Mechanics of Bitcoin

- Bitcoin Transactions
- Bitcoin Scripts
- Applications of Bitcoin Scripts
- Bitcoin Blocks
- The Bitcoin Network
- Limitations and Improvements
**An account-based Ledger (not Bitcoin)**

Create 25 coins and credit to Alice

Transfer 17 coins from Alice to Bob

Transfer 8 coins from Bob to Carol

Transfer 5 coins from Carol to Alice

Transfer 15 coins from Alice to David

**SIMPLIFICATION: only one transaction per block**

**A transaction-based Ledger (Bitcoin)**

1. Inputs: Ø
   Outputs: 25.0 → Alice
2. Inputs: 1[0]
   Outputs: 17.0 → Bob, 8.0 → Alice
3. Inputs: 2[0]
   Outputs: 10.0 → Carol, 7.0 → Bob
4. Inputs: 2[1]
   Outputs: 6.0 → David, 2.0 → Alice

**SIMPLIFICATION: only one transaction per block**
Merging Value

1. Inputs: ...
   Outputs: 17.0→Bob, 8.0→Alice
   \[\text{SIGNED(Alice)}\]

2. Inputs: 1[1]
   Outputs: 6.0→Carol, 2.0→Bob
   \[\text{SIGNED(Alice)}\]

3. Inputs: 1[0], 2[1]
   Outputs: 19.0→Bob
   \[\text{SIGNED(Bob)}\]

Simplification: only one transaction per block

Joint Payments

1. Inputs: ...
   Outputs: 17.0→Bob, 8.0→Alice
   \[\text{SIGNED(Alice)}\]

2. Inputs: 1[1]
   Outputs: 6.0→Carol, 2.0→Bob
   \[\text{SIGNED(Alice)}\]

3. Inputs: 2[0], 2[1]
   Outputs: 8.0→David
   \[\text{SIGNED(Carol), SIGNED(Bob)}\]

Two signatures!

Simplification: only one transaction per block
The Real Deal: a Bitcoin Transaction

```json
{  "hash":"5a42590fbe0a90ee8e8747244d6c8450db1a3a24e8f1b95b10c3e0509990fb860",  "ver":1,  "vin_sz":2,  "vout_sz":1,  "lock_time":0,  "size":404,  "in":{   "prev_out":{      "hash":"3be4ac9728a0823cf5e20eb2e88f25cb2a503a91d30742b7a7617df79280260",      "n":0    },    "scriptSig":"30440..."  },  "out":{   "value":"10.12287097",    "scriptPubKey":"OP_DUP OP_HASH160 69e02e18b5705a05dd6b28ed517716c894b3d42e OP_EQUALVERIFY OP_CHECKSIG"  }}
```

The Real Deal: Transaction Metadata

```json
{  "housekeeping":{  "transaction_hash":null,  "housekeeping":null,  "not_valid_before":null,  "size":404,  "lock_time":0,  "vin_sz":2,  "vout_sz":1,  "ver":1,  "hash":"5a42590fb8b6b",  more on lock_time later...  }}
```
Cryptocurrency Technologies

Mechanics of Bitcoin

The Real Deal: Transaction Inputs

```
"in": [
  {
    "prev_out": {
      "hash": "3be4...80260",
      "n": 0
    },
    "scriptSig": "30440....3f3a4ce81"
  },
  ...
],
```

previous transaction

signature

(more inputs)

The Real Deal: Transaction Outputs

```
"out": [
  {
    "value": "10.12287097",
    "scriptPubKey": "OP_DUP OP_HASH160 69e...3d42e
    OP_EQUALVERIFY OP_CHECKSIG"
  },
  ...
]
```

output value

recipient address??

(more outputs)

more on this soon...
Mechanics of Bitcoin

- Bitcoin Transactions
- **Bitcoin Scripts**
  - Applications of Bitcoin Scripts
  - Bitcoin Blocks
  - The Bitcoin Network
  - Limitations and Improvements

---

Output “Addresses” are really **Scripts**

```plaintext
OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG
```
Input “Addresses” are also Scripts

Sig-script

30440220...
0467d2e9...

PubKey-script

OP_DUP
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG

TO VERIFY: Concatenated script must execute completely with no errors

Why Scripts?!

Redeem previous transaction by signing with correct key

“This can be redeemed by a signature from the owner of address X”

Recall: address X is hash of public key

What is public key associated with X?!

“This can be redeemed by a public key that hashes to X, along with a signature from the owner of that public key”
Cryptocurrency Technologies

Mechanics of Bitcoin

Bitcoin Scripting Language ("Script")

Design “goals“:
- Built for Bitcoin (inspired by Forth)
- Simple, compact
- Stack-based
- No looping
- Support for cryptography
- Limits on time/memory
- Not Turing complete!

I am not impressed

Bitcoin Script Execution Example

30440220...
0467d2c9...
<
OP_DUP
<
OP_HASH160
69e02e18...
OP_EQUALVERIFY OP_CHECKSIG

<sig> <pubKey> OP_DUP OP_HASH160 <pubKeyHash?> OP_EQUALVERIFY OP_CHECKSIG
256 opcodes total (15 disabled, 75 reserved)
- Arithmetic
- If/then
- Logic/data handling
- Crypto!

<table>
<thead>
<tr>
<th>OP</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OP_DUP</td>
<td>Duplicates the top item on the stack</td>
</tr>
<tr>
<td>OP_HASH160</td>
<td>Hashes twice: first using SHA-256 and then RIPEMD-160</td>
</tr>
<tr>
<td>OP_EQUALVERIFY</td>
<td>Returns true if the inputs are equal. Returns false and marks the transaction as invalid if they are unequal</td>
</tr>
<tr>
<td>OP_CHECKSIG</td>
<td>Checks that the input signature is a valid signature using the input public key for the hash of the current transaction</td>
</tr>
<tr>
<td>OP_CHECKMULTISIG</td>
<td>Checks that the k signatures on the transaction are valid signatures from k of the specified public keys.</td>
</tr>
</tbody>
</table>

**OP_CHECKMULTISIG**

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

Incidentally: There is a bug in the multisig implementation. Extra data value popped from the stack and ignored.
Scripts in Practice (as of 2015)

**Theory:** Scripts let us specify *arbitrary conditions* that must be satisfied to spend coins.

Q: Is any of this used in practice?

- 99.9% are simple signature checks
- ~0.01% are **MULTISIG**
- ~0.01% are **Pay-to-Script-Hash**
- Remainder are errors, proof-of-burn

Most nodes *whitelist* known scripts

---

Proof-of-Burn

this script can never be redeemed 😞

```plaintext
OP_RETURN
<arbitrary data>
```

Uses for Proof-of-Burn:
- *Destroy coins* and transfer them to alternative currency
- *Add arbitrary data* to block chain
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!

Pay-to-Script-Hash (P2SH) Workflow

Bob
- creates a redeem script with whatever script he wants
- hashes the redeem script
- sends redeem script hash to Alice.

Alice
- creates a P2SH-style output containing Bob’s redeem script hash.

When Bob wants to spend the output
- provides his signature along with the redeem script in the signature script.

P2P network then
- ensures the redeem script hashes to the same value as the script hash Alice put in her output;
- it then processes the redeem script exactly as it would if it were the primary pubkey script,
- letting Bob spend the output if the redeem script does not return false.
Solution: Use the Hash of Redemption Script

I'm ready to pay for my purchases!

Great! Here's our address: 0x3454
Mechanics of Bitcoin

- Bitcoin Transactions
- Bitcoin Scripts
- Applications of Bitcoin Scripts
- Bitcoin Blocks
- The Bitcoin Network
- Limitations and Improvements

Example 1: Escrow Transactions

PROBLEM: Alice wants to buy online from Bob. Alice doesn't want to pay until after Bob ships. Bob doesn't want to ship until after Alice pays.

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

To: Alice
From: Bob

SIGNED(ALICE)

SIGNED(ALICE, BOB)

SIGNED(ALICE, JUDY)
Example 2: Green Addresses

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

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

Pay x to Bob, y to Bank

No double spend

SIGNED(BANK)

Example 3: Efficient Micro-Payments

PROBLEM: Alice wants to pay Bob for each minute of phone service. She doesn’t want to incur a transaction fee every minute.

Input: x; Pay 42 to Bob, 58 to Alice

SIGNED(ALICE) SIGNED(BOB)

I’m done!

I’ll publish!

all of these could be double-spends!

...
**Example 3: Efficient Micro-Payments**

What if Bob never signs??

Input: \( x \); Pay 42 to Bob, 58 to Alice

Alice demands a **timed refund transaction** before starting

Input: \( x \); Pay 100 to Bob/Alice (MULTISIG)

What if Bob never signs??

Input: \( y \); Pay 100 to Alice, LOCK until time \( t \)

Alice demands a **timed refund transaction** before starting

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

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

Mechanics of Bitcoin

More advanced Scripts

- Multiplayer Lotteries
- Coin-swapping Protocols

“Smart Contracts”

Mechanics of Bitcoin

- Bitcoin Transactions
- Bitcoin Scripts
- Applications of Bitcoin Scripts
- Bitcoin Blocks
- The Bitcoin Network
- Limitations and Improvements
1. Requiring consensus for each transaction separately would reduce transaction acceptance rate.
2. Hash-chain of blocks is much shorter.
3. Faster to verify history.

Q: Why bundle transactions together?
The Real Deal: a Bitcoin Block

The Real Deal: a Bitcoin Block Header
**coinbase Transaction**

New coins are created with `coinbase` transaction:

- **Single input** and **single output**
- Does not redeem previous output
  - Hash pointer is null
- **Output value** is miner’s **revenue** from block:
  - output value = mining reward + transaction fees
  - transaction fees come from all transactions in block
- Special `coinbase` parameter
  - contains arbitrary value

---

**The Real Deal: coinbase Transaction**

```
"in": [
  {
    "prev_out": {
      "hash": "000000.....0000000",
      "n": 4294967295
    },
    "coinbase": "..."
  }
],
"out": [
  {
    "value": "25.03371419",
    "scriptPubKey": "OPDUP OPHASH160 ...
  }
]
```

Null hash pointer

First ever coinbase parameter: "The Times 03/Jan/2009 Chancellor on brink of second bailout for banks"

block reward + transaction fees
Mechanics of Bitcoin

- Bitcoin Transactions
- Bitcoin Scripts
- Applications of Bitcoin Scripts
- Bitcoin Blocks
- The Bitcoin Network
- Limitations and Improvements

Bitcoin P2P Network

Participants can
- publish transactions
- insert transactions into block chain

The network:
- Ad-hoc protocol (runs on TCP port 8333)
- Ad-hoc network with random topology
- All nodes are equal
- New nodes can join at any time
- Forget non-responding nodes after 3 hr
Joining the Bitcoin P2P Network

Hello World! I'm ready to Bitcoin!

getaddr()

Transaction Propagation (Flooding)

Already heard that!
Should I relay a proposed Transaction?

- Transaction valid with current block chain
- (default) script matches a whitelist
  - Avoid unusual scripts
- Haven’t seen before
  - Avoid infinite loops
- Doesn’t conflict with others I’ve relayed
  - Avoid double-spends

Nodes may differ on Transaction Pool

Sanity checks only... Some nodes may ignore them!
## Race Conditions

Transactions or blocks may **conflict**
- This is called **“race condition”**
- **Default behavior**: accept what you hear first
- Tie broken by whoever mines next block
  - picks only one transaction/block
- **Network position** matters
- Miners may implement other logic!

## Orphaned Blocks

![Number Of Orphaned Blocks](chart.png)

*The total number of blocks mined but ultimately not attached to the main Bitcoin blockchain.*

*Source: blockchain.info*
Block Propagation

Propagation of blocks is nearly identical:

Relay a new block when you hear it if:
1. Block meets the hash target
2. Block has all valid transactions
   - Run all scripts, even if you wouldn’t relay
3. Block builds on current longest chain
   - Avoid forks

Latency of Flooding Algorithm

Source: Yonatan Sompolinsky and Aviv Zohar: “Accelerating Bitcoin’s Transaction Processing” 2014
Size of the Network

Q: How big is the Network?

Impossible to measure exactly
- Estimates-up to 1M IP addresses/month
- Only about 5-10k "full nodes"
  - Permanently connected
  - Fully-validating
- This number may be dropping!

Fully-validating Nodes:
- Permanently connected
- Store entire block chain
- Hear and forward every node/transaction

Storage Costs

Blockchain Size
The total size of all block headers and transactions. Not including database indexes.
Source: blockchain.info

[Graph showing the growth of blockchain size from 2008 to 2017]
Unspent Transaction Output fits in RAM

Thin/SPV Clients (not fully-validating)

Idea: don’t store everything
- Store block headers only
  Request transactions as needed
  - To verify incoming payment
Trust fully-validating nodes

1000x cost savings!
Software Diversity

• About 90% of nodes run “Core Bitcoin” (C++)
  - Some are out of date versions
• Other implementations running successfully
  - BitcoinJ (Java)
  - Libbitcoin (C++)
  - btcd (Go)
• “Original Satoshi client”

Mechanics of Bitcoin

• Bitcoin Transactions
• Bitcoin Scripts
• Applications of Bitcoin Scripts
• Bitcoin Blocks
• The Bitcoin Network
• Limitations and Improvements
Cryptocurrency Technologies

Mechanics of Bitcoin

Hard-coded Limits in Bitcoin

- 10 min. average creation time per block
- 1 M bytes in a block
- 20,000 signature operations per block
- 100 M satoshis per bitcoin
- 23M total bitcoins maximum
- 50,25,12.5… bitcoin mining reward

These affect economic balance of power too much to change now

Throughput Limits in Bitcoin

Blocks are limited to 1 M bytes each (10 min)
With
at least 250 bytes/transaction
this gives about 7 transactions/sec!

Compare to:
- VISA: 2,000–10,000 transactions/sec
- PayPal: 50–100 transactions/sec
Cryptographic Limits in Bitcoin

1. Only 1 signature algorithm (ECDSA/P256)
2. Hard-coded hash functions

Crypto primitives might break by 2040...

Changing the Protocol: Hard vs. Soft Forks

Q: So, you want to change the protocol. What to do about “old” nodes?

Hard Fork: Change introduces new features that were previously considered invalid.
"Hard-forking" Changes to Bitcoin

PROBLEM: Old nodes will never catch up.

Changing the Protocol: Hard vs. Soft Forks

Q: So, you want to change the protocol. What to do about "old" nodes?

Hard Fork: Change introduces new features that were previously considered invalid.

Soft Fork: Change introduces new features that make validation rules stricter.
**Soft Forks**

**Observation:** We can add new features that only *limit* the set of valid transactions.

- Need majority of nodes to enforce new rules.
- Old nodes will approve.

**RISK:** Old nodes might *mine now-invalid* blocks.

---

**Soft Fork Example: Pay-to-Script-Hash**

```
<signature>
<<pubkey> OP_CHECKSIG>

OP_HASH160
<hash of redemption script>
OP_EQUAL
```

Old nodes will just approve the hash, not run the embedded script.
Soft Fork Possibilities

- New signature schemes
- Extra per-block metadata
  - Stricter formatting of the coinbase parameter
  - Add Merkle Tree of UTXOs (Unspent Transaction Outputs) in each block

Hard Forks

- New op codes
- Changes to size limits
- Changes to mining rate
- Many small bug fixes (e.g. MULTI-SIG)

Currently very unlikely to happen.

We will revisit this when we discuss altcoins.