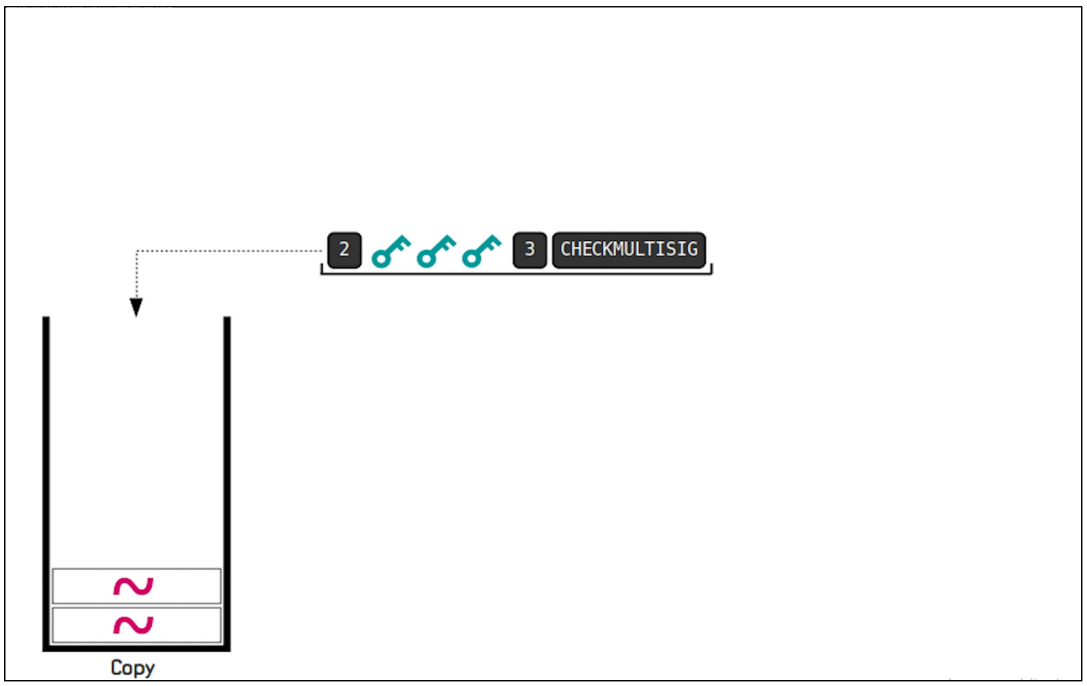
Bitcoin script

- P2SH



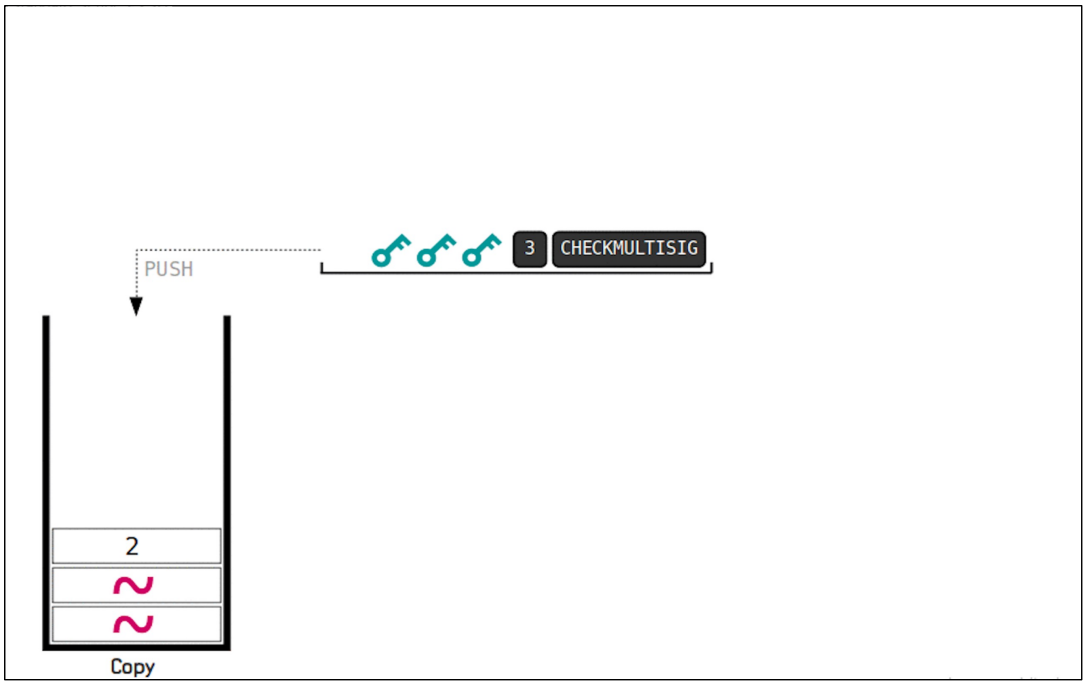
Bitcoin script

- P2SH



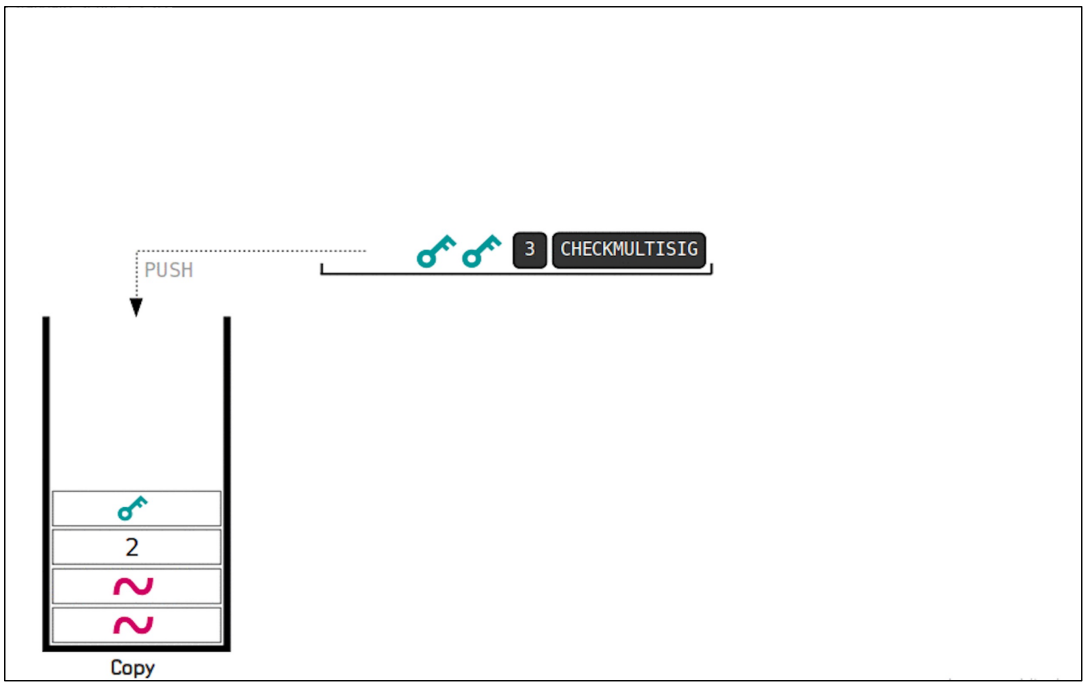
Bitcoin script

- P2SH



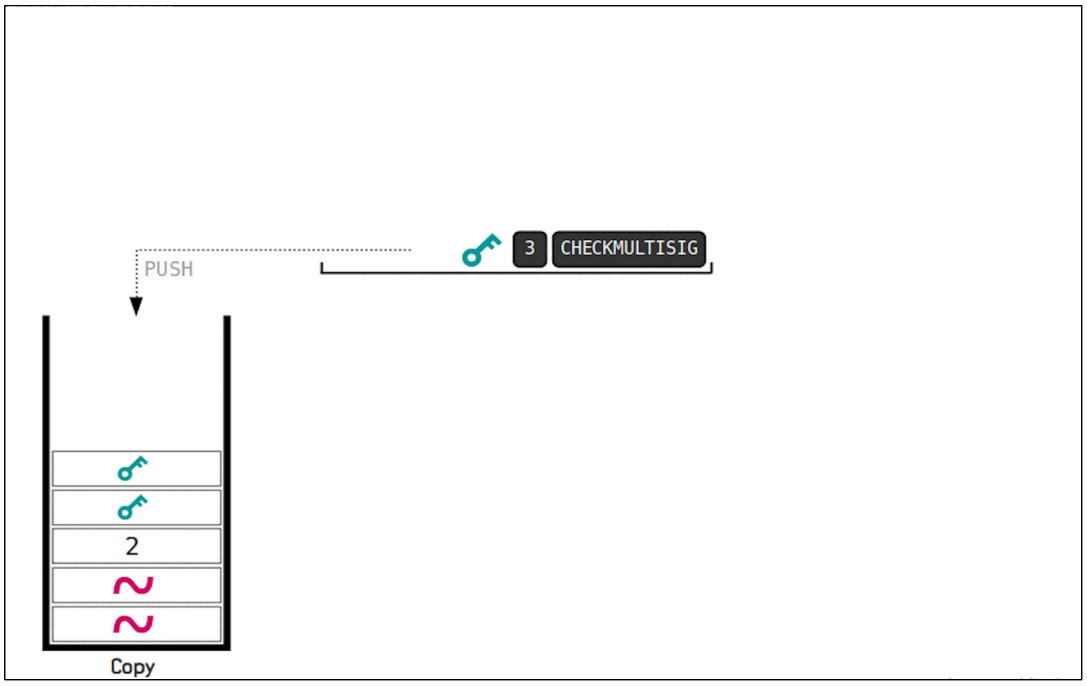
Bitcoin script

- P2SH



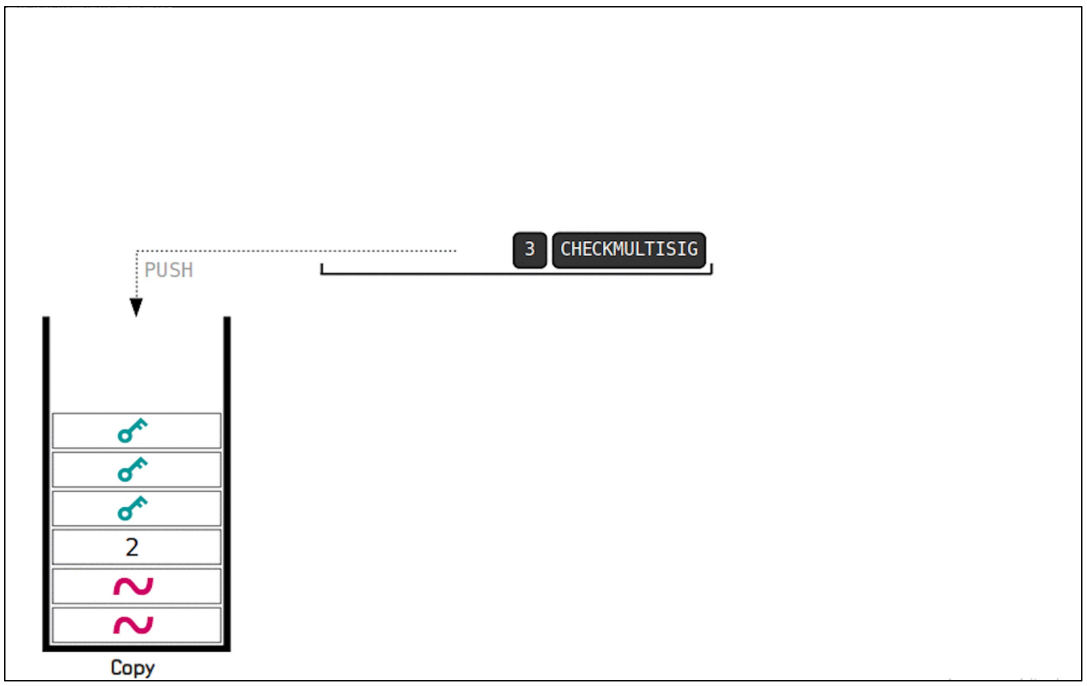
Bitcoin script

- P2SH



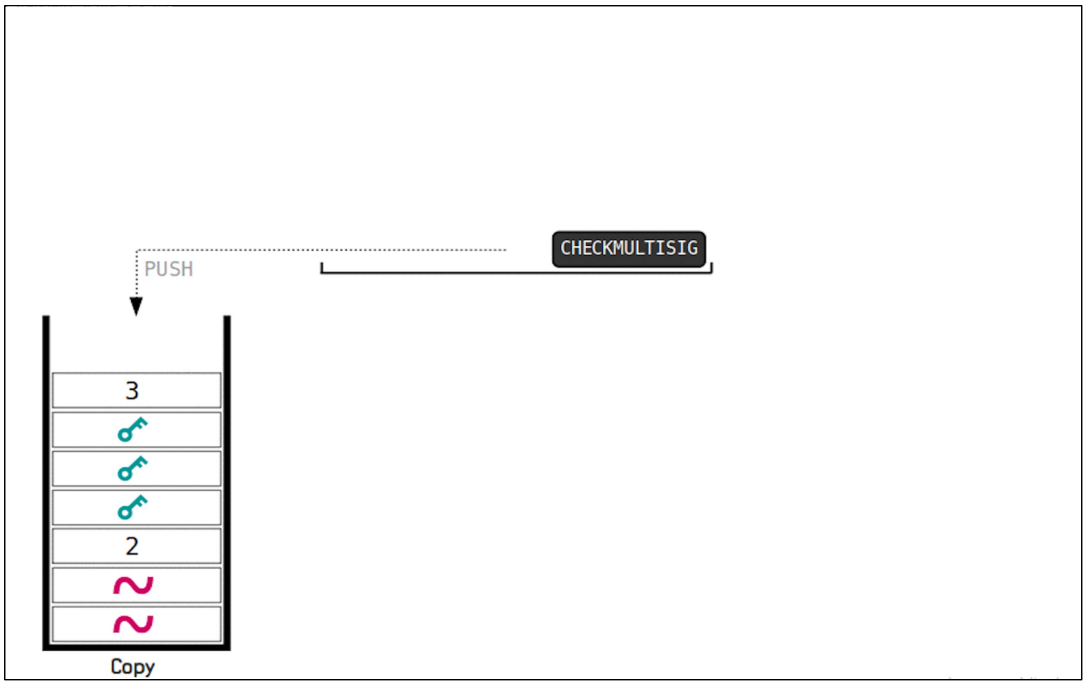
Bitcoin script

- P2SH



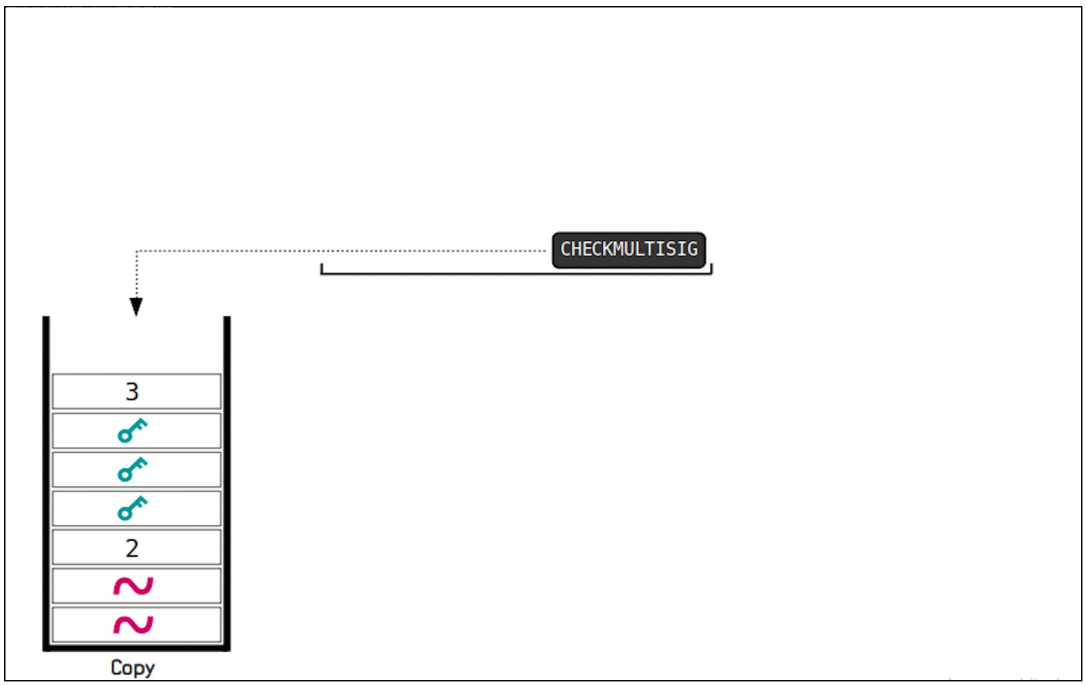
Bitcoin script

- P2SH



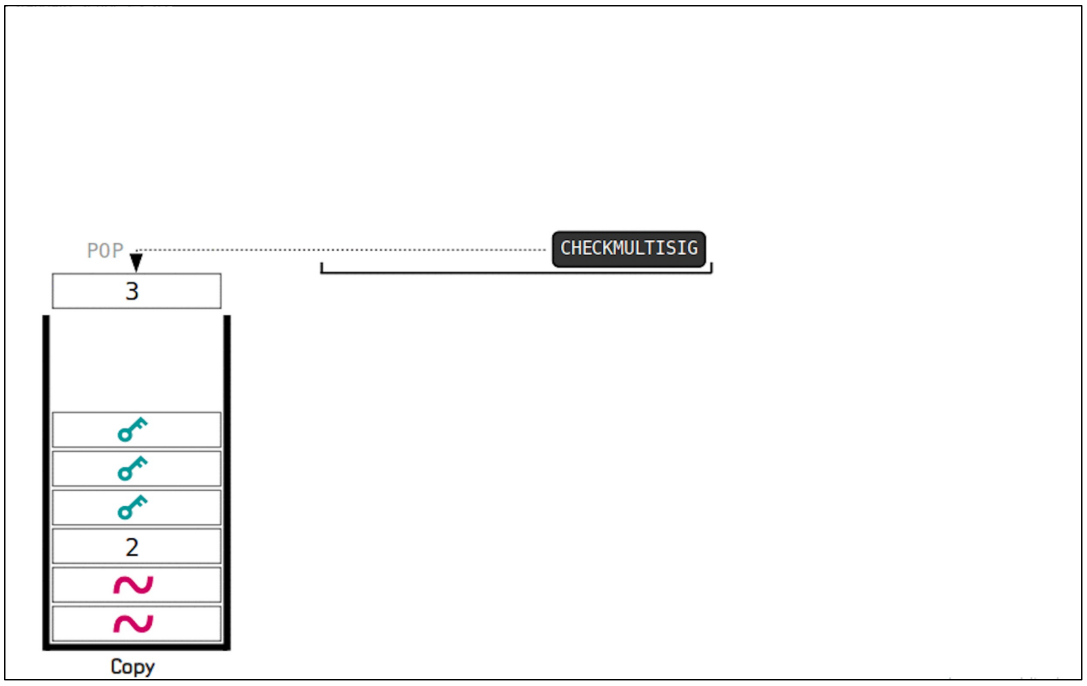
Bitcoin script

- P2SH



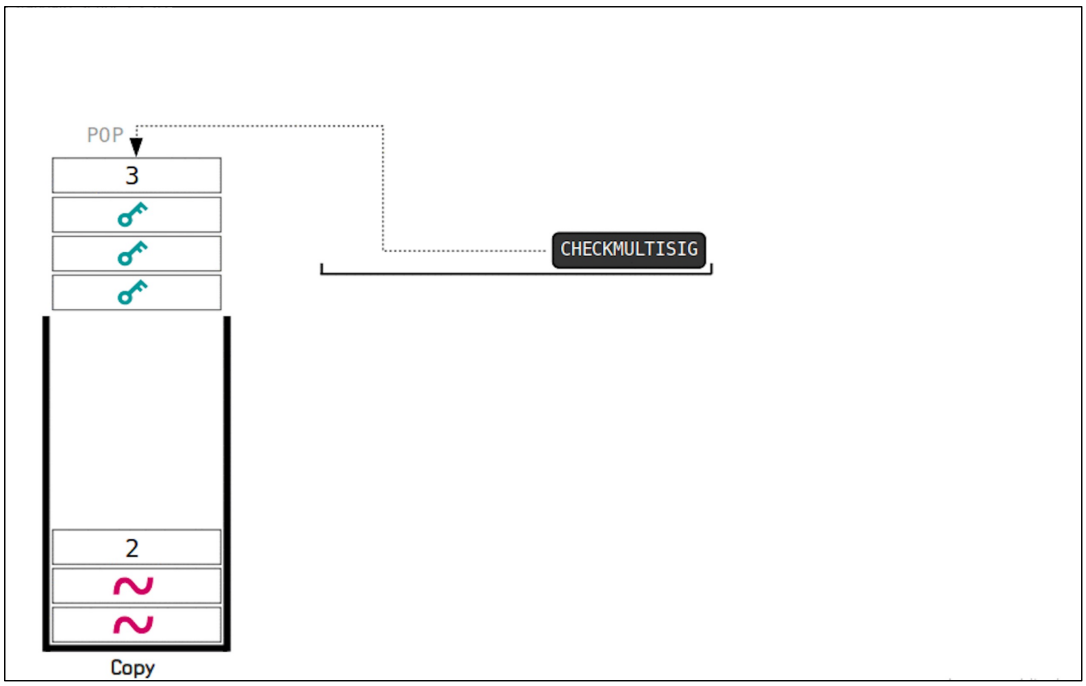
Bitcoin script

- P2SH



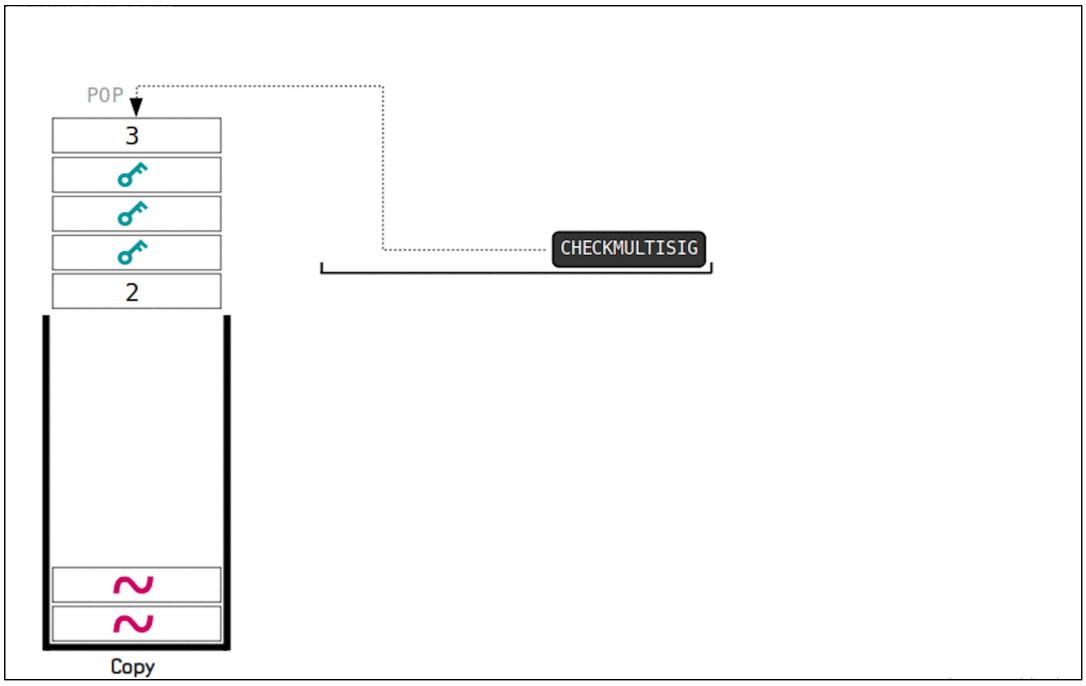
Bitcoin script

- P2SH



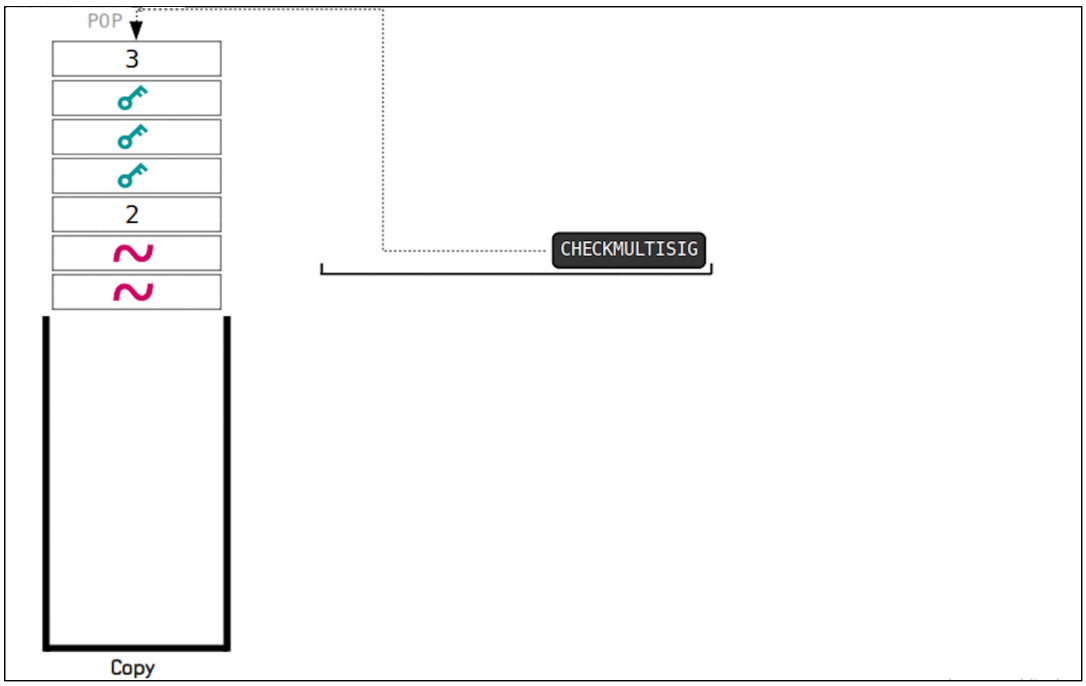
Bitcoin script

- P2SH



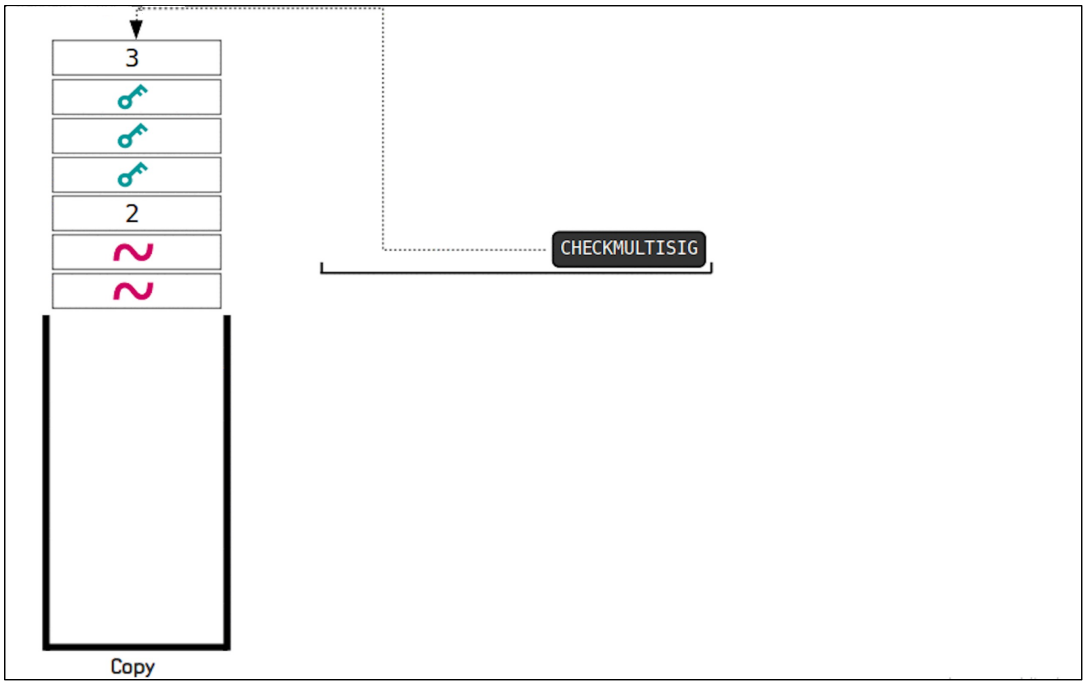
Bitcoin script

- P2SH



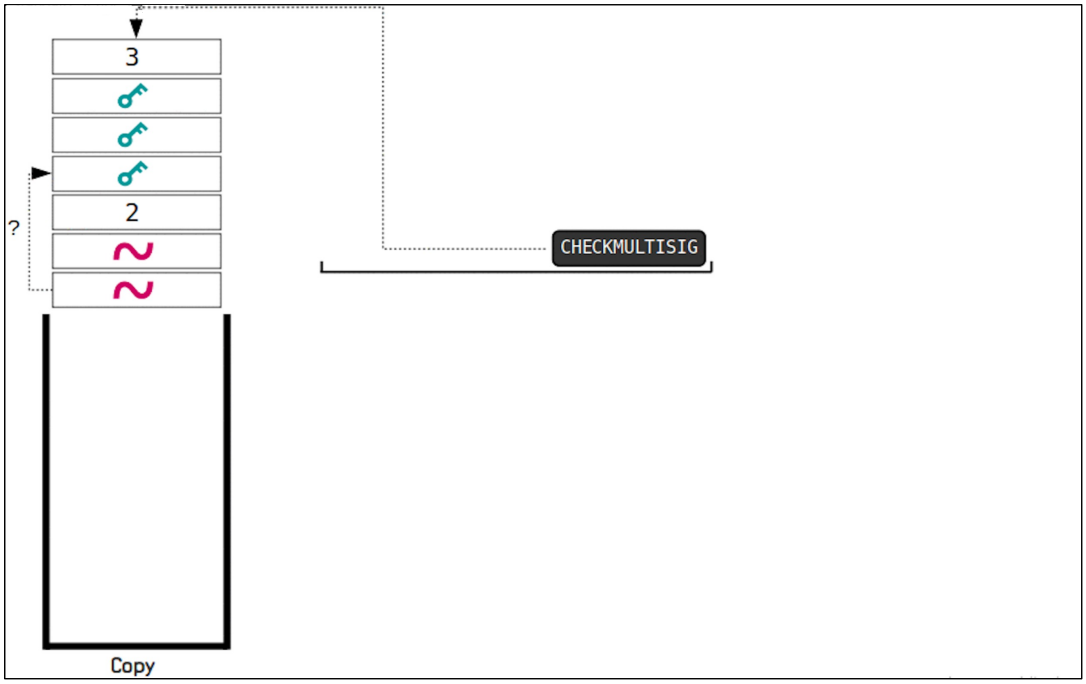
Bitcoin script

- P2SH



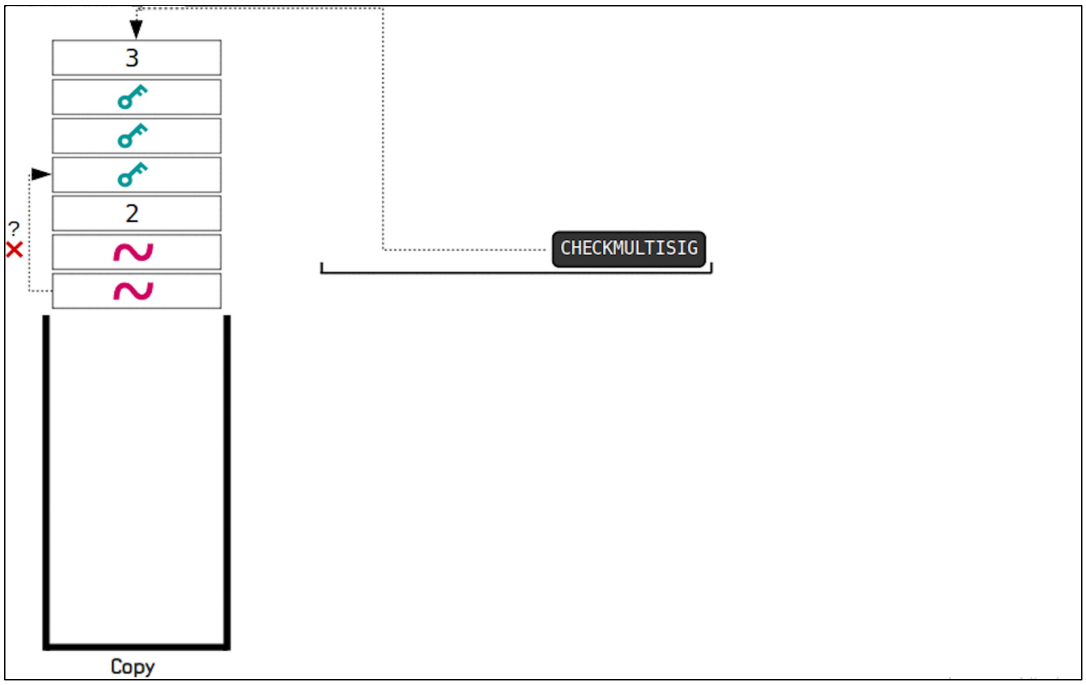
Bitcoin script

- P2SH



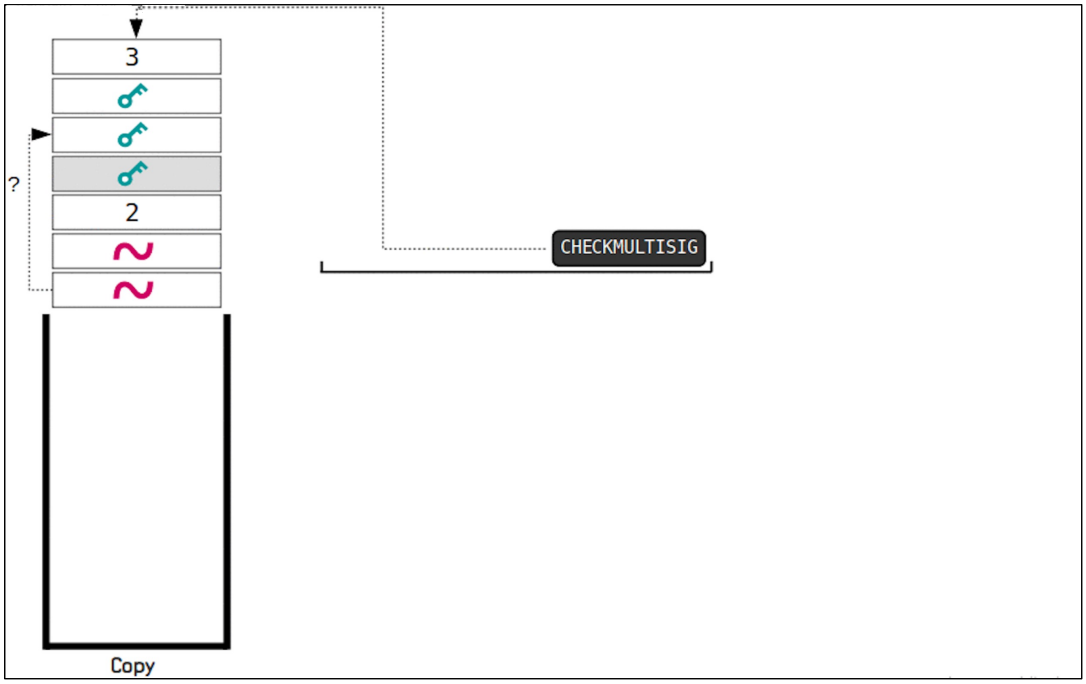
Bitcoin script

- P2SH



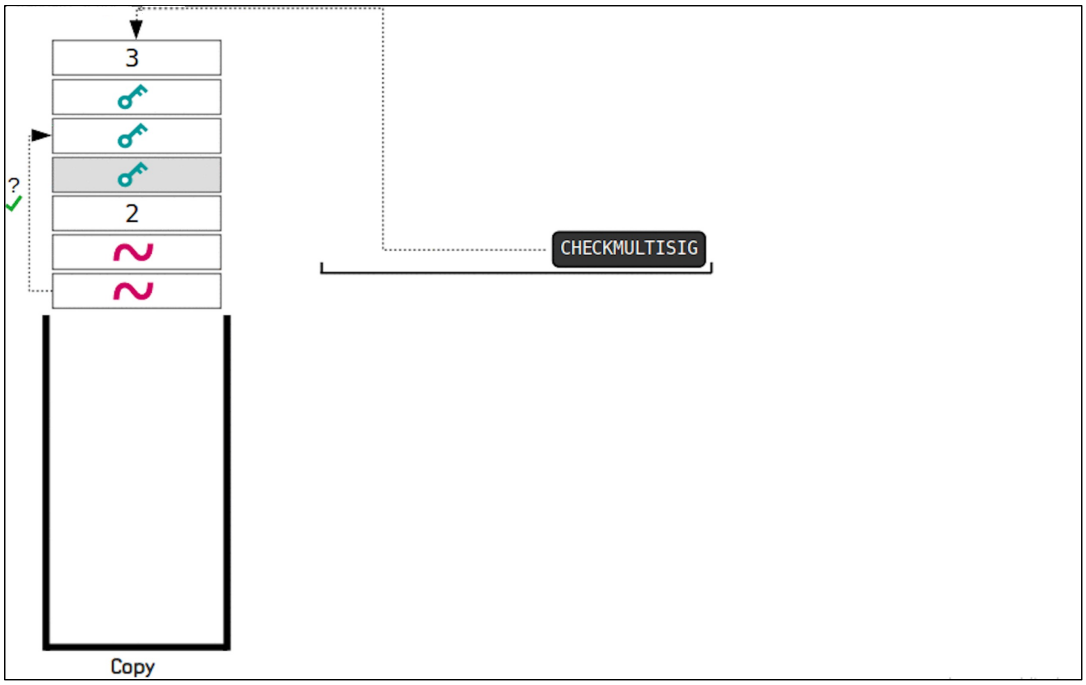
Bitcoin script

- P2SH



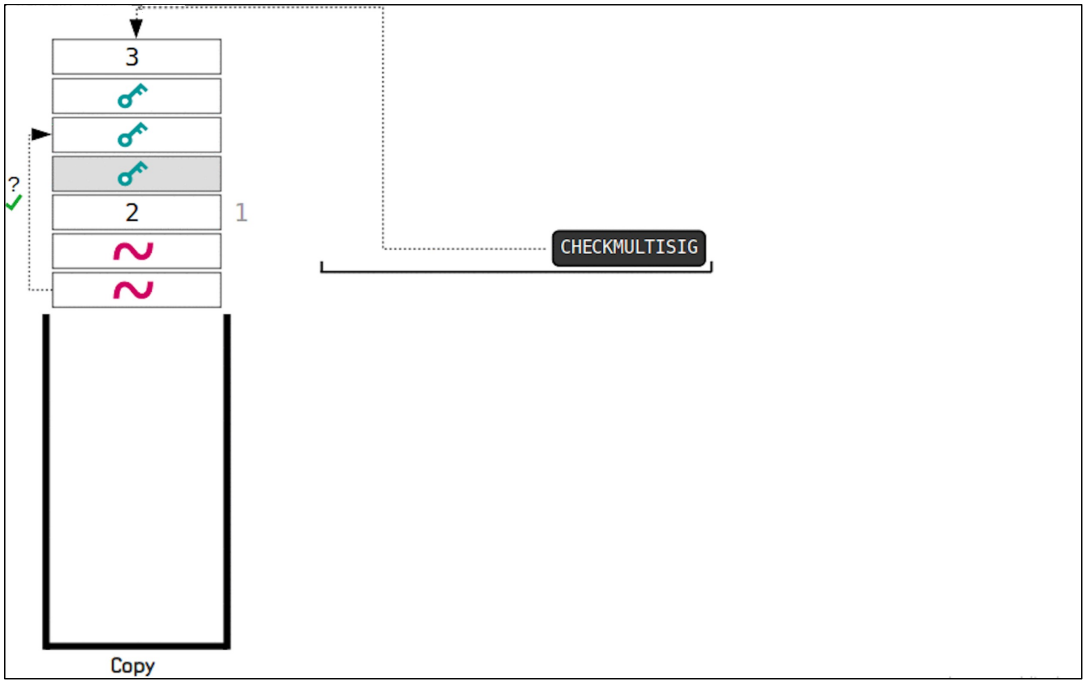
Bitcoin script

- P2SH



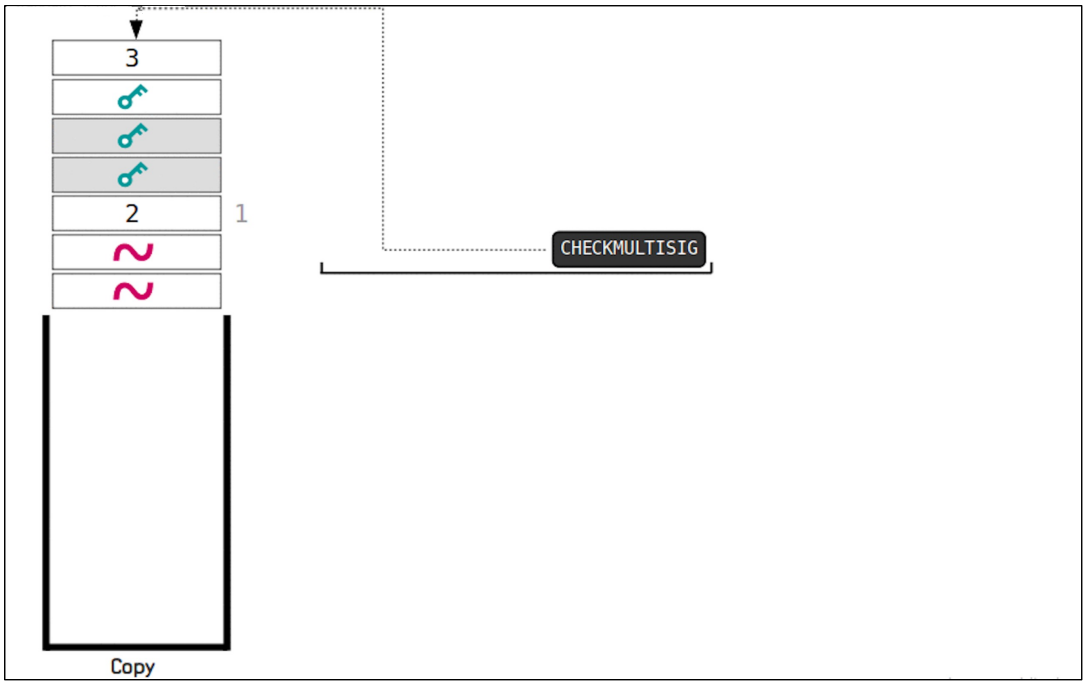
Bitcoin script

- P2SH



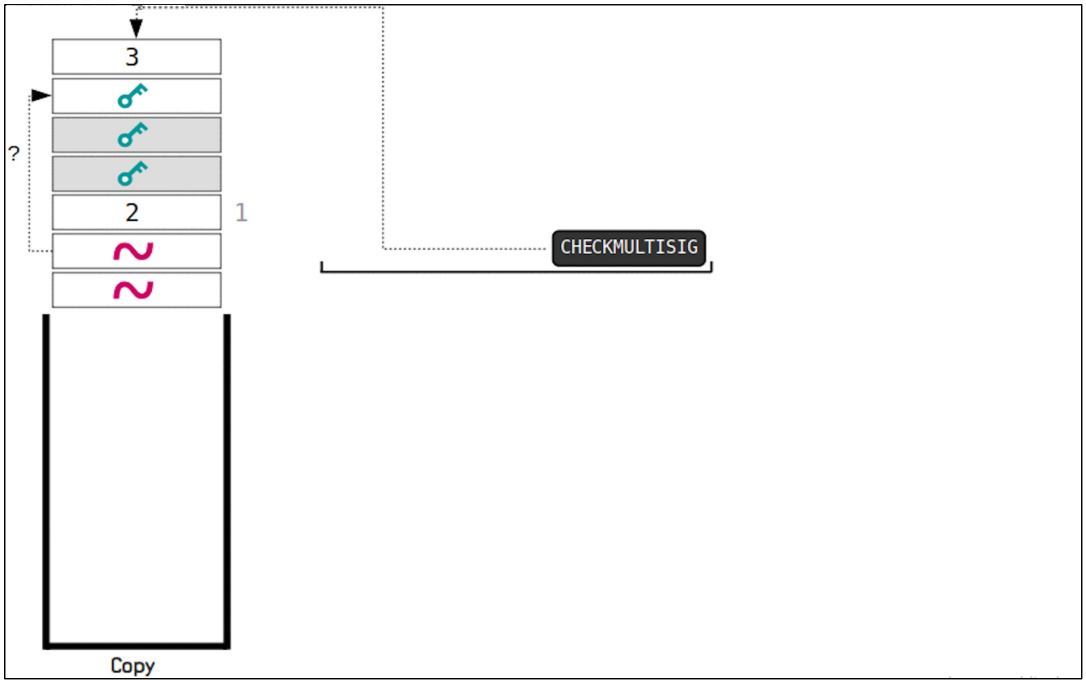
Bitcoin script

- P2SH



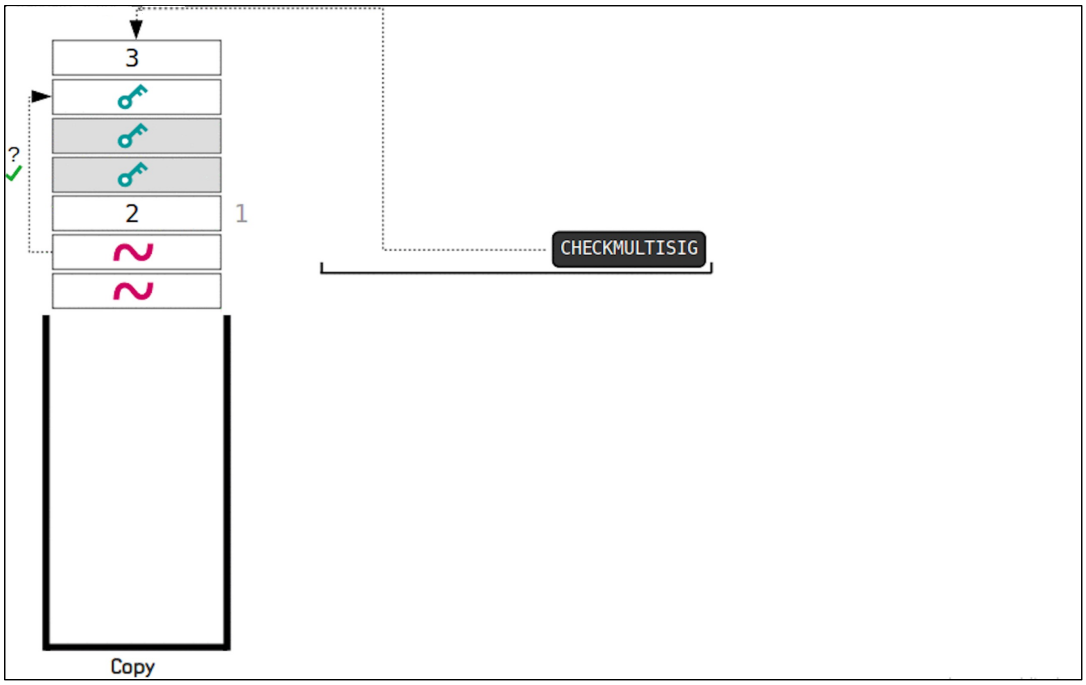
Bitcoin script

- P2SH



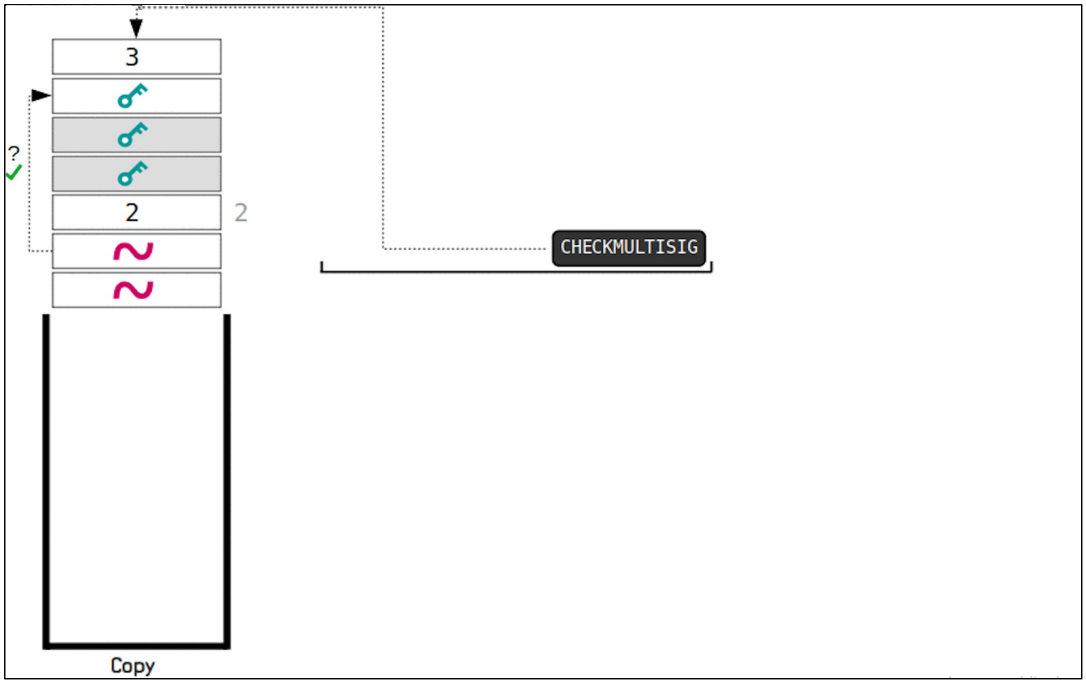
Bitcoin script

- P2SH



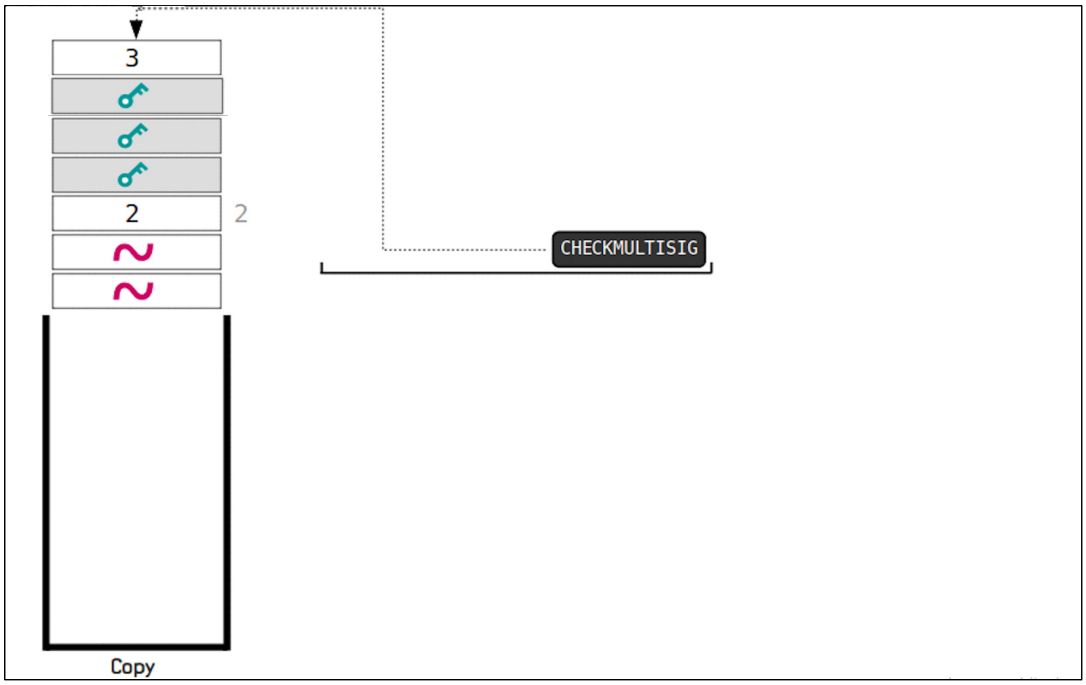
Bitcoin script

- P2SH



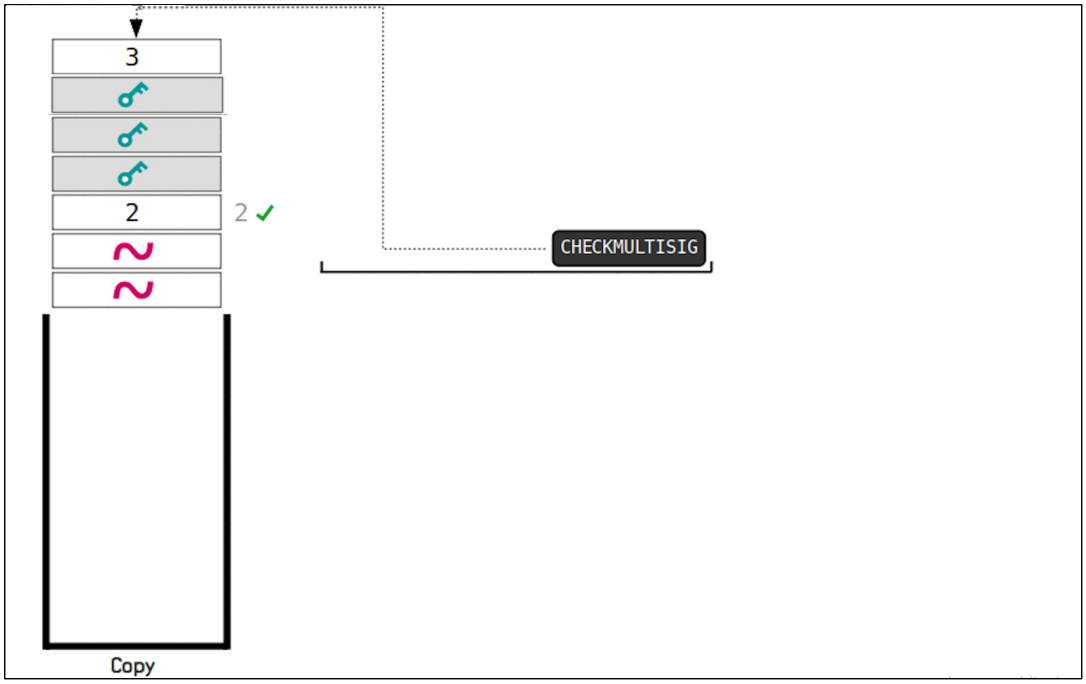
Bitcoin script

- P2SH



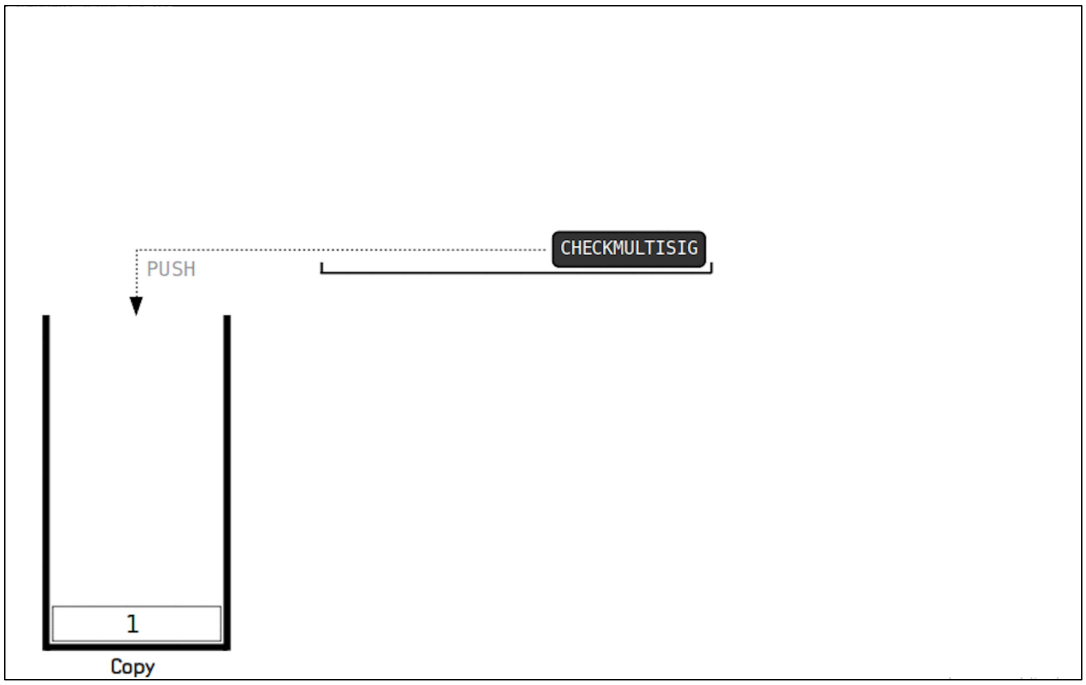
Bitcoin script

- P2SH



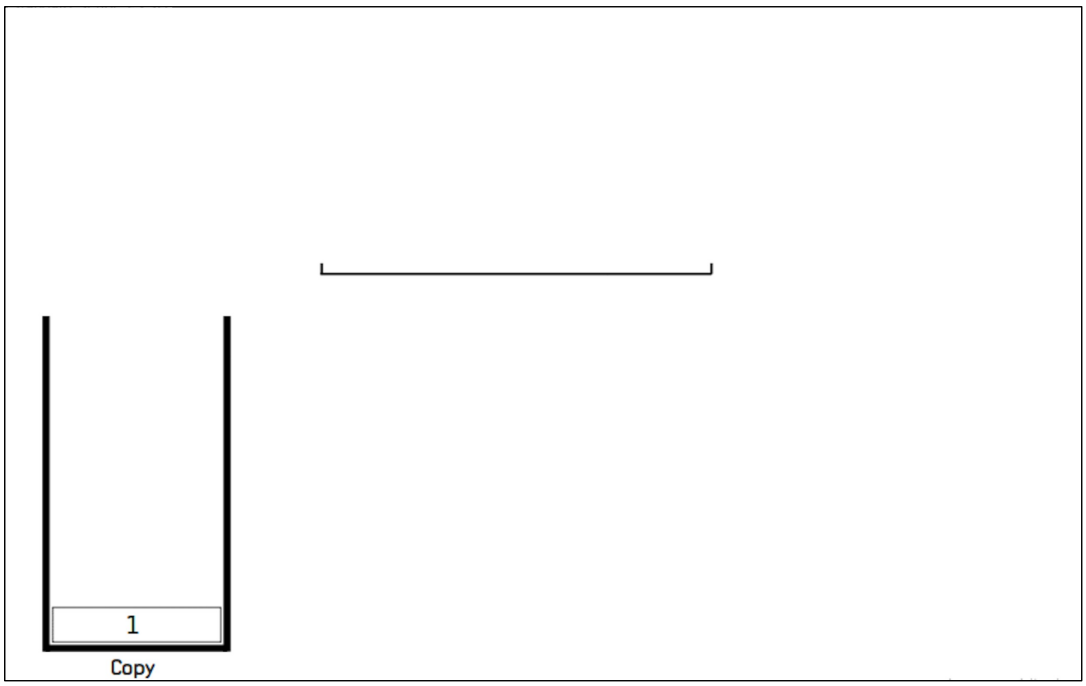
Bitcoin script

- P2SH



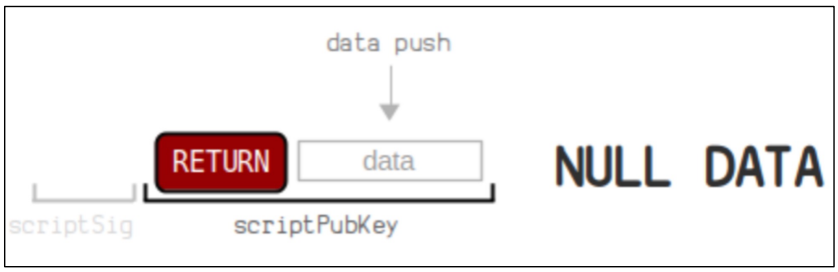
Bitcoin script

- P2SH



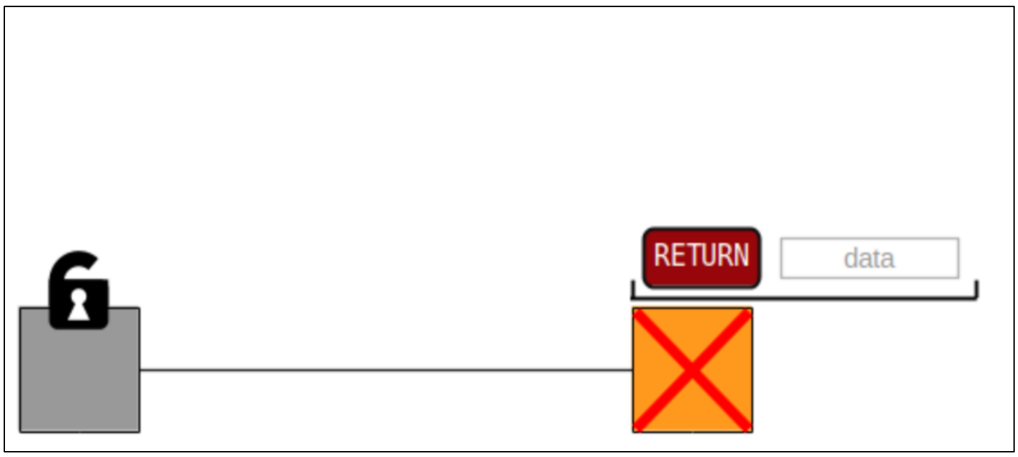
Bitcoin script

- NULL DATA is used to store arbitrary data on the blockchain
  - Store up to 80 bytes of data
  - *OP_RETURN* terminates script execution, marks Tx invalid
  - Tx output with NULL DATA locking script is provably unspendable
  - Nodes relay Tx with max 1 NULL DATA locking script
  - Mere presence of *OP_RETURN* doesn't render script invalid
    - Branching
    - *OP_RETURN* must be evaluated



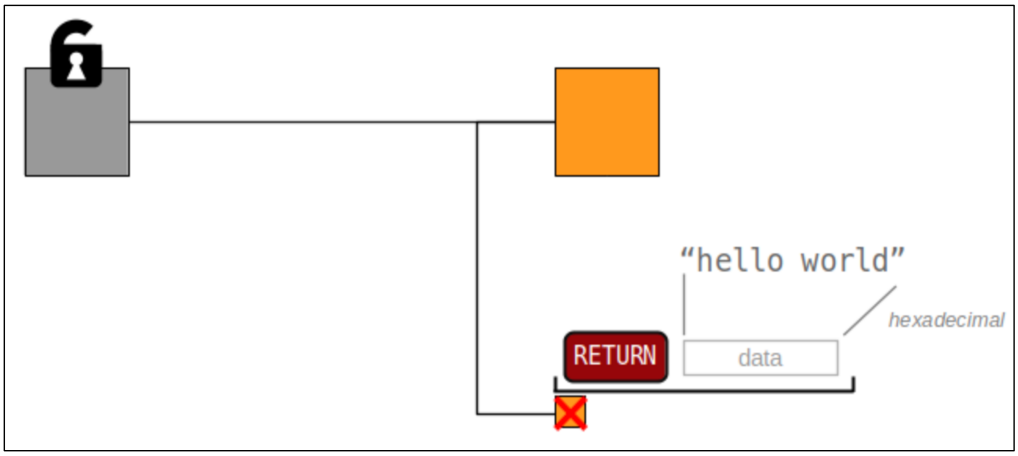
Bitcoin script

- NULL DATA



Bitcoin script

- NULL DATA



Bitcoin script

- NULL DATA

6dfb16dd580698242bcfd8e433d557ed8c642272a368894de27292a8844a4e75

Bitcoin script

- Other applications
  - Escrow Tx
    - A->B, B->A | no trust between A & B | multi-signature Tx with (A+B+C)
  - Green addresses
    - Guarenter's Bitcoin address
      - Credit card | multi-signature | no wait | reputation
      - Failed idea Instawallet, Mt. Gox
  - lock_time
    - Lock funds in future
  - Smart contracts
    - Better in ETH than BTC