MADS

Emmanuelle Anceaume

Blockchain Technology and the Bitcoin Cryptocurrency

http://people.irisa.fr/Emmanuelle.Anceaume/
Content of this lesson

- Crypto background
  - hash functions
  - digital signatures
  - hash pointers
  - Merkle trees
- Bitcoin principles
  - Peer-to-peer networks
  - Transactions
  - Blocks
- Agreement protocols
  - Leader election
  - « Nakamoto Consensus »
Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

Abstract. A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.
An overview of Bitcoin

- Bitcoin is a cryptocurrency and payment system
  - It allows users to anonymously exchange goods against digital currency
- All the valid transactions are recorded in a public ledger, the blockchain
  - Allows anyone to audit and check the integrity of all the transactions

Ledger:

- Bob -> Alice ฿0.001
- Chunk -> Sara ฿0.05
- Eva -> Alice ฿0.009
- Alice -> John ฿0.02
- Bob -> Chunk ฿0.7
- Peter -> Bob ฿0.008
- Bob -> Alice ฿0.05
- Bob -> Alice ฿0.046
- Bob -> Alice ฿0.008
Public ledgers

No trustworthy centralized control. Everyone

- maintains its own copy of the ledger
- disseminates all the transactions
- can read all the transactions

Achieving

- consistency, efficiency and availability of the ledger
- auditability, immutability of the transactions

While preventing

- transaction censorship and double-spending attacks
Basic principles

- **Crypto currency**
  - relies on cryptographic tools
- **Decentralized system**
  - peer-to-peer architecture
- **Trustless model**
  - does not require a central server to validate/abort financial transactions but requires participants to be online
- **Anonymous users**
  - neither sellers nor buyers use their real identities to use Bitcoins but if you are not careful your transactions can be tied together

Bitcoin relies on a set of distributed algorithms

**Cryptographic primitives**
- Bitcoins are immutable
  - Hash functions
  - Digital signature
  - Merkle tree

**Mining & Incentive**
- Approximate the selection of a leader randomly chosen
- Proportional to a resource we hope nobody can monopolize (denial of service)
  - proof of work using hash puzzle (moderately difficult to compute, parametrizable cost, trivial to verify)
- Regulate the creation of new bitcoins

**Blockchain & Agreement**
- No trusted entity in charge of validating transactions nor creating money
- Total and secure ordering of all the valid transactions ever created
- Append only data structure

**Peer-to-Peer Network**
- Dissemination of transactions and blocks in the open and large scale Bitcoin system
Preliminaries on crypto

- cryptographic hash functions
- digital signatures
- Merkle tree
hash functions

All currencies need some way to control supply and prevent counterfeiting money

- Fiat currencies (Dollar, Euro, Yen, Yuan)
  - central banks mint physical currency
  - integrity of bank notes is guaranteed by anti-counterfeiting features to physical currency

- Digital currencies
  - a string of « 0 » and « 1 »
  - no central bank to prevent double-spending attacks
  - heavy use of cryptography
Hash functions

A hash function is an algorithm that allows to compute a fingerprint of fixed size from data of arbitrary size

\[ H : 0, 1^* \rightarrow 0, 1^n \]

\[ M \mapsto H(M) \]

- rather than manipulating data of arbitrary size, a fingerprint is associated to each data which makes operation easier
Hash functions

A hash function satisfies the following properties

- The input space is the set of strings of arbitrarily length
  - « hello world » and « hellohellohello world » are perfectly fine inputs
- The output space is the set of strings of fixed length
  - \( H(« \text{hello world } ») = 000223 \)
  - \( H(« \text{hellohellohello world } ») = 130554 \)
- \( H \) is deterministic
- \( H \) is efficiently computable
  - Given a string \( s \) of length \( n \) the complexity to compute \( H(s) \) is \( O(n) \)

In addition to these properties, crypto-hash functions have additional requirements
Properties of cryptographic hash functions

- **Collision resistance**
  
  It must be difficult to find two inputs $x$ and $x'$ such that
  
  $$H(x) = H(x')$$

- **Second pre-image resistance**
  
  Given an input $x$, it must be difficult to find an input value $x' \neq x$ such that
  
  $$H(x') = H(x)$$

- **Pre-image resistance**
  
  Given $z$, it must be difficult to find an input value $x$ such that
  
  $$H(x) = z$$
Collision resistance

Find two inputs $x$ and $x'$ such that $H(x) = H(x')$
Collision resistance

collisions do exist

possible inputs

possible outputs

Image source: Bitcoin and Cryptocurrency Technologies.
Collision resistance

collisions do exist

possible inputs

possible outputs

but can anyone find them?

Image source: Bitcoin and Cryptocurrency Technologies.
Collision resistance property

Find two inputs $x$ and $x'$ such that $H(x) = H(x')$

Generic attack (i.e., a technique capable of attacking any $n$-bit hash function)

- Choose $2^{n/2}$ random messages (birthday paradox)
- Compute the hashed values and store them
- Find one pair $(x, x')$ such that $H(x) = H(x')$
Birthday paradox

Birthday paradox is about the probability that, in a set of \( m \) randomly chosen people, some pair of them will have the same birthday.

- if \( m = 23 \) the probability to have collision is 50%
- if \( m = 70 \) then \( p \) is equal to 99.9%
Birthday paradox

Let us first compute the probability that no two persons have the same birthday. Let \( p'(m) \) be this probability

\[
p'(m) = \frac{365}{365} \frac{364}{365} \ldots \frac{365 - (m - 1)}{365} \frac{365!}{365^m (365 - m)!}
\]

Thus the probability \( p(m) \) that there exists two persons having the same birthday is

\[
p(m) = 1 - p'(m) = 1 - \frac{365!}{(365 - m)!} \frac{1}{365^m}
\]

\[
\approx 1 - e^{-\frac{m(m-1)}{2 \times 365}}
\]

Thus

\[
m(p) \approx \sqrt{2 \times 365 \ln\left(\frac{1}{1 - p}\right)}
\]
Birthday paradox

\[ m(p) \approx \sqrt{2 \times 365 \ln(\frac{1}{1-p})} \]

we get

\[ m(1/2) = 23 \]

In our case, the set of possible values is equal to \(2^n\) with \(n\) the length of the binary string of the fingerprint.

Thus

\[ m(1/2) \approx \sqrt{2 \ln 2} \ 2^{n/2} \]
\[ \approx 2^{n/2} \]
Collision resistance property

Find two inputs $x$ and $x'$ such that $H(x) = H(x')$

Generic attack (i.e., a technique capable of attacking any hash function)

- Choose $2^{n/2}$ random messages
- Compute the hashed values and store them
- Find one pair $(x, x')$ such that $H(x) = H(x')$

If a computer calculates 10,000 hashes/s

- it would take $10^{27}$ years to output $2^{128}$ hashes, and
- thus $10^{27}$ years to produce a collision with probability $1/2$

Astronomical number of computations!!

So far no hash functions have been proven to be collision resistant
Collision resistance property

To summarize:

Collision resistant hash functions allows us

- to identify data by its hashed value (i.e digest, fingerprint)
  - if $H(x) = H(y)$ then it is safe to assume that $x = y$
- Bitcoin :
  - to identify blocks in the blockchain
  - to make blocks resistant to tampering (modifying a single bit changes the fingerprint)
Second-preimage resistance

Given an input $x$, it is difficult to find an input value $x' \neq x$ such that $H(x') = H(x)$

Generic Attack: probabilistic search

- Given $x$ and its hashed value $H(x)$ ($n$ bits value)
- Randomly choose $x_i$ and compute $z_i = H(x_i)$
- $\text{Proba}(z_i = H(x)) = 1/2^n$
- Thus after having chosen $2^n$ inputs it is likely that one can find a pre-image $x_i \neq x$ such that $H(x_i) = H(x)$
File integrity

Assume you have file 1

\[ H(\text{file1}) = 3214\ 5670\ ab67\ 0123\ 8760\ 2123\ 34BF\ 0A23 \]

(256 bits)

Is it possible to build another file, file 2, such that both files have the same fingerprint?
Preimage resistance

Given $z$, find an input value $x$ such that $H(x) = z$

Generic Attack : probabilistic search

- Given a hashed value $z$
- Randomly choose $x_i$ and compute $z_i = H(x_i)$
- $\text{Proba}(z_i = z) = 1/2^n$
- Thus after having chosen $2^n$ inputs it is likely that one can find a pre-image $x_i$ such that $H(x_i) = z$
Passwords storage

- In your machine, passwords are not stored. Only their hashed value is stored.
- When you want to authenticate, the login pg computes the hashed value, which is compared with the one stored in `/etc/passwd`.

Property: Given the hashed value $y$, it must be difficult to find $x$ such that $H(x) = H(password) = y$. 

Merkle-Damgard construction
A hash pointer is the cryptographic hash value of the pointed information.
Hash pointers

Hash pointers allows the construction of a log data structure that allows the detection of any manipulation

\[ H(\text{bloc}) = 45etb \]
\[ H(\text{bloc}) = 6736a \]
\[ H(\text{bloc}) = 1b781 \]
\[ H(\text{bloc}) = 56ac3 \]
Hash pointers allows the construction of a log data structure that allows the detection of any manipulations.
Hash pointers allows the construction of a log data structure that allows the detection of any manipulations.
Hash pointers

Hash pointers allows the construction of a log data structure that allows the detection of any manipulations.

By only keeping the hash pointer of the head of the data structure, we have a tamper-evident hash of a possibly very long list.
Hash tree : Merkle Tree

A Merkle tree\(^1\) is a tree of hashes

- Leaves of the tree are data blocks
- Nodes are the hashes of their children
- Root of tree is the fingerprint of the tree

Hash tree: Merkle Tree

$h = h( h_0 \parallel h_1 )$

$h_0 = h( h_{00} \parallel h_{01} )$

$h_{00} = h( h_{000} \parallel h_{001} )$

$h_{000} = h( b_0 )$

$h_{001} = h( b_1 )$

$h_{01} = h( h_{010} \parallel h_{011} )$

$h_{010} = h( b_2 )$

$h_{011} = h( b_3 )$

$h_{0} = h( h_{00} \parallel h_{01} )$

$h_{10} = h( h_{100} \parallel h_{101} )$

$h_{100} = h( b_4 )$

$h_{101} = h( b_5 )$

$h_1 = h( h_{10} \parallel h_{11} )$

$h_{11} = h( h_{110} \parallel h_{111} )$

$h_{110} = h( b_6 )$

$h_{111} = h( b_6 )$
Hash tree: Merkle Tree

✓ Checking the integrity of the $n$ data blocks of the tree
  ▶ easy due to collision resistance property of crypto. hash functions
  ▶ Data blocks membership
    ▶ checked with $\log n$ pieces of information and in $\log n$ operations
Hash tree: Merkle Tree

\[ h = h( h_0 \| h_1 ) \]

\[ h_0 = h( h_{00} \| h_{01} ) \]
\[ h_{00} = h( h_{000} \| h_{001} ) \]
\[ h_{000} = h(b_0) \]
\[ h_{001} = h(b_1) \]
\[ h_{01} = h( h_{010} \| h_{011} ) \]
\[ h_{010} = h(b_2) \]
\[ h_{011} = h(b_3) \]

\[ h_1 = h( h_{10} \| h_{11} ) \]
\[ h_{10} = h( h_{100} \| h_{101} ) \]
\[ h_{100} = h(b_4) \]
\[ h_{101} = h(b_5) \]
\[ h_{11} = h( h_{110} \| h_{111} ) \]
\[ h_{110} = h(b_6) \]

\[ b_0 \]
\[ b_1 \]
\[ b_2 \]
\[ b_3 \]
\[ b_4 \]
\[ b_5 \]
\[ b_6 \]
Hash tree : Merkle Tree

I know the root of the Merkle tree, and I would like to know whether data block $b_3$ belongs to the tree?

Question : How can I do that without looking for the full tree?
Hash tree: Merkle Tree

\[ h = h(h_0 \| h_1) \]
Hash tree: Merkle Tree

\[ h = h( h_0 \| h_1 ) \]

\[ h_{011} = h(b_3) \]

\[ b_3 \]
Hash tree: Merkle Tree

\[ h = h( h_0 \| h_1 ) \]

\[ h_{01} = h( h_{010} \| h_{011} ) \]

\[ h_{010} = h(b2) \]

\[ h_{011} = h(b3) \]

\[ b_3 \]
Hash tree: Merkle Tree

\[ h_{00} = h( h_{000} \| h_{001} ) \]

\[ h_{010} = h(b2) \]

\[ h_{011} = h(b3) \]

\[ h_{01} = h( h_{010} \| h_{011} ) \]

\[ h_0 = h( h_{00} \| h_{01} ) \]
Hash tree: Merkle Tree

$h = h( h_0 || h_1 )$

$h_0 = h( h_{00} || h_{01} )$

$h_{00} = h( h_{000} || h_{001} )$

$h_{01} = h( h_{010} || h_{011} )$

$h_{010} = h( b_2 )$

$h_{011} = h( b_3 )$

$b_3$
Hash tree: Merkle Tree

$h = h(\, h_0 \, || \, h_1 \, )$

$h_0 = h(\, h_{00} \, || \, h_{01} \, )$

$h_{00} = h(\, h_{000} \, || \, h_{001} \, )$

$h_{01} = h(\, h_{010} \, || \, h_{011} \, )$

$h_{010} = h(b_2)$

$h_{011} = h(b_3)$

$b_3$
Hash tree: Merkle Tree

\[
\begin{align*}
&h = h(h_0 \| h_1) \\
&h_0 = h(h_{00} \| h_{01}) \\
&h_1 = h(h_{10} \| h_{11}) \\
&h_{00} = h(h_{000} \| h_{001}) \\
&h_{01} = h(h_{010} \| h_{011}) \\
&h_{010} = h(b_2) \\
&h_{011} = h(b_3)
\end{align*}
\]
Hash tree: Merkle Tree

I know the root of the Merkle tree, and I would like to know whether data block $b_3$ belongs to the tree?

Question: How can I do that without looking for the full tree?

I need $\log n$ pieces of information and $\log n$ hash operations
Digital signature primitive

A digital signature is just like a signature on a document

- Only the creator of the document can sign, but anyone can verify it
- Signature is tied to a particular document

How can we build such a digital signature?
Digital signature

Three functions:

- \((s_k, p_k) := \text{generateKeys}(\text{keysize})\)
  - \(s_k\) : private signing key
  - \(p_k\) : public verification key

- \(\text{sig} := \text{sign}(s_k, H(\text{message}))\)

- \(\text{ver} := \text{verify}(p_k, \text{sig})\)
Digital signature

Requirements:

- The verify operation must return true when fed with valid signatures
  \[ \text{verify}(p_k, \text{sign}(s_k, H(\text{message}))) = H(\text{message}) \]

- The signature scheme is unforgeable
  An adversary that knows \( p_k \) and can choose any messages to be signed cannot produce a verifiable signature for another message
Digital signature

Alice

\[ M \]

\[ H(M) \rightarrow H(M) = 01011011 \]

\[ \text{SIG}(H(M), s_k) = \text{sig} \]

Bob

\[ (M, \text{sig}) \rightarrow H(M) \]

\[ \text{VER}(p_k, \text{sig}) = \text{ver} \]

if (ver = H(M)) then sig is valid
Digital signature

- The algorithms to generate keys and sign must have access to a good source of randomness
- Signing the hash of a message is as safe as signing the message itself

In Bitcoin, the signature scheme is ECDSA (Elliptic Curve Digital Signature Algorithm)\(^2\)

- private key = 256 bits
- Public key = 512 bits
- Message = 256 bits
- signature = 512 bits

---

Using verification public key as an identity

Idea: use the verification key of a signature as an identity

- If you see a msg such that the signature verifies under $p_k$ (i.e. $\text{verify}(p_k, \text{sig}) = \text{msg}$) then on can see $p_k$ as a party saying statements by signing them
- To speak on behalf of $p_k$ one must know $s_k$
- So there is an identity in the system such that only a single one can speak for it which is what we want for a signature

✓ By looking at public keys as identities you can generate as many identities as you want!
Using verification public key as an identity

- Create new identities:
  - Eric creates a new pair \((s_k, p_k)\)
  - \(p_k\) is the public name Eric uses
  - Eric is the only person that can speak on behalf of \(p_k\) because he knows \(s_k\)
  - \(p_k\) is sufficient! nobody needs to know that Eric created it
- Creation of identities as often as you want!
  - no central authority in charge of registering new identities!
  - this is the way Bitcoin creates identities (called addresses)
    - address = Hash(public key)
Using verification public key as an identity

Some words on privacy

- no relationships between $p_k$ based identities and real identities
- by using the same $p_k$ (identity) an adversary can infer some relationships based on the activity of $p_k$
Transaction

The state of the Bitcoin world is represented by transactions

- data structure that allows Alice to transfer bitcoins to Bob and Charly
- does not contain any confidential information
- verifiable by anyone → no trusted third-party!

- Alice’s account (input address)
- Bob’s and Charly’s accounts (output address)
- possibly some change
- transactions fees
Transaction

An account is a « one shot object »

- an account is a $<(\text{private key}, \text{public key}), \text{amount of bitcoins}>$
- debited once!
- each time you are the recipient of a transaction, it's safer if you create a new account (privacy reasons)

Checking the ownership of an account

- how can anyone check that I am the legitimate owner of an account since there are no identities in a transaction?
- using « public keys » as identities
Transaction

- Bitcoin relies on a (limited) script language
- to lock an output, the script provides a challenge
  - *i.e.*, fingerprint of the recipient public key, $H(p_k)$
- to unlock an input, the script provides the solution of the challenge
  - *i.e.*, public key $p_k$ together with $s_k$’s signature
Decoded:

- **Header**: `ver=1, vin.size=1, vout.size=2, nLockTime=438069`
- **Inputs**:  
  - ID: `4983442ccd814372b8c8614bf81942ba17c6e3470f21897524a63caf54aa54ba`
  - Index: 1 (input value: 0.030 980 35 BTC)
  - `scriptSig`:
    - Signature: `304402201e0b0555330b9ba6dc689aeebecf0464391a882c41a6650ab66f803179860a1802207d1b46c45d37e8a88fee49c3e02adb9cd3ccb4bb96ca313135e00a5c01f71a6b[ALL]`
    - Key: `02374f390070a14763707fe9310a73eaf2b2221734d0ff0a0684078571e2a12e9e`
- **nSeq**: `4294967294`
Decoded (ctd):

Outputs:

- n: 0
  - value: 0.01685815 BTC
  - ScriptPubKey: 0P_DUP 0P_HASH160 3d5b9da23ff21a211f101ee2adec37d6b79d7c 0P_EQUALVERIFY 0P_CHECKSIG

- n: 1
  - value: 0.01000000 BTC
  - ScriptPubKey: 0P_DUP 0P_HASH160 4ce03f31d4bdbc2932f14cea99f4d96edcdeb0c 0P_EQUALVERIFY 0P_CHECKSIG
Script language

Transaction T

Input ----

Output

OP_DUP OP_HASH160 <pubKeyHash>
OP_EQUALVERIFY OP_CHECKSIG

Transaction T'

Input

<sig> <pubKey>

Output ----

<sig> <pubKey>

OP_DUP

<sig> <pubKey>

OP_HASH

<sig> <pubKey>

OP_EQUALVERIFY

<sig> <pubKey>

OP_CHECKSIG

if true empty
Relying on past transactions to create new ones

Transaction 20ab3701i

Input #1
Output ref. + script

Output #1
account + ฿ + script

Output #2
account + ฿ + script

UTXO

Transaction 74201ab3c

Input #1
Output ref. + script

Output #1
account + ฿ + script

Input #2
Output ref. + script

Output #1
account + ฿ + script

UTXO

Transaction 1206ac34e

Input #1
Output ref. + script

Output #1
account + ฿ + script

Input #2
Output ref. + script

Output #2
account + ฿ + script

Output #3
account + ฿ + script

UTXO

UTXO = Unspent Transaction Output

Do not forget Tx fees, i.e., ฿ input > ฿ output
Peer-to-Peer Network

- Topology formed through a randomized process
- No coordinating entity
- Broadcast is based on flooding/gossiping neighbors’ link
Peer-to-Peer Network
Resistance to « double-spending attacks »

The diagram illustrates the network of transactions. Each node represents a part of the network, and transactions are exchanged between them. The Internet serves as the backbone connecting these nodes. The diagram also shows a bag of transactions, indicating the flow of transactions within the network.
Required properties

1. Connectivity
   ▶ Each node should receive any broadcast information

2. Low latency
   \[
   \frac{\text{msg. transmission time}}{\text{block time interval}} \ll 1
   \]
   ✓ Allows to keep the probability of fork small (i.e. \(10^{-3}\))
   ✓ PoW mechanism allows this ratio to continuously hold
   ✗ No more than 7 trans/s can be permanently confirmed in average!!

A chain of blocks: the blockchain

A publicly, immutable, and totally ordered log of transactions
Block of transactions

Transaction 945846
Transaction 58801a
Transaction 665389
Transaction 7654ab
.
.
.
Transaction 321456
Blocks of transactions

- Transaction 945846
- Transaction 58801a
- Transaction 665389
- Transaction 7654ab
- ...
- Transaction 321456

- Transaction 437621
- Transaction 8593ab
- Transaction 12367b
- Transaction 793154
- ...
- Transaction 653278

- Transaction 336789
- Transaction 7245ab
- Transaction 635566
- Transaction 12f4a22
- ...
- Transaction 232356
Resistant to attacks
Resistant to attacks
A chain of blocks: the blockchain

A publicly, immutable, and totally ordered log of transactions

- Why digital signatures are not sufficient?
- How can we securely link blocks?
- How can we prevent transactions in a block from being manipulated?
- Who is allowed to create blocks?
- Who is in charge of checking that blocks are correctly created?
A chain of blocks: the blockchain

A publicly, immutable, and totally ordered log of transactions

- Why digital signatures are not sufficient?
  - do not prevent double-spending attacks
- How can we securely link blocks?
- How can we prevent transactions in a block from being manipulated?
- Who is allowed to create blocks?
- Who is in charge of checking that blocks are correctly created?
A chain of blocks : the blockchain

A publicly, immutable, and totally ordered log of transactions

- Why digital signatures are not sufficient?
  - ✗ do not prevent double-spending attacks
- How can we securely link blocks?
  - ✓ by linking them with cryptographic fingerprints
- How can we prevent transactions in a block from being manipulated?
- Who is allowed to create blocks?
- Who is in charge of checking that blocks are correctly created?
A chain of blocks: the blockchain

A publicly, immutable, and totally ordered log of transactions

- Why digital signatures are not sufficient?
  - do not prevent double-spending attacks

- How can we securely link blocks?
  - by linking them with cryptographic fingerprints

- How can we prevent transactions in a block from being manipulated?
  - efficient cryptographic fingerprint of the transactions

- Who is allowed to create blocks?

- Who is in charge of checking that blocks are correctly created?
A chain of blocks: the blockchain

A publicly, immutable, and totally ordered log of transactions

- Why digital signatures are not sufficient?
  - ✗ do not prevent double-spending attacks
- How can we securely link blocks?
  - ✓ by linking them with cryptographic fingerprints
- How can we prevent transactions in a block from being manipulated?
  - ✓ efficient cryptographic fingerprint of the transactions
- Who is allowed to create blocks?
  - ✓ any node in the network capable of solving a given challenge
- Who is in charge of checking that blocks are correctly created?
A chain of blocks: the blockchain

A publicly, immutable, and totally ordered log of transactions

- Why digital signatures are not sufficient?
  - ❌ do not prevent double-spending attacks
- How can we securely link blocks?
  - ✔ by linking them with cryptographic fingerprints
- How can we prevent transactions in a block from being manipulated?
  - ✔ efficient cryptographic fingerprint of the transactions
- Who is allowed to create blocks?
  - ✔ any node in the network capable of solving a given challenge
- Who is in charge of checking that blocks are correctly created?
  - ✔ everyone!!
How can we securely link blocks?

Cryptographic Hash functions

- allows you to compute a fixed-size fingerprint from any type of data
- deterministic function
- efficiently computable
- practically impossible to invert (« one-way function »)

More on hash functions
How can we prevent transactions of a block from being manipulated?

**Merkle tree**

A Merkle tree is a tree
- Leaves of the tree are data blocks
- Nodes are the cryptographic hashes of their children
- Root of tree is the fingerprint of the tree

*More on Merkle tree*
Who is allowed to create blocks?

Miners, i.e., any node capable of solving a highly complex computational puzzle

- A leader election algorithm is run among all the miners
  - Probabilistic oracle
  - Synchronize the creation of blocks
  - Prevent Sybil attacks
  - Incentivize correct behavior through currency creation
Minting money to incentivize correct behavior

- Nodes are working (racing) to solve the computational puzzle
- This is in exchange for monetary rewards
  - Coinbase transaction = 12.5 bitcoins + transaction fees
  - Started at 50 bitcoins and is halved every 210,000 blocks
- This incentivizes miners to only work on valid blocks
  - « valid block »: for miners, this means blocks for which the majority will accept to build upon
Blockchain Proof-of-Work

Version: 536870912
Parent block: 000000000000000000000287d...
Merkle root: e1f2e6de5580056c0469ab8c8b3e4e04f611701cf36
0253182103dccec494a2b441ac1dc93314f1577ff8ce44cfe59...
fcbd4a74e1659

Transactions

2478

2016-11-09
14:54:31
Bits: 402936180
Nonce: 1381339156
Blockchain Proof-of-Work

Version: 536870912
Parent block: 000000000000000000000287d...
e1f2e6de5580056c0469ab8c8b3e4e04f611701cf36
Merkle root:
0253182103dccec494a2b441ac1dc93314f1577ff8ce44cfe59...
fcbd4a74e1659

Goal: SHA256 ◦ SHA256(header) ≤ target

Every 2016 blocks:
\[ \text{target} \leftarrow \text{target} \times \max\left(\frac{1}{4}, \min(4, \frac{\text{real}}{\text{expected}})\right) \]

- Comes down to compute a hash partial inversion
  - find pre-image for a partially specified hash function output
Proof-of-Work

string=HelloWorld!, nonce=0, difficulty=000
Proof-of-Work

string=HelloWorld!, nonce=0, difficulty=000

HelloWorld!0:3f6fc92516327a1cc4d3dca5ab2b27aeedf2d459a77fa06fd3c6b19fb6
Proof-of-Work

string=HelloWorld!, nonce=1, difficulty=000
Proof-of-Work

string=HelloWorld!, nonce=2, difficulty=000

HelloWorld!0:3f6fc92516327a1cc4d3dca5ab2b27aeedf2d459a77fa06fd3c6b19fb6
HelloWorld!1:b5690c48c2d0a09481186aaa99e4e090901ff2ac4d572e6706dfd30eef
HelloWorld!2:5b6fd9c27fcb54ca23404d9428f081b7c9280ba6370e33a6a20b16f40c
Proof-of-Work

string=HelloWorld!, nonce=3, difficulty=000

HelloWorld!0:3f6fc92516327a1cc4d3dca5ab2b27aeedf2d459a77fa06fd3c6b19fb65
HelloWorld!1:b5690c48c2d0a09481186aaa99e4e090901ff2ac4d572e6706dfd30eef
HelloWorld!2:5b6fd9c27fcb54ca23404d9428f081b7c9280ba6370e33a6a20b16f40c8
HelloWorld!3:9c5d769416aa0ca894abf22bd17bd30fbb6959291423ae1903a9f86a1f7
....
Proof-of-Work

string=HelloWorld!, nonce=94, difficulty=000

HelloWorld!0:3f6fc92516327a1cc4d3dca5ab2b27aeedf2d459a77fa06fd3c6b19fb6
HelloWorld!1:b5690c48c2d0a09481186aaa99e4e090901ff2ac4d572e6706dfd30eef
HelloWorld!2:5b6fd9c27fcb54ca23404d9428f081b7c9280ba6370e33a6a20b16f40c
HelloWorld!3:9c5d769416aa0ca894abf22bd17bd30fbb6959291423ae1903a9f86a1f
....
HelloWorld!94:7090a0e5d88cff635e42ea33fcd6091a058e9cdd58ab8cd5c21c1c704
Proof-of-Work

string=HelloWorld!, nonce=95, difficulty=000

HelloWorld!0:3f6fc92516327a1cc4d3dca5ab2b27aeedf2d459a77fa06fd3c6b19fb6
HelloWorld!1:b5690c48c2d0a09481186aaa99e4e090901ff2ac4d572e6706dfd30eff
HelloWorld!2:5b6fd9c27fcb54ca23404d9428f081b7c9280ba6370e33a6a20b16f40c
HelloWorld!3:9c5d769416aa0ca894abf22bd17bd30fbb6959291423ae1903a9f86a1f
....
HelloWorld!94:7090a0e5d88cff635e42ea33fcd6091a058e9cdd58ab8cd5c21c1c704
HelloWorld!95:b74f3b2cf1061895f880a99d1d0249a8cedf223d3ed061150548aa621
HelloWorld!96:447ca2fa886965af084808d22116edde4383cbaa16fd1fbcf3db61421
Proof-of-Work

string=HelloWorld!, nonce=97, difficulty=000,

HelloWorld!0:3f6fc92516327a1cc4d3dca5ab2b27aeedf2d459a77fa06fd3c6b19fb6
HelloWorld!1:b5690c48c2d0a09481186aaa99e4e090901ff2ac4d572e6706dfd30eef
HelloWorld!2:5b6fd9c27fcb54ca23404d9428f081b7c9280ba6370e33a6a20b16f40c
HelloWorld!3:9c5d769416aa0ca894abf22bd17bd30fbb6959291423ae1903a9f86a1f
....
HelloWorld!94:7090a0e5d88cff635e42ea33fcd6091a058e9cdd58ab8cd5c21c1c704
HelloWorld!95:b74f3b2cf1061895f880a99d1d0249a8cedf223d3ed061150548aa621
HelloWorld!96:447ca2fa886965af084808d22116edde4383cbba16fd1fbcf3db61421
HelloWorld!97:000ba61ca46d1d317684925a0ef070e30193ff5fa6124aff76f513d96
Blockchain Proof-of-Work

Properties

- Algorithmically computable
- Difficult to solve but quickly verified
- Difficulty proportional to computation power
  - average generation time: 10 minutes
  - probability $p$ that one try is successful is $p = D/2^\kappa$ with $\{0, 1\}^\kappa$ the range of $H(\cdot)$ and $D$ the difficulty
  - around 23,000,000 $\times 10^{12}$ hash/s
- Memoryless process
Blockchain construction

(B₀, ⊥) (B₀, ⊥) (B₀, ⊥) (B₀, ⊥) (B₀, ⊥) (B₀, ⊥) (B₀, ⊥)

A  B  C  D  E  F  G
Blockchain construction

\[(B_0, \bot)\]  \[(B_0, \bot)\]  \[(B_0, \bot)\]  \[(B_0, \bot)\]  \[(B_0, \bot)\]  \[(B_0, \bot)\]  \[(B_0, \bot)\]
Blockchain construction

A

Block !

B

C

D

E

F

G
Blockchain construction

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>((B_1, B))</td>
<td>((B_1, B))</td>
<td>((B_1, B))</td>
<td>((B_1, B))</td>
<td>((B_1, B))</td>
<td>((B_1, B))</td>
<td>((B_1, B))</td>
</tr>
<tr>
<td>((B_0, \bot))</td>
<td>((B_0, \bot))</td>
<td>((B_0, \bot))</td>
<td>((B_0, \bot))</td>
<td>((B_0, \bot))</td>
<td>((B_0, \bot))</td>
<td>((B_0, \bot))</td>
</tr>
</tbody>
</table>

A  B  C  D  E  F  G
Blockchain construction

\[(B_5, E)\]
\[(B_4, A)\]
\[(B_3, C)\]
\[(B_2, E)\]
\[(B_1, B)\]
\[(B_0, \bot)\]

\[(B_0, \bot)\]
\[(B_1, B)\]
\[(B_2, E)\]
\[(B_3, C)\]
\[(B_4, A)\]

\[(B_5, E)\]
\[(B_4, A)\]
\[(B_3, C)\]
\[(B_2, E)\]
\[(B_1, B)\]

\[(B_0, \bot)\]
\[(B_1, B)\]
\[(B_2, E)\]
\[(B_3, C)\]
\[(B_4, A)\]

\[(B_5, E)\]
\[(B_4, A)\]
\[(B_3, C)\]
\[(B_2, E)\]
\[(B_1, B)\]

\[(B_0, \bot)\]
\[(B_1, B)\]
\[(B_2, E)\]
\[(B_3, C)\]
\[(B_4, A)\]

\[(B_5, E)\]
\[(B_4, A)\]
\[(B_3, C)\]
\[(B_2, E)\]
\[(B_1, B)\]

\[(B_0, \bot)\]
\[(B_1, B)\]
\[(B_2, E)\]
\[(B_3, C)\]
\[(B_4, A)\]

\[(B_5, E)\]
\[(B_4, A)\]
\[(B_3, C)\]
\[(B_2, E)\]
\[(B_1, B)\]

\[(B_0, \bot)\]
\[(B_1, B)\]
\[(B_2, E)\]
\[(B_3, C)\]
\[(B_4, A)\]
Conflict: Transient inconsistencies

\[(B_6, A)\]
\[(B_5, E)\]
\[(B_4, A)\]
\[(B_3, C)\]
\[(B_2, E)\]
\[(B_1, B)\]
\[(B_0, \bot)\]

\[(B_0, \bot)\]
\[(B_1, B)\]
\[(B_2, E)\]
\[(B_3, C)\]
\[(B_4, A)\]
\[(B_5, E)\]

\[(B'_6, F)\]

A Block!
B
C
D
E
F Block!
G
Nakamoto conflict chain rule

Mine on the longest branch!

- Proba. that a block is removed from the blockchain decreases exponentially with the number $k$ of blocks appended to it.
- $k = \text{confirmation level}$
- When $k \geq 6$, this proba is very small
Conflict: Transient inconsistencies

\[
\begin{align*}
(B_7, G) & \quad (B_7', C) & \quad (B_7', C) & \quad (B_7, G) \\
(B_6, A) & \quad (B_6', F) & \quad (B_6', F) & \quad (B_6, A) \\
(B_5, E) & \quad (B_5, E) & \quad (B_5, E) & \quad (B_5, E) \\
(B_4, A) & \quad (B_4, A) & \quad (B_4, A) & \quad (B_4, A) \\
(B_3, C) & \quad (B_3, C) & \quad (B_3, C) & \quad (B_3, C) \\
(B_2, E) & \quad (B_2, E) & \quad (B_2, E) & \quad (B_2, E) \\
(B_1, B) & \quad (B_1, B) & \quad (B_1, B) & \quad (B_1, B) \\
(B_0, \bot) & \quad (B_0, \bot) & \quad (B_0, \bot) & \quad (B_0, \bot)
\end{align*}
\]
### Conflict: Transient inconsistencies

<table>
<thead>
<tr>
<th></th>
<th>(B₈, A)</th>
<th>(B₇, G)</th>
<th>(B₆, A)</th>
<th>(B₅, B)</th>
<th>(B₄, A)</th>
<th>(B₃, C)</th>
<th>(B₂, E)</th>
<th>(B₁, B)</th>
<th>(B₀, ⊥)</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>Block!</td>
<td>B</td>
<td>C</td>
<td>D</td>
<td>E</td>
<td>F</td>
<td>G</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*Note: The diagram depicts the various configurations of blocks (A, B, C, D, E, F, G) and their associated inconsistencies.*
Nakamoto consensus protocol

1. Leader election among all the miners
   - Probabilistic oracle
   - Synchronize the creation of blocks
   - Prevent Sybil attacks
   - Incentivize correct behavior through currency creation

2. A simple and local arbitration rule
   - Select the chain which required the greatest among of work
Nakamoto consensus protocol

1. Leader election among all the miners
   ▶ Probabilistic oracle
   ▶ Synchronize the creation of blocks
   ▶ Prevent Sybil attacks
   ▶ Incentivize correct behavior through currency creation

2. A simple and local arbitration rule
   ▶ Select the chain which required the greatest among of work
   ▶ Random nature of the PoW = one chain will be extended more than the others

\[ B_0 \rightarrow B_i \rightarrow B_{i+1} \]
The 50% attack

- Once a transaction has been inserted in a block $B$ of the blockchain, the sender of coins will typically expect some product or service in return.
- This is undermined if the sender would be able, after receiving the product, to broadcast a conflicting transaction sending the same coin back to itself, and such that by chance this non-legitimate transaction would appear in some concurrent block $B'$.
- By applying the Bitcoin rule, this concurrent block $B'$ might invalidate the legitimate one $B$ if $B'$ belongs to a longest chain than the one of $B$.

![Diagram of the 50% attack]
The 50% attack (con’t)

Thus as long as a legitimate recipient of a transaction is not sure that the coins it received will not be redirected to another party, it is safer for him to wait before delivering the product (otherwise the recipient would lose its product without receiving any coins)

The question is how long the recipient of transaction $T'$ must wait before delivering its product/service?

If an attacker controls a substantial computational power he may succeed in computing sufficiently many concurrent blocks so that eventually «his» branch becomes longer than the one mined by honest miners, and win its attack
The 50% attack

- A transaction is said to have \( n \) confirmations if it included in a block \( B \) which is part of the longest chain, and there are \( n \) blocks in the path from that block \( B \) to the leaf of the chain
- It is generally assumed that a transaction with enough confirmations is safe from double spending attacks
- We now study what it means « enough confirmations »
A game between two teams

We consider two teams:

- A bad team $Q$ which tries to make a successful double-spending attack
- A honest team $P$ which behave correctly

Suppose that the legitimate transaction $T$ belong to block $B$, and the conflicting one $T'$ belongs to block $B'$
A game between two teams

1. Secretly team $Q$ mines a chain of blocks $B' \ldots$ which builds on the latest block that precedes block $B$

2. Team $Q$ waits until block $B$ receives enough confirmations and the recipient, confident in its payment, sends the product

3. Team $Q$ continues to mine blocks in the secret chain (which contradicts block $B$) until this secret chain is longer than the public chain (the one that contains $B$)

4. Then team $B$ broadcasts its secret chain. This chain being longer that the public one, it is considered as the new valid one, and thus the payment to the recipient will be replaced by the payment to the attacker
A game between two teams

- Let $n$ be the number of blocks after the fork in the public chain (mined by team $P$)
- Let $m$ be the number of blocks that team $Q$ succeeded to mine after the fork
- Both teams are trying to win the competition
- The question is « Is team $Q$ will be able to catch-up with a longer chain? » or will the gap just keep increasing, leaving team $Q$ behind with a chain which will never be valid?
A game between two teams

- We suppose that the total hashrate of team $P$ and team $Q$ is constant and equal to $H$
  - Team $P$ has a hashrate of $p.H$
  - Team $Q$ has a hashrate of $q.H$, where $p + q = 1$

- Let $z = n - m$

- $z$ is the number of blocks by which the honest team ($P$) has an advantage on the attacker ($Q$)

- When team $P$ finds a block, $z$ increases by 1 and when team $Q$ finds a block, $z$ decreases by 1

- $z = -1$ then the attack of team $Q$ succeeds. If it never happens, the attack fails
A game between two teams

- This process is equivalent to a discrete time Markov chain.
- A step is defined as finding a block by team $P$ or team $Q$.
- We have at step $i+1$
  \[ z_{i+1} = z_i + 1 \] with probability $p$ and
  \[ z_{i+1} = z_i - 1 \] with probability $q$. 
A game between two teams

- Let $a_z$ be the probability that the attacker will be able to catch up when it is $z$ block behind the honest team (initial disadvantage = $z$)
- if $z < 0$ then $a_z = 1$ (the attacker has a longer branch)
- now, if $z \geq 1$, what is the value of $a_z$?
- if the next block which is found is found by the honest team, which happens with probability $p$, then the attacker will be $z + 1$ blocks behind, and its probability of success will be $a_{z+1}$
- if the next block which is found is found by the attacker, which happens with probability $q$, then the attacker will be $z - 1$ blocks behind, and its probability of success will be $a_{z-1}$.
- This leads to the following relation

$$a_z = pa_{z+1} + qa_{z-1}$$
A game between two teams

- We have
  \[ a_z = p a_{z+1} + q a_{z-1} \]
- Remember that \( p + q = 1 \), which leads to
  \[ a_z = 1 \text{ if } z < 0 \text{ or if } q > p \]
  \[ a_z = (q/p)^{z+1} \text{ if } z \geq 0 \text{ and } q \leq p \]

Remarks:

- If the attacker owns more than 50\% of the computational power, then he always succeeds catching up, from any gap \( z \).
- When \( q < p \) the probability of success decreases exponentially with gap \( z \). The lower \( q \) is, the faster the decay.
A game between two teams

- We have just seen that the probability that the attacker succeeds depends on $z$
- So what is a reasonable value for the number of confirmations $n$ for a merchant to be sure that his transaction will stay forever in the blockchain?
- We model $m$ as the number of successes (blocks found by the attacker) before $n$ failures (blocks found by the honest team). Thus $m$ is a negative binomial variable.
- The probability $P(q, m, n)$ of having $m$ success before $n$ failures is
  \[ P(q, m, n) = \binom{m + n - 1}{m} p^n q^m \]
- Once $n$ blocks have been mined by the honest team in a period of time during which $m + 1$ blocks have been found by the attacker, the race starts with $z = n - m - 1$
A game between two teams

Thus the probability for a double-spending attack to succeed when the merchant waits for \( n \) confirmations blocks is the probability of the attacker progressing from 1 pre-computed block to \( m \) blocks and then catching up from a difference of \( m - m \) blocks:

\[
d = \sum_{m=0}^{\infty} a_{n-m-1} P(q, m, n)
\]

\[
d = \sum_{m=0}^{n} a_{n-m-1} P(m) + \sum_{m=n+1}^{\infty} a_{n-m-1} P(m)
\]

\[
d = \sum_{m=0}^{n} a_{n-m-1} \binom{m + n - 1}{m} p^n q^m + \sum_{m=n+1}^{\infty} a_{n-m-1} \binom{m + n - 1}{m} p^n q^m
\]

\[
d = 1 - \sum_{m=0}^{n} \binom{m + n - 1}{m} (p^n q^m - q^n p^m) \text{ if } q < p
\]

\[
d = 1 \text{ if } q \geq p
\]
Probability of a successful double spend attack as a function of the attacker’s hashrate $q$ and the number of confirmations $n$

<table>
<thead>
<tr>
<th>$q$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>2%</td>
<td>4%</td>
<td>0.237%</td>
<td>0.016%</td>
<td>0.001%</td>
<td>≈ 0</td>
<td>≈ 0</td>
<td>≈ 0</td>
<td>≈ 0</td>
<td>≈ 0</td>
<td>≈ 0</td>
</tr>
<tr>
<td>4%</td>
<td>8%</td>
<td>0.934%</td>
<td>0.120%</td>
<td>0.016%</td>
<td>0.002%</td>
<td>≈ 0</td>
<td>≈ 0</td>
<td>≈ 0</td>
<td>≈ 0</td>
<td>≈ 0</td>
</tr>
<tr>
<td>6%</td>
<td>12%</td>
<td>2.074%</td>
<td>0.394%</td>
<td>0.078%</td>
<td>0.016%</td>
<td>0.003%</td>
<td>0.001%</td>
<td>≈ 0</td>
<td>≈ 0</td>
<td>≈ 0</td>
</tr>
<tr>
<td>8%</td>
<td>16%</td>
<td>3.635%</td>
<td>0.905%</td>
<td>0.235%</td>
<td>0.063%</td>
<td>0.017%</td>
<td>0.005%</td>
<td>0.001%</td>
<td>≈ 0</td>
<td>≈ 0</td>
</tr>
<tr>
<td>10%</td>
<td>20%</td>
<td>5.600%</td>
<td>1.712%</td>
<td>0.546%</td>
<td>0.178%</td>
<td>0.059%</td>
<td>0.020%</td>
<td>0.007%</td>
<td>0.002%</td>
<td>0.001%</td>
</tr>
<tr>
<td>12%</td>
<td>24%</td>
<td>7.949%</td>
<td>2.864%</td>
<td>1.074%</td>
<td>0.412%</td>
<td>0.161%</td>
<td>0.063%</td>
<td>0.025%</td>
<td>0.010%</td>
<td>0.004%</td>
</tr>
<tr>
<td>14%</td>
<td>28%</td>
<td>10.662%</td>
<td>4.400%</td>
<td>1.887%</td>
<td>0.828%</td>
<td>0.369%</td>
<td>0.160%</td>
<td>0.075%</td>
<td>0.034%</td>
<td>0.016%</td>
</tr>
<tr>
<td>16%</td>
<td>32%</td>
<td>13.722%</td>
<td>6.352%</td>
<td>3.050%</td>
<td>1.497%</td>
<td>0.745%</td>
<td>0.375%</td>
<td>0.190%</td>
<td>0.097%</td>
<td>0.050%</td>
</tr>
<tr>
<td>18%</td>
<td>36%</td>
<td>17.907%</td>
<td>8.741%</td>
<td>4.626%</td>
<td>2.499%</td>
<td>1.369%</td>
<td>0.758%</td>
<td>0.423%</td>
<td>0.237%</td>
<td>0.134%</td>
</tr>
<tr>
<td>20%</td>
<td>40%</td>
<td>20.800%</td>
<td>11.584%</td>
<td>6.669%</td>
<td>3.916%</td>
<td>2.331%</td>
<td>1.401%</td>
<td>0.848%</td>
<td>0.516%</td>
<td>0.316%</td>
</tr>
<tr>
<td>22%</td>
<td>44%</td>
<td>24.781%</td>
<td>14.887%</td>
<td>9.227%</td>
<td>5.828%</td>
<td>3.729%</td>
<td>2.407%</td>
<td>1.565%</td>
<td>1.023%</td>
<td>0.672%</td>
</tr>
<tr>
<td>24%</td>
<td>48%</td>
<td>29.030%</td>
<td>18.650%</td>
<td>12.339%</td>
<td>8.310%</td>
<td>5.664%</td>
<td>3.895%</td>
<td>2.696%</td>
<td>1.876%</td>
<td>1.311%</td>
</tr>
<tr>
<td>26%</td>
<td>52%</td>
<td>33.530%</td>
<td>22.868%</td>
<td>16.031%</td>
<td>11.427%</td>
<td>8.238%</td>
<td>5.988%</td>
<td>4.380%</td>
<td>3.220%</td>
<td>2.377%</td>
</tr>
<tr>
<td>28%</td>
<td>56%</td>
<td>38.259%</td>
<td>27.530%</td>
<td>20.319%</td>
<td>15.232%</td>
<td>11.539%</td>
<td>8.810%</td>
<td>6.766%</td>
<td>5.221%</td>
<td>4.044%</td>
</tr>
<tr>
<td>30%</td>
<td>60%</td>
<td>43.200%</td>
<td>32.616%</td>
<td>25.207%</td>
<td>19.762%</td>
<td>15.645%</td>
<td>12.475%</td>
<td>10.003%</td>
<td>8.055%</td>
<td>6.511%</td>
</tr>
<tr>
<td>32%</td>
<td>64%</td>
<td>48.333%</td>
<td>38.105%</td>
<td>30.687%</td>
<td>25.037%</td>
<td>20.611%</td>
<td>17.080%</td>
<td>14.226%</td>
<td>11.897%</td>
<td>9.983%</td>
</tr>
<tr>
<td>34%</td>
<td>68%</td>
<td>53.638%</td>
<td>43.970%</td>
<td>36.738%</td>
<td>31.058%</td>
<td>26.470%</td>
<td>22.695%</td>
<td>19.548%</td>
<td>16.900%</td>
<td>14.655%</td>
</tr>
<tr>
<td>36%</td>
<td>72%</td>
<td>59.098%</td>
<td>50.179%</td>
<td>43.330%</td>
<td>37.807%</td>
<td>33.226%</td>
<td>29.356%</td>
<td>26.044%</td>
<td>23.182%</td>
<td>20.692%</td>
</tr>
<tr>
<td>38%</td>
<td>76%</td>
<td>64.691%</td>
<td>56.698%</td>
<td>50.421%</td>
<td>45.245%</td>
<td>40.854%</td>
<td>37.062%</td>
<td>33.743%</td>
<td>30.811%</td>
<td>28.201%</td>
</tr>
<tr>
<td>40%</td>
<td>80%</td>
<td>70.400%</td>
<td>63.488%</td>
<td>57.958%</td>
<td>53.314%</td>
<td>49.300%</td>
<td>45.769%</td>
<td>42.621%</td>
<td>39.787%</td>
<td>37.218%</td>
</tr>
<tr>
<td>42%</td>
<td>84%</td>
<td>76.205%</td>
<td>70.508%</td>
<td>65.882%</td>
<td>61.938%</td>
<td>58.480%</td>
<td>55.390%</td>
<td>52.595%</td>
<td>50.042%</td>
<td>47.692%</td>
</tr>
<tr>
<td>44%</td>
<td>88%</td>
<td>82.086%</td>
<td>77.715%</td>
<td>74.125%</td>
<td>71.028%</td>
<td>68.282%</td>
<td>65.801%</td>
<td>63.530%</td>
<td>61.431%</td>
<td>59.478%</td>
</tr>
<tr>
<td>46%</td>
<td>92%</td>
<td>88.026%</td>
<td>85.064%</td>
<td>82.612%</td>
<td>80.480%</td>
<td>78.573%</td>
<td>76.836%</td>
<td>75.234%</td>
<td>73.742%</td>
<td>72.342%</td>
</tr>
<tr>
<td>48%</td>
<td>96%</td>
<td>94.003%</td>
<td>92.508%</td>
<td>91.264%</td>
<td>90.177%</td>
<td>89.201%</td>
<td>88.307%</td>
<td>87.478%</td>
<td>86.703%</td>
<td>85.972%</td>
</tr>
<tr>
<td>50%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
<td>100%</td>
</tr>
</tbody>
</table>
Looking at the blockchain as a shared object
Looking at the blockchain as a shared object

\((B_0, \bot)\)
Looking at the blockchain as a shared object

\((B_0, \bot)\)
Looking at the blockchain as a shared object

$(B_0, \bot) \quad (B_1, B)$
Looking at the blockchain as a shared object

(read()) = (B₀, ⊥) (B₁, B)
Looking at the blockchain as a shared object

$(B_0, \bot)$  $(B_1, B)$  $(B_2, E)$  $(B_3, C)$  $(B_4, A)$  $(B_5, E)$

A  B  C  D  E  F  G
Looking at the blockchain as a shared object

$(B_0, \bot) \quad (B_1, B) \quad (B_2, E) \quad (B_3, C) \quad (B_4, A) \quad (B_5, E)$

write $(B_6, A) \quad \text{read()} = ?$
Looking at the blockchain as a shared object

\[(B_0, \perp) \quad (B_1, B) \quad (B_2, E) \quad (B_3, C) \quad (B_4, A) \quad (B_5, E)\]
The main steps to write and read the blockchain

- As you have seen, each miner needs to locally manage a data structure from which it extracts the blockchain, i.e., the longest chain (more exactly the one which required the most amount of work)
- Actually this data structure is a tree
- the root of the tree is the common block called the genesis block
- Let $\mathcal{TB}$ be this tree, and $\mathcal{B}$ be the longest chain in $\mathcal{TB}$
- What about the read and write operations:
  - a read operation on $\mathcal{TB}$ should return $\mathcal{B}$
  - a write operation on $\mathcal{TB}$ should try to change the value of $\mathcal{TB}$ with a new block $\mathcal{B}$, that is $\mathcal{B}$ to which is appended $\mathcal{B}$
The main steps to write and read the blockchain

- When a miner wishes to augment the blockchain with a new block
  - it first invokes a read on $TB$ to get the current blockchain $B$
  - then the miner creates its block $B_1$, appends it to $B$:
    $$B ← B ⊕ B_1$$
  - then it invokes a write on $TB$ with $B$ as parameter
  - finally it broadcasts $B$ in the system (actually from a practical point of view, only $B$ is broadcast)

Blockchain-as-a-shared-object Algorithm // run by a miner

{  
repeat forever{  
  $B = DLR\.read()$
  create the well-formed block $b$ from $B$
  append $b$ to $B$
  DLR\.write($TB,B$)
}
}
Classical Shared Registers

**Liveness**: Any read / write operation terminates

1. **Safe register** Safety:

   \[
   \text{read()} = \begin{cases} 
   \text{last written value,} \\
   \text{any value of the register domain if concurrent write}
   \end{cases}
   \]

2. **Regular register** Safety:

   \[
   \text{read()} = v \begin{cases} 
   \text{last written value,} \\
   \text{value written by a concurrent write}
   \end{cases}
   \]

3. **Atomic register** Safety:

   \[(\text{Regular register safety}) + (\text{no old / new inversions})\]
Blockchain
Classical Shared Registers

- If we look at the blockchain as a shared object, we observe that it is quite different from a read/write register.
- In a classic R/W register, we do not impose that write operations depend on each other.
- We need to introduce a new object that better captures the semantics of blockchains.
Algorithm run by a miner in terms of read and write operations

Blockchain-as-a-shared-object Algorithm // run by a miner
{
    repeat forever{
        \( B = DLR.read() \)
        create the well-formed block \( b \) from \( B \)
        append \( b \) to \( B \)
        write(\( TB, B \))
    }
}
Algorithm run by a miner in terms of read and write operations

```
Operation read()
{
    return best_chain(B)
}
```
Algorithm run by a miner in terms of read and write operations

Operation write($TB, B$)
{
    update_tree($TB, B$)
    broadcast($B$)

    return
}

Algorithm run by a miner in terms of read and write operations

- Recall that an important property of Bitcoin is the number of confirmations \( n \)
- Once a block has \( n \) (e.g. \( n = 6 \)) confirmations then one may have confidence that this block will stay forever in the blockchain
Algorithm run by a miner in terms of read and write operations

Operation write($TB, B$)
{
    update_tree($TB, B$)
    broadcast($B$)
    repeat
        $B' = \text{read}()$
    until length($B'$) $\geq$ length($B$) $+$ $n$
    if $B = \text{prefix}(B')$
        return true
    else
        return abort
    upon deliver($B$)
        update_tree($TB, B$)
}
Distributed Ledger Register (DLR)

- A DLR should mimic the behavior of a blockchain and
- A DLR should allow any number of writers (miners) to write it and
- A DLR should allow any number of readers (anyone) to read it
Distributed Ledger Register (DLR)

- DLR = tree structure $T_B$
  - Root = genesis block
  - Branch = sequence of blocks
- Value of DLR
  - longest sequence of blocks starting from the genesis block (initial value of the register)
  - called the blockchain, and denoted $B$
- Two operations:
  - DLR.read() : returns the blockchain
  - DLR.write($B$) : update the value of DLR with $B$
**write() operation**

**Definition (n-valid)**

Operation write(\(\mathcal{B}\)) invoked at time \(t_w > 0\) is *n-valid* if and only if for any read operation \(r\) invoked at time \(t_r, t_r > t_w\), \(r\) returns \(\mathcal{B}'\) such that

- \(\mathcal{B}\) is a prefix of \(\mathcal{B}'\) and \(\text{length}(\mathcal{B}') \geq \text{length}(\mathcal{B}) + n\).

**write(\(\mathcal{B}\)):**
- Update \(\mathcal{T}\mathcal{B}\) with \(\mathcal{B}\)
- Returns true if it is *n-valid*, abort otherwise

![Diagram](image-url)

- \(\mathcal{B}_0\) is mapped to \(\mathcal{B}\) via write operation.
- \(\mathcal{B}'\) is returned by read operation.
- \(\mathcal{B}''\) is a prefix of \(\mathcal{B}'\) and its length satisfies the condition for *n-valid*.
- write(\(\mathcal{B}\)) \(\rightarrow\) true.
- write(\(\mathcal{B}''\)) \(\rightarrow\) abort.
Specification of the Distributed Ledger Registers

A Multi-Reader/Multi-Writer Distributed Ledger is defined by the following properties:

- **Liveness**  Any invocation of a read or a write($B$) operation terminates

- **n-consistency**  Any read() operation returns $B$ whose prefix $B'$ is the value of the register written by the last $n$-valid write($B'$) operation that precedes read().

In other words the $n$-consistency property says that the returned blockchain except for the last $n$ appended blocks is exactly the value written by the last $n$-valid write() operation.
Lemma

*Blockchain-as-a-shared-object Algorithm (A) satisfies the liveness property of the DLR*

Proof

- The read() operation (see code) always returns since the read is executed locally
- For the write() operation (see code), the only blocking part of the code is the repeat loop
- By assumption of A, miners continuously try to create blocks which gives rise to the invocation of write() operation periodically
- Thus the loop stops, which allows the write() operation to either return true or abort
- This ends the proof
Blockchain-as-a-shared-object Algorithm proof: $n$-validity

Lemma

Each non aborted write() operation invoked by $A$ satisfies the $n$-validity property of the DLR

Proof

- Let $w$ be any non aborted write($B$) operation that completes at time $t$
- Note that this operation returns only when the best chain in the tree, say $B'$, has $B$ as a prefix and has at least $k$ additional blocks (see code)
- Let $r$ be a read() operation that happens after op $w$.
  - If $r$ is invoked by the miner that invoked $w$ then necessarily the $n$-validity holds
  - Otherwise, by the property of the broadcast, there is a time $\tau$, such that by time $t + \tau$, $B'$ has reached all the nodes. Hence any read() invoked after $t + \tau$ returns at least $B'$
- This completes the proof
Blockchain-as-a-shared-object Algorithm proof: 
n-consistency

Lemma

*Each* read() *operation invoked by A* after that $\tau$ *time units have elapsed since the last* $n$-*valid operation satisfies the* $n$-*consistency*