Blockchain and Cryptocurrencies

Week 4 — Chapter 3: Mechanics of Bitcoin

Prof. Dr. Peter Thiemann

Albert-Ludwigs-Universität Freiburg, Germany

SS 2020

Mechanics of Bitcoin

draws on material from

- Bitcoin and Cryptocurrency Technologies
- Bitcoin, Blockchain, and Cryptoassets
- Antonopoulos: Mastering Bitcoin

Contents

The Bitcoin Ledger

No Accounts

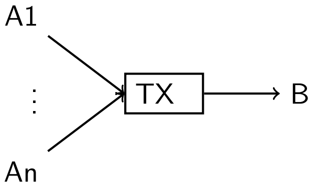
- balances constructed from the ledger
- (cf. Scroogecoin)
- for efficiency: full nodes maintain **UTXO**s (unspent transaction outputs)

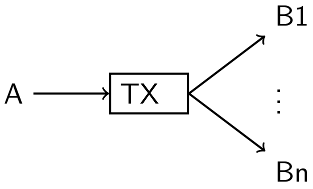
Example Ledger

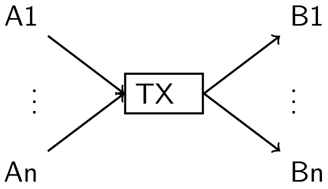
Output [Input]

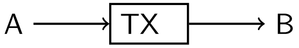
1	Inputs: $\emptyset$ Outputs: 6.25→Alice	
2	Inputs: 1[0] Outputs: 5.0→Bob, 1.25→Alice	signed(Alice)
3	Inputs: 2[0] Outputs: 3.0→Carol, 2.0→Bob	signed(Bob)
4	Inputs: 2[1] ⟶ Previous output. Outputs: 1.0→David, 0.25→Alice	signed(Alice)

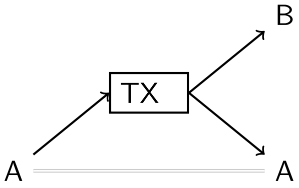
Possible Transactions











Transaction Summary

Many inputs, one output

- joint payment
- consolidating funds

One input, many outputs

Multiparty payment

Many input, many outputs

- Payment with fan-in and fan-out
- Combination of the above

Change address

Coins are immutable; change goes back to sender or another address controlled by sender

Transaction Contents, I

Metadata

```
{ "hash":"744cb999c4f052fb438c033f3a01adf6300797038719708a7e7679c11b5c0879",
  "ver": 1,
  "vin_sz":1,   input
  "vout_sz":2,  outputs.
  "lock_time": 0,
  "size": 380,
```

- unique identifier
- Bitcoin version
- nr of inputs and outputs
- lock time: do not accept before till [illegible]

Transaction Contents, II

Inputs

```
"in": [
  {"prev_out": {
     "hash": "bc1qwqdg6squsna38e46795at95yu9atm8azzmyvckulcc7kytlcckxswvvzej",
     "n": 0},
   "scriptSig": "30440..."
  }
],
```

- array of inputs
- each input refers to a previous output
- described by transaction hash and index of output
- signed by owner of that output

Transaction Contents, III

Outputs

```
"out": [
  {"value": 0.33696388,
   "scriptPubkey":
     "OP_HASH160 b0d3b60e930b0af317e781f835868b6e6692b638 OP_EQUAL"
  },
  {"value": 0.12840190,
   "scriptPubkey":
     "OP_0 701a8d401c84fb13e6baf169d59684e17abd9fa216c8cc5b9fc63d622ff8c58d"
  }
],
```

- array of outputs
- each output is value and target hash
- but the scriptPubkey field contains something special...

Contents

What's in a script?

- A transaction does **not** just contain public keys . . .

What’s in a script?

- A transaction does **not** just contain public keys . . .
- but rather several short programs in **script**

What's in a script?

- A transaction does **not** just contain public keys . . .
- but rather several short programs in **script**

Why script?

What’s in a script?

- A transaction does **not** just contain public keys . . .
- but rather several short programs in **script**

Why script?

- just having a pubkey hash is not sufficient

What's in a script?

- A transaction does **not** just contain public keys ...
- but rather several short programs in **script**

Why script?

- just having a pubkey hash is not sufficient
- need to express a statement like
  this output can be collected by a public key that hashes to some X along with a signature from the owner of that public key

What's in a script?

- A transaction does **not** just contain public keys . . .
- but rather several short programs in **script**

Why script?

- just having a pubkey hash is not sufficient
- need to express a statement like
  this output can be collected by a public key that hashes to some X along with a signature from the owner of that public key
- procedure may be different depending on the kind of transaction

How does (the) script work?

- each input contains `scriptSig`, an **unlocking script** that states a signature and public key for the input
- each output contains `scriptPubkey`, a **locking script** that checks the validity of the transaction
- for each input
  - the miner obtains the corresponding UTXO ?
  - runs the concatenation of the unlocking script of the input with the UTXO's locking script (conceptually)
  - stops the transaction if the script run fails (i.e., it does not return with final value True)
- the transaction is valid if all script runs are successful

Script

- Script is a stack-based language (like Forth or the JVM)
  - a script acts on a stack of values
  - each operation removes its arguments (0, 1, or more) from the top of the stack
  - leaves its results (0, 1, or more) on top of the stack
- operations are encoded by one-byte **op codes**
- **no loops**: intentionally **not** Turing complete
- special op codes for hashing and signing

Example Script

Script

2	3	OP_ADD	5	OP_EQUAL

Example Script

Script

2	3	OP_ADD	5	OP_EQUAL

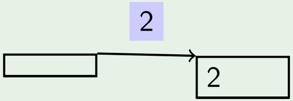
Execution

Example Script

Script

2	3	OP_ADD	5	OP_EQUAL

Execution

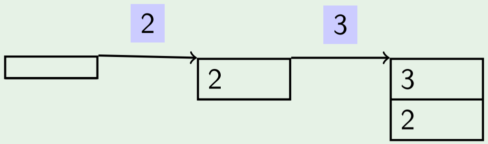


Example Script

Script

2	3	OP_ADD	5	OP_EQUAL

Execution

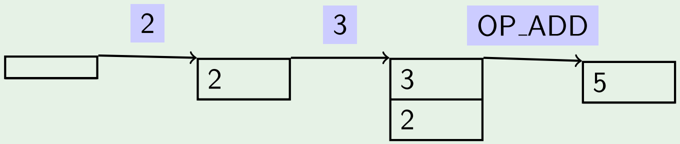


Example Script

Script

2	3	OP_ADD	5	OP_EQUAL

Execution

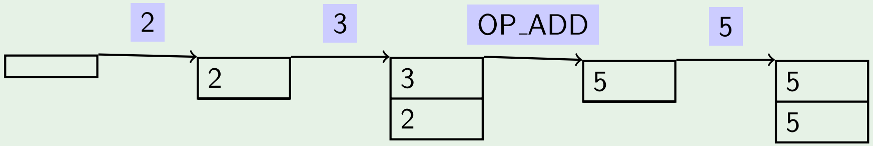


Example Script

Script

2	3	OP_ADD	5	OP_EQUAL

Execution

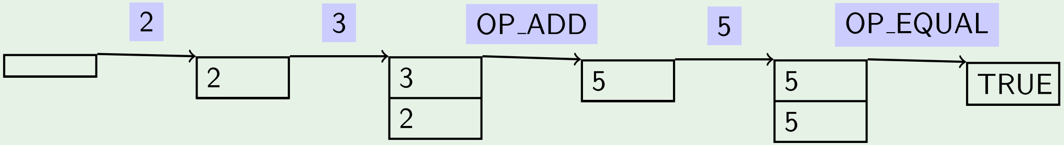


Example Script

Script

2	3	OP_ADD	5	OP_EQUAL

Execution



Common script op codes

Name	in	out	Function
OP_DUP	1	2	duplicate top of stack
OP_HASH160	1	1	apply SHA-256 to top of stack and then the RIPEMD-160 hash function
OP_EQUALVERIFY	2	0	continues if the top two items on stack are equal, otherwise stops marks transaction as invalid
OP_CHECKSIG	2	1	expects the public key and the (script's) signature on the stack; returns TRUE if the signature is valid and FALSE otherwise
OP_CHECKMULTISIG	$2 + m + n$	1	expects the number m followed by m keys and the number n followed by n signatures on the stack; returns TRUE if every signature can be validated with one of the keys

$n \leq m$

How to combine this?!

The most common scripts

Authoring scripts

- In principle, any logic can be implemented in a script
- In practice, miners do not accept arbitrary scripts, but they have a whitelist
- Best practice has narrowed down the scripts in use

The most common scripts

Authoring scripts

- In principle, any logic can be implemented in a script
- In practice, miners do not accept arbitrary scripts, but they have a whitelist
- Best practice has narrowed down the scripts in use

Five standard types of transaction scripts

- Pay-to-Public-Key-Hash (P2PKH),
- Pay-to-Public-Key,
- Multi-Signature (limited to 15 keys),
- Data Output (OP_RETURN), and
- Pay-to-Script-Hash (P2SH)

Contents

Pay-to-Public-Key-Hash

Whenever I want to use something

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

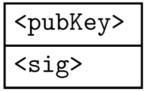
unlocking script from input `scriptSig`

---

locking script from UTXO's `scriptPubkey`

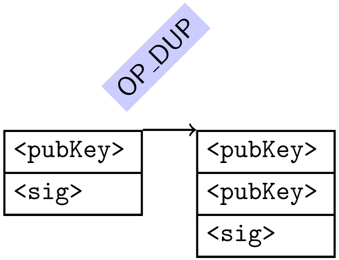
Operation of P2PKH

- Assume the unlocking script already completed
- the locking script starts on the stack left behind by the unlocking script



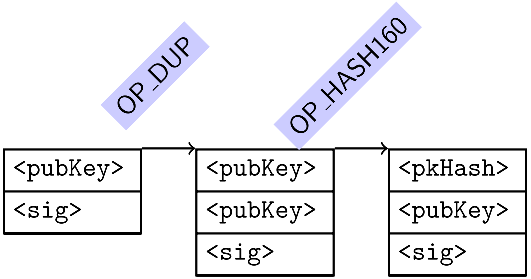
Operation of P2PKH

- Assume the unlocking script already completed
- the locking script starts on the stack left behind by the unlocking script



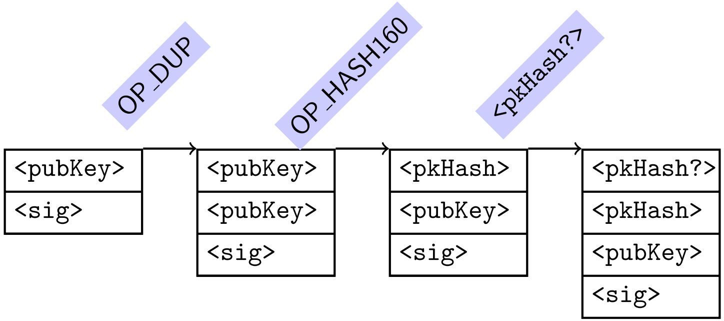
Operation of P2PKH

- Assume the unlocking script already completed
- the locking script starts on the stack left behind by the unlocking script



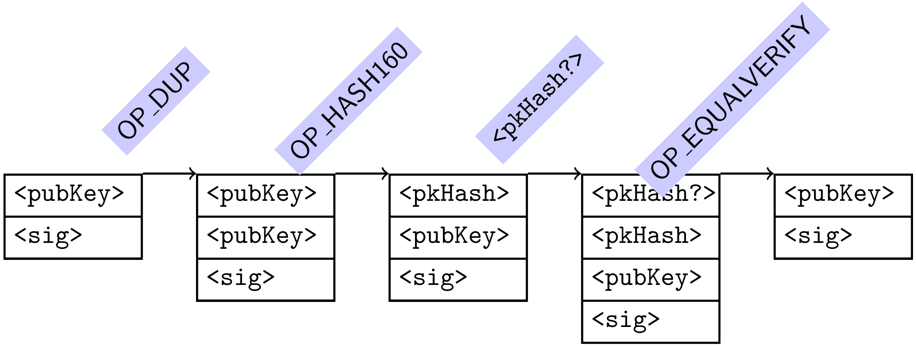
Operation of P2PKH

- Assume the unlocking script already completed
- the locking script starts on the stack left behind by the unlocking script



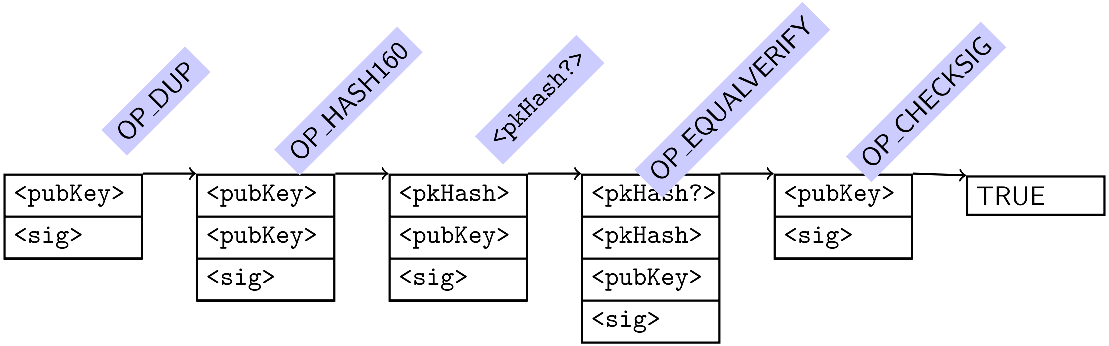
Operation of P2PKH

- Assume the unlocking script already completed
- the locking script starts on the stack left behind by the unlocking script



Operation of P2PKH

- Assume the unlocking script already completed
- the locking script starts on the stack left behind by the unlocking script



Contents

Pay-to-Public-Key

- Most simple form of payment
- Needs more space as P2PKH because the hash is shorter
- Still used in coinbase transactions by miners

Pay-to-Public-Key

- Most simple form of payment
- Needs more space as P2PKH because the hash is shorter
- Still used in coinbase transactions by miners

P2PK scripts

`<sig>`	unlocking script from input `scriptSig`
`<pubKey>` `OP_CHECKSIG`	locking script from UTXO's `scriptPubkey`

Contents

Multi-Signature

- used to redeem a payment where m out of n signers are sufficient
- consider an account jointly owned by $n \geq 2$ entities
  - if everyone is allowed individually: 1 out of n
  - if everyone must sign off: n out of n
  - if a majority is needed: $\lfloor n/2 \rfloor + 1$ out of n
- limited to $n \leq 15$ (as of June 2020)

Multi-Signature scripts for M out of N

Locking script

```
M <PubKey 1> <PubKey 2> ... <PubKey N> N OP_CHECKMULTISIG
```

Multi-Signature scripts for M out of N

Locking script

```
M <PubKey 1> <PubKey 2> ... <PubKey N> N OP_CHECKMULTISIG
```

Unlocking script

```
OP_0 <Sig 1> <Sig 2> ... <Sig M>
```

- `OP_0` is due to an implementation error that cannot be amended

Multi-Signature scripts for M out of N

Locking script

```
M <PubKey 1> <PubKey 2> ... <PubKey N> N OP_CHECKMULTISIG
```

Unlocking script

```
OP_0 <Sig 1> <Sig 2> ... <Sig M>
```

- `OP_0` is due to an implementation error that cannot be amended

Example N=3, M=2

- Locking: `2 <PubKey 1> <PubKey 2> <PubKey 3> 3 OP_CHECKMULTISIG`
- Unlocking: `OP_0 <Sig A> <Sig B>`

Contents

Data Output / Proof of Burn

Storing data on the blockchain (initial attempt)

- non-monetary use
- each output of a transaction defines a target address (hash)
- one could misuse this space as 20 bytes of storage

$\Rightarrow$ would create an unspendable UTXO as no corresponding private key exists

- inefficient as full nodes keep track of UTXOs

Data Output / Proof of Burn

Storing data on the blockchain (initial attempt)

- non-monetary use
- each output of a transaction defines a target address (hash)
- one could misuse this space as 20 bytes of storage

$\Rightarrow$ would create an unspendable UTXO as no corresponding private key exists

- inefficient as full nodes keep track of UTXOs

Storing data (best practice)

- create transaction output with 0 BTC and locking script OP_RETURN
- comes with 40 bytes of data storage
- usually a SHA-256 hash (32 bytes) plus some tag
- trying to spend this output $\rightarrow$ running OP_RETURN $\rightarrow$ script ends immediately $\rightarrow$ transaction invalid

Contents

Pay-to-Script-Hash (P2SH)

A use case of Multi-Signature

- A company wants to safeguard incoming payments
- Incoming payments can only be released by 2 out of 5 employees ($\Rightarrow$ Multi-Signature)

Pay-to-Script-Hash (P2SH)

A use case of Multi-Signature

- A company wants to safeguard incoming payments
- Incoming payments can only be released by 2 out of 5 employees ($\Rightarrow$ Multi-Signature)

Multi-Signature is very powerful but awkward to use

- customers (that send payments) need special locking scripts/ specialized software
- the transaction is about five times larger than a simple payment
- large UTXO puts burden on full nodes

Pay-to-Script-Hash (P2SH)

A use case of Multi-Signature

- A company wants to safeguard incoming payments
- Incoming payments can only be released by 2 out of 5 employees ($\Rightarrow$ Multi-Signature)

Multi-Signature is very powerful but awkward to use

- customers (that send payments) need special locking scripts/ specialized software
- the transaction is about five times larger than a simple payment
- large UTXO puts burden on full nodes

P2SH to the rescue

- replaces the complex locking script by its hash
- a subsequent transaction trying to spend the UTXO needs to present the script matching the hash (the redeem script) in addition to the unlocking script
- P2SH: "pay to a script matching this hash, corresponding to a script which will be presented later when this output is spent"

Example: 2 of 5 signatures to pay

Direct use of Multi-Signature

locking script: `2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 OP_CHECKMULTISIG`

unlocking script: `Sig1 Sig2`

Example: 2 of 5 signatures to pay

Direct use of Multi-Signature

locking script: `2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 OP_CHECKMULTISIG`

unlocking script: `Sig1 Sig2`

Using P2SH

locking script: `OP_HASH160 <20-byte hash of redeem script> OP_EQUALVERIFY`

unlocking script: `Sig1 Sig2 <redeem script>`

redeem script: `2 PubKey1 PubKey2 PubKey3 PubKey4 PubKey5 5 OP_CHECKMULTISIG`

To validate must run in sequence: unlocking, locking, redeem script

Properties of P2SH

- Complex scripts replaced by shorter fingerprints

Properties of P2SH

- Complex scripts replaced by shorter fingerprints
- Scripts can be encoded as an address for use with standard wallets

Properties of P2SH

- Complex scripts replaced by shorter fingerprints
- Scripts can be encoded as an address for use with standard wallets
- Recipient constructs the script

Properties of P2SH

- Complex scripts replaced by shorter fingerprints
- Scripts can be encoded as an address for use with standard wallets
- Recipient constructs the script
- Data no longer stored in UTXO

Properties of P2SH

- Complex scripts replaced by shorter fingerprints
- Scripts can be encoded as an address for use with standard wallets
- Recipient constructs the script
- Data no longer stored in UTXO
- Recipient carries transaction fees

Properties of P2SH

- Complex scripts replaced by shorter fingerprints
- Scripts can be encoded as an address for use with standard wallets
- Recipient constructs the script
- Data no longer stored in UTXO
- Recipient carries transaction fees

Warning

An invalid script hash will lock up the UTXO

Possible Redeem Scripts

- redeem scripts originally limited to standard transactions: P2PK, P2PKH or Multi-Sig (neither OP_RETURN nor P2SH)

Possible Redeem Scripts

- redeem scripts originally limited to standard transactions: P2PK, P2PKH or Multi-Sig (neither OP_RETURN nor P2SH)
- presently any valid script is allowed, except P2SH and OP_RETURN

Contents

Escrow Transactions

Scenario

Alice and Bob want to do business. Alice orders goods from Bob, but she only wants to pay once the goods are received in good order.

Escrow Transactions

Scenario

Alice and Bob want to do business. Alice orders goods from Bob, but she only wants to pay once the goods are received in good order.

Solution

- Alice and Bob set up a 2-of-3 Multi-Signature scheme with Conor, a mediator
- Alice makes her payment using Multi-Signature (more likely P2SH)
- If Alice and Bob agree, they can redeem the payment together (if fact, it is sufficient for Alice to send her signature to Bob)
- In case of a dispute, the mediator Conor needs to be convinced to either redeem the payment with Alice (transfer back) or Bob (for goods received).

Micropayments

Scenario

Lois is providing some service to Clark, which is charged by actual use (e.g., online time, cell phone use). Lois insists that charges are deducted as they occur, but having many transactions for small amounts is inefficient.

Micropayments

Scenario

Lois is providing some service to Clark, which is charged by actual use (e.g., online time, cell phone use). Lois insists that charges are deducted as they occur, but having many transactions for small amounts is inefficient.

Solution

- Clark sends a Multi-Signature (for Lois and Clark) transaction paying a deposit.
- Whenever Clark uses the service, he signs a transaction that pays the acrued charges to Lois and returns the rest to himself.
- He sends these transaction to Lois.
- Usually, Lois just keeps the last of these transactions.
- If Clark wants to quit, Lois co-signs the last transaction, submits it, and stops the service.

Lock Time

Scenario

In the previous Micropayments scenario, if Lois never co-signs any transaction, then the deposit is lost for Clark.

Lock Time

Scenario

In the previous Micropayments scenario, if Lois never co-signs any transaction, then the deposit is lost for Clark.

Solution

- Lois and Clark sign a transaction that returns the deposit to Clark.
- This transaction contains a **lock time**, which indicates the earliest time that the transaction can be submitted.
- This time (block number) serves as a deadline for Lois to cash the charges.
- If Lois charges in time, then the return transaction would be rejected as double spending.

Smart Contracts

- Even though Bitcoin script is quite limited, it already enables some interesting applications
- Other blockchains have taken this idea further, most prominently Ethereum
- Their scripting languages are more powerful (sometimes Turing complete)
- Scripts are often called **smart contracts**

Thanks!