Blockchain and Cryptocurrencies

Chapter 1: Basic Tools

Prof. Dr. Peter Thiemann

Albert-Ludwigs-Universität Freiburg, Germany

SS 2020

Literature for this lecture

- Chapter 1 of Bitcoin and Cryptocurrency Technologies Book
- Einführung in die Kryptographie (German), Johannes Buchmann, Springer
- Serious Cryptography, Jean-Philippe Aumasson

Contents

1 Blockchain

2 Cryptographic Hash Functions
- Finding a Collision
- Hash from Compression
- Blockchain-specific Properties and Applications
- SHA-256

What Does Wikipedia Say?

Blockchain

From Wikipedia, the free encyclopedia

A **blockchain**,[1][2][3] originally **block chain**,[4][5] is a growing list of records, called *blocks*, that are linked using cryptography.[1][6] Each block contains a cryptographic hash of the previous block,[6] a timestamp, and transaction data (generally represented as a Merkle tree).

By design, a blockchain is resistant to modification of the data. It is "an open, distributed ledger that can record transactions between two parties efficiently and in a verifiable and permanent way".[7] For use as a distributed ledger, a blockchain is typically managed by a peer-to-peer network collectively adhering to a protocol for inter-node communication and validating new blocks. Once recorded, the data in any given block cannot be altered retroactively without alteration of all subsequent blocks, which requires consensus of the network majority. Although blockchain records are not unalterable, blockchains may be considered secure by design and exemplify a distributed computing system with high Byzantine fault tolerance. Decentralized consensus has therefore been claimed with a blockchain.[8]

Blockchain was invented by a person (or group of people) using the name Satoshi Nakamoto in 2008 to serve as the public transaction ledger of the cryptocurrency bitcoin.[1] The identity of Satoshi Nakamoto is unknown. The invention of the blockchain for bitcoin made it the first digital currency to solve the double-spending problem without the need of a trusted authority or central server. The bitcoin design has inspired other applications,[1][3] and blockchains that are readable by the public are widely used by cryptocurrencies. Blockchain is considered a type of payment rail.[9] Private blockchains have been proposed for business use. Sources such as *Computerworld* called the marketing of such blockchains without a proper security model "snake oil".[10]

What Does Wikipedia Say?

Blockchain

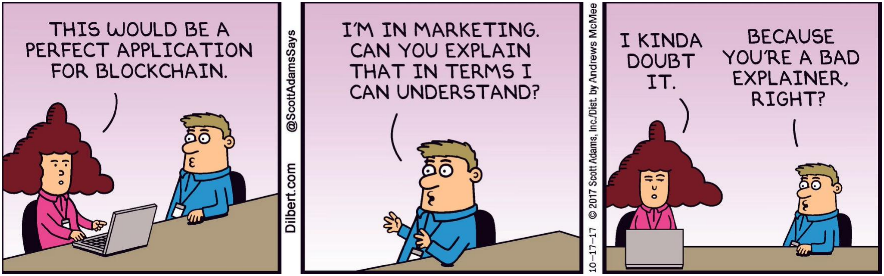
From Wikipedia, the free encyclopedia

A **blockchain**,[1][2][3] originally **block chain**,[4][5] is a growing list of records, cal … e linke … .[1][6] Each block contains a cryptographic hash of the previous block,[6] a timestamp, and trans … ata (gen … Merkle tree).

By design, a blockchain is resistant to modification of the data. It is "an open, distributed ledger … ons between two parties efficiently and in a verifiable and permanent way".[7] For use as a distributed ledger, a … anaged by a peer-to-peer network collectively adhering to a protocol for inter-node communication and valida … recorded, the data in any given block cannot be altered retroactively without alteration of all subsequent blocks, … sensus of the network majority. Although blockchain records are not unalterable, blockchains may be considered … and exemplify a distributed computing system with high Byzantine fault tolerance. Decentralized consensus has therefo … d with a blockchain.[8]

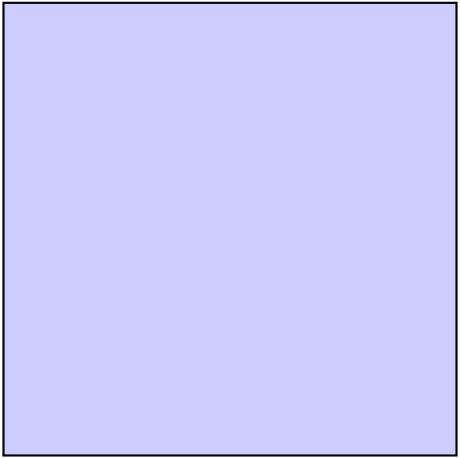
Blockchain was invented by a person (or group of people) using the name Satoshi Nakamo … e as the public transaction ledger of the cryptocurrency bitcoin.[1] The identity of Satoshi Nakamoto is unknown. The i … kchain for bitcoin made it the first digital currency to solve the double-spending problem without the need of a trusted … l server. The bitcoin design has inspired other applications,[1][3] and blockchains that are readable by the public are widely … ptocurrencies. Blockchain is considered a type of payment rail.[9] Private blockchains have been proposed for business use. Sources such as *Computerworld* called the marketing of such blockchains without a proper security model "snake oil".[10]

Without Words

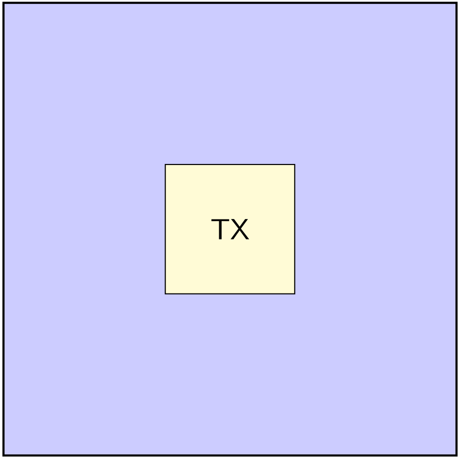


"the number of people who understand blockchain has within the past year jumped from two to an incredible four." [cryptonews.com, Sead Fadilpašić, June 30, 2018]

In The Beginning Was The Block



In The Beginning Was The Block



Transaction

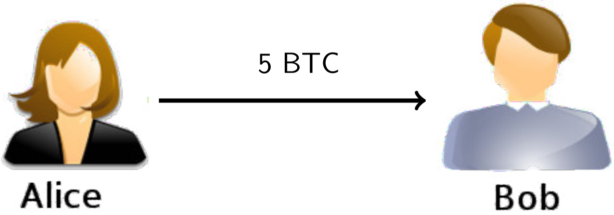
TX is a transaction

What is a transaction?

What is a transaction?

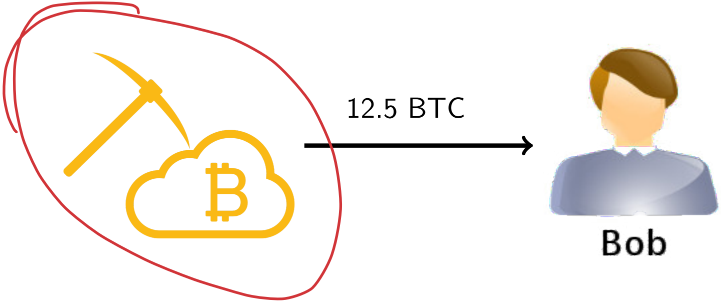
Movement of a monetary item!

Transaction "Transfer"



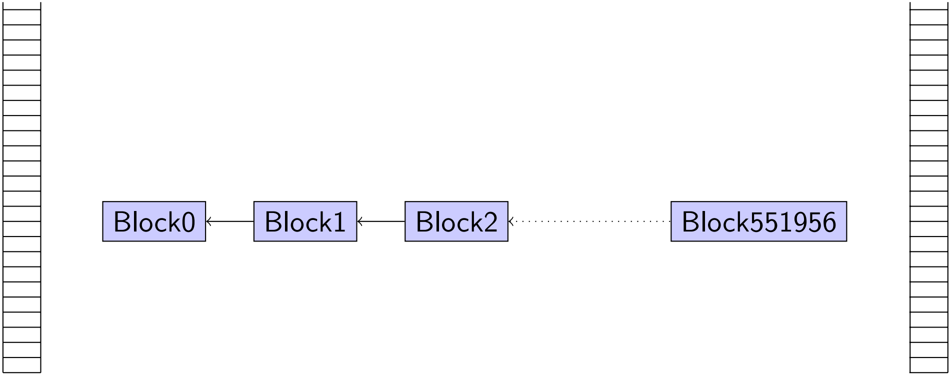
- Alice transfers 5 BTC to Bob

Transaction "Mining"



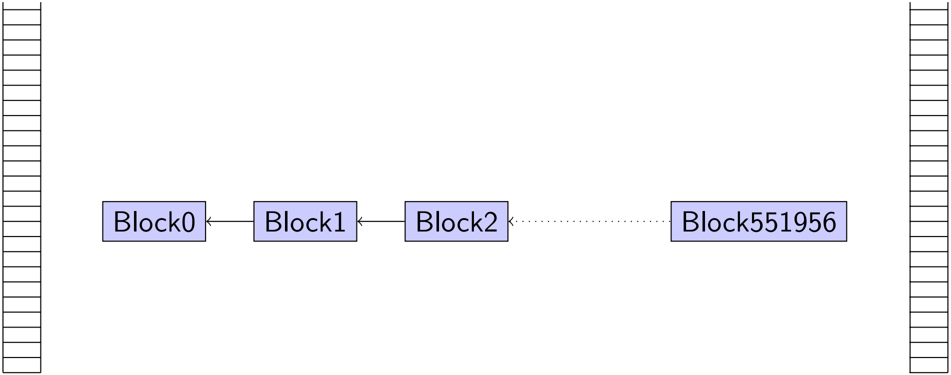
- Creation of new BTC by mining

Development of the Bitcoin-Blockchain



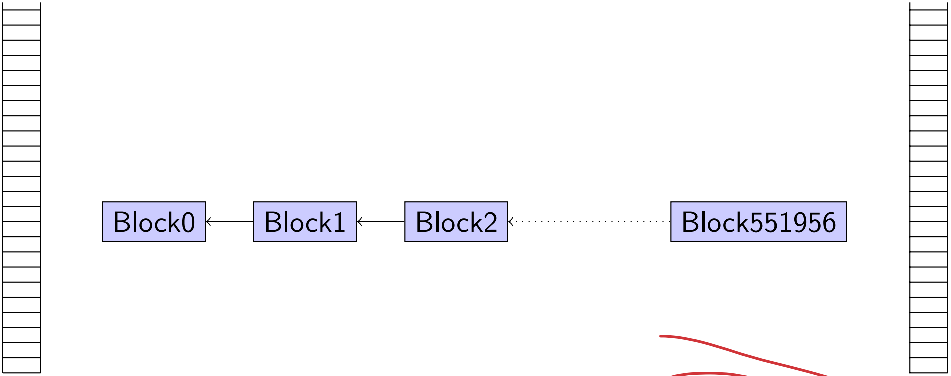
- Genesis: January 2009

Development of the Bitcoin-Blockchain



- Genesis: January 2009
- All account balances = 0

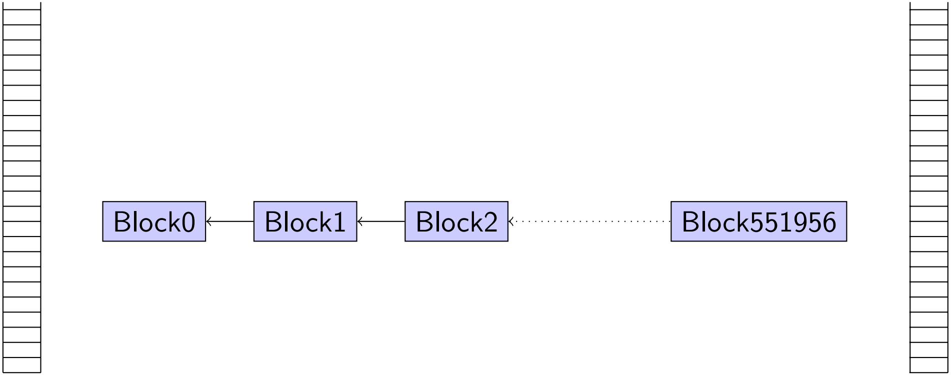
Development of the Bitcoin-Blockchain



- Genesis: January 2009
- All account balances = 0

- 29.11.2018

Development of the Bitcoin-Blockchain



- Genesis: January 2009
- All account balances = 0

- 29.11.2018
- Balances according to transaction history

Transaction History

Transactions	Balances			
	A	B	C	⋯
Genesis January 2009	0	0	0	⋯

Transaction History

Transactions	Balances			
	A	B	C	$\cdots$
Genesis January 2009	0	0	0	$\cdots$
Mining: 50 BTC $\rightarrow$ B	0	50	0	$\cdots$

Transaction History

Transactions	Balances			
	A	B	C	⋯
Genesis January 2009	0	0	0	⋯
Mining: 50 BTC $\rightarrow$ B	0	50	0	⋯
Transfer: 7.5 BTC B $\rightarrow$ A	7.5	42.5	0	⋯

Transaction History

Transactions	Balances A	B	C	
Genesis January 2009	0	0	0	$\cdots$
Mining: 50 BTC $\rightarrow$ B	0	50	0	$\cdots$
Transfer: 7.5 BTC B $\rightarrow$ A	7.5	42.5	0	$\cdots$
Transfer: 2.5 BTC A,B $\rightarrow$ C	5	40	5	$\cdots$

Transaction History

Transactions	Balances			
	A	B	C	$\cdots$
Genesis January 2009	0	0	0	$\cdots$
Mining: 50 BTC $\rightarrow$ B	0	50	0	$\cdots$
Transfer: 7.5 BTC B $\rightarrow$ A	7.5	42.5	0	$\cdots$
Transfer: 2.5 BTC A,B $\rightarrow$ C	5	40	5	$\cdots$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Block 551956

Zusammenfassung	
Height	551956 (Main chain)
Hash	000000000000000000017231299a46f025ba24207245856f6bec0678fe0f55a03
Vorheriger Block	0000000000000000000c700dc6fc74bc1330867ed748a97379daab8ef4def4cf
Nächster Block	00000000000000000000005ee21e560dcc6821ae201248ed87f30a7eb18e314b3a4
Zeit	2018-11-29 19:02:07
Empfangene Zeit	2018-11-29 19:02:07
Weitergeleitet von	BTC.com
Schwierigkeit	6,653,303,141,405.96
Bits	388648495
Anzahl der Transaktionen	3071
Ausgang insgesamt	15,944.50826868 BTC
Geschätztes Transaktionsvolumen	3,536.3279512 BTC
Größe	1298.52 KB
Ausführung	0x20C00000
Merkle Root	67f036c0f1e7b77d303c8bae81eefb17002a8a5bbd52b0c38a6119167527c80d
Nonce	133892812
Block Reward	12.5 BTC
Transaktions Gebühren	0.7427863 BTC

Source: https://www.blockchain.com/btc/block-height/551956 (defunct: see `https://www.blockchain.com/de/explorer`, current block 630718)

Cryptocurrency

- Transaction history is public

Cryptocurrency

- Transaction history is public
- Everyone can check every balance

Cryptocurrency

- Transaction history is public
- Everyone can check every balance
- Everyone can check every transaction

Cryptocurrency

- Transaction history is public
- Everyone can check every balance
- Everyone can check every transaction
- If there is consensus about the sequence of the transactions!

Cryptocurrency

- Transaction history is public
- Everyone can check every balance
- Everyone can check every transaction
- If there is consensus about the sequence of the transactions!
- If the history cannot be rewritten/faked.

Cryptocurrency

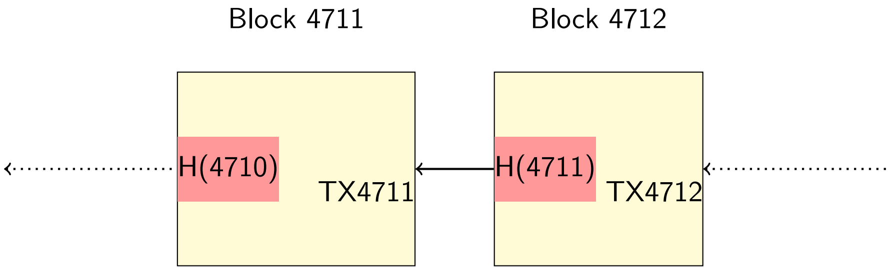
- Transaction history is public
- Everyone can check every balance
- Everyone can check every transaction
- If there is consensus about the sequence of the transactions!
- If the history cannot be rewritten/faked.
- Cryptography helps!

Contents

1 Blockchain

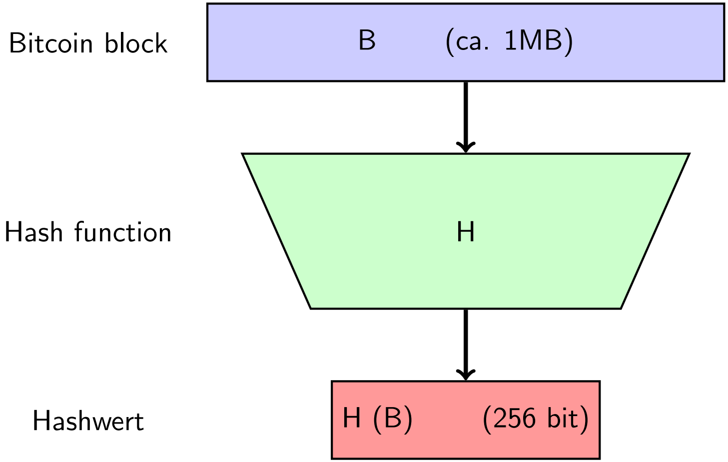
2 Cryptographic Hash Functions
- Finding a Collision
- Hash from Compression
- Blockchain-specific Properties and Applications
- SHA-256

Hash me!



- Every block consists of the text of the transactions and the hash of the preceding block. (simplified)

(Cryptographic) Hash



Hash Functions

Definition

A **hash function** is a mapping $h : B^* \to B^n$, for some $n > 0$.

$B = \{0, 1\}$

Hash Functions

Definition

A **hash function** is a mapping $h : B^* \to B^n$, for some $n > 0$. $B = \{0, 1\}$

Example

The mapping $P : B^* \to B^1$ defined by $P(b_1 \dots b_k) = b_1 \oplus \cdots \oplus b_k$ is a hash function that computes the parity of its input. $\oplus$ is xor

$P(01101) = 1$ Just 2 possible outputs.

Hash Functions

Definition

A **hash function** is a mapping $h : B^* \to B^n$, for some $n > 0$. $B = \{0, 1\}$

Example

The mapping $P : B^* \to B^1$ defined by $P(b_1 \dots b_k) = b_1 \oplus \cdots \oplus b_k$ is a hash function that computes the parity of its input. $\oplus$ is xor

$P(01101) = 1$

Remark

Hash functions are never injective. (by a counting argument)

Compression Functions

Definition

A **compression function** is a mapping $h : B^m \to B^n$, for some $m > n > 0$.

Compression Functions

Definition

A **compression function** is a mapping $h : B^m \to B^n$, for some $m > n > 0$.

Example

The mapping $b_1 \ldots b_m \mapsto b_1 \ldots b_n$ is a compression function.

Compression Functions

Definition

A **compression function** is a mapping $h : B^m \to B^n$, for some $m > n > 0$.

Example

The mapping $b_1 \dots b_m \mapsto b_1 \dots b_n$ is a compression function.

Remark

Hash and compression functions have many uses for efficient search algorithms. In a cryptographic context, further properties for hash functions are necessary.

One-way functions

Let $h : D \to B^n$ be a hash or compression function, efficiently computable for each $x \in D$.

Definition

Function h is a **one-way function** (or **preimage resistant**) if, given some $s \in B^n$, it is practically impossible to find a preimage $x \in D$ such that $h(x) = s$.

One-way functions

Let $h : D \to B^n$ be a hash or compression function, efficiently computable for each $x \in D$.

Definition

Function h is a **one-way function** (or **preimage resistant**) if, given some $s \in B^n$, it is practically impossible to find a preimage $x \in D$ such that $h(x) = s$.

Remark

It is not known whether one-way functions exist. According to today's knowledge there are functions, for which there is no efficiently computable inverse function.

One-way functions

Let $h : D \to B^n$ be a hash or compression function, efficiently computable for each $x \in D$.

Definition

Function h is a **one-way function** (or **preimage resistant**) if, given some $s \in B^n$, it is practically impossible to find a preimage $x \in D$ such that $h(x) = s$.

Remark

It is not known whether one-way functions exist. According to today's knowledge there are functions, for which there is no efficiently computable inverse function.

Example

Let p be a large prime number (e.g., more than 2048 bits) and g a primitive root modulo p. Then $f(x) = g^x \mod p$ is a one-way function because there is no know efficient algorithm to compute discrete logarithms.

Collisions

Definition

A **collision** of h is a pair (x, x') with $x \neq x' \in D$ such that $h(x) = h(x')$.

Collisions

Definition

A **collision** of h is a pair (x, x') with $x \neq x' \in D$ such that $h(x) = h(x')$.

Remark

Every hash or compression function has a collsion because it is not injective.

Collisions

Definition

A **collision** of h is a pair (x, x') with $x \neq x' \in D$ such that $h(x) = h(x')$.

Remark

Every hash or compression function has a collsion because it is not injective.

Example

Consider the function $P : B^* \to B^1$ with $P(b_1 \dots b_k) = b_1 \oplus \dots \oplus b_k$.
Clearly $P(111) = P(010) = 1$ is a collision.

Second Preimage Resistance & Collision Resistance

Definition

A function h is **second preimage resistant** (weakly collision resistant) if, given some $x \in D$, it is practically impossible to find x' such that (x, x') is a collision of h.

Second Preimage Resistance & Collision Resistance

Definition

A function h is **second preimage resistant** (weakly collision resistant) if, given some $x \in D$, it is practically impossible to find x' such that (x, x') is a collision of h.

Definition

A function h is **collision resistant** if it is practically impossible to find a collision (x, x') of h.

Properties

Any second preimage resistant function h is also one-way.

Suppose $FP(h, s)$ computes a preimage of h for s.
Then $SP(h, x) = FP(h, h(x))$ computes a second preimage.

Properties

Any second preimage resistant function h is also one-way.

Suppose $FP(h, s)$ computes a preimage of h for s.
Then $SP(h, x) = FP(h, h(x))$ computes a second preimage.

Any collision resistant function is also second preimage resistant

Suppose $SP(h, x)$ computes a second preimage of h for x.
Then $C(h) = (x, SP(h, x))$, for some random x, computes a collision for h.

Properties of cryptographic hash functions

Collision Resistance

If $B_1 \neq B_2$, then almost certainly $H(B_1) \neq H(B_2)$.

Properties of cryptographic hash functions

Collision Resistance

If $B_1 \neq B_2$, then almost certainly $H(B_1) \neq H(B_2)$.

Example (H=SHA-256)

Properties of cryptographic hash functions

Collision Resistance

If $B_1 \neq B_2$, then almost certainly $H(B_1) \neq H(B_2)$.

Example (H=SHA-256)

- $B_1 =$ "Zwei Warzenschweine spielen Fussball im Regen."

Properties of cryptographic hash functions

Collision Resistance

If $B_1 \neq B_2$, then almost certainly $H(B_1) \neq H(B_2)$.

Example (H=SHA-256)

- $B_1 =$ "Zwei Warzenschweine spielen Fussball im Regen."
- $H(B_1) =$
  `7f2c6d75c99218fe6f4b742b469c0e67b319387e3a12dadb4d9dea4cd9143611`

Properties of cryptographic hash functions

Collision Resistance

If $B_1 \neq B_2$, then almost certainly $H(B_1) \neq H(B_2)$.

Example (H=SHA-256)

- $B_1 =$ "Zwei Warzenschweine spielen Fussball im Regen."
- $H(B_1) =$ `7f2c6d75c99218fe6f4b742b469c0e67b319387e3a12dadb4d9dea4cd9143611`
- $B_2 =$ "Zwei Warzenschweine spielen Fussball im Regen!"

Properties of cryptographic hash functions

Collision Resistance

If $B_1 \neq B_2$, then almost certainly $H(B_1) \neq H(B_2)$.

Example (H=SHA-256)

- $B_1 =$ "Zwei Warzenschweine spielen Fussball im Regen."
- $H(B_1) =$
  `7f2c6d75c99218fe6f4b742b469c0e67b319387e3a12dadb4d9dea4cd9143611`
- $B_2 =$ "Zwei Warzenschweine spielen Fussball im Regen!"
- $H(B_2) =$
  `ffc094012cf2eef9a528287f4f8efd85d7ec787ac1622903d166f2d45fe8024c`

Contents

1 Blockchain

2 Cryptographic Hash Functions
- Finding a Collision
- Hash from Compression
- Blockchain-specific Properties and Applications
- SHA-256

Finding a Collision

How hard?

Finding a Collision

How hard?

- Consider h $=$ SHA-256 where the output is B^{256} (Bitcoin's choice)

Finding a Collision

How hard?

- Consider h $=$ SHA-256 where the output is B^{256} (Bitcoin's choice)
- Choose $2^{256}+1$ distinct input values. Then there must be two equal outputs.

Finding a Collision

How hard?

- Consider h $=$ SHA-256 where the output is B^{256} (Bitcoin's choice)
- Choose $2^{256}+1$ distinct input values. Then there must be two equal outputs.
- Worst case: $2^{256} \approx 1.2 \cdot 10^{78}$ attempts

Finding a Collision

How hard?

- Consider h = SHA-256 where the output is B^{256} (Bitcoin's choice)
- Choose $2^{256} + 1$ distinct input values. Then there must be two equal outputs.
- Worst case: $2^{256} \approx 1.2 \cdot 10^{78}$ attempts
- with a million attempts per second: $\approx 3.6 \cdot 10^{63}$ years

Finding a Collision

How hard?

- Consider h $=$ SHA-256 where the output is B^{256} (Bitcoin's choice)
- Choose $2^{256} + 1$ distinct input values. Then there must be two equal outputs.
- Worst case: $2^{256} \approx 1.2 \cdot 10^{78}$ attempts
- with a million attempts per second: $\approx 3.6 \cdot 10^{63}$ years
- age of the universe: $\approx 1.4 \cdot 10^{10}$ years

Finding a Collision

How hard?

- Consider h $=$ SHA-256 where the output is B^{256} (Bitcoin's choice)
- Choose $2^{256}+1$ distinct input values. Then there must be two equal outputs.
- Worst case: $2^{256} \approx 1.2 \cdot 10^{78}$ attempts
- with a million attempts per second: $\approx 3.6 \cdot 10^{63}$ years
- age of the universe: $\approx 1.4 \cdot 10^{10}$ years

Conclusion

It is practially impossible to find a collision (x, x') for h.

The Birthday Paradox

Question

How many people have to be in the room such that two of them have the same birthday with probability greater than $1/2$?

Answer

Let's compute the probability that all k people in the room have a different birthday.
There are $n = 365$ different birthdays.
Clearly, there are n^k possible birthday scenarios.
To obtain all different birthdays, the first person has n choices, the second $n-1$, and so on.
So the number of all-different-birthday scenarios is $\prod_{i=0}^{k-1}(n-i)$.
So the probability of all-different-birthday is

$$q = \frac{1}{n^k}\prod_{i=0}^{k-1}(n-i) = \prod_{i=0}^{k-1}(1-\frac{i}{n})$$

The Birthday Paradox approximated

We obtained

$$q = \frac{1}{n^k} \prod_{i=0}^{k-1} (n - i) = \prod_{i=0}^{k-1} (1 - \frac{i}{n})$$

Observe that $1 - x \leq e^{-x}$ to obtain

$$q \leq \prod_{i=0}^{k-1} e^{-i/n} = e^{-\sum_{i=0}^{k-i} i/n} = e^{-k(k-1)/(2n)}$$

Using $q = e^{\ln q}$ and taking the logarithm on both sides, it remains to solve a quadratic equation. Solving for $q \leq 1/2$ yields

$$k \geq (1 + \sqrt{1 + (8 \ln 2) 2^n})/2$$

The Birthday Attack

- Let n be the hash size and

$$f(n) = (1 + \sqrt{1 + (8 \ln 2) 2^n})/2$$

- If we choose $k \geq f(n)$ input values x uniformly randomly, then there exist x_1, x_2 such that the hash values $h(x_1) = h(x_2)$ with probability $1/2$.
- Hence, to find a collision with probability $1/2$, it is sufficient to compare the results for $f(n)$ different input values. It turns out that $\log_2(f(n)) \approx n/2$ so that for $n = 256$ (SHA-256) we can find a collision with probability $> 1/2$ when testing 2^{130} inputs.

Contents

1 Blockchain

2 Cryptographic Hash Functions
- Finding a Collision
- Hash from Compression
- Blockchain-specific Properties and Applications
- SHA-256

Hash function from compression function

Merkle-Damgaard procedure

Let $f : B^m \to B^n$ be a compression function and let $r = m - n \geq 2$.
The goal is to construct a hash function $h : B^* \to B^n$ from f.

Hash function from compression function

Merkle-Damgaard procedure

Let $f : B^m \to B^n$ be a compression function and let $r = m - n \geq 2$.
The goal is to construct a hash function $h : B^* \to B^n$ from f.

Preprocessing Step 1

Given $x \in B^*$, prepend the minimal number $0 \leq k < r$ of zeroes such that the new length is a multiple of r and append 0^r. Result: $x' = 0^k \| x \| 0^r$

Hash function from compression function

Merkle-Damgaard procedure

Let $f : B^m \to B^n$ be a compression function and let $r = m - n \geq 2$.
The goal is to construct a hash function $h : B^* \to B^n$ from f.

Preprocessing Step 1

Given $x \in B^*$, prepend the minimal number $0 \leq k < r$ of zeroes such that the new length is a multiple of r and append 0^r. Result: $x' = 0^k \| x \| 0^r$

Preprocessing Step 2

Calculate the binary representation b of the original length of x and prepend zeroes such that its length is divisible by $r - 1$. Starting at the beginning insert 1 at every $r - 1$st position of the resulting string. The length of the resulting string b' is a multiple of r.

Hash function from compression function

Merkle-Damgaard procedure

Let $f : B^m \to B^n$ be a compression function and let $r = m - n \geq 2$.
The goal is to construct a hash function $h : B^* \to B^n$ from f.

Preprocessing Step 1

Given $x \in B^*$, prepend the minimal number $0 \leq k < r$ of zeroes such that the new length is a multiple of r and append 0^r. Result: $x' = 0^k \| x \| 0^r$

Preprocessing Step 2

Calculate the binary representation b of the original length of x and prepend zeroes such that its length is divisible by $r - 1$. Starting at the beginning insert 1 at every $r - 1$st position of the resulting string. The length of the resulting string b' is a multiple of r.

Preprocessing Step 3

Prepend b' to obtain a string $b' \| 0^k \| x \| 0^r$ of length $t \cdot r$. Decompose into $x_1 \| x_2 \| \ldots \| x_t$ with $x_i \in B^r$.

Preprocessing (Merkle-Damgaard)

Example

- Let $r = 4$ and $x = 111011$ of length 6.

Preprocessing (Merkle-Damgaard)

Example

- Let $r = 4$ and $x = 111011$ of length 6.
- Step 1 results in $x' = 00 \| x \| 0000 = 001110110000$.

Preprocessing (Merkle-Damgaard)

Example

- Let $r = 4$ and $x = 111011$ of length 6.
- Step 1 results in $x' = 00\|x\|0000 = 001110110000$.
- Step 2: the binary representation of 6 is $b = 110$ which happens to be of length $3 = r - 1$. Hence, no zero padding is needed, but we need to insert one(s) to obtain $b' = 1110$.

Preprocessing (Merkle-Damgaard)

Example

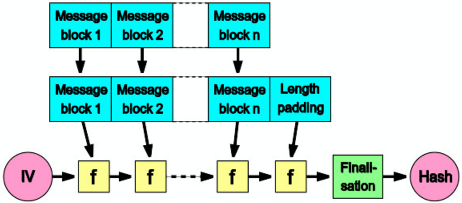
- Let $r = 4$ and $x = 111011$ of length 6.
- Step 1 results in $x' = 00\|x\|0000 = 001110110000$.
- Step 2: the binary representation of 6 is $b = 110$ which happens to be of length $3 = r - 1$. Hence, no zero padding is needed, but we need to insert one(s) to obtain $b' = 1110$.
- Step 3 results in $b'\|x' = 1110\|0011\|1011\|0000$ so there are four packets of size $r = 4$.

Constructing the hash function

Definition

Define $h(x) = H_t$ where

- $x_1 \| x_2 \| \dots \| x_t$ with $x_i \in B^r$ is the result of preprocessing $x \in B^*$.
- $H_0 = 0^n$ (or a different, but fixed initialization vector).
- $H_i = f(H_{i-1} \| x_i)$ for $1 \leq i \leq t$.



By Davidgothberg - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=1906913

Properties

Merkle has shown (in his 1979 thesis)

Result

If f is collision resistant, then the Merkle-Damgaard construction yields a function h that is also collision resistant.

Contents

1 Blockchain

2 Cryptographic Hash Functions
- Finding a Collision
- Hash from Compression
- Blockchain-specific Properties and Applications
- SHA-256

Application: Message Authentication Code (MAC)

Suppose you want to guarantee the integrity of a program (or a file you stored online). You compute the hash of its executable file and store it in a safe place. Before you run the program you compute the hash and compare it to the saved value. If the program was modified (e.g., by a virus), the new hash will be different.
The hash is small and thus more efficient to store and compare.
For this application, second preimage resistance is sufficient.

Property: Hiding

- Consequence of using a one-way function: We cannot compute a preimage efficiently.

Property: Hiding

- Consequence of using a one-way function: We cannot compute a preimage efficiently.
- But what if the input is from a small finite set (e.g., the result of a coin flip)?

Property: Hiding

- Consequence of using a one-way function: We cannot compute a preimage efficiently.
- But what if the input is from a small finite set (e.g., the result of a coin flip)?
- An adversary can see two outputs: h(heads) and h(tails).

Property: Hiding

- Consequence of using a one-way function: We cannot compute a preimage efficiently.
- But what if the input is from a small finite set (e.g., the result of a coin flip)?
- An adversary can see two outputs: h(heads) and h(tails).
- They can immediately figure out when two inputs were equals and they could easily precompute the two possible hashes.

Property: Hiding

- Consequence of using a one-way function: We cannot compute a preimage efficiently.
- But what if the input is from a small finite set (e.g., the result of a coin flip)?
- An adversary can see two outputs: h(heads) and h(tails).
- They can immediately figure out when two inputs were equals and they could easily precompute the two possible hashes.

Solution: Harden the one-way function

Instead of computing $h(x)$, compute $h(r \| x)$ where r is a suitably chosen random number.

Application: Commitments

Definition

A **commitment scheme** consists of two algorithms

- com = commit (msg, nonce)
- verify (com, msg, nonce)
  returns true iff com == commit(msg, nonce)

Requirements

Hiding given com it is infeasible to find msg.

Binding It is infeasible to find two pairs (msg, nonce) and (msg', nonce') such that msg != msg' and commit(msg, nonce) == commit(msg', nonce').

[illegible]

Concept: Nonce

Definition

A **nonce** (contraction of "number used once") is a random number drawn from a probability distribution with high min-entropy.
It is intended to add perturbation into encrypted messages and hashes, to avoid detection of recurring message contents.
Hence, each nonce must only be used once

A Commitment Scheme

Implementation

Let h be a collision resistant hash function.
Define $\mathsf{commit}(\mathsf{msg}, \mathsf{nonce}) = \mathsf{h}(\mathsf{nonce} \mathbin{||} \mathsf{msg})$ where nonce is a random 256-bit value.

A Commitment Scheme

Implementation

Let h be a collision resistant hash function.
Define $\mathsf{commit}(\mathsf{msg}, \mathsf{nonce}) = \mathsf{h}(\mathsf{nonce} \mathbin{||} \mathsf{msg})$ where nonce is a random 256-bit value.

Checking the properties

Hiding It is infeasible to find msg from com because h is collision resistant (and hence one-way).

Binding Immediate from collision resistance.

Property: Puzzle Friendliness

Definition

A hash function h is **puzzle friendly** if for every possible n-bit output value and every k chosen from a distribution with high min-entropy, then it is infeasible to find x such that $h(k \| x) = y$ in time significantly less than 2^n.

Application: Search Puzzle

Search Puzzle

Suppose we are given

- a hash function hn
- a value k (called *puzzle ID*) chosen from a high min-entropy distribution, and
- a target set Y contained in the range of h.

A solution to this puzzle is a value, x, such that $h(k \| x) \in Y$.

Application: Search Puzzle

Search Puzzle

Suppose we are given

- a hash function hn
- a value k (called *puzzle ID*) chosen from a high min-entropy distribution, and
- a target set Y contained in the range of h.

A solution to this puzzle is a value, x, such that $h(k\|x) \in Y$.

- The size of Y determines hardness of the puzzle: $Y = B^n$ trivial, $Y = \{y\}$ maximum difficulty

Application: Search Puzzle

Search Puzzle

Suppose we are given

- a hash function *hn*
- a value k (called *puzzle ID*) chosen from a high min-entropy distribution, and
- a target set Y contained in the range of h.

A solution to this puzzle is a value, x, such that $h(k \| x) \in Y$.

- The size of Y determines hardness of the puzzle: $Y = B^n$ trivial, $Y = \{y\}$ maximum difficulty
- High min-entropy of k ensures that precomputing x for expected values of k is not useful.

Application: Search Puzzle

Search Puzzle

Suppose we are given

- a hash function hn
- a value k (called *puzzle ID*) chosen from a high min-entropy distribution, and
- a target set Y contained in the range of h.

A solution to this puzzle is a value, x, such that $h(k\|x) \in Y$.

- The size of Y determines hardness of the puzzle: $Y = B^n$ trivial, $Y = \{y\}$ maximum difficulty
- High min-entropy of k ensures that precomputing x for expected values of k is not useful.
- No better solving strategy than trying random values for x.

Contents

1 Blockchain

2 Cryptographic Hash Functions
- Finding a Collision
- Hash from Compression
- Blockchain-specific Properties and Applications
- SHA-256

SHA-256

- Hash function used in Bitcoin (though dated)
- SHA = Secure Hash Algorithm
- standard hash function(s) defined by NIST for use by non-military federal goverment agencies in the US
- SHA-256 is defined via the Merkle-Damgaard construction from a compression function
- the compression function is designed such that flipping a bit in the input changes at least 50% of the bits in the output
- Not know to be compromised, but successors exist since 2012: SHA-3 (Keccak)

Thanks!