CBS COPENHAGEN BUSINESS SCHOOL
HANDELSHØJSKOLEN

Elective Course on Mastering Blockchain: Foundations to Consensus, session-05

Bitcoin Scripting Fundamentals

Raghava Mukkamala

Associate Professor & Director, Centre for Business Data Analytics
Copenhagen Business School, Denmark
Email: rrm.digi@cbs.dk, Centre: https://cbsbda.github.io/

Course Coordinator at SRMIST:

Prof. K. Shantha Kumari

Associate Professor
Data Science and Business Systems Department,
SRM Institute of Science and Technology, India
Shanthak@srmist.edu.in

EQUIS ACCREDITED AACSB ACCREDITED ASSOCIATION AMBA ACCREDITED CEMS PIM

Outline

- Splitting and Sharing Keys
- Bitcoin scripts
- Script: Language Features
- Script: Multisignatures
- Script: Pay-to-Script-Hash
- Applications of Bitcoin scripts

Splitting and Sharing Keys

Slides based on Bitcoin and Cryptocurrency Technologies: http://bitcoinbook.cs.princeton.edu/

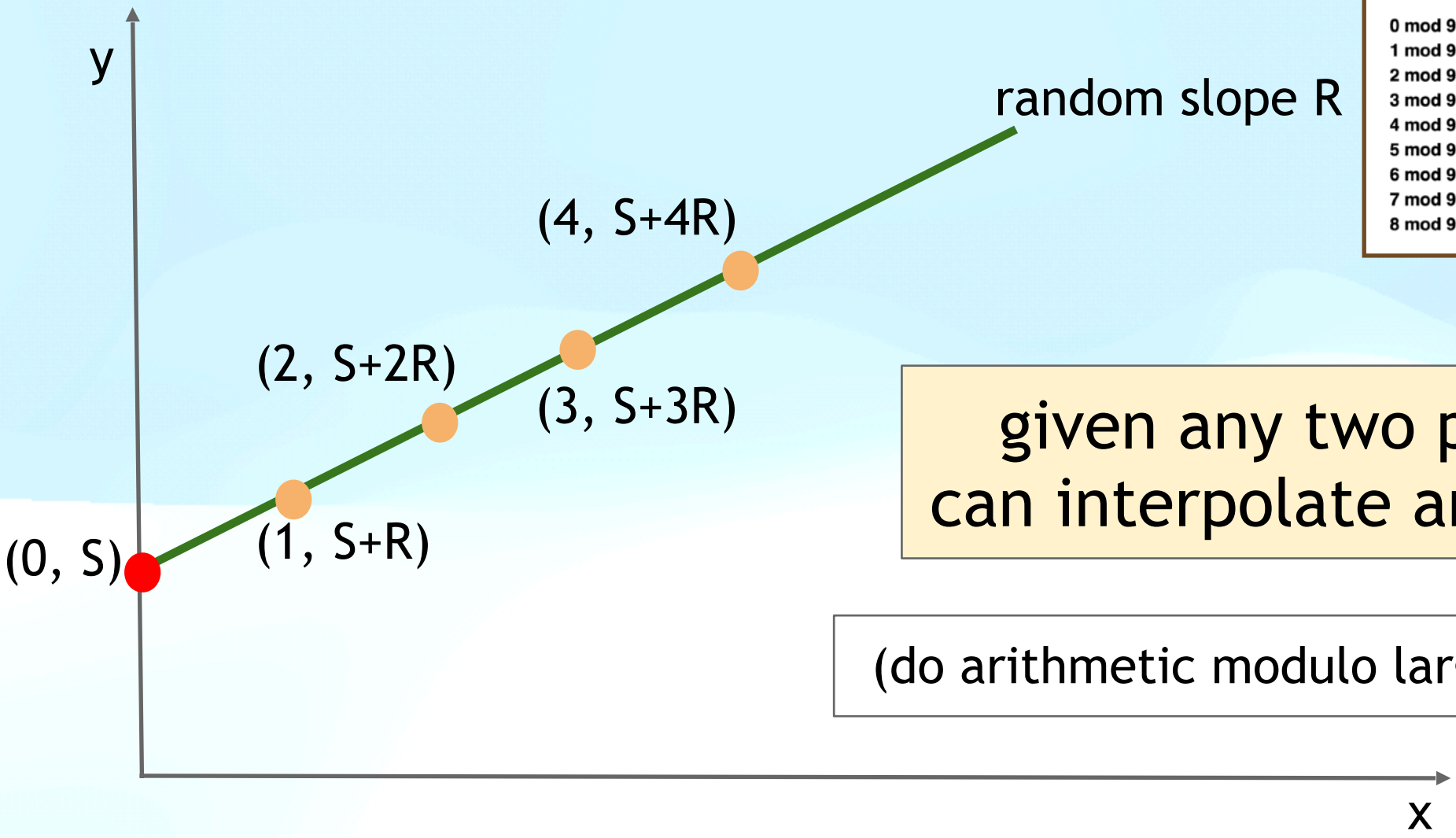
Image by pch.vector on Freepik

CBS

Enhancing Key Security: Splitting Keys

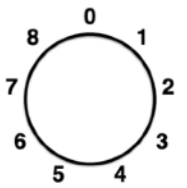
- We've looked at different ways of storing and managing the secret keys that control bitcoins, but we've always put a key in a single place: whether locked in a safe, in software, or on paper.
- This leaves us with a single point of failure.
- We could create and store backups of the key material, but while this will decrease the risk of lost or corrupted (availability), it increases the risk of theft (security)
- ***Can we take a piece of data and store it in such a way that availability and security increase at the same time?***
- One more cryptographic trick - *Secret Sharing*

Arvind Narayanan et. al., (2016). Bitcoin and Cryptocurrency Technologies. p.111



Modulus 9

0 mod 9 = 0	9 mod 9 = 0
1 mod 9 = 1	10 mod 9 = 1
2 mod 9 = 2	11 mod 9 = 2
3 mod 9 = 3	12 mod 9 = 3
4 mod 9 = 4	13 mod 9 = 4
5 mod 9 = 5	14 mod 9 = 5
6 mod 9 = 6	15 mod 9 = 6
7 mod 9 = 7	16 mod 9 = 7
8 mod 9 = 8	17 mod 9 = 8



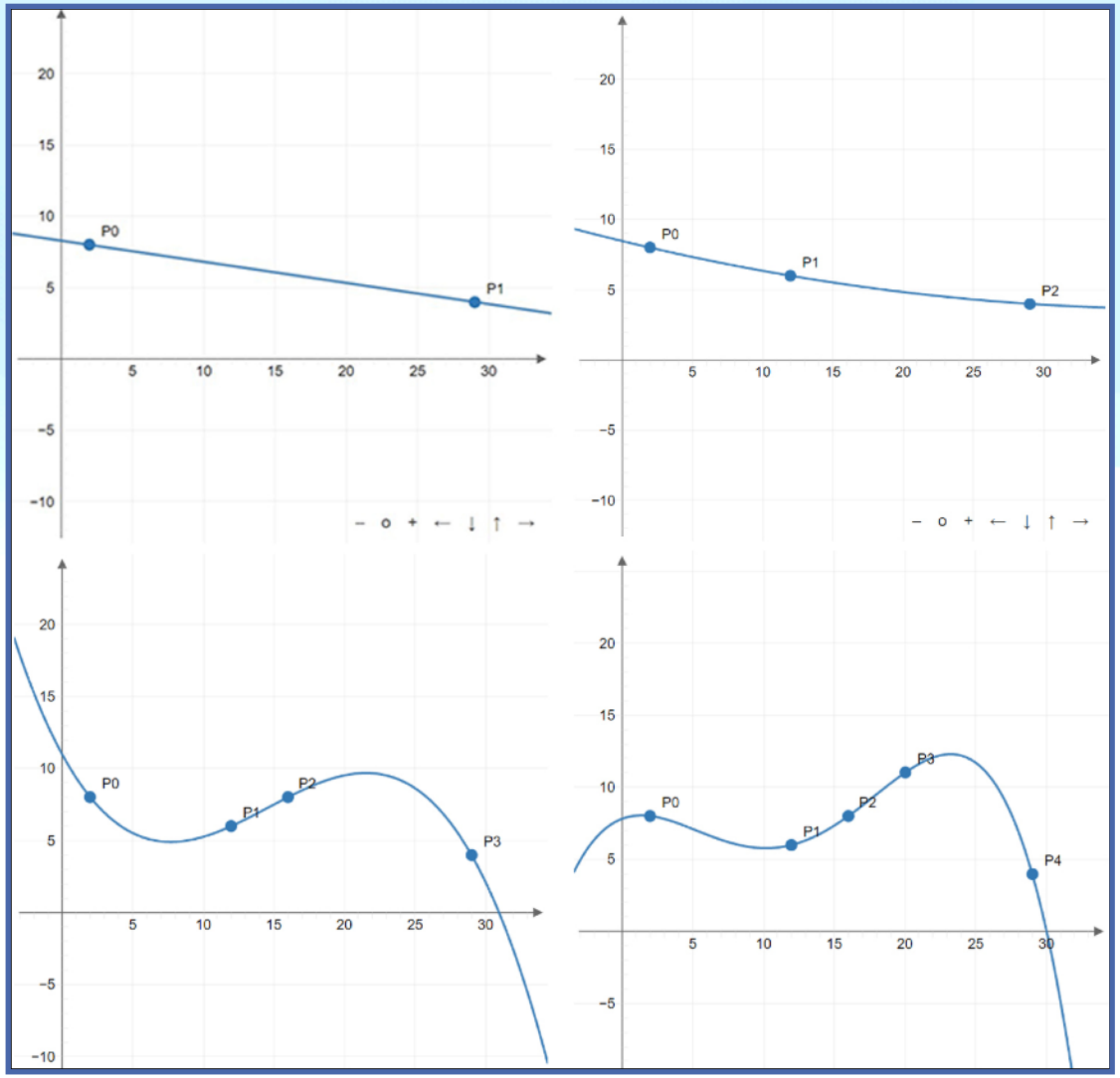
given any two points,
can interpolate and find S

(do arithmetic modulo large prime P)

Splitting Keys

Equation	Degree	Shape	Random parameters	Number of points (K) needed to recover S
$(S + RX) \bmod P$	1	Line	R	2
$(S + R_1X + R_2X^2) \bmod P$	2	Parabola	R_1, R_2	3
$(S + R_1X + R_2X^2 + R_3X^3) \bmod P$	3	Cubic	R_1, R_2, R_3	4

- a random polynomial curve of degree K-1 allows the secret to be reconstructed if, and only if, at least K of the points (“shares”) are available.
- There is a formula called Lagrange interpolation that allows you to reconstruct a polynomial of degree K-1 from any K points on its curve.



https://www.geeksforgeeks.org/implementing-shamirs-secret-sharing-scheme-in-python/

Secret sharing

Good: Store shares separately, adversary must compromise several shares to get the key.

Bad: To sign, need to bring shares together, reconstruct the key. ⇐ vulnerable

Multi-sig

We will see later multi-sig which can solve this problem.

Lets you keep shares apart, approve transaction without reconstructing key at any point.

Multi Signatures

- Instead of splitting a single key, Bitcoin script directly allows you to stipulate that control over an address be split between different keys.
- These keys can then be stored in different locations, and the signatures can be produced separately.
- The completed, signed transaction will be constructed on some device, but even if the adversary controls this device, all that he can do is prevent it from being broadcast to the network
- He can’t produce valid multi-signatures of some other transaction without the involvement of the other devices.

Arvind Narayanan et. al., (2016). Bitcoin and Cryptocurrency Technologies. p.111

Example

Andrew, Arvind, Ed, and Joseph are co-workers.
Their company has lots of Bitcoins.

Each of the four generates a key-pair,
puts secret key in a safe, private, offline place.

The company's cold-stored coins use multi-sig, so that three of the four keys must sign to release a coin.

Bitcoin scripts

Image by MichaelWuensch from Pixabay

CBS

Recap: Bitcoin transaction

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"
     }
    ]
}
```

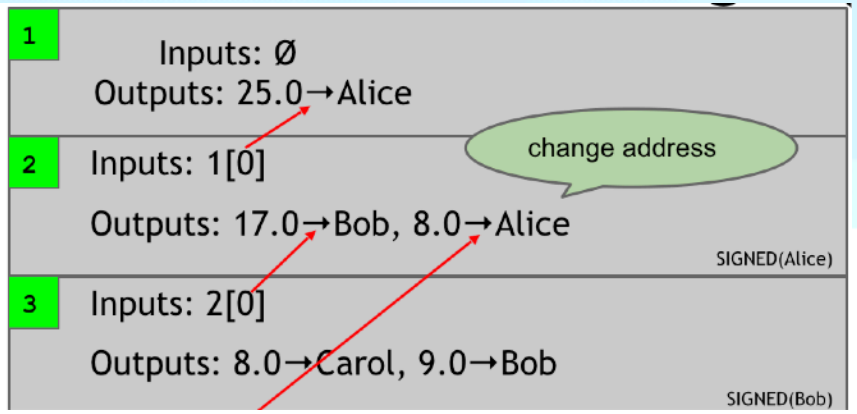
Recap: Peek at Transactions of a bitcoin block

https://blockchain.info/rawblock/00000000000000000333d265acedd64e126788ce876b46450023bddb4e6a06

https://blockchain.info/rawblock/00000000000000000333d265acedd64e126788ce876b46450023bddb4e6a06

Redeem a Previous Transaction

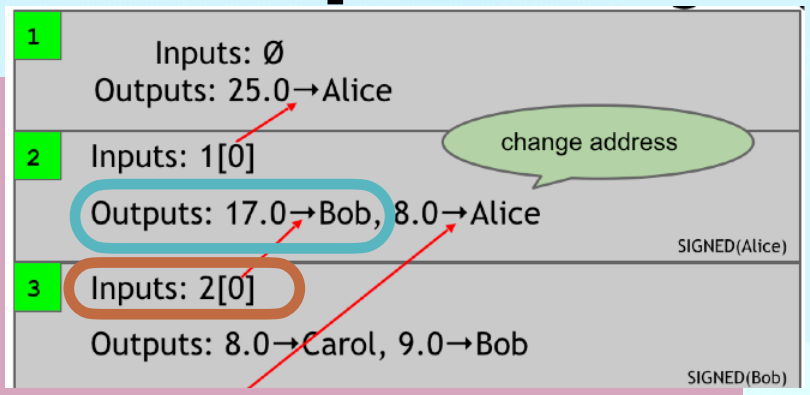
- The most common type of transaction in Bitcoin is to redeem a previous transaction output by signing with the correct key (aka Pay-to-Public-Key-Hash)
- In this case, we want the transaction output to say, “This can be redeemed by a signature from the owner of address X.”
- Recall that an address is a hash of a public key. Merely specifying the address X doesn’t tell us how to redeem this transaction (it may say what the public key is)
- This can be redeemed by a public key that hashes to X, along with a signature from the owner of that public key



Output “addresses” are really *scripts*

```
OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG
```

Input “addresses” are *also* scripts



scriptSig

unlocking script

30440220…
0467d2c9…

scriptPubKey

locking script

OP_DUP
OP_HASH160
69e02e18…
OP_EQUALVERIFY OP_CHECKSIG

TO VERIFY: Concatenated script must execute completely with no errors

addresses are really *scripts*

1AMDs5sXM-xLjqPhZsx
Pkscript
OP_DUP
OP_HASH160
668c307043452c8f6af9f042da546ab022f1fe5b
OP_EQUALVERIFY
OP_CHECKSIG

To (output)
Locking Script

1AMDs5sXM-xLjqPhZsx
Pkscript
OP_DUP
OP_HASH160
668c307043452c8f6af9f042da546ab022f1fe5b
OP_EQUALVERIFY
OP_CHECKSIG
Sigscript
493046022100de8ad3ea1c369ed22a9dacd329a662401113a991e1b71f
1ab090887d313584ed022100f3694f19e5ba86d1554d51f5982f44c8d0
bf78c3884d426c76d3d29deff0a16a0141048024e78901513df0115bac
6af24eaa4fe4b6530763a68b2b222e8e16fb66e676bba4c9ca9a9869a5
1cb7de2fb8af698bbbb3fc812985c5ab768071eab166328c
Witness

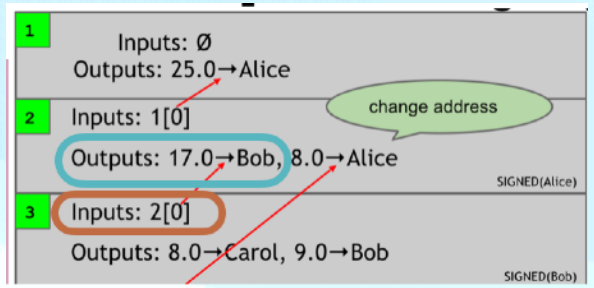
From (input)
Unlocking Script

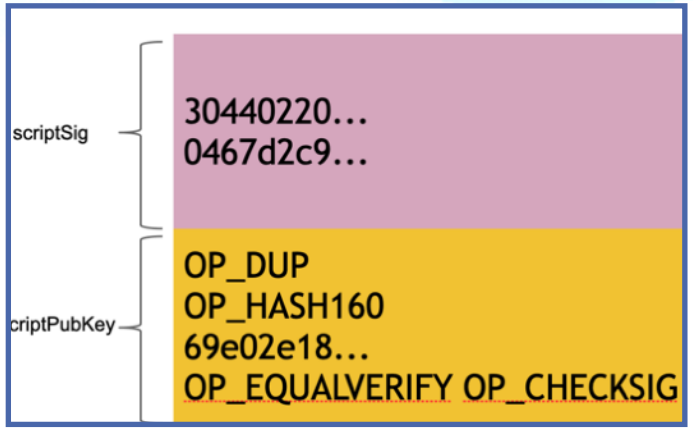
https://www.blockchain.com/explorer/transactions/btc/ec028b2dc22eb24dd5229619a33a6f1dd24a3521383a9673ac9f68421899836d

https://www.blockchain.com/explorer/transactions/btc/ca509df6924a5c738a7de96feed07c33a61c2db148692c73f05c1c419d287061

Pay-to-Public-Key-Hash

- But what happens to this script? Who runs it, and how exactly does this sequence of instructions enforce the above statement?
- The secret is that the inputs also contain scripts instead of signatures.
- We combine the new transaction's **input** script and the **earlier transaction's output script** to validate that a transaction redeems a previous transaction output correctly.
- We simply concatenate them, and the resulting script must run successfully in order for the transaction to be valid.





Arvind Narayanan et. al., (2016). Bitcoin and Cryptocurrency Technologies. p.111

Script Construction (Lock)

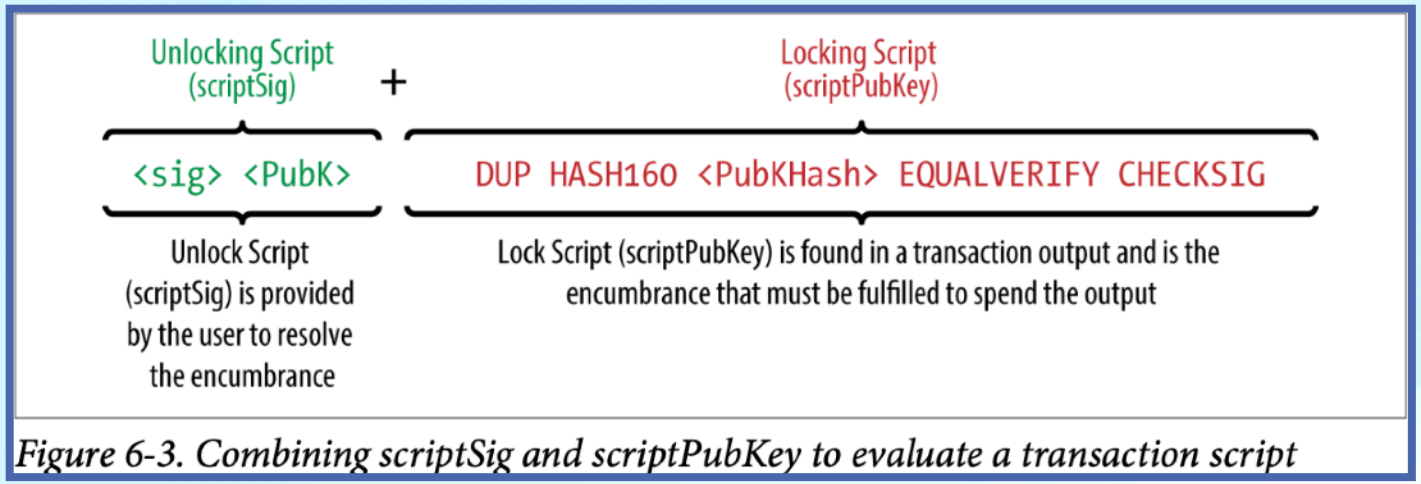
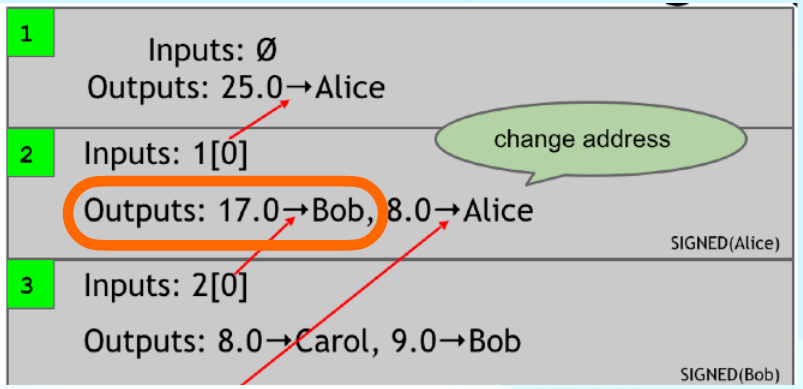


Figure 6-3. Combining scriptSig and scriptPubKey to evaluate a transaction script



- A locking script is a **spending condition/lock** placed on an output: it specifies the conditions that must be met to spend the output in the future.
- Locking script is called a scriptPubKey, as it contains a public key or bitcoin address (public key hash).
- In most bitcoin applications, locking script will appear in the source code as scriptPubKey. You will also see the locking script referred to as a witness script or generally as a *cryptographic puzzle*.

Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.". p.133

Script Construction (Unlock)

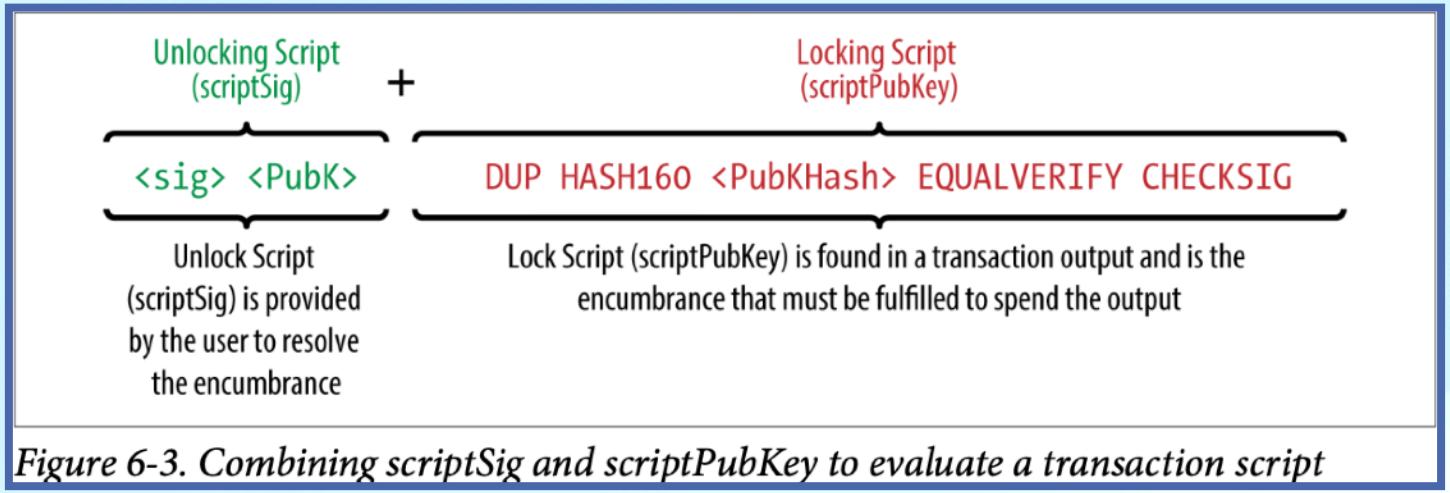
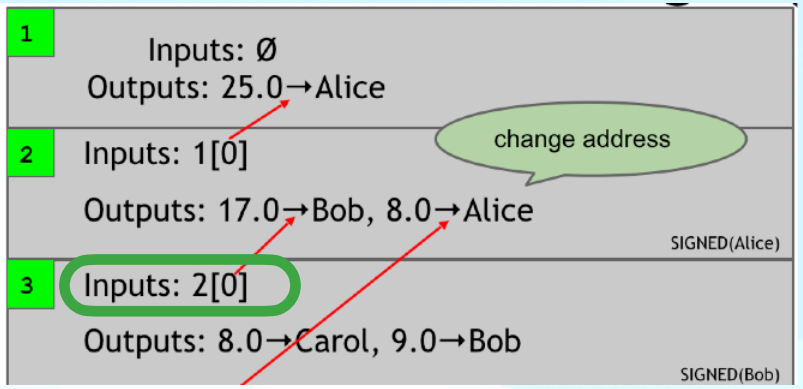


Figure 6-3. Combining scriptSig and scriptPubKey to evaluate a transaction script



- An unlocking script is a script that "**solves**," or **satisfies**, the conditions placed on an output by a locking script and allows the output to be spent and are part of every transaction input.
- The unlocking script was called scriptSig, because it usually contained a digital signature.
- "unlocking script" to acknowledge the much broader range of locking script requirements, because not all unlocking scripts must contain signatures.

Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.". p.133

Script Language Features

https://alexey-shepelev.medium.com/hierarchical-key-generation-fc27560f786

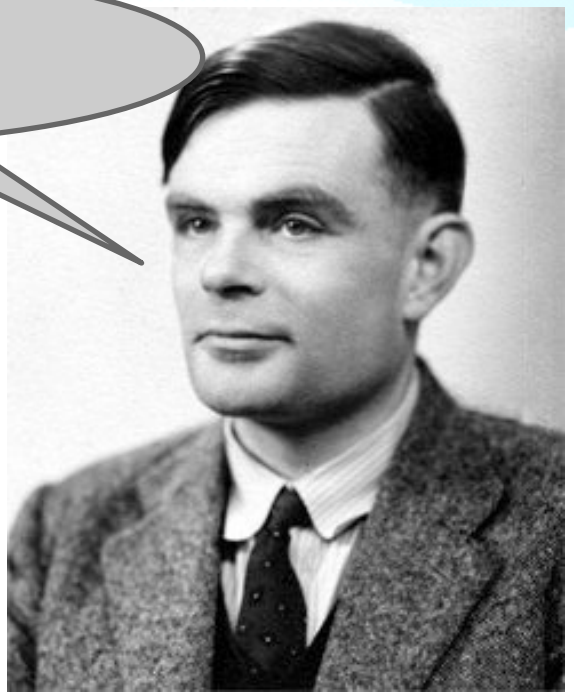
Slides based on
1) Bitcoin and Cryptocurrency Technologies: http://bitcoinbook.cs.princeton.edu/
2) Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.".

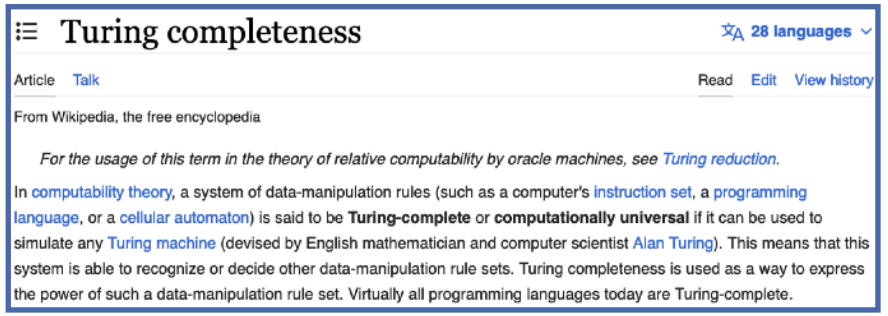
CBS

Bitcoin scripting language (“Script”)

Design goals

- Built for Bitcoin (inspired by Forth)
- Simple, compact
- Support for cryptography
- Stack-based
- Limits on time/memory
- No looping





Turing completeness

28 languages

Article Talk Read Edit View history

From Wikipedia, the free encyclopedia

For the usage of this term in the theory of relative computability by oracle machines, see Turing reduction.

In computability theory, a system of data-manipulation rules (such as a computer's instruction set, a programming language, or a cellular automaton) is said to be **Turing-complete** or **computationally universal** if it can be used to simulate any Turing machine (devised by English mathematician and computer scientist Alan Turing). This means that this system is able to recognize or decide other data-manipulation rule sets. Turing completeness is used as a way to express the power of such a data-manipulation rule set. Virtually all programming languages today are Turing-complete.

image via Jessie St. Amand

Bitcoin scripting language

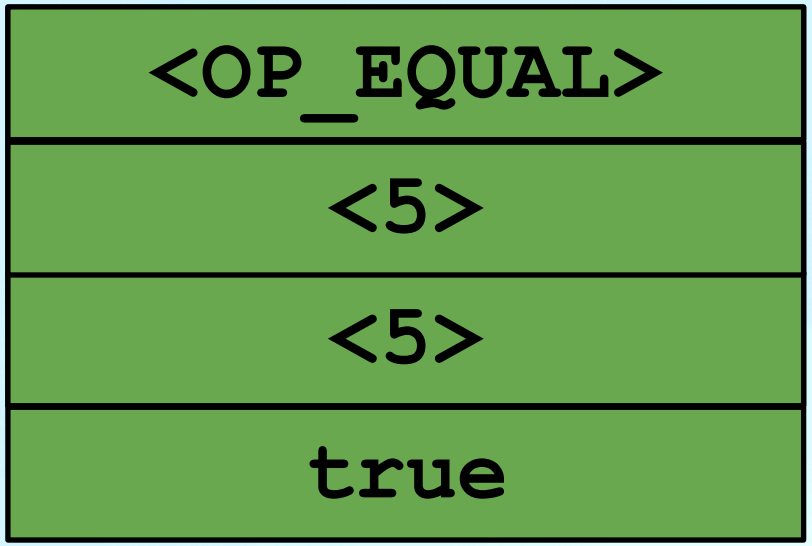
- Scripting is stack-based means every instruction is executed only once linearly.
- No looping constructs -> can be computed exactly how many steps
  - No infinite loops
- Not Turing complete mean not possible to encode powerful functions
- Possible outcomes: Success or Fail

Bitcoin scripting language

- Script was designed to be limited in scope and executable on a range of hardware, perhaps as simple as an embedded device.
- It requires minimal processing and cannot do many of the fancy things modern programming languages can do.
- This is a deliberate security feature for its use in validating programmable money.

A Simple Script

```
2 3 OP_ADD 5 OP_EQUAL
```



Use part of the arithmetic example script as the locking script:

```
3 OP_ADD 5 OP_EQUAL
```

which can be satisfied by a transaction containing an input with the unlocking script:

```
2
```

The validation software combines the locking and unlocking scripts and the resulting script is:

```
2 3 OP_ADD 5 OP_EQUAL
```

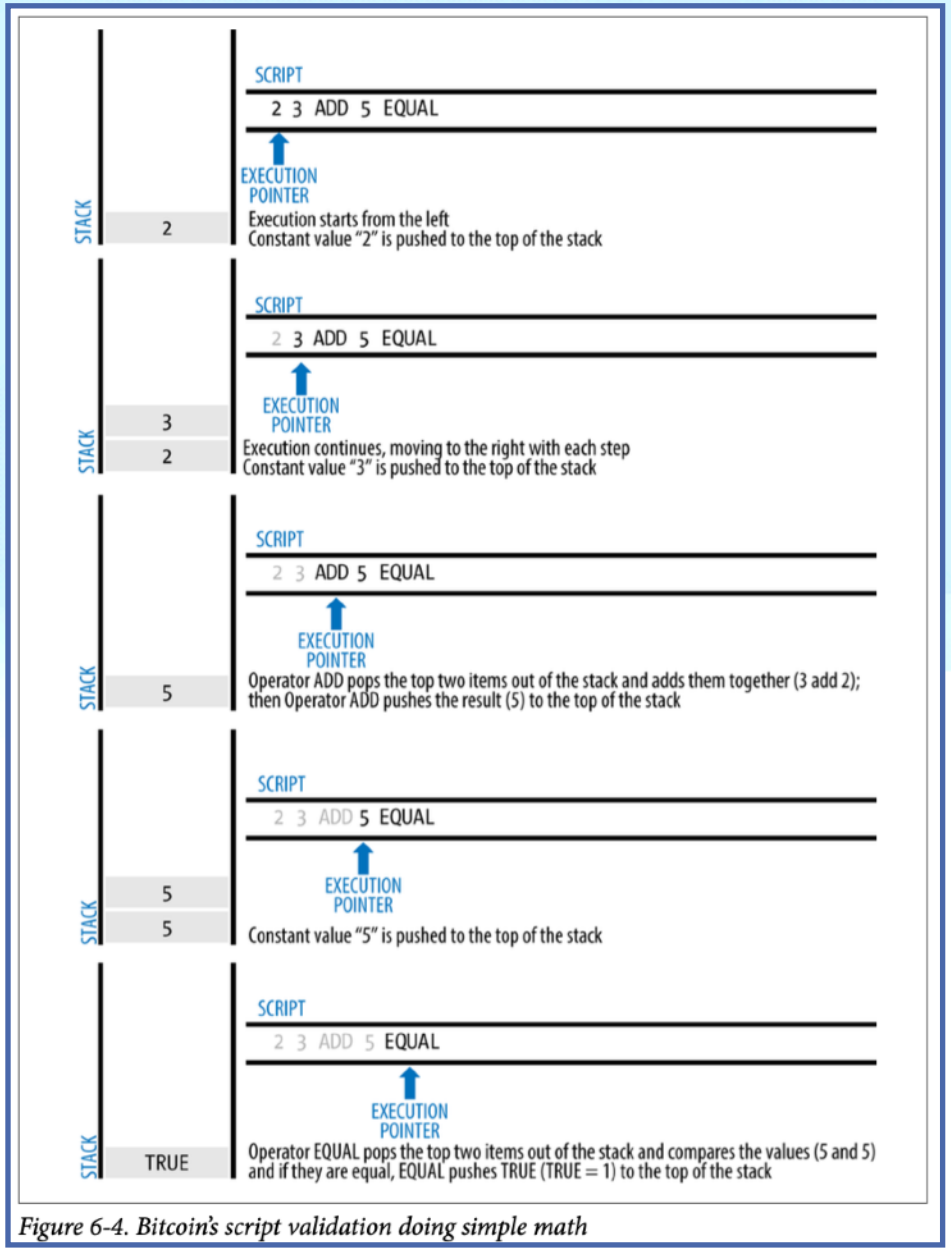
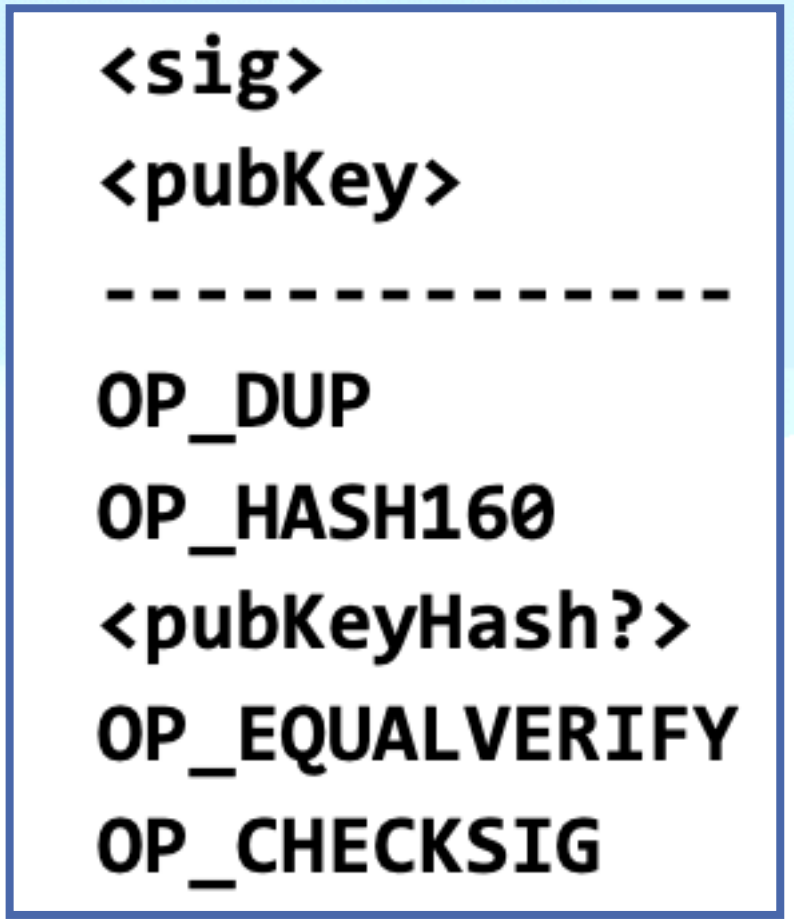


Figure 6-4. Bitcoin's script validation doing simple math

Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.". p.135

Redeem a Previous Transaction

- To check if a transaction correctly redeems an output, we create a combined script by appending the **scriptPubKey** of the referenced output transaction (**bottom**) to the **scriptSig** of the redeeming transaction (**top**).
- Notice that <pubKeyHash?> contains a '?'. We use this notation to indicate that we will later check to confirm that this is equal to the hash of the public key provided in the redeeming script.

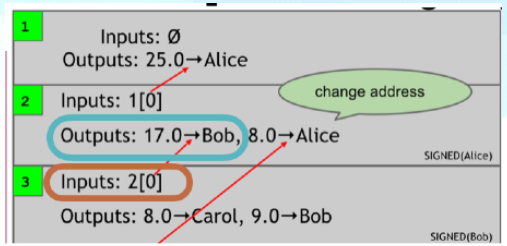


https://www.geeksforgeeks.org/implementing-shamirs-secret-sharing-scheme-in-python/

Bitcoin script execution example

<sig>
<pubKey>

OP_DUP
OP_HASH160
<pubKeyHash?>
OP_EQUALVERIFY
OP_CHECKSIG



<pubKeyHash?>
<pubKeyHash>
<pubKey_b>
true

<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash?> OP_EQUALVERIFY OP_CHECKSIG

Pay-to-Public-Key-Hash

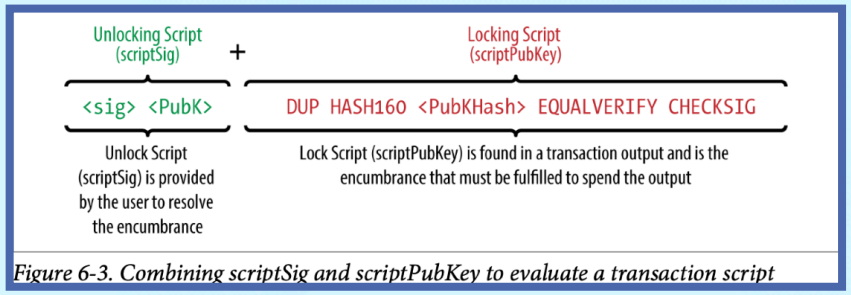


Figure 6-3. Combining scriptSig and scriptPubKey to evaluate a transaction script

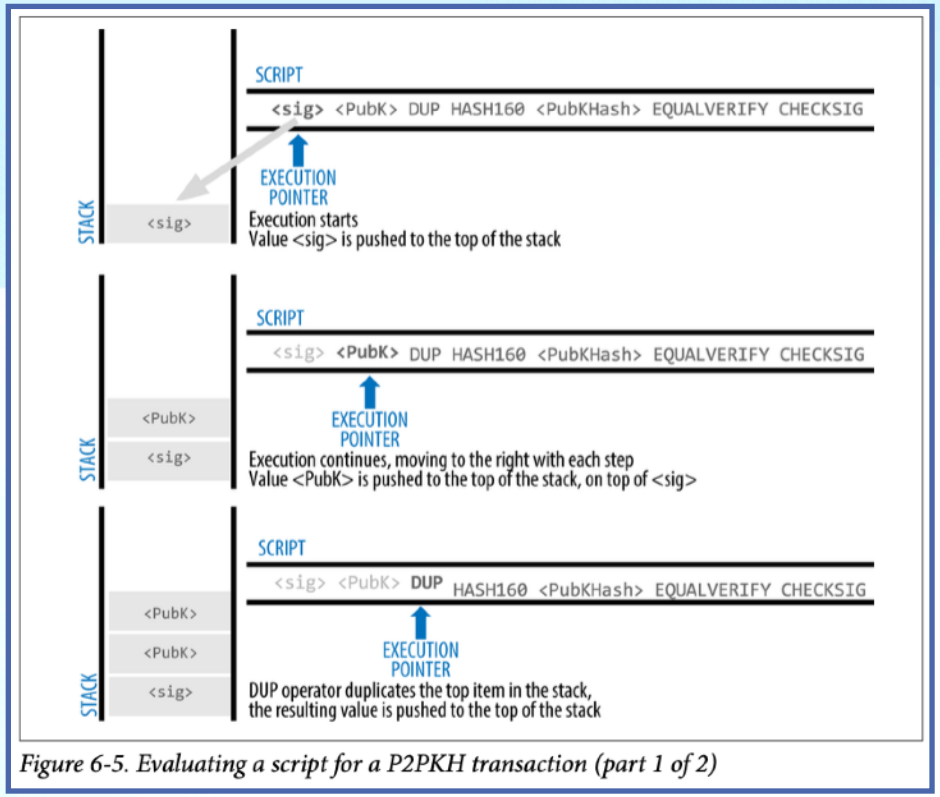


Figure 6-5. Evaluating a script for a P2PKH transaction (part 1 of 2)

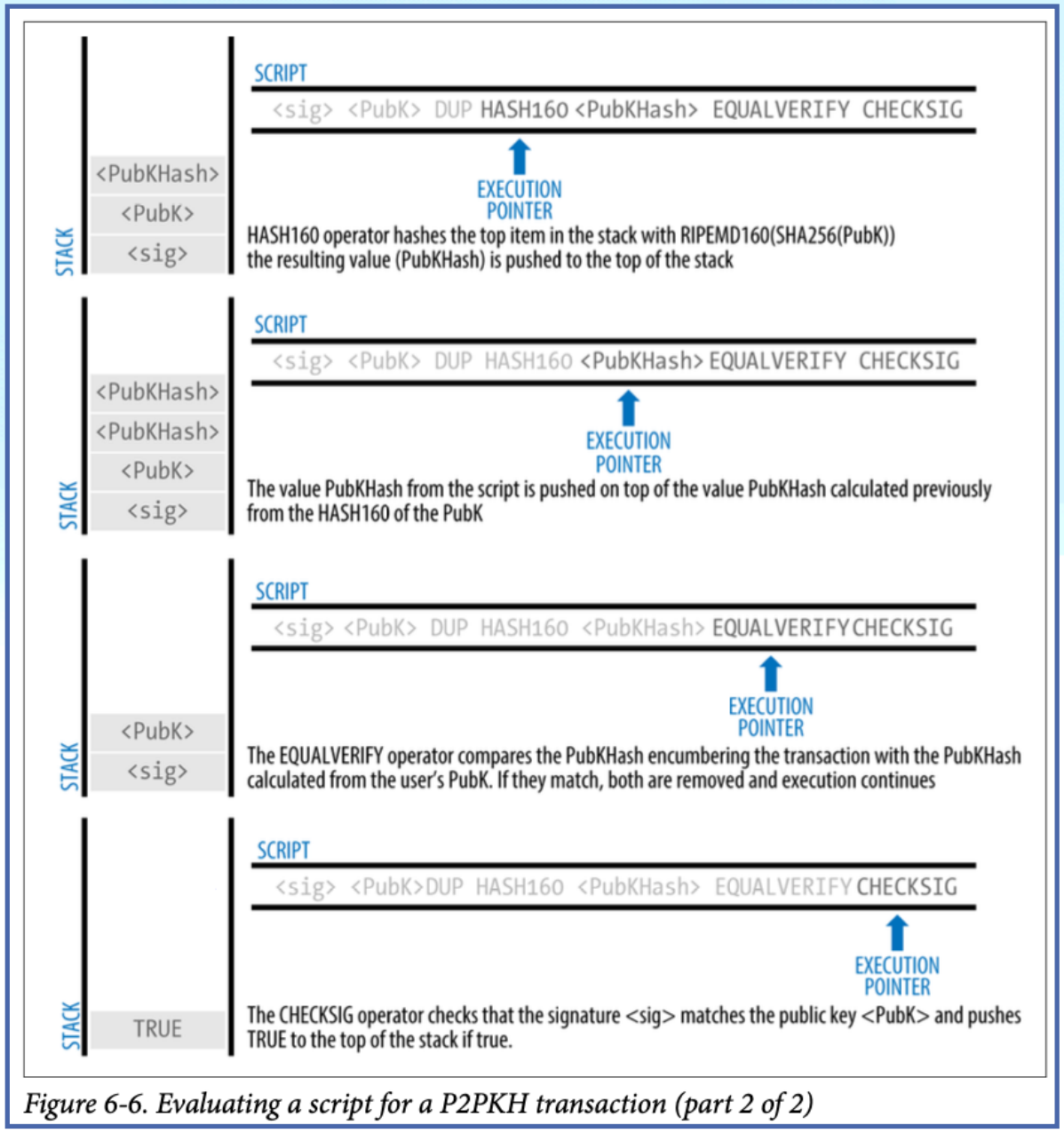


Figure 6-6. Evaluating a script for a P2PKH transaction (part 2 of 2)

Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.". p.135

Bitcoin script instructions

256 opcodes total (15 disabled, 75 reserved)

- Arithmetic
- If/then
- Logic/data handling
- Crypto!
  - Hashes
  - Signature verification
  - Multi-signature verification

Look into the full set of operations:
https://en.bitcoin.it/wiki/Script

Script Multisignatures

https://alexey-shepelev.medium.com/hierarchical-key-generation-fc27560f786

Slides based on
1) Bitcoin and Cryptocurrency Technologies: http://bitcoinbook.cs.princeton.edu/
2) Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.".

CBS

OP_CHECKMULTISIG

- Built-in support for joint signatures
- Specify *n* public keys
- Specify *t*
- Verification requires *t* signatures

BUG ALERT: Extra data value popped from the stack and ignored

Multisignatures

- Multisignature scripts set a condition where N public keys are recorded in the script and at least M of those must provide signatures to unlock the funds.
- Known as an M-of-N scheme, where N is the total keys and M is the threshold of signatures required for validation. e.g. a 2-of-3 is one where three public keys are listed as potential signers and at least two signatures needed to spend the funds.
- At this time, multisignature scripts are limited to at most 15 listed public keys, so 1-of-1 to a 15-of-15 multisignature or any combination within that range.

The general form of a locking script setting an M-of-N multisignature condition is:

```
M <Public Key 1> <Public Key 2> ... <Public Key N> N CHECKMULTISIG
```


Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.". p.149

Multisignatures (locking/unlocking)

A locking script setting a 2-of-3 multisignature condition looks like this:

```
2 <Public Key A> <Public Key B> <Public Key C> 3 CHECKMULTISIG
```

The preceding locking script can be satisfied with an unlocking script containing pairs of signatures and public keys:

```
<Signature B> <Signature C>
```

or any combination of two signatures from the private keys corresponding to the three listed public keys.

The two scripts together would form the combined validation script:

```
<Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3
CHECKMULTISIG
```

When executed, this combined script will evaluate to TRUE if, and only if, the unlocking script matches the conditions set by the locking script. In this case, the condition is whether the unlocking script has a valid signature from the two private keys that correspond to two of the three public keys set as an encumbrance.

Bug in Multisignatures

Because this bug became part of the consensus rules, it must now be replicated forever. Therefore the correct script validation would look like this:

```
0 <Signature B> <Signature C> 2 <Public Key A> <Public Key B> <Public Key C> 3
CHECKMULTISIG
```

Thus the unlocking script actually used in multisig is not:

```
<Signature B> <Signature C>
```

but instead it is:

```
0 <Signature B> <Signature C>
```

From now on, if you see a multisig unlocking script, you should expect to see an extra `0` in the beginning, whose only purpose is as a workaround to a bug that accidentally became a consensus rule.

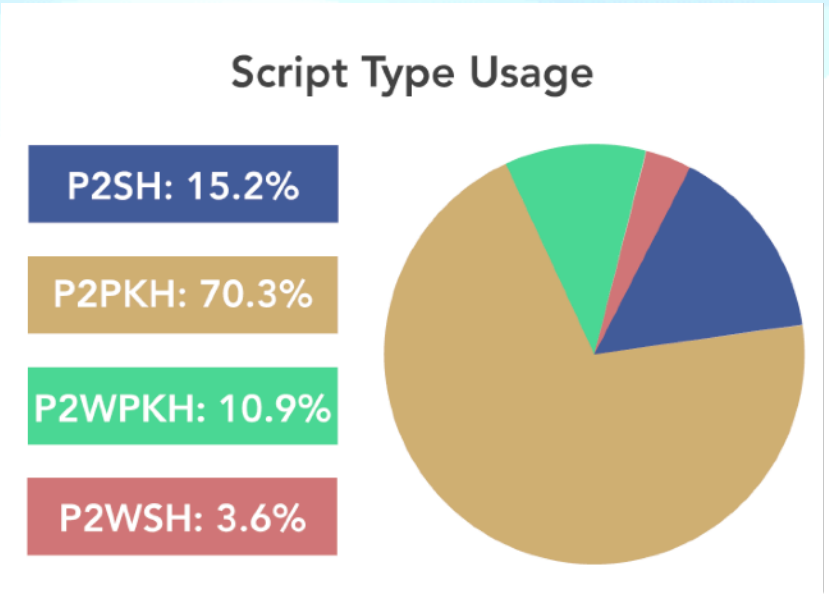
Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.". p.149

Bitcoin scripts in practice (as of 2014)

- Most nodes whitelist known scripts
- 99.9% are simple signature checks (P2PKH)
- ~0.01% are MUL
- ~0.01% are Pay-to-Script-Hash (P2SH)
- Remainder are errors, proof-of-burn

More on this soon

As of now



https://river.com/learn/terms/s/script-bitcoin/
https://river.com/learn/terms/p/p2wpkh/
https://river.com/learn/terms/p/p2wsh/

Proof-of-burn

nothing’s going to redeem that ☹

OP_RETURN
<arbitrary data>

OP_RETURN

- RETURN allows developers to add 80 bytes of nonpayment data to a transaction output.
- However, unlike the use of “fake” UTXO, the RETURN operator creates an explicitly provably unspendable output, which does not need to be stored in the UTXO set.
- Keep in mind that there is no “unlocking script” that corresponds to RETURN that could possibly be used to “spend” a RETURN output.
- The whole point of RETURN is that you can’t spend the money locked in that output, RETURN is provably unspendable
- any bitcoin assigned to such an output is effectively lost forever.

RETURN scripts look like this:

```
RETURN <data>
```


Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.”. p.155

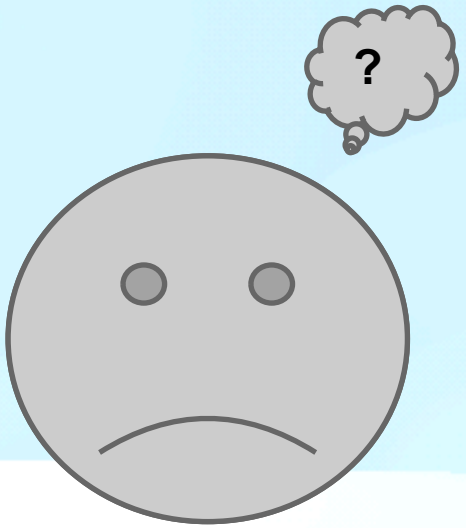
Script
Pay-to-Script-Hash

https://alexey-shepelev.medium.com/hierarchical-key-generation-fc27560f786

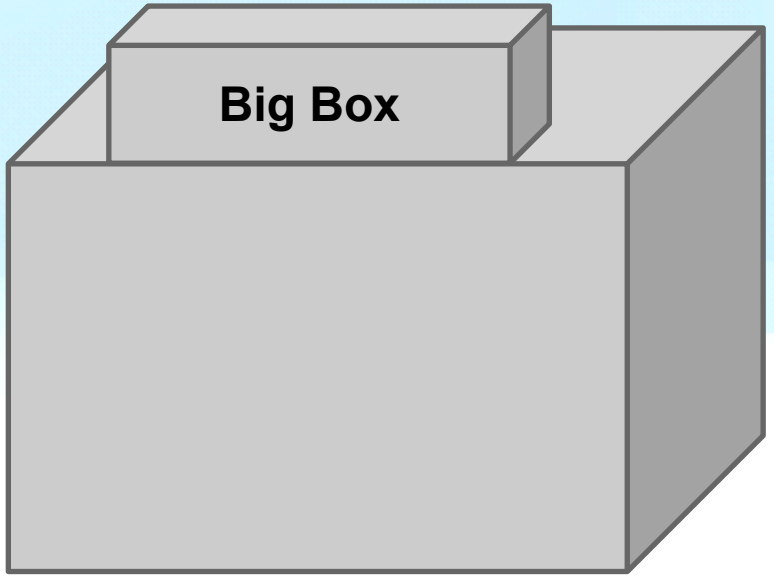
Slides based on
1) Bitcoin and Cryptocurrency Technologies: http://bitcoinbook.cs.princeton.edu/
2) Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.".

CBS

Should senders specify scripts?

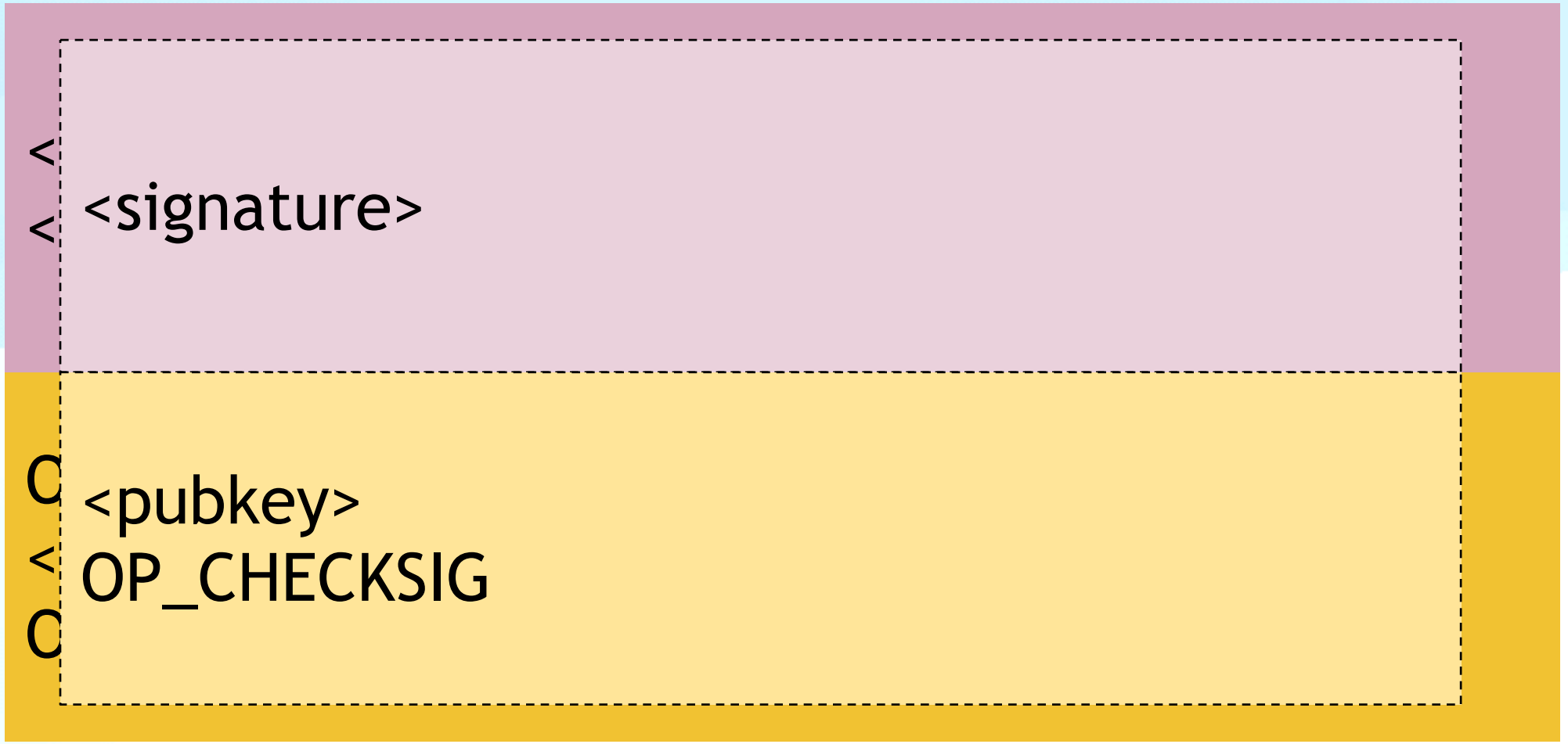


I'm ready to pay for my purchases!

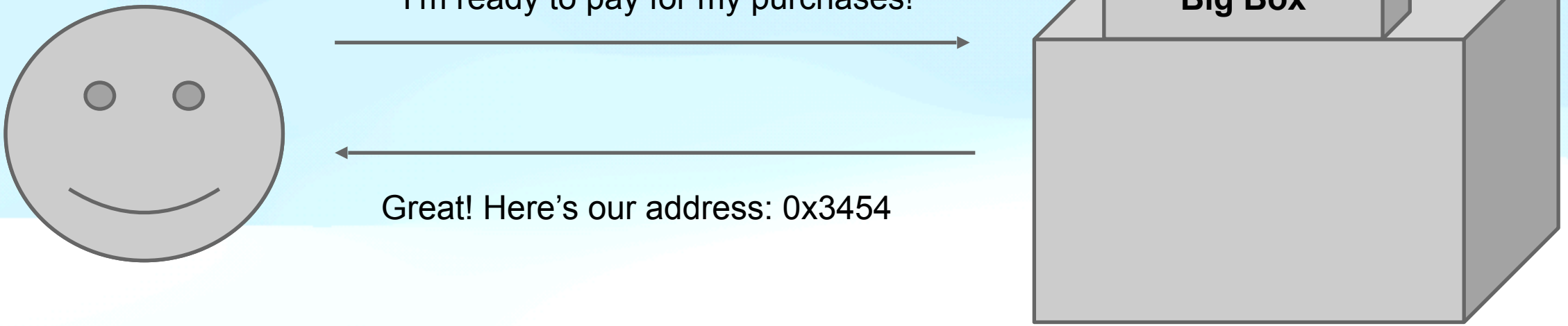


Cool! Well we're using MULTISIG now, so include a script requiring 2 of our 3 account managers to approve. Don't get any of those details wrong. Thanks for shopping at Big Box!

Idea: use the hash of redemption script



Pay to script hash



Pay-to-Script-Hash Example

- Mohammed's company uses bitcoin's multisignature feature extensively for its corporate accounts.
- With the multisignature scheme, any payments made by customers are locked in such a way that they require at least two signatures to release, from Mohammed and one of his partners or from his attorney who has a backup key.
- multisignature scheme like that offers corporate governance controls and protects against theft, embezzlement, or loss.
- Although multisignature scripts are a powerful feature, they are cumbersome to use.

The resulting script is quite long and looks like this:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3
Public Key> <Attorney Public Key> 5 CHECKMULTISIG
```


Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.". p.155

Pay-to-Script-Hash Example

The resulting script is quite long and looks like this:

```
2 <Mohammed's Public Key> <Partner1 Public Key> <Partner2 Public Key> <Partner3
Public Key> <Attorney Public Key> 5 CHECKMULTISIG
```

```
2
04C16B8698A9ABF84250A7C3EA7EEDEF9897D1C8C6ADF47F06CF73370D74DCCA01CDCA79DCC5C395
D7EEC6984D83F1F50C900A24DD47F569FD4193AF5DE762C58704A2192968D8655D6A935BEAF2CA23
E3FB87A3495E7AF308EDF08DAC3C1FCBFC2C75B4B0F4D0B1B70CD2423657738C0C2B1D5CE65C97D7
8D0E34224858008E8B49047E63248B75DB7379BE9CDA8CE5751D16485F431E46117B9D0C1837C9D5
737812F393DA7D4420D7E1A9162F0279CFC10F1E8E8F3020DECDBC3C0DD389D99779650421D65CBD
7149B255382ED7F78E946580657EE6FDA162A187543A9D85BAAA93A4AB3A8F044DADA618D0872274
40645ABE8A35DA8C5B73997AD343BE5C2AFD94A5043752580AFA1ECED3C68D446BCAB69AC0BA7DF5
0D56231BE0AABF1FDEEC78A6A45E394BA29A1EDF518C022DD618DA774D207D137AAB59E0B000EB7E
D238F4D800 5 CHECKMULTISIG
```


Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.". p.155

Pay-to-Script-Hash Example

```
54c557e07dde5bb6cb791c7a540e0a4796f5e97e
```

A P2SH transaction locks the output to this hash instead of the longer script, using the locking script:

```
HASH160 54c557e07dde5bb6cb791c7a540e0a4796f5e97e EQUAL
```

which, as you can see, is much shorter. Instead of "pay to this 5-key multisignature script," the P2SH equivalent transaction is "pay to a script with this hash." A customer making a payment to Mohammed's company need only include this much shorter locking script in his payment. When Mohammed and his partners want to spend this UTXO, they must present the original redeem script (the one whose hash locked the UTXO) and the signatures necessary to unlock it, like this:

```
<Sig1> <Sig2> <2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG>
```

The two scripts are combined in two stages. First, the redeem script is checked against the locking script to make sure the hash matches:

```
<2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG> HASH160 <redeem scriptHash> EQUAL
```

If the redeem script hash matches, the unlocking script is executed on its own, to unlock the redeem script:

```
<Sig1> <Sig2> 2 PK1 PK2 PK3 PK4 PK5 5 CHECKMULTISIG
```


Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.". p.155

Pay-to-Script-Hash Example

Benefits of P2SH

The P2SH feature offers the following benefits compared to the direct use of complex scripts in locking outputs:

- Complex scripts are replaced by shorter fingerprints in the transaction output, making the transaction smaller.
- Scripts can be coded as an address, so the sender and the sender's wallet don't need complex engineering to implement P2SH.
- P2SH shifts the burden of constructing the script to the recipient, not the sender.
- P2SH shifts the burden in data storage for the long script from the output (which is in the UTXO set) to the input (stored on the blockchain).
- P2SH shifts the burden in data storage for the long script from the present time (payment) to a future time (when it is spent).
- P2SH shifts the transaction fee cost of a long script from the sender to the recipient, who has to include the long redeem script to spend it.

Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.". p.155

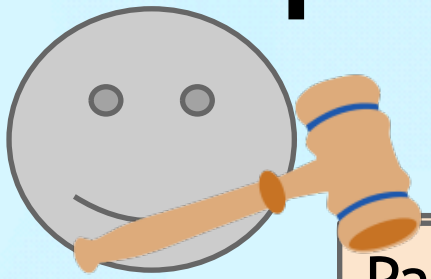
Applications of Bitcoin scripts

Image by MichaelWuensch from Pixabay

CBS

Example 1: Escrow transactions

(dispute case)



Judy

Pay *x* to Alice

SIGNED(ALICE, JUDY)

Alice

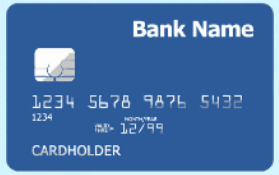
Bob

Pay *x* to 2-of-3 of Alice, Bob, Judy (MULTISIG)

SIGNED(ALICE)

Bob doesn't want to ship until after Alice pays.

Example 2: Green addresses

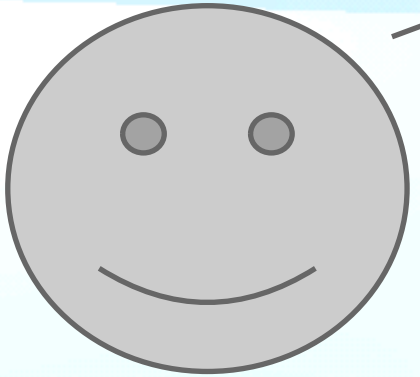


It's me, Alice! Could you make out a green payment to Bob?

Bank

004 days since last double spend!

Faraday cage



Alice

Pay x to Bob, y to Bank

No double spend

SIGNED(BANK)

Bob

PROBLEM: Alice wants to pay Bob.
Bob can't wait 6 verifications to guard against double-spends, or is offline completely.

lock_time


```
{
    "hash":"5a42590...b8b6b",
      "ver":1,
      "vin_sz":2,
      "vout_sz":1,
      "lock_time":315415,
      "size":404,
...
}
```


Block index or real-world timestamp before which this transaction can’t be published

Lock Time

Transaction Locktime (nLocktime)

From the beginning, bitcoin has had a transaction-level timelock feature. Transaction locktime is a transaction-level setting (a field in the transaction data structure) that defines the earliest time that a transaction is valid and can be relayed on the network or added to the blockchain. Locktime is also known as `nLocktime` from the variable name used in the Bitcoin Core codebase. It is set to zero in most transactions to indicate immediate propagation and execution. If `nLocktime` is nonzero and below 500 million, it is interpreted as a block height, meaning the transaction is not valid and is not relayed or included in the blockchain prior to the specified block height. If it is above 500 million, it is interpreted as a Unix Epoch timestamp (seconds since Jan-1-1970) and the transaction is not valid prior to the specified time. Transactions with `nLocktime` specifying a future block or time must be held by the originating system and transmitted to the bitcoin network only after they become valid. If a transaction is transmitted to the network before the specified `nLocktime`, the transaction will be rejected by the first node as invalid and will not be relayed to other nodes. The use of `nLocktime` is equivalent to postdating a paper check.

Antonopoulos, A. M. (2014). Mastering Bitcoin: unlocking digital cryptocurrencies. " O'Reilly Media, Inc.". p.155

More advanced scripts

- Multiplayer lotteries
- Hash pre-image challenges
- Coin-swapping protocols
  - Don’t miss the lecture on anonymity!

“Smart contracts”

Thank you!

Raghava Mukkamala

rrm.digi@cbs.dk

https://www.cbs.dk/staff/rrmdigi

https://raghavamukkamala.github.io/

https://cbsbda.github.io/