IC3 The Initiative For CryptoCurrencies & Contracts

# Deep dive into the security of cryptocurrencies

Patrick McCorry

KING'S College LONDON

# Questions Questions Questions!

Today - I'll focus on **concrete instantiations** of the principles taught yesterday, alongside tips on designing secure smart contracts.

But…. we can't do it without your help!

Please interrupt at any time. Ask lots of questions.

**You Goal:** Stop me from finishing my slide deck!

# Part 1: The “Crypto” in Cryptocurrency

# Two Cryptographic Primitives

# Cryptographic Hash Function

h = Hash(x)

One-way pseudorandom (and collision-resistant) function

***Given h, it is computationally difficult to compute the pre-image x.***

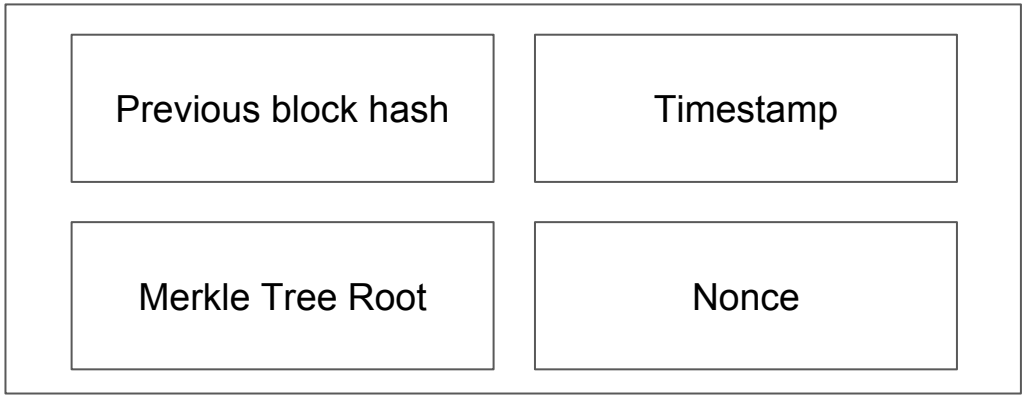
***1:1 matching of inputs to output (probabilistically)***

*In the cryptography world - we often call it a “commitment” - since there is no concept of “decrypting”*

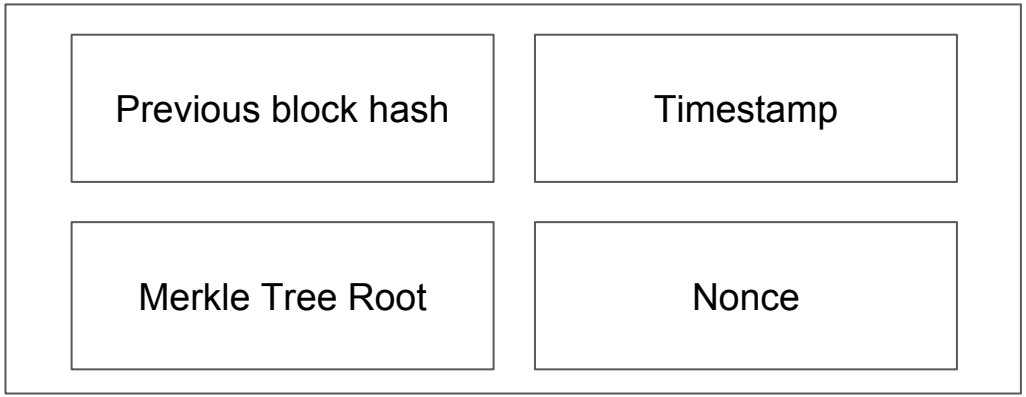
# Hash Example:

How can I verify a transaction is in the blockchain?

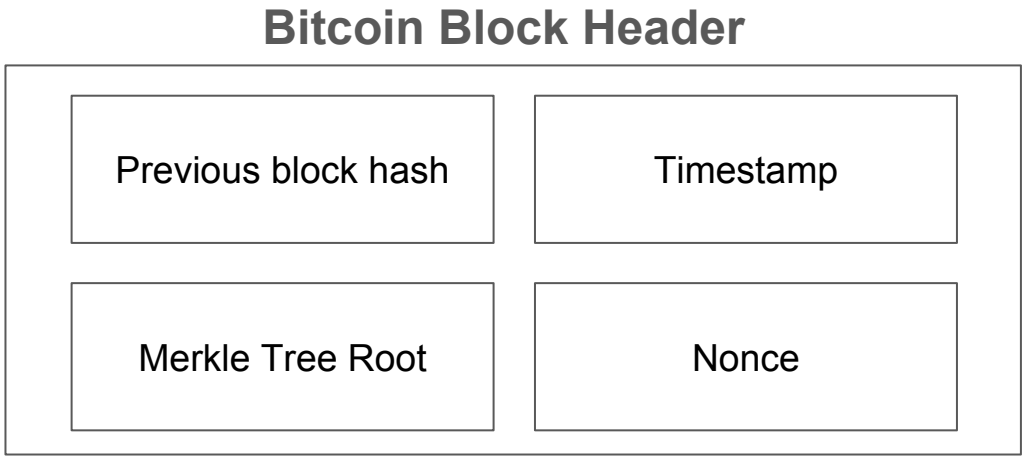
# Bitcoin Block Header



## Bitcoin Block Header

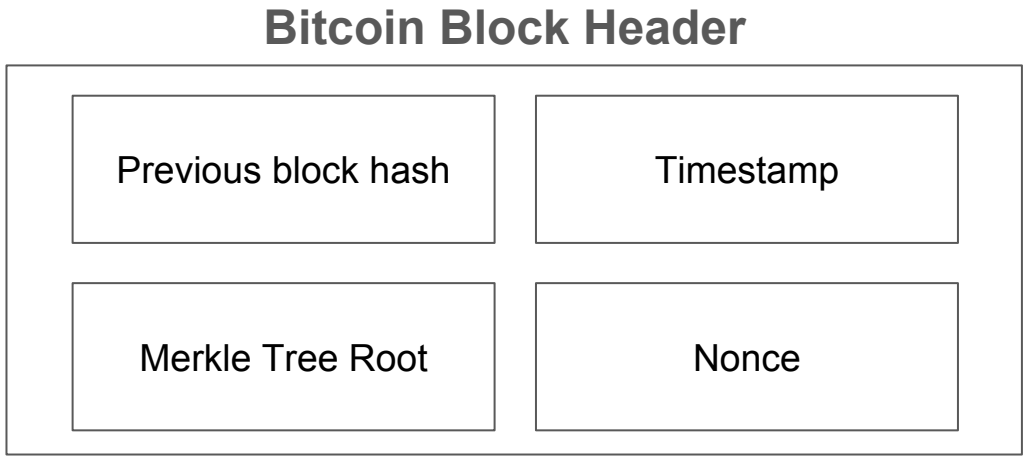


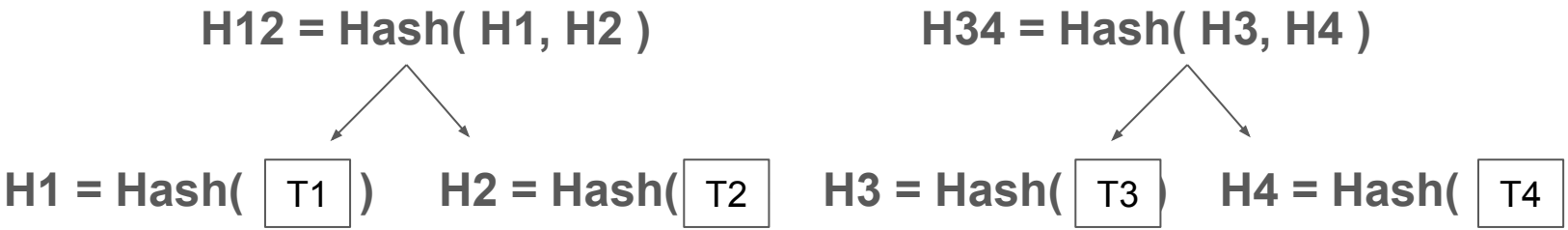

T1 T2 T3 T4



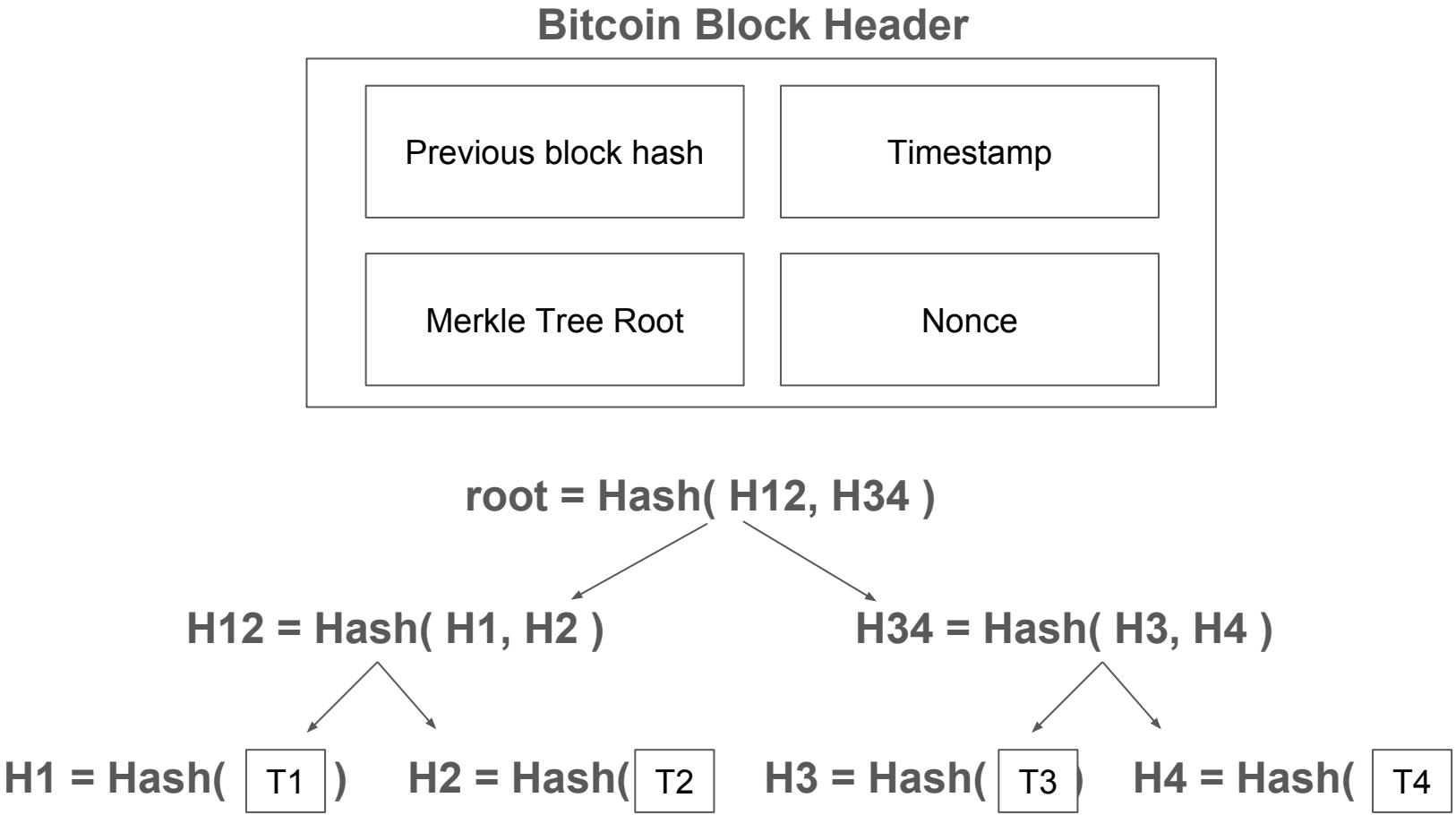

**H1 = Hash( T1 )** **H2 = Hash( T2 )** **H3 = Hash( T3 )** **H4 = Hash( T4 )**

**Step 1:** Hash all transactions individually

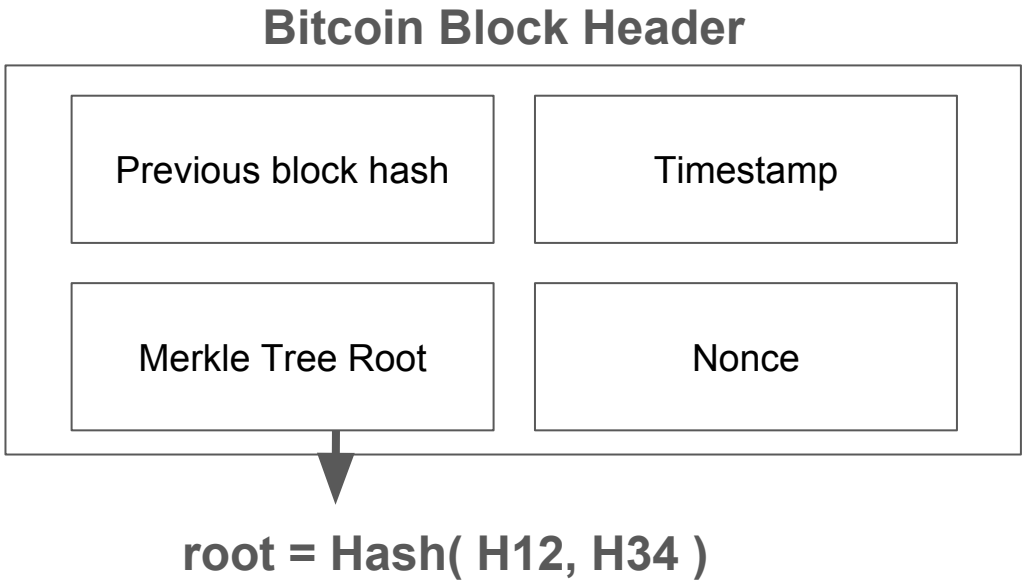



**Step 2:** Hash pairs H1,H2 and H3,H4

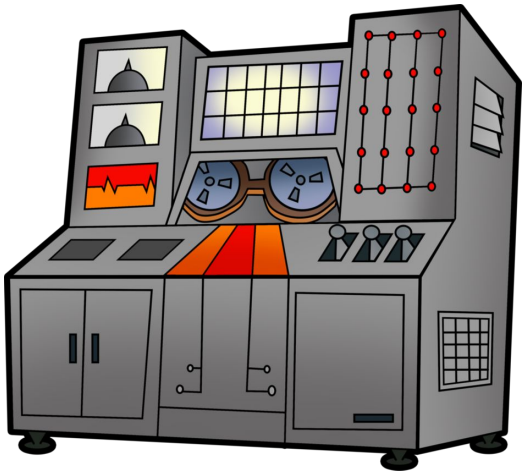


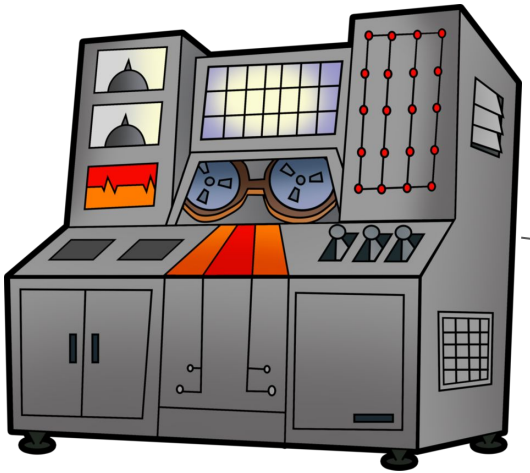

**Step 3:** Hash H12 and H34!




**Merkle Root:** Commitment to all transactions in the block

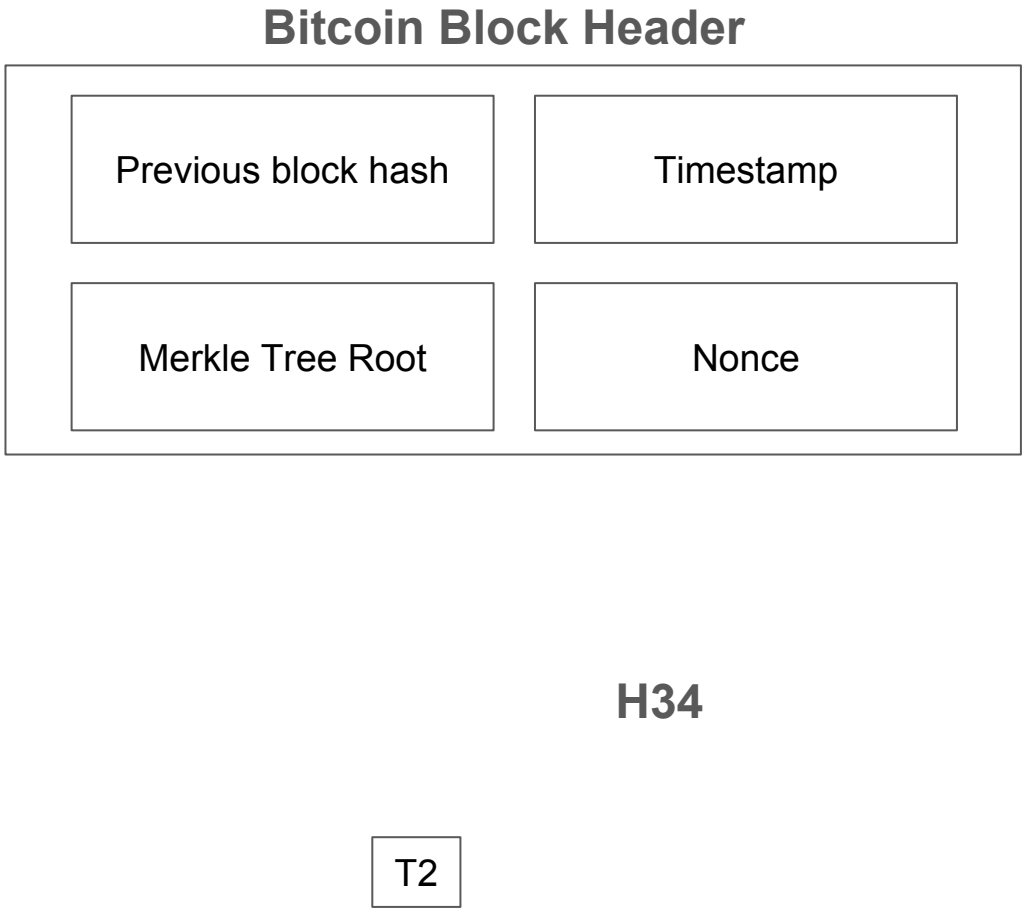
**Task:** Prove T2 are in a block?



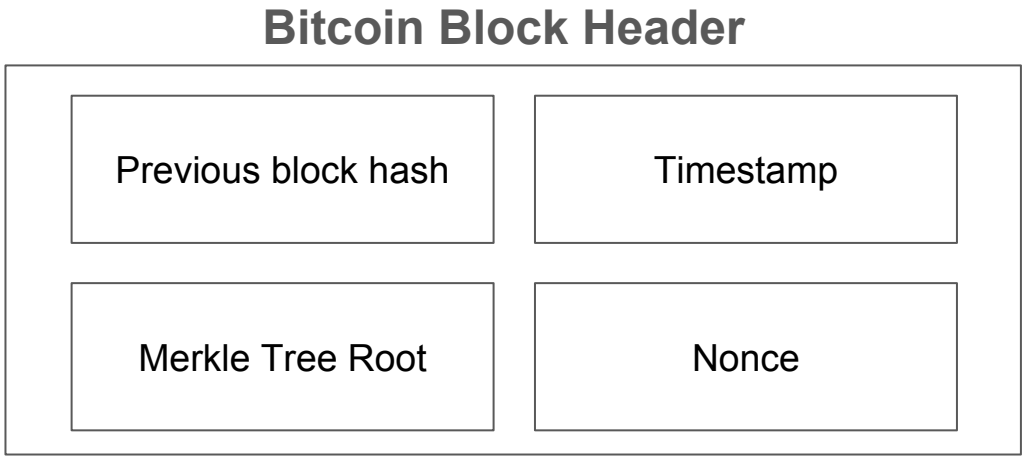


T2 *Block ID, H1, H34*

**Step 1:** Send client H1 and H34



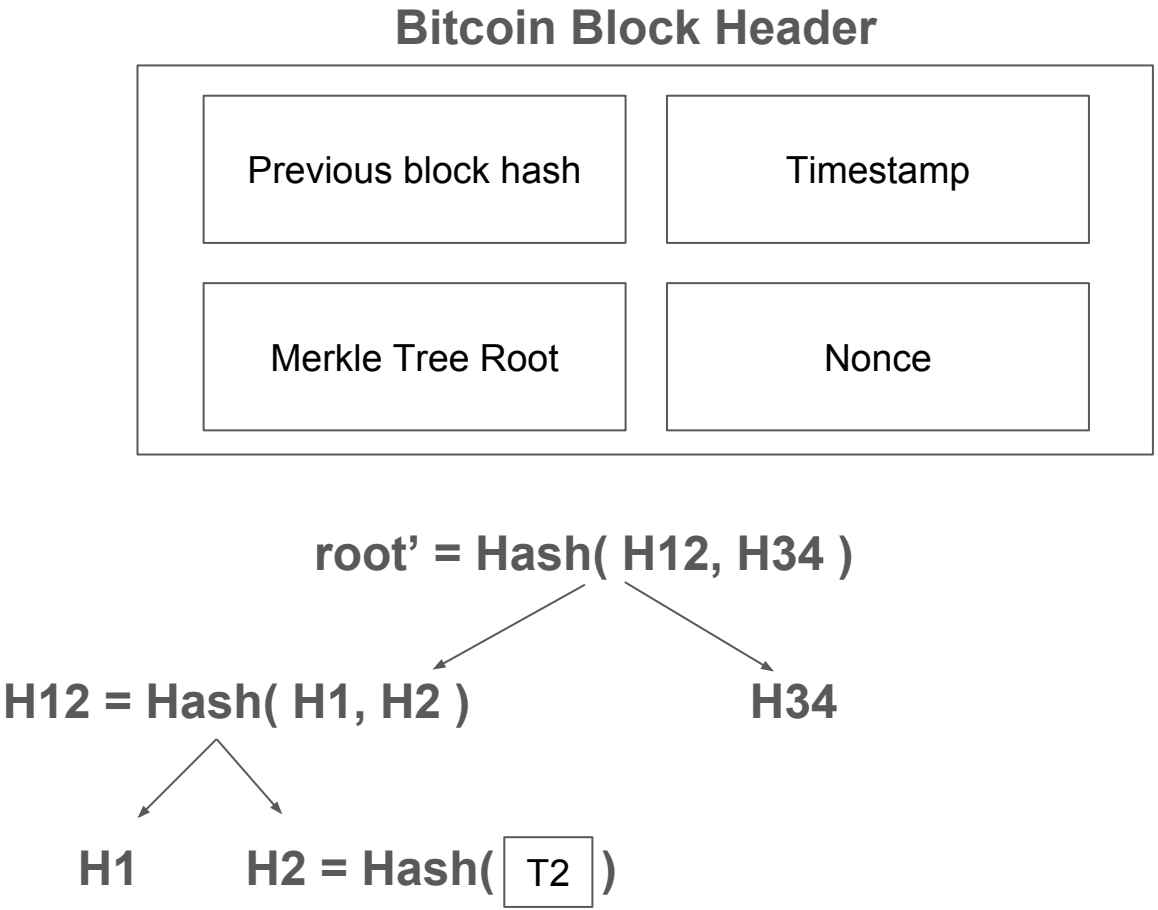

**Step 2:** Organise your data! Fetch block header!



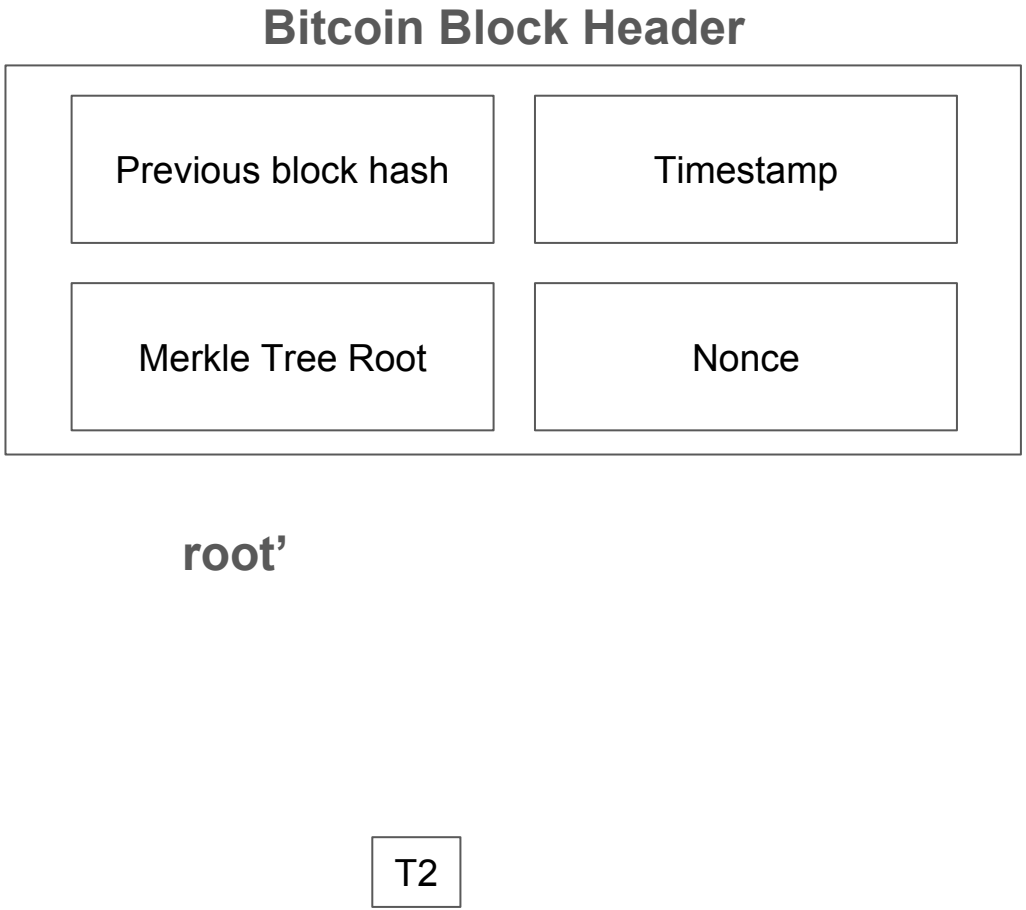

H34

H1 H2 = Hash( T2 )

**Step 3:** Compute H2 by re-hashing transaction

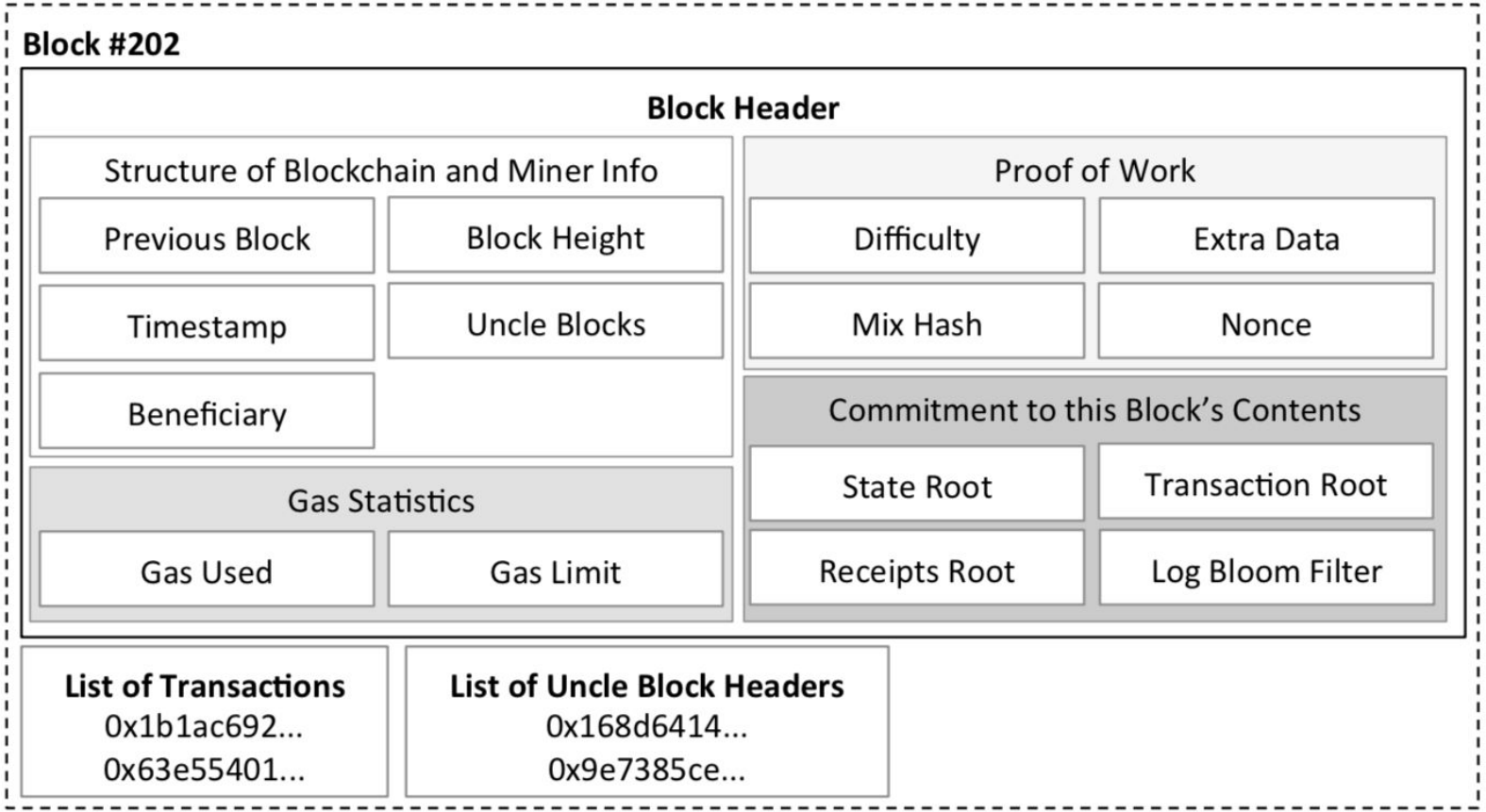



**Step 4:** Compute H12 and root'




**Verify:** root’ == block header’s root?

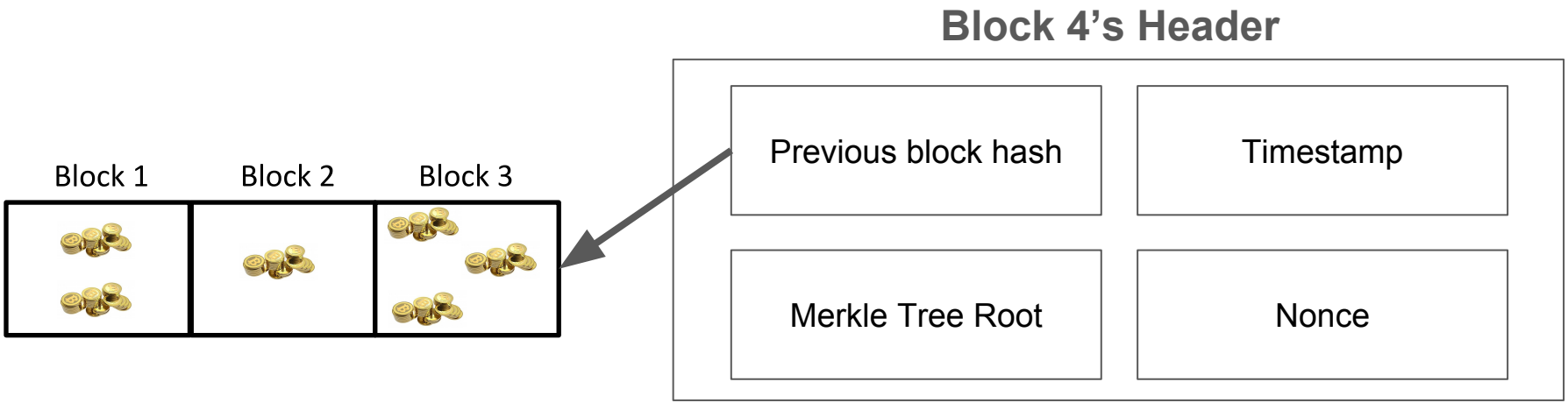
# **Ethereum Block Header:** Roots and Trees



# Hash Example:

How is the blockchain “chained” together?

# Chain of Hashes




Previous block hash is **Hash**(block_header)

# Hash Chain + Merkle Tree = ?

# **Unlocked:** Simplified Payment Verification Mode!

- “Light weight clients” store the list of block headers
  - Discard (or simply do not receive) all transactions.
- Transaction Inclusion Proof
  - Transaction + Merkle Tree Branch + Block Header
- **Weaker Trust assumption**
  - Cannot validate all transactions, assumes chain with most proof of work is correct
  - “Fraud Proof” is a research direction - prove that a block is invalid.

# Digital Signatures

***Signing and Verification functions:***

Sign(sk, msg) returns sig

Verify(sig, msg, pk) returns boolean

***Given sig, pk, msg:***

*It is computationally infeasible to find a second msg' such that*

*Verify(sig, msg', pk) == TRUE*

Secret Key = sk

Public key = pk

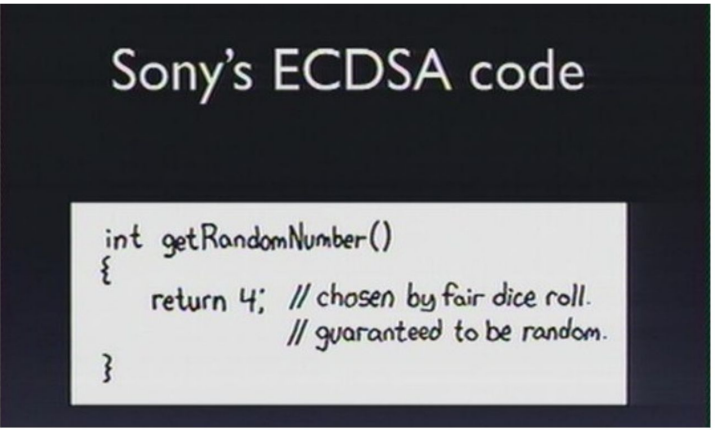
***Key point:*** *Digital signatures are unforgeable*

# Secure random number generators are required!

## Hackers obtain PS3 private cryptography key due to epic programming fail? (update)

Sean Hollister
12.29.10

1
Shares




## Android Security Vulnerability

**11 August 2013**

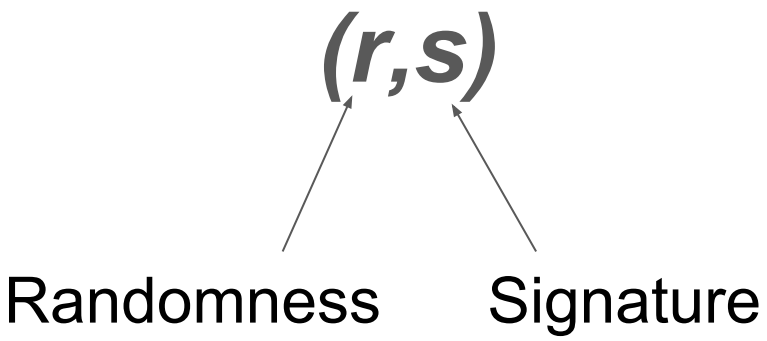
### What happened

We recently learned that a component of Android responsible for generating secure random numbers contains critical weaknesses, that render all Android wallets generated to date vulnerable to theft. Because the problem lies with Android itself, this problem will affect you if you have a wallet generated by any Android app. An incomplete list would be Bitcoin Wallet, blockchain.info wallet, BitcoinSpinner and Mycelium Wallet. Apps where you don't control the private keys at all are not affected. For example, exchange frontends like the Coinbase or Mt Gox apps are not impacted by this issue because the private keys are not generated on your Android phone.

### What has been done

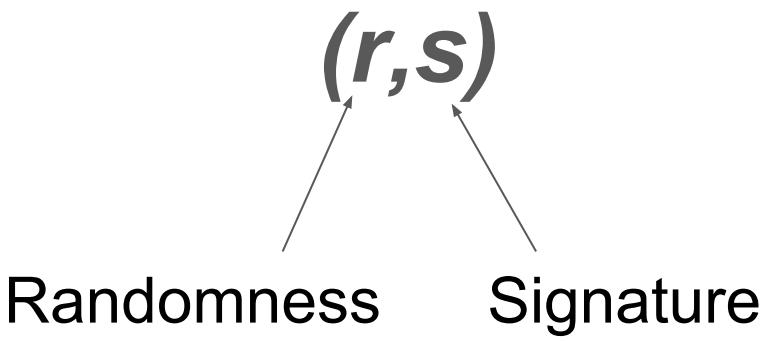
# Secure random number generators are required!

*An ECDSA Signature is denoted:*



# Secure random number generators are required!

*An ECDSA Signature is denoted:*

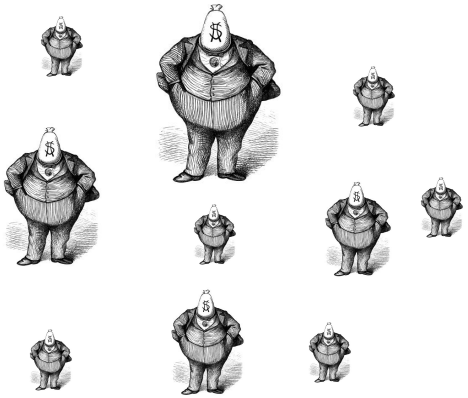



**Key Point:** Re-using randomness can be detected publicly!

Cryptography’s goal is not always data privacy.

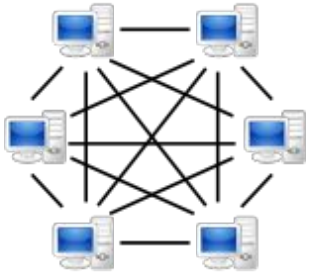
In cryptocurrencies…. cryptography is used to guarantee its integrity and to authenticate users.

# Part 2: The Myth of Decentralisation



**Nakamoto Consensus**

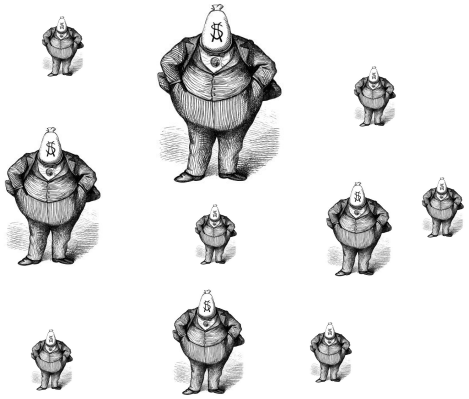
**Users of the network**



**Public Verifiability**

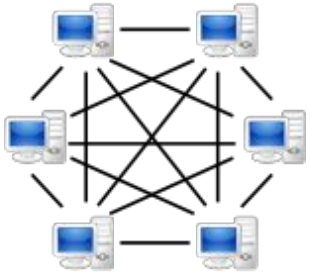
**Security Paradigm:**

Trust, but verify



**Nakamoto Consensus**
**Distributed**

**Users of the network**
**$$ Tradeoff**

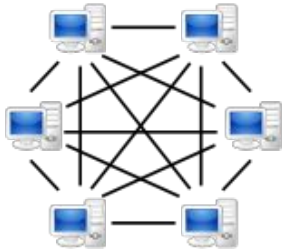


**Public Verifiability**
**Throughput Tradeoff**

**Security Paradigm:**
Trust, but verify

**Users of the network**
**Decentralised**

**Big block**

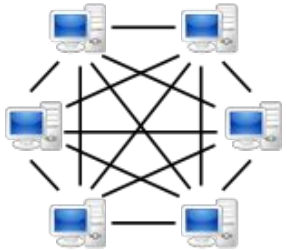


**Public Verifiability**
**Centralised**

**Reliability:** Slower to propagate, more forks - less reliable confirmation time
**Cheap:** Fit more transactions, lower fee fee (knapsack problem)
**Datacentres:** Large resources required to fully verify the blockchain

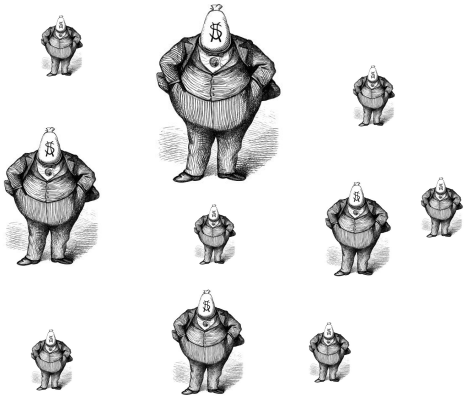
**Users of the network**
**Centralised**

**Small block**



**Public Verifiability**
**Decentralised**

**Reliability:** Faster to propagate, less forks. Reliable confirmation time.
**Expensive:** Less transactions - competitive fee market
**Home Computer:** More resources required to fully verify the blockchain



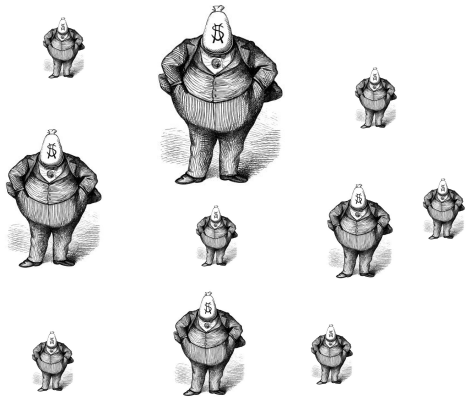
**Nakamoto Consensus**
**Distributed**


**Threat Model:**

Miners are mostly peaceful, but there is a tragedy of the commons

Mining is DISTRIBUTED

Dangerous?



**Increasing Power**

0 - 51%

**Delay or Censor Transactions**

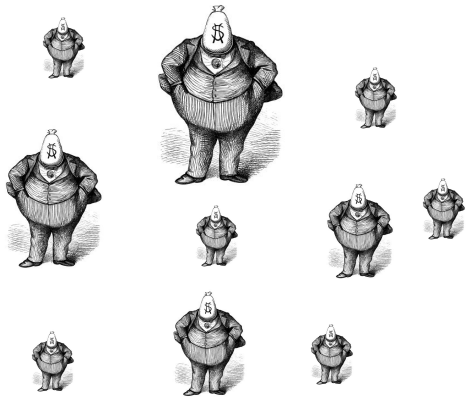
Any protocols that require an action to be performed with a “grace period”

**Withhold Blocks**

Using block hash as a random beacon? (e.g. gambling). Miner can bias the beacon.

**Impact execution by ordering transactions**

Their transaction is the first one processed in the block.



**Full control**

Over 51% cartel

**+Reverse Transactions**

Re-organise the blockchain (i.e. fork it) to add/remove transactions.

# Part 3: Smart Contract Features And World-View

# You are developing for a distributed system!

This means:

- No privacy (normally)
- No local storage
- No non-deterministic execution
- No external lookups of data
- Everything is expensive!!! (if done naively)
- Security is extremely important!
- Not being careful can lock your contract forever!

# Smart Contracts have TWO forms of memory

```
contract Storage {
  uint256 st = 1; // STORAGE

  function f(){
    uint256 mem; // MEMORY
  }

}
```

**Storage** is long-term
(and expensive)

**Memory** is short-term
(and cheap)

# Contract Environment Variables

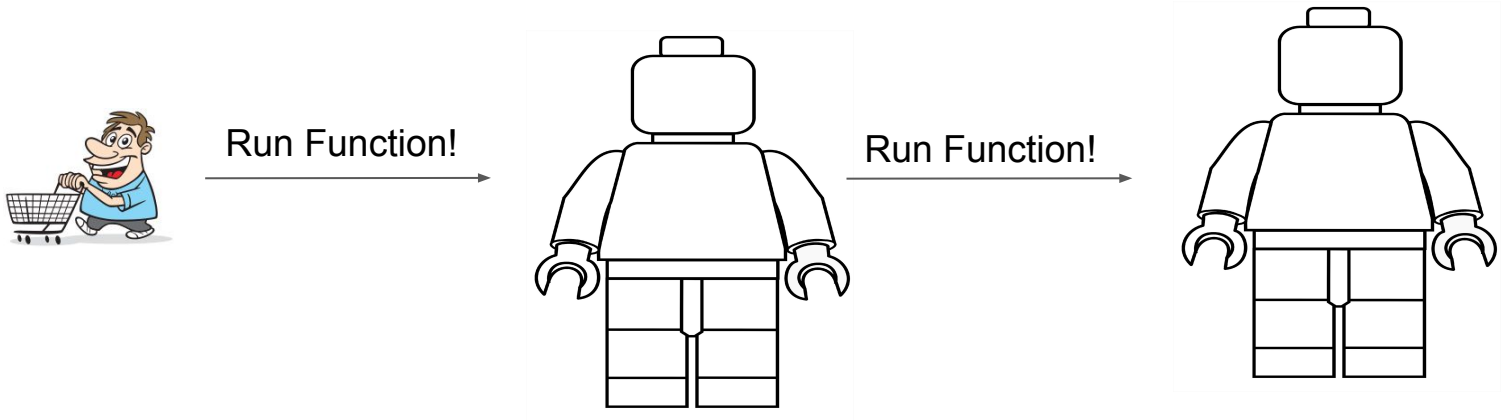
Blocks

- Gas limit, network difficulty, PREVIOUS hashes, miner’s address

# Contract Environment Variables

Blocks

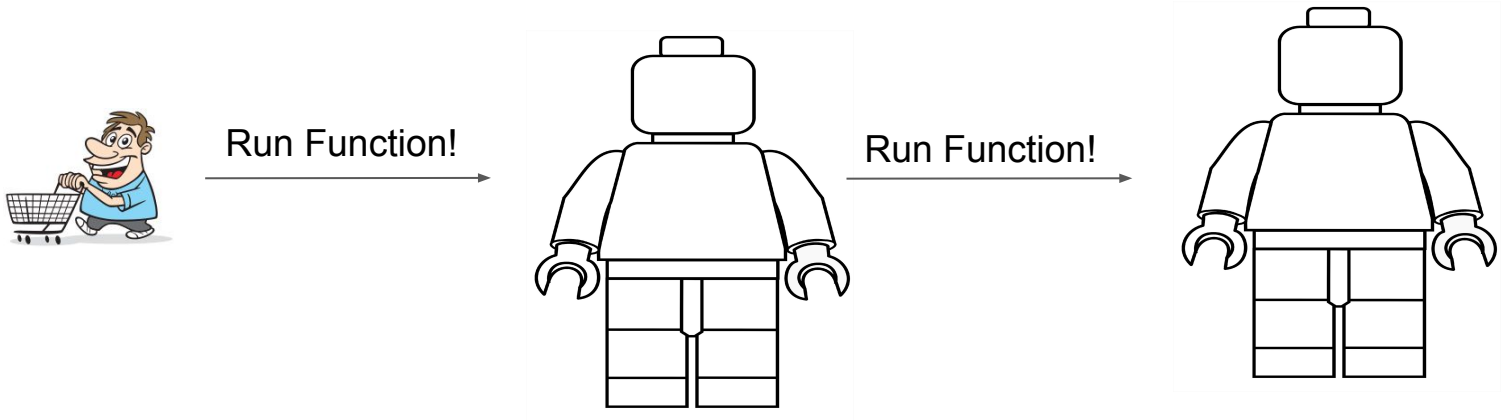
- Gas limit, network difficulty, PREVIOUS hashes, miner’s address



# Contract Environment Variables

Blocks

- Gas limit, network difficulty, PREVIOUS hashes, miner’s address

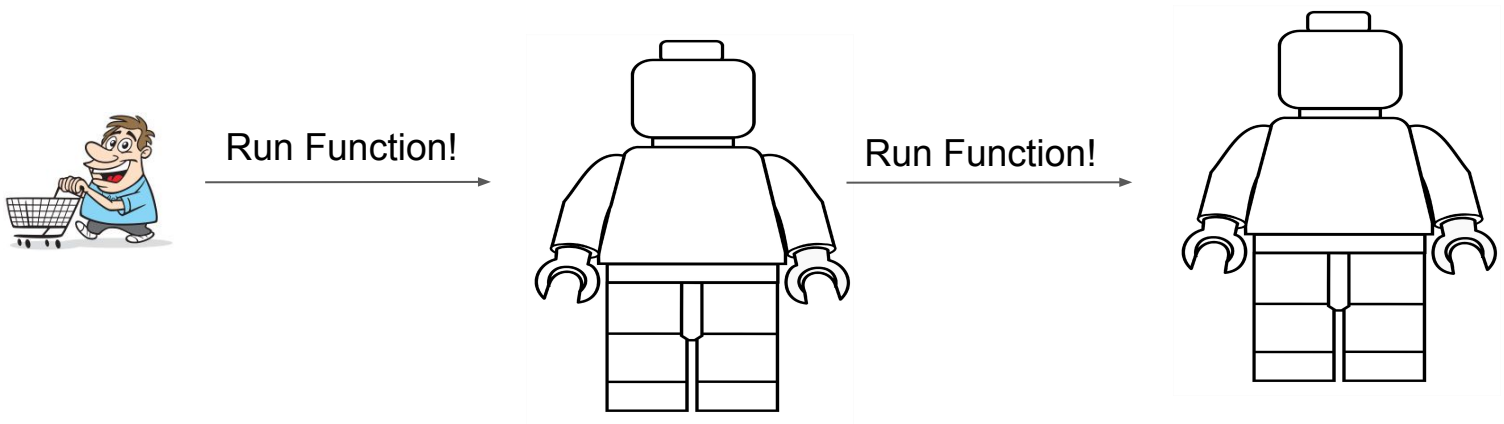



**Who is calling me?**

# Contract Environment Variables

Blocks

- Gas limit, network difficulty, PREVIOUS hashes, miner's address

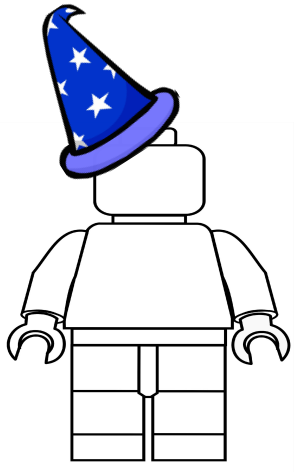



**tx.origin()** **msg.sender()** **Who is calling me?**

http://solidity.readthedocs.io/en/develop/units-and-global-variables.html

Contract’s are blind and cannot see the outside world.

# Oracle Contracts

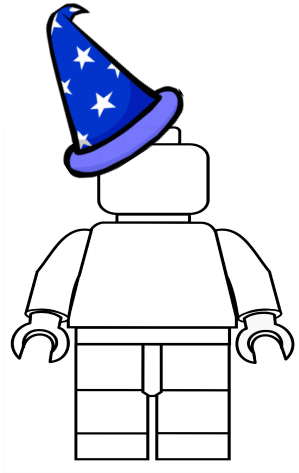


**Purpose:** Verify and store information from the external world

# Oracle Contracts

BBC NEWS

Football results

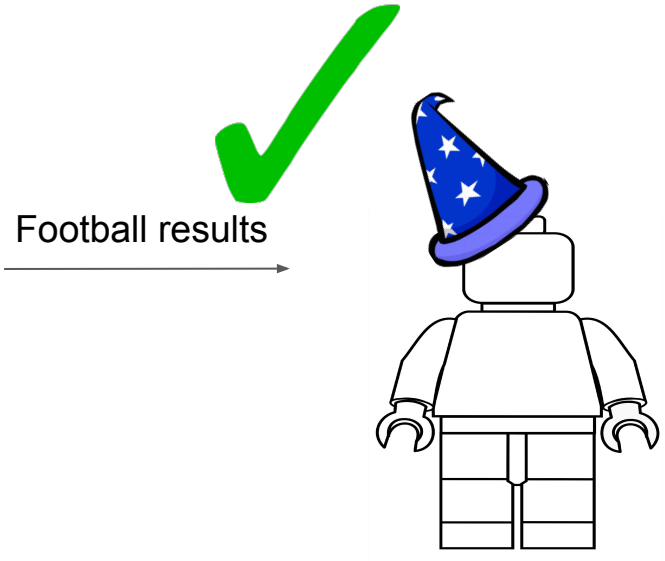


**Step 1:** BBC sends football results (and digitally signs it!)

# Oracle Contracts

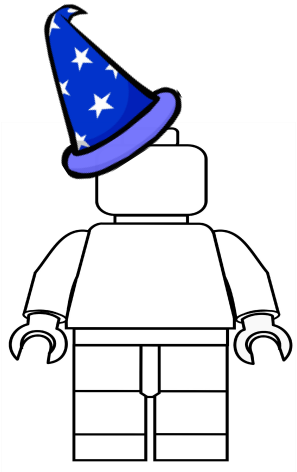
BBC NEWS

Football results



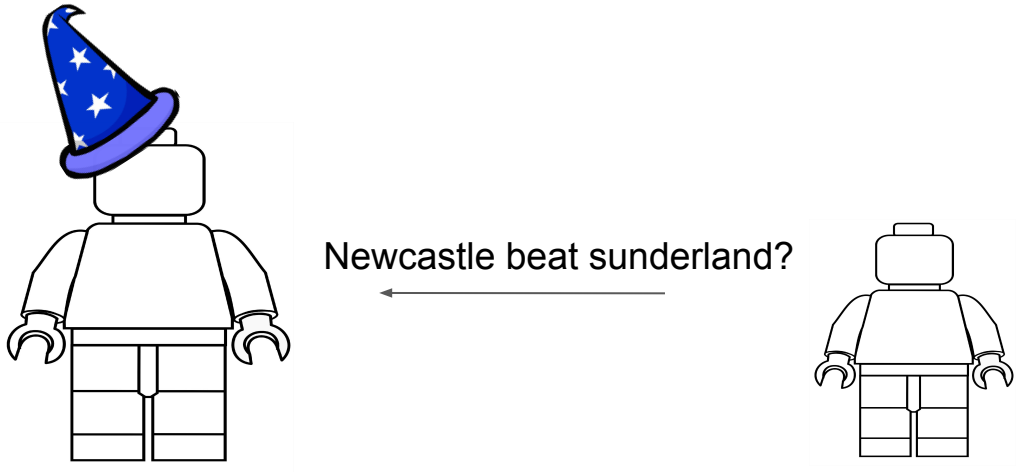
**Step 2:** Oracle verifies BBC’s signature and stores results. (We must trust BBC)

# Oracle Contracts



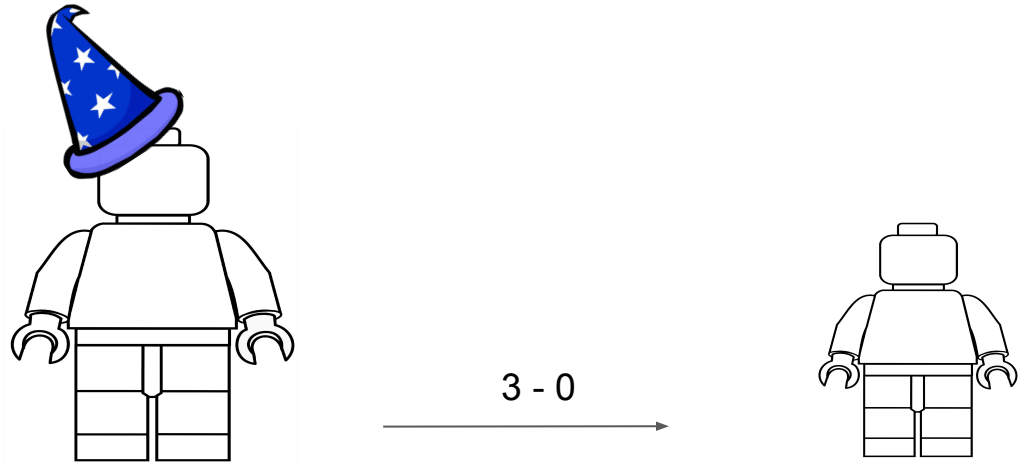
**Step 3:** Wait for a request from another smart contract

# Oracle Contracts




**Step 4:** Request comes in. Did Newcastle United win today?

# Oracle Contracts




**Step 5:** Yes sir! Newcastle United won :)

# Not turing-complete.. But that doesn’t matter anyway

Solidity (and the EVM) supports for-loops, if statements, etc.

… but the computation is bounded by the gas limit!

**Today - a block’s gas limit ~8 million gas….**

# Part 4: Smart Contract dangers that may arise

Be careful not to run out of gas.. might lock up coins!

```
contract loop {

    uint[] array;
    function doLoop() public pure {

        for(uint i=0; i>array.length; i++) {
            // do a nice trick here
        }

        // send money to someone

    }
}
```

All execution is deterministic…

but what *actually is* executed is not!

```
contract NotGuaranteed {

    boolean ready;
    uint deposit;

    function doSomething() public pure returns(byte) {

        if(ready) {
            return deposit;
        } else {
            uint h = 0;
            for(uint i=0; i=1000; i++) {
                h = sha3("lol" + h);
            }
        }
    }
}
```

**If “ready” is true, send caller the deposit OR waste caller’s gas.**

**If “ready” is true, send caller the deposit OR waste caller’s gas.**

**Oh! Looks like I can get the deposit!**

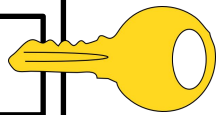
**Epppp!!**

# If “ready” is true, send caller the deposit OR waste caller’s gas.

**Invocation Transaction**

| From | To | Value |
| --- | --- | --- |
| Bob | Contract | 0 eth |

| Gas Price | Total Gas | Nonce |
| --- | --- | --- |
| $2 * 10^{-8}$ | 90,000 | 321 |



Data

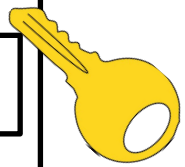
**Contract.doSomething()**

# If “ready” is true, send caller the deposit OR waste caller’s gas.

**Invocation Transaction**

| From | To | Value |
|---|---|---|
| Bob | Contract | 0 eth |

| Gas Price | Total Gas | Nonce |
|---|---|---|
| $2 * 10^{-8}$ | 90,000 | 321 |



Data

**Contract.doSomething()**

# If “ready” is true, send caller the deposit OR waste caller’s gas.

**Invocation Transaction**

| From | To | Value |
|---|---|---|
| Bob | Contract | 0 eth |

| Gas Price | Total Gas | Nonce |
|---|---|---|
| $2 * 10^{-8}$ | 90,000 | 321 |

Data

**Contract.doSomething()**

# If “ready” is true, send caller the deposit OR waste caller’s gas.

**Invocation Transaction**

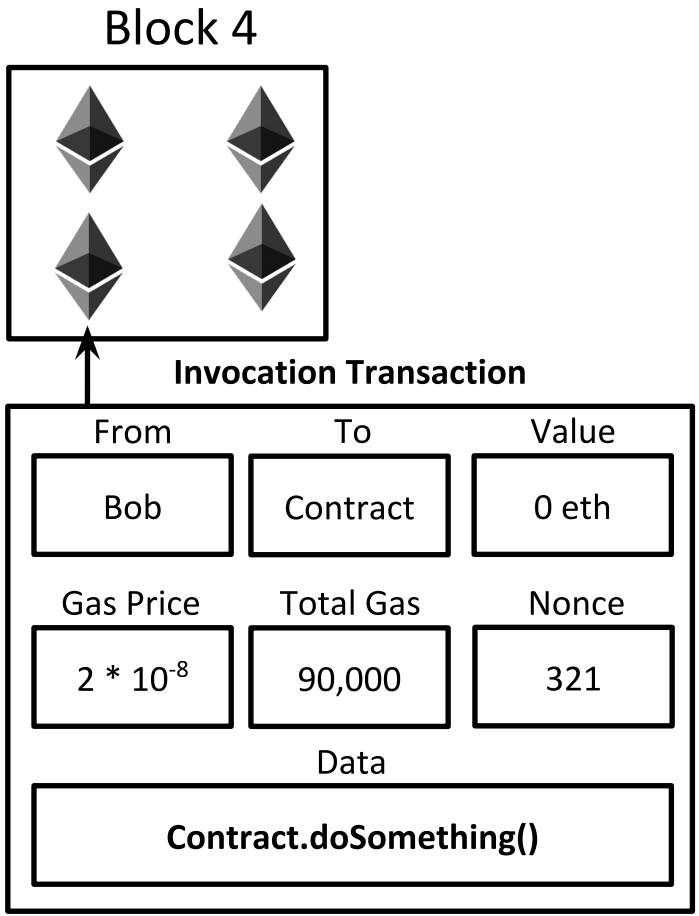
| From | To | Value |
|---|---|---|
| Bob | Contract | 0 eth |

| Gas Price | Total Gas | Nonce |
|---|---|---|
| $2 * 10^{-8}$ | 90,000 | 321 |

Data

**Contract.doSomething()**

# If “ready” is true, send caller the deposit OR waste caller’s gas.

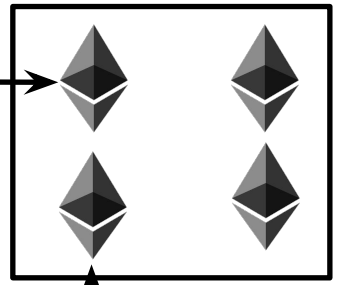



# If “ready” is true, send caller the deposit OR waste caller’s gas.

**Front-Running Transaction**

Updates “ready” to FALSE.

Block 4



**Invocation Transaction**

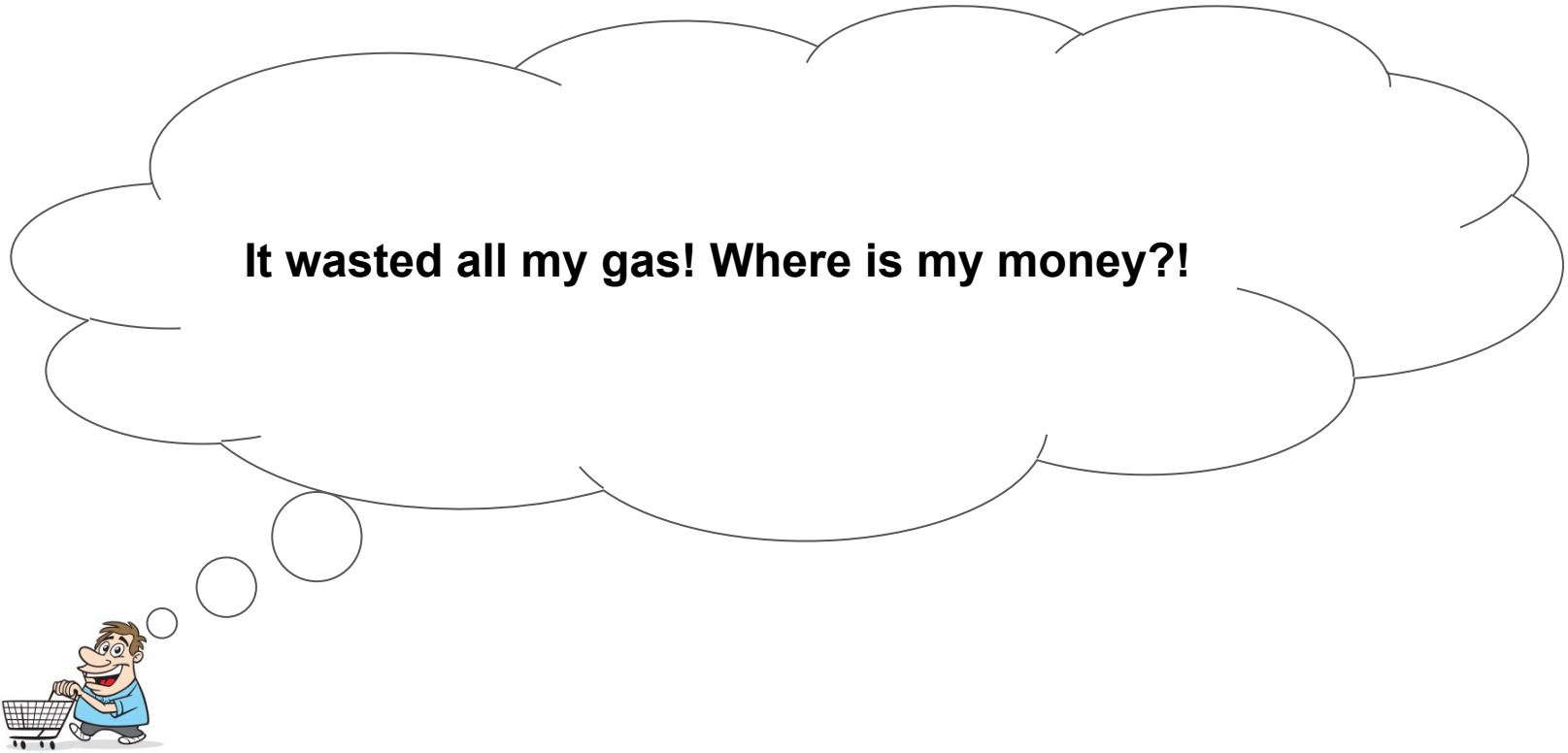
| From | To | Value |
|---|---|---|
| Bob | Contract | 0 eth |

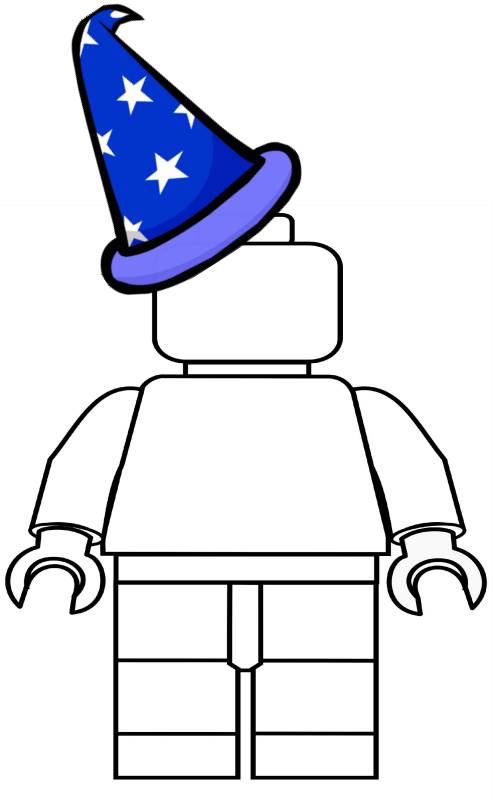
| Gas Price | Total Gas | Nonce |
|---|---|---|
| $2 * 10^{-8}$ | 90,000 | 321 |

Data

**Contract.doSomething()**

**If “ready” is true, send caller the deposit OR waste caller’s gas.**





## Unpredictable Execution

Transactions are signed to execute contracts, but this execution is not decided until it is in the blockchain!

The “predictability” is fully dependent on the implementation! Don’t do fancy tricks!

This (alongside super-clever tricks) has led to the rise of Honey Pot contracts!

https://medium.com/@gerhard.wagner/the-phenomena-of-smart-contract-honeypots-755c1f943f7b

Be mindful not to call other contracts….
By mistake…

```
contract Send {

    uint256 money = 0;
    uint256 allowed_widthdraw = 1;
    address someone_else = 0x123...;

    function fund() payable {
        money += msg.value;
    }

    function pay() {
        someone_else.send(allowed_withdraw);
        allowed_withdraw = 0;
    }
}
```

**someone_else can be ANOTHER contract**

```
11      function pay() {
12          someone_else.send(allowed_withdraw);
13          allowed_withdraw = 0;
14      }
```

**someone_else can be ANOTHER contract**

Pay() is invoked.
At line 12 - the **someone_else.send()** invokes the contract

```
11      function pay() {
12          someone_else.send(allowed_withdraw);
13          allowed_withdraw = 0;
14      }
```

**someone_else can be ANOTHER contract**

Pay() is invoked.
At line 12 - the **someone_else.send()** invokes the contract

someone_else **re-invokes Pay()**
Again, at line 12 - the **someone_else.send()** invokes the contract

```
11    function pay() {
12        someone_else.send(allowed_withdraw);
13        allowed_withdraw = 0;
14    }
```

**someone_else can be ANOTHER contract**

Pay() is invoked.
At line 12 - the **someone_else.send()** invokes the contract

someone_else **re-invokes Pay()**
Again, at line 12 - the **someone_else.send()** invokes the contract

This can result in several withdrawals;
but the contract assumes only one withdrawal has happened!

```
11    function pay() {
12        someone_else.send(allowed_withdraw);
13        allowed_withdraw = 0;
14    }
```


**Simon de la Rouviere** @simondlr · May 25

Ethereum inside joke game on point at @ethbuenosaires. Door is labeled as "Reentrancy". 😅

2 5 48

# The “Checks / Effects / Interactions” paradigm

A Best Practice guideline for safe smart contract behavior

When receiving a message, do the following in order:

1. Perform all input validation and checks on current state. Discard the message if validation fails.

2. Update local state.

3. Finally, pass on interactions to trigger other contracts.

**Contract A:**

```
public address callee;
public int balance = 100;
…


function withdraw()
only(callee) {
 if (balance <= 0) return;
 var toSend = balance;
 balance = 0;
 callee.recv.value(toSend)();
}
```

Stole slide from andrew miller

# Don’t trust the contract yet? Reactionary Security

# Don’t trust the contract yet? Reactionary Security

- Build in a **failsafe()** function
  - K out of N signers (distribute trust)
  - Shuts down contract and withdraws coins.

- Time-delayed withdrawals
  - All transfers are “pending” for a fixed-set of time
  - Provides time for **failsafe()** to be called and cancel any heist.

- Build in a **removefailsafe()** function
  - Disables the fail-safe function (and removes time-delay) if contract survives the test of time.

This human oversight is **controversial** in the community.

# If you only take away a few tips….

**Everything is public**
And replicated across the entire peer to peer network!

**Event Driven**
Contract's don't do anything by themselves. It must be notified by a transaction (and someone has to pay for that!)

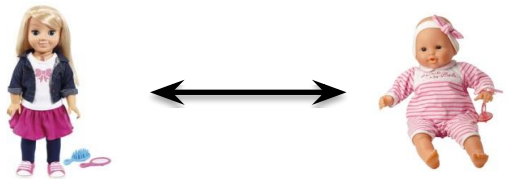
**Adversaries everywhere**
If someone else can negatively impact your interaction with the contract - it'll happen!

# Bonus Content

So-called “Layer 2” and “Off-chain” solutions

# What is a Payment Network?

**Payment Channel**



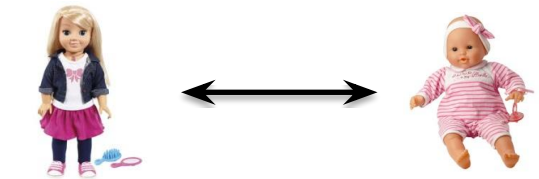
1.0 BTC 1.0 BTC

Both parties must authorise a payment.

No activity? Parties are eventually refunded their deposit.

# What is a Payment Network?

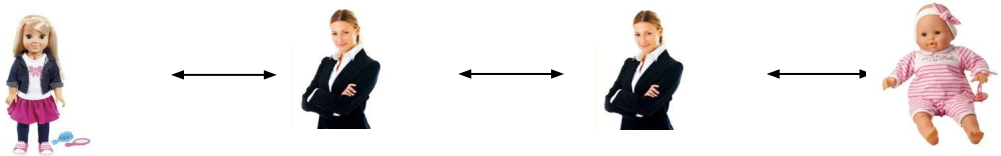
## Payment Channel




Both parties must authorise a payment.

No activity? Parties are eventually refunded their deposit.

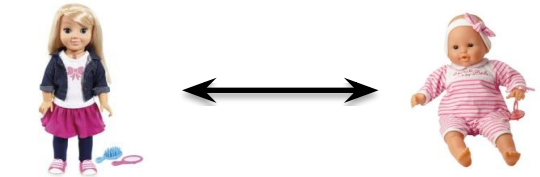
## Fair Exchange of bitcoins



Alice can send coins to Bob via an intermediary channel.

# What is a Payment Network?

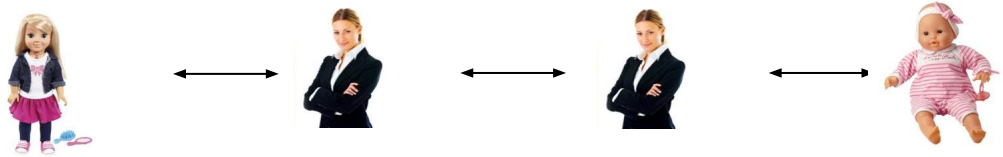
## Payment Channel




Both parties must authorise a payment.

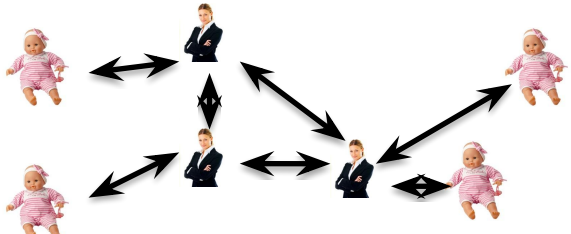
No activity? Parties are eventually refunded their deposit.

## Fair Exchange of bitcoins



Alice can send coins to Bob via an intermediary channel.

## Network of Payment Channels



# Why Payment Networks?

# Why Payment Networks?

## Settlement System

Blockchain is only consulted to
settle final balance and resolve disputes.

# Why Payment Networks?

## Settlement System

Blockchain is only consulted to settle final balance and resolve disputes.

## Scalability

Support thousands of transactions without consulting the Bitcoin network

# Why Payment Networks?

## Settlement System

Blockchain is only consulted to settle final balance and resolve disputes.

## Scalability

Support thousands of transactions without consulting the Bitcoin network

## Optimistic Privacy

Network can only see two transactions – not all exchanges between Alice and Bob.

# Why not just use Bitcoin?

# Why not just use Bitcoin?

**Transactions per second**

3.3-7 transactions per second

# Why not just use Bitcoin?

| **Transactions per second** | **Why?** |
| --- | --- |
| 3.3-7 transactions per second | Block Size + Frequency of new Blocks |

# Why not just use Bitcoin?

**Transactions per second**

3.3-7 transactions per second

**Why?**

Block Size + Frequency of new Blocks

**Who decided that???**

# Why not just use Bitcoin?

**Transactions per second**

3.3-7 transactions per second

**Why?**

Block Size + Frequency of new Blocks

**Who decided that???**

**Also…. Micropayments**

Average fee is currently $1…

Cannot send 10 cents or less anymore!

# Simple Payment Channels

Unidirectional (one-way)

# One-way Payment Channel

1.0 BTC

0.0 BTC

**Step 1:** Alice deposits coins into the channel

# One-way Payment Channel

1.0 BTC

0.0 BTC

**Step 1:** Alice deposits coins into the channel

**Condition to release coins:**
Both Alice and Bob authorise payment
OR
Alice waits until time T

# One-way Payment Channel

1.0 BTC

0.0 BTC

**Step 2:** Alice pays Bob 0.1 BTC

# One-way Payment Channel


**Step 2:** Alice pays Bob 0.1 BTC

Alice sends Bob the signature!

# One-way Payment Channel

0.9 BTC

0.1 BTC

**Step 3:** Bob must decide

Mutually sign payment and publish it
OR
Wait for a new payment from Alice

# One-way Payment Channel

0.8 BTC

0.2 BTC

**Step 4:** Alice pays Bob another 0.1 BTC

# One-way Payment Channel

0.0 BTC

1.0 BTC

**Step K:** Alice keeps paying bob… until she has no more coins!

# One-way Payment Channel

0.0 BTC

1.0 BTC

**Final step:** Bob signs and publishes the final payment he receives.

**Overall Two transactions:**

1 deposit transaction

1 payment transaction

# One-way Payment Channel

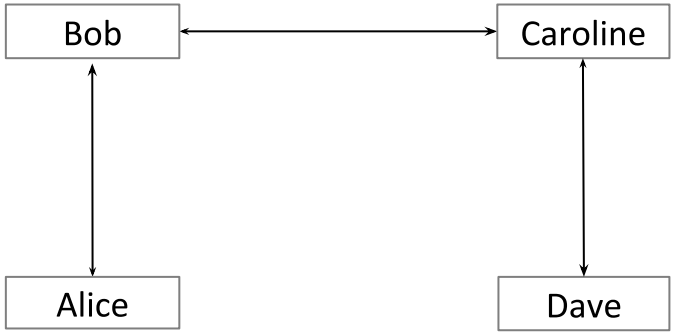
0.0 BTC

1.0 BTC

**Final step:** Bob signs and publishes the final payment he receives.

Raiden’s payment channels is a pair of one way channels!

# Synchronising Channels (HTLC)




**Alice wants to pay Dave, via Bob and Caroline**

# Synchronising Channels (HTLC)

Bob

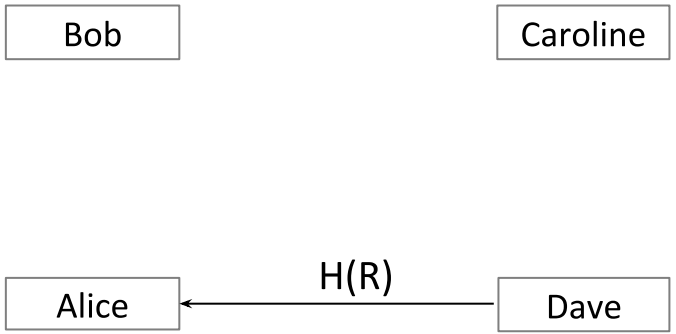
Caroline

Alice

Dave

**Round 1:** Set up the HTLC Transfer

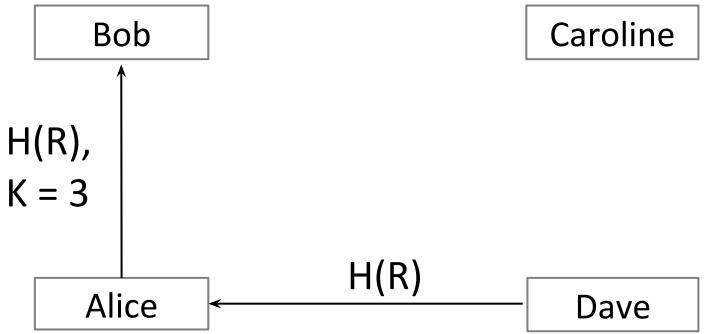
# Synchronising Channels (HTLC)




**Round 1:** Set up the HTLC Transfer

**Step 1:** Dave sends Alice H(R), where “R” is his secret!

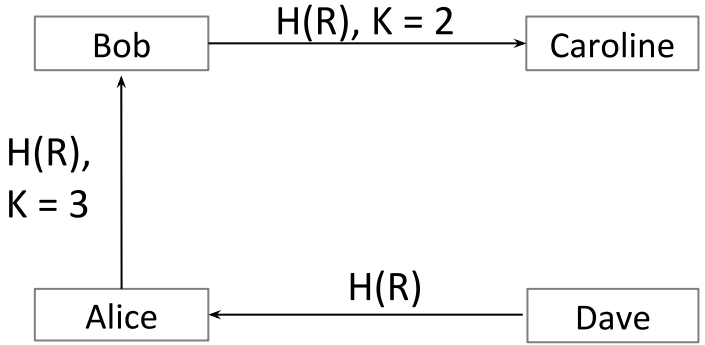
# Synchronising Channels (HTLC)




**Round 1:** Set up the HTLC Transfer

**Step 1:** Dave sends Alice H(R), where “R” is his secret!
**Step 2:** Alice sets up a “Conditional Transfer” with Bob. Expires after time 3 days.

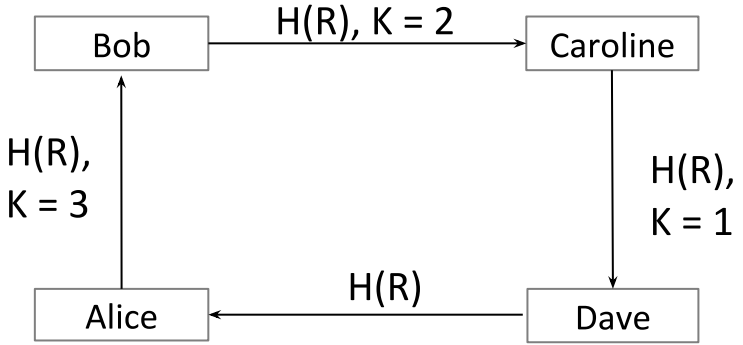
# Synchronising Channels (HTLC)




**Round 1:** Set up the HTLC Transfer

**Step 1:** Dave sends Alice H(R), where “R” is his secret!
**Step 2:** Alice sets up a “Conditional Transfer” with Bob. Expires after time 3 days.
**Step 3:** Bob sets up a “Conditional Transfer” with Caroline. Expires after 2 days.

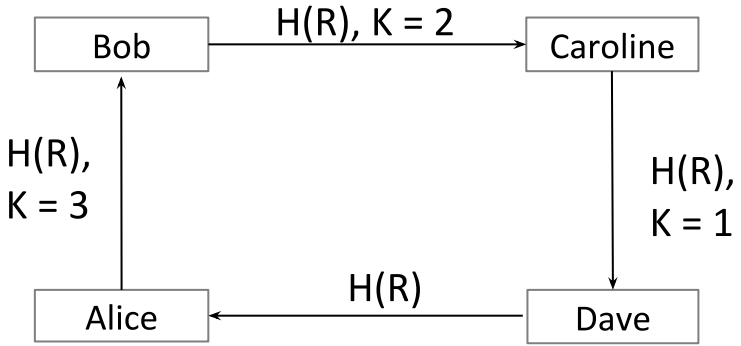
# Synchronising Channels (HTLC)




**Round 1:** Set up the HTLC Transfer

**Step 1:** Dave sends Alice H(R), where “R” is his secret!
**Step 2:** Alice sets up a “Conditional Transfer” with Bob. Expires after time 3 days.
**Step 3:** Bob sets up a “Conditional Transfer” with Caroline. Expires after 2 days.
**Step 4:** Caroline sets up a “Conditional Transfer” with Dave. Expires after 1 day.

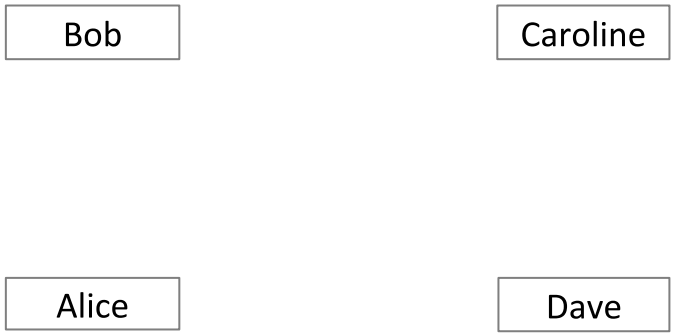
# Synchronising Channels (HTLC)




**The Payment Route is all set up!**

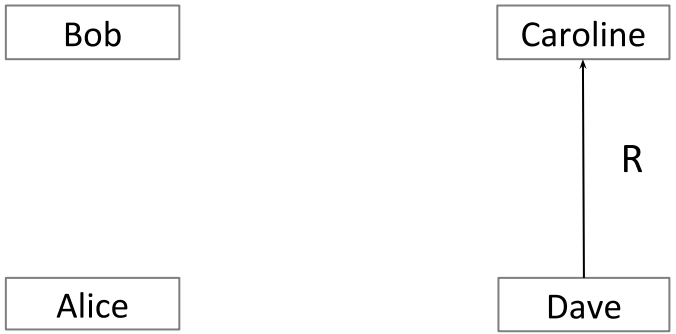
**Time to “finalise” the payment!**

# Synchronising Channels (HTLC)




**Round 2:** Retrieve payment

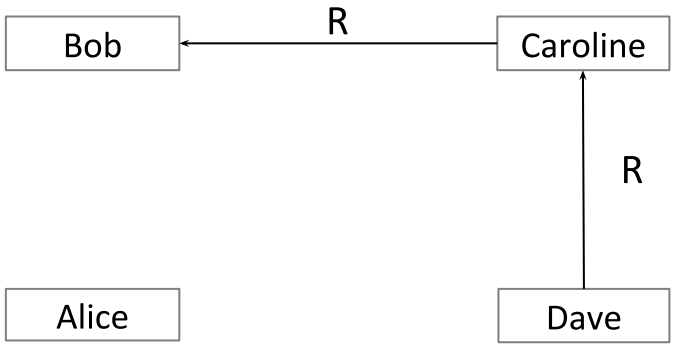
# Synchronising Channels (HTLC)




**Round 2:** Retrieve payment

**Step 1:** Dave reveals “R” to Caroline, and takes his coins!

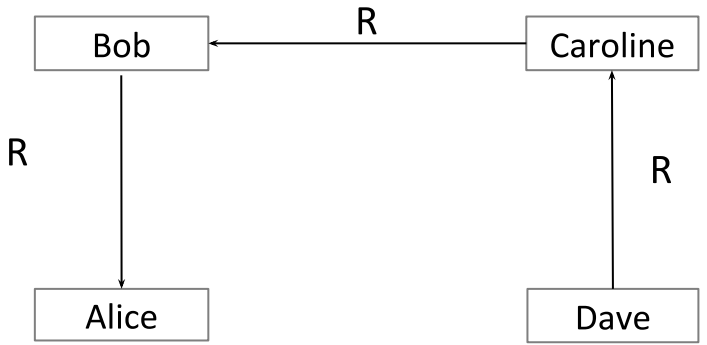
# Synchronising Channels (HTLC)




**Round 2:** Retrieve payment

**Step 1:** Dave reveals “R” to Caroline, and takes his coins!
**Step 2:** Caroline reveals “R” to Bob, and takes her coins!

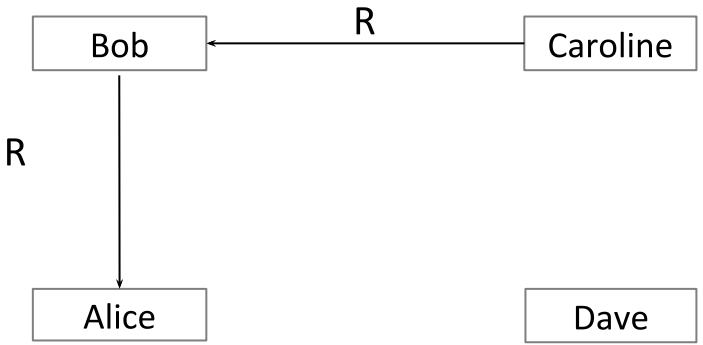
# Synchronising Channels (HTLC)




**Round 2:** Retrieve payment

**Step 1:** Dave reveals “R” to Caroline, and takes his coins!
**Step 2:** Caroline reveals “R” to Bob, and takes her coins!
**Step 3:** Bob reveals “R” to Alice, and takes his coins!

# Synchronising Channels (HTLC)




**Payment is complete!**

Everything happened “off-chain”

We are effectively just synchronising all channels on a single payment!

# Hiring 1-2 Javascript Developers

ethereum foundation grants

## Grantee List: May 2018

**Scalability:**

Perun – $250K. State channels R&D.

PISA (by Patrick McCorry et al.) – $250K. State channels R&D.

Sprites Implementation (by Enuma) – $200K. Payment channels implementation.

General Computation on Plasma (by Parsec Labs) – $50K. Plasma implementation.

Plasma (by Kyokan) – $50K. Plasma implementation.

Plasma (by Fourth State) – $32K. Plasma implementation.

Can we build a channel (in Bitcoin) with no expiry time and support bi-directional payments?

# Lightning Channels

Poon & Dryja (2014)

(with later help from Adam Back/Rusty Russell!)

**Old State**

**Both parties cooperate to authorise every payment**

# Lightning Channels

Poon & Dryja (2014)

(with later help from Adam Back/Rusty Russell!)

**Old State**

**New State**

**Both parties cooperate to authorise every payment**

**Step 1:** Create a new state (i.e. new balance for both parties)

# Lightning Channels

Poon & Dryja (2014)

(with later help from Adam Back/Rusty Russell!)

**Old State**

**New State**

**Both parties cooperate to authorise every payment**

**Step 1:** Create a new state (i.e. new balance for both parties)

**Step 2:** Both parties authorise the new state (i.e. exchange digital signatures)

# Lightning Channels

Poon & Dryja (2014)

(with later help from Adam Back/Rusty Russell!)

**REVOKED STATE**

**New State**


**Both parties cooperate to authorise every payment**

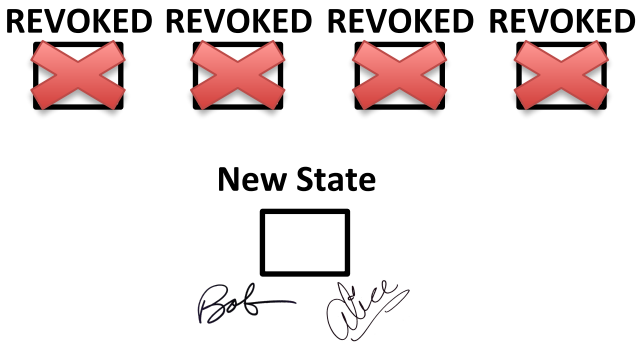
**Step 1:** Create a new state (i.e. new balance for both parties)

**Step 2:** Both parties authorise the new state (i.e. exchange digital signatures)

**Step 3:** Both parties revoke the old state (i.e. by exchanging secrets)

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)




**Both parties cooperate to authorise every payment**

**Step 1:** Create a new state (i.e. new balance for both parties)

**Step 2:** Both parties authorise the new state (i.e. exchange digital signatures)

**Step 3:** Both parties revoke the old state (i.e. by exchanging secrets)

**Repeat**

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)


**New State**

**Alice's TX**

**Bob's TX**

**Key Points**

**Copy of latest state:** Alice and Bob have a transaction that ONLY they can broadcast!

**No Expiry:** Any party can broadcast their tx at any time to claim their coins

**Safe Revocation:** Latest state is always valid before revoking the old one!

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)


**New State**

**Alice's TX**

**Bob's TX**

**Key Points**

**Copy of latest state:** Alice and Bob have a transaction that ONLY they can broadcast!

**No Expiry:** Any party can broadcast their tx at any time to claim their coins

**Safe Revocation:** Latest state is always valid before revoking the old one!

**…. But** what if someone broadcasts a revoked state? And claims it is the latest?

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**Alice's TX**

**Bob's TX**

**Bob wants to withdraw (honestly)**

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**Alice's TX**

**Bob's TX**

Block 1

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**Alice's TX**

**Bob's TX**

Block 1

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**Alice's TX**

Block 1

**Bob's TX**

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)

(with later help from Adam Back/Rusty Russell!)

**Alice's TX**

| Block 1 | Block 2 |
|---|---|
| **Bob's TX** | |

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**Alice's TX**

| Block 1 | Block 2 | Block ... |
|---|---|---|
| **Bob's TX** | | |

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**Alice's TX**

| Block 1 | Block 2 | Block ... | Block N |
|---|---|---|---|
| **Bob's TX** | | | |

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**Alice's TX**

**WITHDRAW**

| Block 1 | Block 2 | Block ... | Block N |
|---|---|---|---|
| **Bob's TX** | | | |

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)

(with later help from Adam Back/Rusty Russell!)

**Alice's TX**

**WITHDRAW**

| Block 1 | Block 2 | Block ... | Block N |
|---|---|---|---|
| **Bob's TX** | | | |

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**Alice's TX**

| Block 1 | Block 2 | Block ... | Block N |
|---|---|---|---|
| **Bob's TX** | | | **WITHDRAW** |

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**Alice's TX**

**Bob's TX**

**What if Alice cheats?**

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**Alice's**
**REVOKED TX**

Block 1

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**Alice's**
**REVOKED TX**

Block 1

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

Block 1

**Alice's REVOKED TX**

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**REVOKE PROOF**

Block 1 | Block 2

| **Alice's REVOKED TX** | |
|---|---|

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**REVOKE PROOF**

| Block 1 | Block 2 |
| --- | --- |
| **Alice's REVOKED TX** | |

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)
(with later help from Adam Back/Rusty Russell!)

**Alice's REVOKED TX**

**REVOKE PROOF**

| Block 1 | Block 2 |
| --- | --- |
| **Alice's REVOKED TX** | **REVOKE PROOF** |

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# Lightning Channels

Poon & Dryja (2014)

(with later help from Adam Back/Rusty Russell!)

**Alice's REVOKED TX**

**BAM!**
**Bob gets all the coins**

| Block 1 | Block 2 |
|---|---|
| **Alice's REVOKED TX** | **REVOKE PROOF** |

**The Dispute Process**

Bob can initiate the dispute process:

broadcast a state to the blockchain

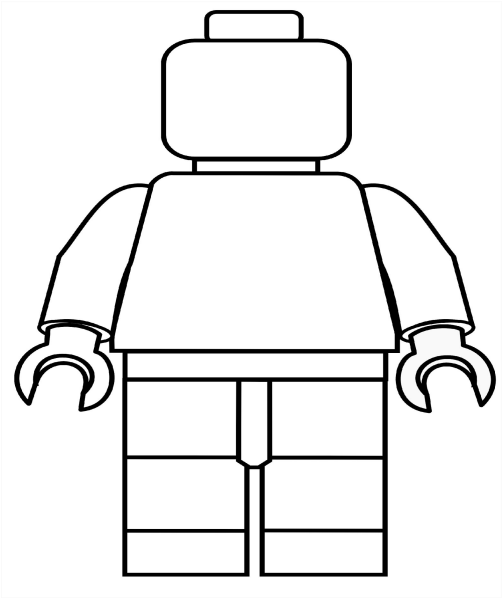
tion

**Chall : Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel

# A Trusted Third Party with Public State

**Hey, I'm a smart contract.**



**I promise you the following:**

1. I will never modify or change your code.
2. I will always run the code you tell me too (assuming the code itself allows me!).
3. I will never let code execution "stop half way" it is ALL or NOTHING with me.
4. I like to gossip and I can't keep secrets - Everything you tell me will be public knowledge.

# Ethereum Payment Channel:
## Easy Approach

**State 1**

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain
i.e. their transaction

Block 1

# Ethereum Payment Channel:
## Easy Approach

**State 1**

Block 1

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain
i.e. their transaction

# Ethereum Payment Channel:
## Easy Approach

**State 1**

Block 1

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain
i.e. their transaction

# Ethereum Payment Channel:
## Easy Approach

**State 1**
**Alice:** 0.9 ETH
**Bob:** 1.1 ETH
**Counter:** 1

Block 1

**State 1**

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain
i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

# Ethereum Payment Channel:
## Easy Approach

**State 1**
**Alice:** 0.9 ETH
**Bob:** 1.1 ETH
**Counter:** 1

Block 1 Block 2

| State 1 | |
|---|---|

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain
i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

# Ethereum Payment Channel:
## Easy Approach

**State 1**
**Alice:** 0.9 ETH
**Bob:** 1.1 ETH
**Counter:** 1

**State 2**

| Block 1 | Block 2 | Block 3 |
|---|---|---|
| **State 1** | | |

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain
i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

# Ethereum Payment Channel:
## Easy Approach

**State 1**
**Alice:** 0.9 ETH
**Bob:** 1.1 ETH
**Counter:** 1

**State 2**

Block 1 | Block 2 | Block 3

| | | |
|---|---|---|
| **State 1** | | |

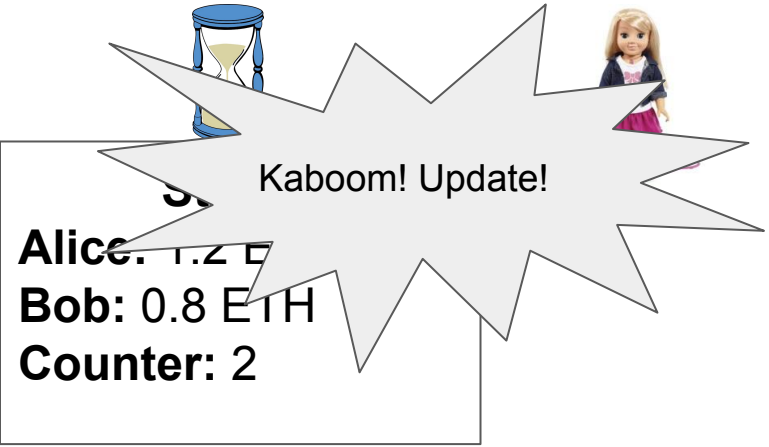
**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain
i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

# Ethereum Payment Channel:
## Easy Approach




| Block 1 | Block 2 | Block 3 |
|---|---|---|
| **State 1** | | **State 2** |

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain
i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

# Ethereum Payment Channel:
## Easy Approach

**State 2**
**Alice:** 1.2 ETH
**Bob:** 0.8 ETH
**Counter:** 2

| Block 1 | Block 2 | Block 3 |
|---|---|---|
| **State 1** | | **State 2** |

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain
i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

# Ethereum Payment Channel:
## Easy Approach

**State 2**
**Alice:** 1.2 ETH
**Bob:** 0.8 ETH
**Counter:** 2

| Block 1 | Block 2 | Block 3 | Block ... | Block N |
|---|---|---|---|---|
| **State 1** | | **State 2** | | |

**The Dispute Process**

Alice or Bob can initiate the dispute process:

**Initiate:** Broadcast a state to the blockchain
i.e. their transaction

**Challenge:** Fixed time period for counterparty to prove state is revoked

**Withdraw:** After fixed time period, withdraw coins from channel