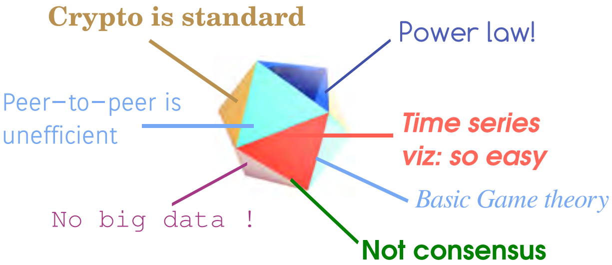
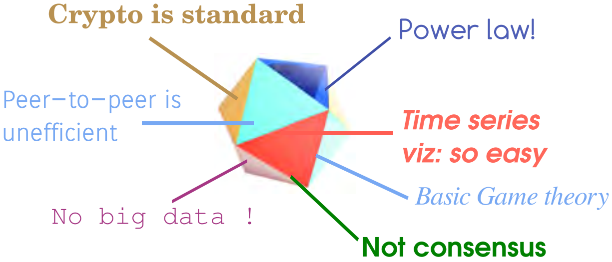
Hash functions in blockchains

Daniel Augot
INRIA Saclay–Île-de-France
Laboratoire d'informatique de l'École polytechnique
Head of project-team Grace (crypto)

This talk

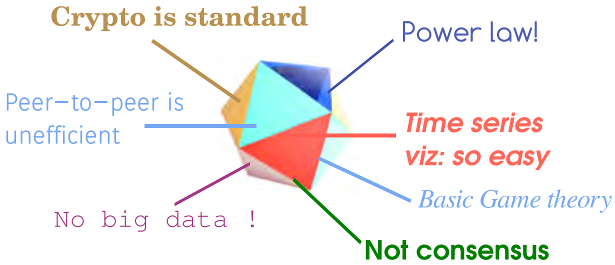

Crypto is standard
Power law!
Peer−to−peer is unefficient
Time series viz: so easy
Basic Game theory
No big data !
Not consensus

This talk



Well, actually, there are very good research topics.

This talk



Well, actually, there are very good research topics.
A very narrow view: hash functions.

Outline

Outline

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.

1. Introduction

Commerce on the Internet has come to rely almost exclusively on financial institutions serving as trusted third parties to process electronic payments. While the system works well enough for most transactions, it still suffers from the inherent weaknesses of the trust based model. Completely non-reversible transactions are not really possible, since financial institutions cannot avoid mediating disputes. The cost of mediation increases transaction costs, limiting the minimum practical transaction size and cutting off the possibility for small casual transactions, and there is a broader cost in the loss of ability to make non-reversible payments for non-reversible services. With the possibility of reversal, the need for trust spreads. Merchants must be wary of their customers, hassling them for more information than they would otherwise need. A certain percentage of fraud is accepted as unavoidable. These costs and payment uncertainties can be avoided in person by using physical currency, but no mechanism exists to make payments over a communications channel without a trusted party.

What is needed is an electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party. Transactions that are computationally impractical to reverse would protect sellers from fraud, and routine escrow mechanisms could easily be implemented to protect buyers. In this paper, we propose a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions. The system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes.

1

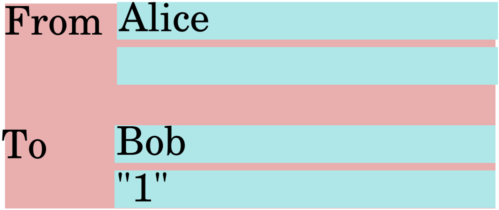
What is needed is an electronic payment system based on cryptographic proof instead of trust, [...] without the need for a trusted third party

We propose a solution [...] using a peer-to-peer distributed timestamp server to generate [...] proof of the chronological order of transactions

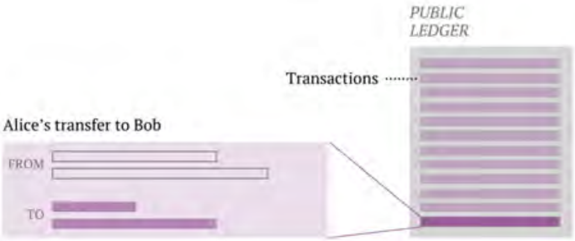
Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. Online, bitcoin.org/bitcoin.pdf. 2008

Transactions I

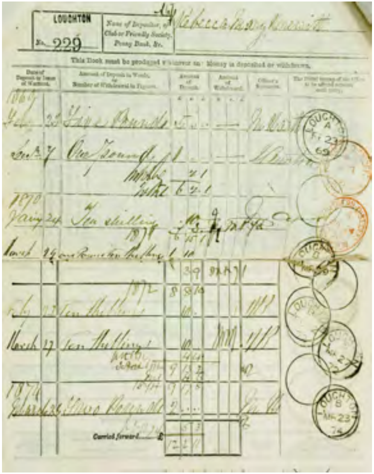
- Alice transfers bitcoins to Bob



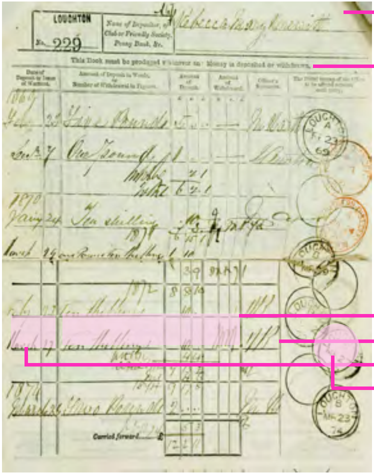
- this is written in a public ledger



Blocks and the ledger



Blocks and the ledger



Rebbeca Mary Marewitt

"This book must be produced whenever any money is deposited or withdraw"

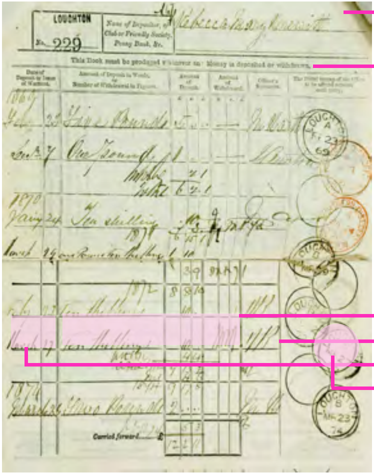
Transaction

Officer's signature

March 27, 1869

Date stamp of the office to be affixed against each entry

Blocks and the ledger



Rebbeca Mary Marewitt — adresse bitcoin

"This book must be produced
whenever any money is deposited
or withdraw" — registre public

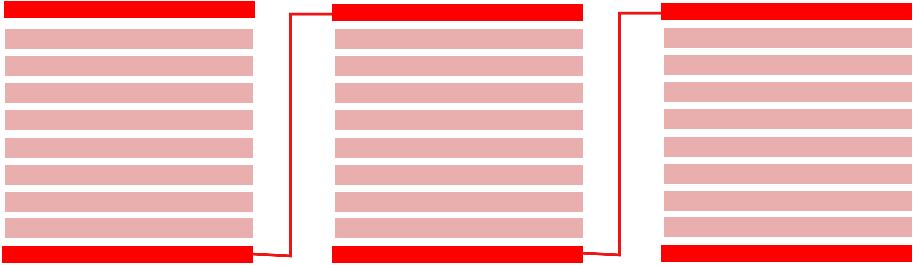
Transaction

Officer's signature — pas d'officier

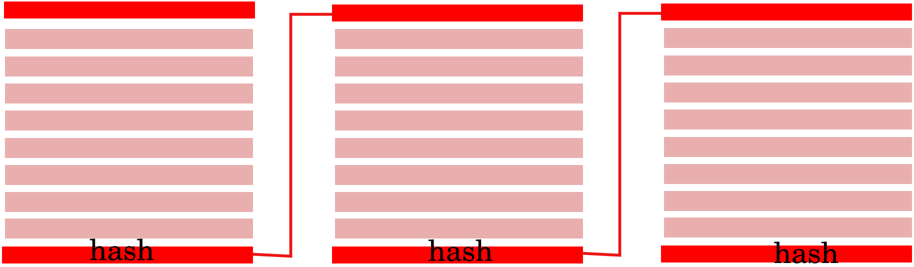
March 27, 1869 — ni de signature

Date stamp of the office
to be affixed against each entry — "minage"

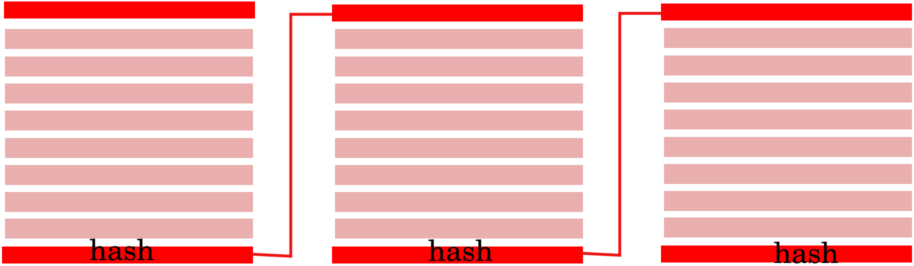
Blocks are chained



Blocks are chained



Blocks are chained



Actually, the hash is hard to find: proof-of-work

Cynthia Dwork and Moni Naor. "Pricing via Processing or Combatting Junk Mail". In: *CRYPTO' 92*. 1993

Outline

Cryptographic hash function

$$H : \begin{cases} \{0,1\}^* & \to & \{0,1\}^m \\ x & \mapsto & y = H(x) \end{cases}$$

- deterministic algorithm
- no secrets, no keys (neither private, public, or secret)
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - ...

Cryptographic hash function

$$H : \begin{cases} \text{any bit string} & \rightarrow & \text{m bits} \\ x & \mapsto & y = H(x) \end{cases}$$

- deterministic algorithm
- no secrets, no keys (neither private, public, or secret)
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - ...

Cryptographic hash function

$$H : \begin{cases} \text{any bit string} & \rightarrow & \text{m/8 bytes} \\ x & \mapsto & y = H(x) \end{cases}$$

- deterministic algorithm
- no secrets, no keys (neither private, public, or secret)
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - ...

Cryptographic hash function

$$H : \begin{cases} \text{any digitalized document} & \rightarrow & \text{m/8 bytes} \\ x & \mapsto & y = H(x) \end{cases}$$

- deterministic algorithm
- no secrets, no keys (neither private, public, or secret)
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - it is **not** signature, **neither** encryption
  - ...

Cryptographic hash function: properties

Standard definition

- First preimage resistance: given $y = H(x)$, impossible to find x
  - no better way than 2^m calls to H
- Second preimage resistance: given x impossible to find x' s.t. $H(x') = H(x)$
  - no better way than 2^m calls to H
- Collision resistance: impossible to find x, x' s.t. $H(x) = H(x')$
  - no better way than $2^{m/2}$ calls to H

Random Oracle Idealization

- Hold a table T
- when queried for x
  - if $x \in T$ return $T[x]$
  - if $x \notin T$ return a random y, and set $T[x] = y$

Why such a thing would be useful

Most unsemantic function: given x, $y = F(x)$ is (hopefully) pure random!

Usage

- Ensuring file integrity: $M \mapsto (M, h(M))$
  If $h(M)$ is secure, there can be non corruption on M
  $\Longrightarrow$ *integrity of the blockchain from last trusted hash h*
- Password storage (with a pinch of salt)
- Blind registration of documents (notarization)

d95b82d3187458f83ad36abd509c7688f60cbda4

Where do they come from

V·T·E	Cryptographic hash functions & message authentication codes
	List · Comparison · Known attacks
Common functions	MD5 · SHA-1 · SHA-2 · SHA-3 · BLAKE2
SHA-3 finalists	BLAKE · Grøstl · JH · Skein · Keccak (winner)
Other functions	CubeHash · ECOH · FSB · GOST · HAS-160 · HAVAL · Kupyna · LM hash · MD2 · MD4 · MD6 · MDC-2 · N-Hash · RIPEMD · RadioGatún · SWIFFT · Snefru · Streebog · Tiger · VSH · Whirlpool
Key derivation functions	bcrypt · crypt · PBKDF2 · scrypt · Argon2
MAC functions	DAA · CBC-MAC · HMAC · OMAC/CMAC · PMAC · VMAC · UMAC · Poly1305 · SipHash
Authenticated encryption modes	CCM · CWC · EAX · GCM · IAPM · OCB
Attacks	Collision attack · Preimage attack · Birthday attack · Brute-force attack · Rainbow table · Side-channel attack · Length extension attack
Design	Avalanche effect · Hash collision · Merkle-Damgård construction · Sponge function · HAIFA construction · Unique Block Iteration
Standardization	CRYPTREC · NESSIE · NIST hash function competition
Utilization	Hash-based cryptography · Key stretching · Merkle tree · Message authentication · Proof of work · Salt
V·T·E	**Cryptography** [show]

Where do they come from

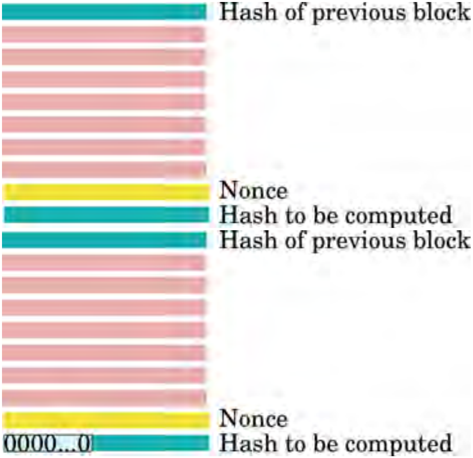
V • T • E	Cryptographic hash functions & message authentication codes
	List • Comparison • Known attacks
Common functions	MD5 • SHA-1 • SHA-2 • SHA-3 • BLAKE2
SHA-3 finalists	BLAKE • Grøstl • JH • Skein • Keccak (winner)
Other functions	CubeHash • ECOH • FSB • GOST • HAS-160 • HAVAL • Kupyna • LM hash • MD2 • MD4 • MD6 • MDC-2 • N-Hash • RIPEMD • RadioGatún • SWIFFT • Snefru • Streebog • Tiger • VSH • Whirlpool
Key derivation functions	bcrypt • crypt • PBKDF2 • scrypt • Argon2
MAC functions	DAA • CBC-MAC • HMAC • OMAC/CMAC • PMAC • VMAC • UMAC • Poly1305 • SipHash
Authenticated encryption modes	CCM • CWC • EAX • GCM • IAPM • OCB
Attacks	Collision attack • Preimage attack • Birthday attack • Brute-force attack • Rainbow table • Side-channel attack • Length extension attack
Design	Avalanche effect • Hash collision • Merkle-Damgård construction • Sponge function • HAIFA construction • Unique Block Iteration
Standardization	CRYPTREC • NESSIE • NIST hash function competition
Utilization	Hash-based cryptography • Key stretching • Merkle tree • Message authentication • Proof of work • Salt
V • T • E	**Cryptography** [show]

SHA-1 is well broken (alongside with pdf)

SHA-1 is well broken (alongside with pdf)

Mining, proof-of-work

Mining is finding a nonce wich contributes to a partially prescribed hash



nonce = an arbitrary meaningless number

Proof-of-work with $\mathrm{SHA256}^2$

Bitcoin uses

$$\begin{array}{rcl} \{0,1\}^{2^{64}} & \to & \{0,1\}^{256} \\ x & \mapsto & \mathrm{SHA256}(\mathrm{SHA256}(x)) \in \{0,1\}^{256} \end{array}$$

- given a "target" $\mathrm{T} \in [0, 2^{256})$
- to "mine" block-data:

  UNTIL hash $<$ T

  nonce $=$ next nonce

  hash $= H(\text{block-data} \,||\, \text{nonce})$

No better way than guessing. Probability of success for one nonce: $T/2^{256}$

$$1/903,262,006,880,187,187,200 \approx 2^{-72} \approx 10^{-24}$$

Proof-of-work with SHA256²

Bitcoin uses

$$\begin{array}{rcl} \{0,1\}^{2^{64}} & \rightarrow & \{0,1\}^{256} \\ x & \mapsto & \mathrm{SHA256}(\mathrm{SHA256}(x)) \in \{0,1\}^{256} \end{array}$$

- given a "target" $T \in [0, 2^{256})$
- to "mine" block-data:

  UNTIL hash $<$ T

  nonce $=$ next nonce

  hash $= H$(block-data $||$ nonce)

No better way than guessing. Probability of success for one nonce: $T/2^{256}$

$$1/903,262,006,880,187,187,200 \approx 2^{-72} \approx 10^{-24}$$

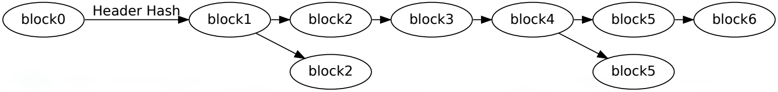
T is readjusted every 2016 blocks, to keep producing a block every 10 min

$$\texttt{difficulty} \leftarrow \texttt{difficulty} \cdot \frac{\texttt{2 weeks}}{\texttt{time to mine last 2016 blocks}}$$

where $\texttt{difficulty} \propto 1/T$

From Satochi's paper

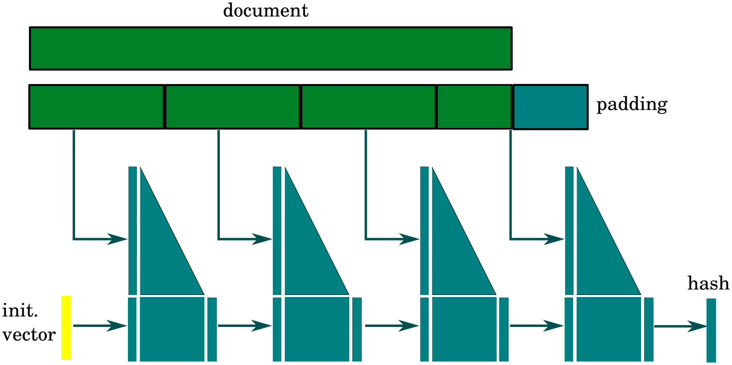
1. *new transactions are broadcast to all*† *nodes*
2. *each*† *node collects new transactions into a block*
3. *each*† *node works on finding a difficult proof-of-work for its block*
4. *when a*† *node finds a proof-of-work, it broadcasts the block to all nodes*
5. *nodes*† *accept the block only if all transactions in it are valid and not already spent*
6. *nodes*† *express their acceptance of the block by working on creating the next block in the chain, using the hash of the accepted block as the previous hash*



Outline

Merkle-Damgard

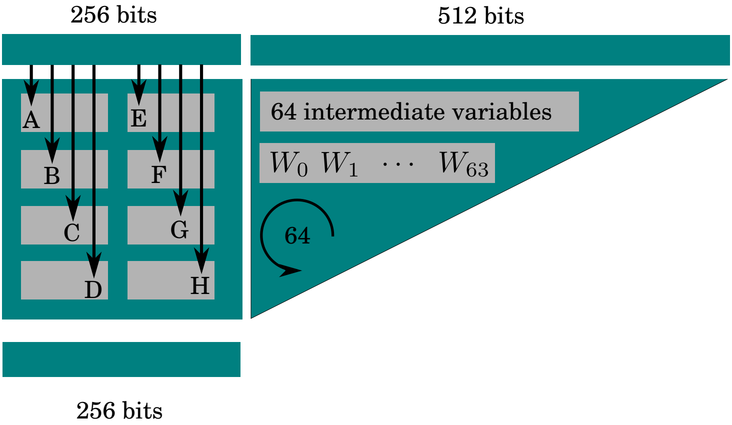
Given a compression fonction $f : \{0,1\}^m \times \{0,1\}^n \to \{0,1\}^m$
build a function $H : \{0,1\}^{2^N} \to \{0,1\}^m$



Theorem

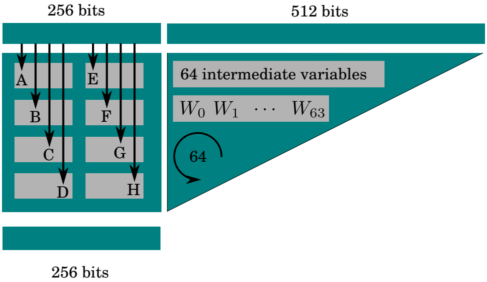
If f is secure then iterated f is secure.

SHA-256

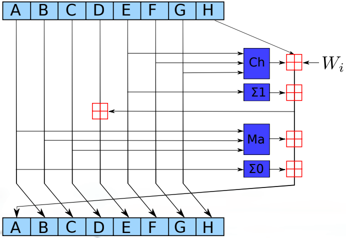


internal states $A, B \ldots, H$ and $W_0, \ldots, W_{31}$ are 32-bit long
$W_0, \ldots, W_{15}$ are equal to the message
the W_i's $i \geq 16$, are computed with a recurrence relation of length 16

Gory details of the compression function f



Repeat 64 times



$$\mathrm{Ch}(E,F,G) = (E \wedge F) \oplus (\neg E \wedge G)$$

$$\mathrm{Ma(A,B,C)} = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C)$$

$$\Sigma_0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$$

$$\Sigma_1(E) = (A \ggg 6) \oplus (A \ggg 11) \oplus (A \ggg 25)$$

$\boxplus$ is addition $\bmod 2^{32}$

$\oplus$ is 32-bit exclusive-or.

More barbarisms: example I

A	B	C	D	E	F	G	H
6a09e667	bb67ae85	3c6ef372	a54ff53a	510e527f	9b05688c	1f83d9ab	5be0cd19
5d6aebcd	6a09e667	bb67ae85	3c6ef372	fa2a4622	510e527f	9b05688c	1f83d9ab
5a6ad9ad	5d6aebcd	6a09e667	bb67ae85	78ce7989	fa2a4622	510e527f	9b05688c
c8c347a7	5a6ad9ad	5d6aebcd	6a09e667	f92939eb	78ce7989	fa2a4622	510e527f
d550f666	c8c347a7	5a6ad9ad	5d6aebcd	24e00850	f92939eb	78ce7989	fa2a4622
04409a6a	d550f666	c8c347a7	5a6ad9ad	43ada245	24e00850	f92939eb	78ce7989
2b4209f5	04409a6a	d550f666	c8c347a7	714260ad	43ada245	24e00850	f92939eb
e5030380	2b4209f5	04409a6a	d550f666	9b27a401	714260ad	43ada245	24e00850
85a07b5f	e5030380	2b4209f5	04409a6a	0c657a79	9b27a401	714260ad	43ada245
8e04ecb9	85a07b5f	e5030380	2b4209f5	32ca2d8c	0c657a79	9b27a401	714260ad
8c87346b	8e04ecb9	85a07b5f	e5030380	1cc92596	32ca2d8c	0c657a79	9b27a401
4798a3f4	8c87346b	8e04ecb9	85a07b5f	436b23e8	1cc92596	32ca2d8c	0c657a79
f71fc5a9	4798a3f4	8c87346b	8e04ecb9	816fd6e9	436b23e8	1cc92596	32ca2d8c
87912990	f71fc5a9	4798a3f4	8c87346b	1e578218	816fd6e9	436b23e8	1cc92596
d932eb16	87912990	f71fc5a9	4798a3f4	745a48de	1e578218	816fd6e9	436b23e8
c0645fde	d932eb16	87912990	f71fc5a9	0b92f20c	745a48de	1e578218	816fd6e9
b0fa238e	c0645fde	d932eb16	87912990	07590dcd	0b92f20c	745a48de	1e578218
21da9a9b	b0fa238e	c0645fde	d932eb16	8034229c	07590dcd	0b92f20c	745a48de

More barbarisms: example II

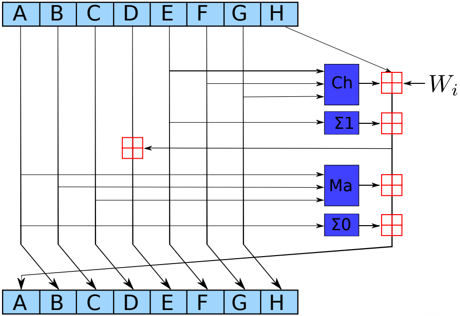
c2fbd9d1	21da9a9b	b0fa238e	c0645fde	846ee454	8034229c	07590dcd	0b92f20c
fe777bbf	c2fbd9d1	21da9a9b	b0fa238e	cc899961	846ee454	8034229c	07590dcd
e1f20c33	fe777bbf	c2fbd9d1	21da9a9b	b0638179	cc899961	846ee454	8034229c
9dc68b63	e1f20c33	fe777bbf	c2fbd9d1	8ada8930	b0638179	cc899961	846ee454
c2606d6d	9dc68b63	e1f20c33	fe777bbf	e1257970	8ada8930	b0638179	cc899961
a7a3623f	c2606d6d	9dc68b63	e1f20c33	49f5114a	e1257970	8ada8930	b0638179
c5d53d8d	a7a3623f	c2606d6d	9dc68b63	aa47c347	49f5114a	e1257970	8ada8930
1c2c2838	c5d53d8d	a7a3623f	c2606d6d	2823ef91	aa47c347	49f5114a	e1257970
cde8037d	1c2c2838	c5d53d8d	a7a3623f	14383d8e	2823ef91	aa47c347	49f5114a
b62ec4bc	cde8037d	1c2c2838	c5d53d8d	c74c6516	14383d8e	2823ef91	aa47c347
77d37528	b62ec4bc	cde8037d	1c2c2838	edffbff8	c74c6516	14383d8e	2823ef91
363482c9	77d37528	b62ec4bc	cde8037d	6112a3b7	edffbff8	c74c6516	14383d8e
a0060b30	363482c9	77d37528	b62ec4bc	ade79437	6112a3b7	edffbff8	c74c6516
ea992a22	a0060b30	363482c9	77d37528	0109ab3a	ade79437	6112a3b7	edffbff8
73b33bf5	ea992a22	a0060b30	363482c9	ba591112	0109ab3a	ade79437	6112a3b7
98e12507	73b33bf5	ea992a22	a0060b30	9cd9f5f6	ba591112	0109ab3a	ade79437
fe604df5	98e12507	73b33bf5	ea992a22	59249dd3	9cd9f5f6	ba591112	0109ab3a
a9a7738c	fe604df5	98e12507	73b33bf5	085f3833	59249dd3	9cd9f5f6	ba591112
65a0cfe4	a9a7738c	fe604df5	98e12507	f4b002d6	085f3833	59249dd3	9cd9f5f6
41a65cb1	65a0cfe4	a9a7738c	fe604df5	0772a26b	f4b002d6	085f3833	59249dd3

More barbarisms: example III

```
34df1604 41a65cb1 65a0cfe4 a9a7738c a507a53d 0772a26b f4b002d6 085f3833
6dc57a8a 34df1604 41a65cb1 65a0cfe4 f0781bc8 a507a53d 0772a26b f4b002d6
79ea687a 6dc57a8a 34df1604 41a65cb1 1efbc0a0 f0781bc8 a507a53d 0772a26b
d6670766 79ea687a 6dc57a8a 34df1604 26352d63 1efbc0a0 f0781bc8 a507a53d
df46652f d6670766 79ea687a 6dc57a8a 838b2711 26352d63 1efbc0a0 f0781bc8
17aa0dfe df46652f d6670766 79ea687a decd4715 838b2711 26352d63 1efbc0a0
9d4baf93 17aa0dfe df46652f d6670766 fda24c2e decd4715 838b2711 26352d63
26628815 9d4baf93 17aa0dfe df46652f a80f11f0 fda24c2e decd4715 838b2711
72ab4b91 26628815 9d4baf93 17aa0dfe b7755da1 a80f11f0 fda24c2e decd4715
a14c14b0 72ab4b91 26628815 9d4baf93 d57b94a9 b7755da1 a80f11f0 fda24c2e
4172328d a14c14b0 72ab4b91 26628815 fecf0bc6 d57b94a9 b7755da1 a80f11f0
05757ceb 4172328d a14c14b0 72ab4b91 bd714038 fecf0bc6 d57b94a9 b7755da1
f11bfaa8 05757ceb 4172328d a14c14b0 6e5c390c bd714038 fecf0bc6 d57b94a9
7a0508a1 f11bfaa8 05757ceb 4172328d 52f1ccf7 6e5c390c bd714038 fecf0bc6
886e7a22 7a0508a1 f11bfaa8 05757ceb 49231c1e 52f1ccf7 6e5c390c bd714038
101fd28f 886e7a22 7a0508a1 f11bfaa8 529e7d00 49231c1e 52f1ccf7 6e5c390c
f5702fdb 101fd28f 886e7a22 7a0508a1 9f4787c3 529e7d00 49231c1e 52f1ccf7
3ec45cdb f5702fdb 101fd28f 886e7a22 e50e1b4f 9f4787c3 529e7d00 49231c1e
38cc9913 3ec45cdb f5702fdb 101fd28f 54cb266b e50e1b4f 9f4787c3 529e7d00
fcd1887b 38cc9913 3ec45cdb f5702fdb 9b5e906c 54cb266b e50e1b4f 9f4787c3
```

More barbarisms: example IV

```
c062d46f fcd1887b 38cc9913 3ec45cdb 7e44008e 9b5e906c 54cb266b e50e1b4f
ffb70472 c062d46f fcd1887b 38cc9913 6d83bfc6 7e44008e 9b5e906c 54cb266b
b6ae8fff ffb70472 c062d46f fcd1887b b21bad3d 6d83bfc6 7e44008e 9b5e906c
b85e2ce9 b6ae8fff ffb70472 c062d46f 961f4894 b21bad3d 6d83bfc6 7e44008e
04d24d6c b85e2ce9 b6ae8fff ffb70472 948d25b6 961f4894 b21bad3d 6d83bfc6
d39a2165 04d24d6c b85e2ce9 b6ae8fff fb121210 948d25b6 961f4894 b21bad3d
506e3058 d39a2165 04d24d6c b85e2ce9 5ef50f24 fb121210 948d25b6 961f4894
```



Outline

Block header

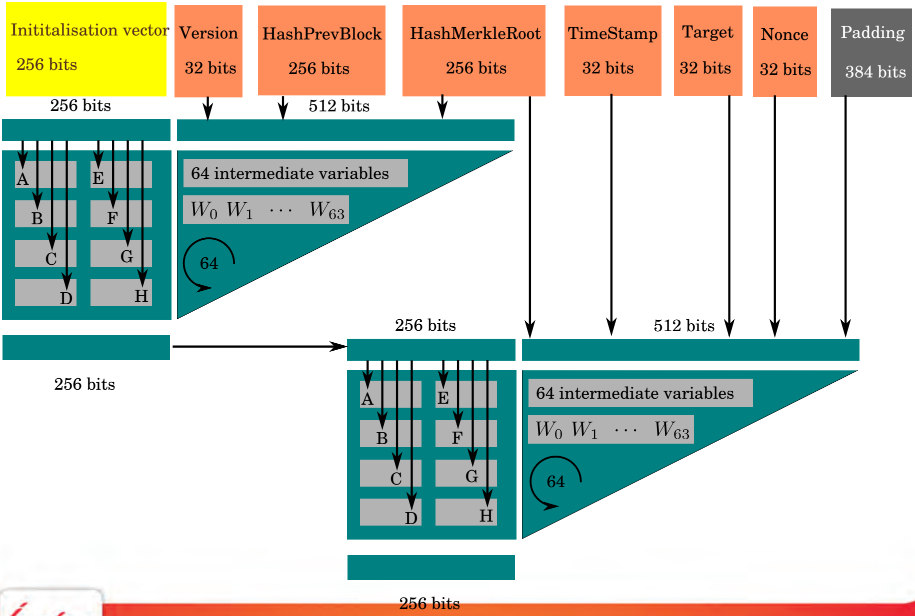
- compute the hash of the transations (HashMerkleRoot),
- link to the hash of the previous block
- write the difficulty (target)
- time stamp
- search for the proof-of-work (nonce)

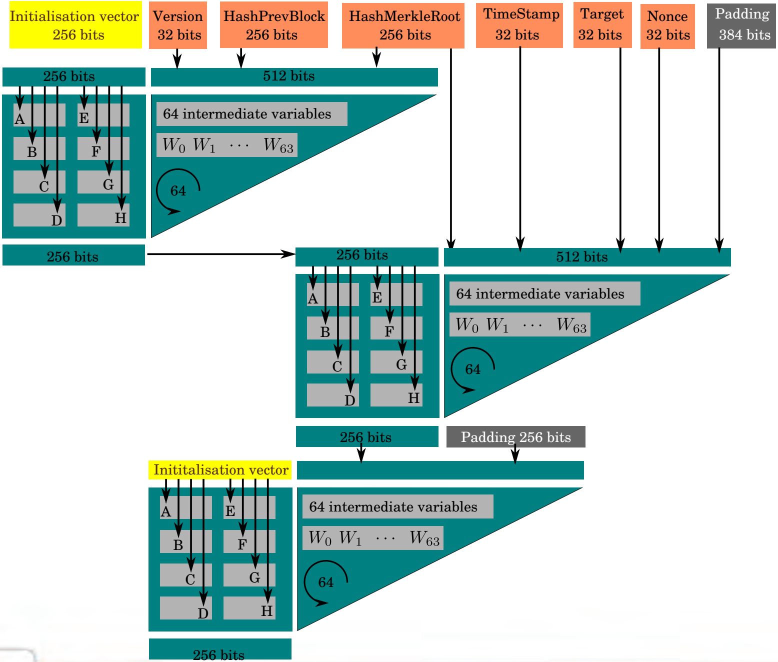
Version	HashPrevBlock	HashMerkleRoot	TimeStamp	Target	Nonce
32 bits	256 bits	256 bits	32 bits	32 bits	32 bits

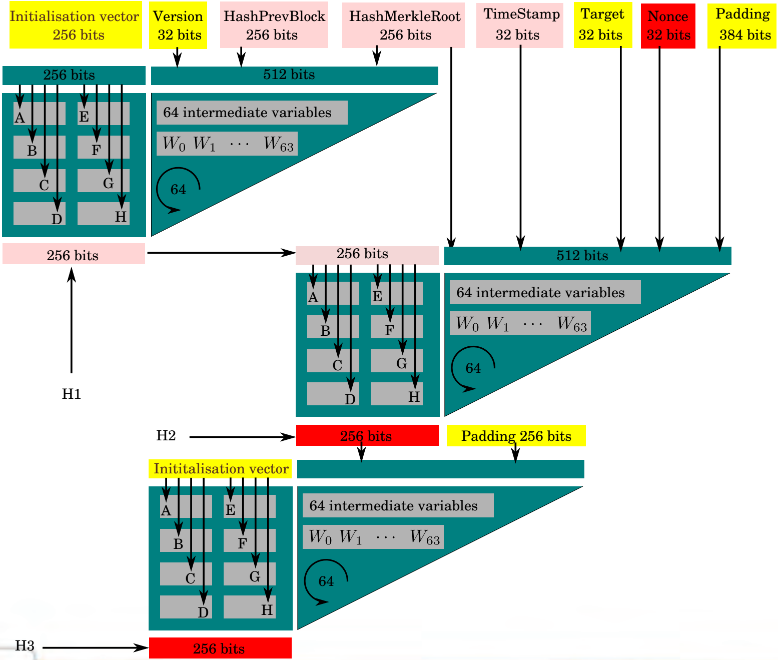
512 bits | 512 bits

That gives the hash of the block, to be put in next block

Proof-of-work




Initialisation vector 256 bits
Version 32 bits
HashPrevBlock 256 bits
HashMerkleRoot 256 bits
TimeStamp 32 bits
Target 32 bits
Nonce 32 bits
Padding 384 bits
256 bits
512 bits
A B C D E F G H
64 intermediate variables
W_0 W_1 $\cdots$ W_{63}
64
256 bits
256 bits
512 bits
A B C D E F G H
64 intermediate variables
W_0 W_1 $\cdots$ W_{63}
64
256 bits
Padding 256 bits
Inititalisation vector
A B C D E F G H
64 intermediate variables
W_0 W_1 $\cdots$ W_{63}
64
256 bits


Initialisation vector 256 bits
Version 32 bits
HashPrevBlock 256 bits
HashMerkleRoot 256 bits
TimeStamp 32 bits
Target 32 bits
Nonce 32 bits
Padding 384 bits
256 bits
512 bits
A B C D E F G H
64 intermediate variables
W_0 W_1 $\cdots$ W_{63}
64
256 bits
H1
256 bits
512 bits
A B C D E F G H
64 intermediate variables
W_0 W_1 $\cdots$ W_{63}
64
H2
256 bits
Padding 256 bits
Inititalisation vector
A B C D E F G H
64 intermediate variables
W_0 W_1 $\cdots$ W_{63}
64
H3
256 bits

Hacking

- amortized number of calls to f: $2 + \frac{1}{2^{32}}$
- saving rounds
  - save 3 rounds at the end for $H2$: $1 + \frac{61}{64}$
  - incrementing the nonce leads to just increment values at round 3
- many other tricks: amortized cost/nonce: 1.89 calls to f

Nicolas T. Courtois, Marek Grajek, and Rahul Naik. "The Unreasonable Fundamental Incertitudes Behind Bitcoin Mining". In: *CoRR* abs/1310.7935 (2013)

In mining hardware, the last three rounds are not computed (early abort).

Timo Hanke. "AsicBoost - A Speedup for Bitcoin Mining". In: *CoRR* abs/1604.00575 (2016). arXiv: 1604.00575

Outline

Changing the hash function

- Given a hash function H with w output bits, and an integer n,
- Given an input X, compute $\texttt{scrypt}^H(X, n)$ as follows
  1. Fill a table
  $$X_0 = X$$
  $$X_i = H(X_{i-1}), \quad i = 1, \ldots, n-1$$
  2. Access the table
  $$S_0 = H(X_{n-1})$$
  $$S_i = h(S_{i-1} \oplus X_{S_{i-1} \bmod n}), \quad i = 1, \ldots, n$$
  3. output S_n.

C. Percival and S Josefsson. *The scrypt Password-Based Key Derivation Function*. RFC – Request for comments 7914. IETF, Aug. 2016

Does it make sense ?

Theorem

Any algorithm $A^H(X, n)$, outputing $\texttt{scrypt}^H(X, n)$ requires cumulative memory complexity

$$\mathrm{cc}(A^H(X, n)) \gtrsim \frac{1}{25} \cdot n^2(w^2 - 4\log n).$$

Joël Alwen et al. "Scrypt Is Maximally Memory-Hard". In: *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part III*. 2017, pp. 33–62

Outline

Ethash: principle

1. block $\rightarrow$ seed
2. seed $\rightarrow$ 32Mb cache
3. cache $\rightarrow$ 1Gb dataset
4. Mining requires to hash random slices from the dataset.
5. Verifying only needs the cache

Generating the cache

The cache production process involves first sequentially filling up 32 MB of memory, then performing two passes of Sergio Demian Lerner's RandMemoHash algorithm[...]

```
def mkcache(cache_size, seed):
    n = cache_size // HASH_BYTES

    # Sequentially produce the initial dataset
    o = [sha3_512(seed)]
    for i in range(1, n):
        o.append(sha3_512(o[-1]))

    # Use a low-round version of randmemohash
    for _ in range(CACHE_ROUNDS):
        for i in range(n):
            v = o[i][0] % n
            o[i] = sha3_512(map(xor, o[(i-1+n) % n], o[v]))

    return o
```

Generating the dataset

```
def calc_dataset_item(cache, i):
    n = len(cache)
    r = HASH_BYTES // WORD_BYTES
    # initialize the mix
    mix = copy.copy(cache[i % n])
    mix[0] ^= i
    mix = sha3_512(mix)
    # fnv it with a lot of random cache nodes based on i
    for j in range(DATASET_PARENTS):
        cache_index = fnv(i ^ j, mix[j % r])
        mix = map(fnv, mix, cache[cache_index % n])
    return sha3_512(mix)

def calc_dataset(full_size, cache):
    return [calc_dataset_item(cache, i) for i in range(
                                  full_size // HASH_BYTES)]

def fnv(x1, x2):
    return ((x1 * 16777619) ^ x2) % 2**32
```

Hashimoto Hash

```
def hashimoto(header, nonce, full_size, dataset_lookup):
  # combine header+nonce into a 64 byte seed
  s = sha3_512(header + nonce[::-1])
  # start the mix with replicated s
  mix = [s for _ in range(MIX_SIZE)]
  # mix in random dataset nodes
  for i in range(ACCESSES):
    p = fnv(i ^ s[0], mix[i % blabla]) % ...blabla
    newdata = [dataset_lookup(p+j) for j in range(MIX_SIZE)]
    mix = map(fnv, mix, newdata)
  # compress mix
  cmix=[fnv4(mix[i],mix[i+1],mix[i+2],mix[i+3]) for i in
                                             range(0,len(mix),4)]
  return {
      "mix digest": serialize_hash(cmix),
      "result": serialize_hash(sha3_256(s+cmix))
  }
```

Invocation

```
def hashimoto(header, nonce, full_size, dataset_lookup):
  ...
  for i in range(ACCESSES):
    ...
    newdata = [dataset_lookup(p+j) for j in range(MIX_SIZE)]
    ...
  ...
  return {
    ...
  }
```

```
def hashimoto_light(full_size, cache, header, nonce):
  return hashimoto(header, nonce, full_size, lambda x:
                                     calc_dataset_item(cache, x))

def hashimoto_full(full_size, dataset, header, nonce):
  return hashimoto(header, nonce, full_size, lambda x:
                                     dataset[x])
```

Outline

Equihash: requirements

Alex Biryukov and Dmitry Khovratovich. "Equihash: asymmetric proof-of-work based on the Generalized Birthday problem". In: *Network and Distributed System Security Symposium (NDSS)*. 2016

- Progress-free
- Large AT-cost
- Small proofs, fast verification
- Steep time-space tradeoffs

$$C(q) = \frac{\text{Time}(M/q)}{\text{Time(M)}}$$

- Flexibility
- Bandwith-hard parallelism
- "Optimization-free": most efficient algorithm is used

General birthday problem

- Given (small) k, and L n-bits words, $X_1, \ldots, X_L$
- find $i_1, \ldots, i_{2^k}$ such that

$$X_1 \oplus \cdots \oplus X_{i_{2^k}} = 0$$

- For instance using a hash function as a pseudo-random generator requires

$$H(i_1) \oplus \cdots \oplus H(i_{2^k}) = 0$$

- For $k = 1$, reduces to the well birthday paradox
- Probabilistic bound

$$|L| \geq 2^{n/2^k}$$

- Wagner's algorithm has complexity

$$2^{n/(k+1)}$$

Wagner's algorithm

- **Input** List L de N mots de n bits
- **Do the following steps**
  1. Enumerate the list as $\{X_1, \dots, X_N\}$ and store pairs (X_i, i) in a table
  2. Sort the table by X_i.
     Find all pairs (i, j) such that X_i and X_j collide on the first $n/(k+1)$ bits
  3. Store all tuples $(X_{i,j} = X_i \oplus X_j, i, j)$ in a table
  4. Redo the previous step to find collisions on the $X_{i,j}$'s in the next $n/(k+1)$ bits
     Store all tuples $(X_{i,j,k,l} = X_{i,j} \oplus X_{k,l}, i, j, k.l)$ in a table

  ... Repeat the previous step for the next $k+1$ bits,

  ... and so on until only the last $2n/(k+1)$ bits are non-zero.

  $k+1$ At the last step, find a collision on the last $2n/(k+1)$ bits.
- **Output** This gives a solution to the original problem

Time and Memory

Proposition

For $N = 2^{n/(k+1)+1}$, Wagner's algorithm finds two solutions on average using memory

$$M(n,k) = (2^{k-1} + n)2^{n/(k+1)+1}$$

and time

$$T(n,k) = (k+1)2^{n/(k+1)+1}.$$

Time-memory tradeoffs

Using $qM(n,k)$ memory, one can find $2q^{k+1}$ solutions with time $qT(n,k)$, thus an amortized cost divided by q^{k-1}.

Proposal

Parameters

- a cryptographic hash function H
- n, k, d, l

Given a seed I, the prover has to find a nonce V and $x_1, \ldots, x_{2^k}$ such that

1. Birthday
$$H(I|V|i_1) \oplus \cdots \oplus H(I|V|i_{2^k}) = 0$$
2. Difficulty
$$H(I|V|x_1|\ldots|x_{2^k})\text{has } d \text{ leading zeros}$$
3. Algorithm binding
$$H(I|V|x_{u2^l+1}) \oplus \cdots \oplus H(I|V|x_{u2^l+2^l})\text{has } nl/(k+1) \text{ leading zeros}$$
and
$$(x_{u2^l+1}|\ldots|x_{u2^l+2^{l-1}}) <_{\text{lex}} (x_{u2^l+2^{l+1}+1}|\ldots|x_{u2^l+2^l})$$

Claim

Constrained algorithms, parallelism

Using $M(n,k)/q$ memory, a solution can be found in time

$$2^k q^{k/2} k^{k/2-1}.$$

With $p \ll T(n,k)$ processors and $M(n,k)$ shared memory a user can find 2 algorithm-bound solutions in time $T(n,k)$. Additionally, the memory bandwidt grows by the factor of p.

a reference implementation of a proof-of-work requiring 700 MB of RAM runs in 30 seconds on a 1.8 GHz CPU, increases the computations by the factor of 1000 if memory is halved[...]

Epistemology

These results are not lower bound, or “negative” (i.e. “positive” for crypto).

Conclusion

1. SHA256 (bitcoin): actually designed for software/hardwate $\Longrightarrow$ ASIC
2. Scrypt (litecoin): symmetry of memory use for finding/verifying the nounce
3. Ethash (ethereum): asymmetry, yet quite heuristic
4. Equihash (zerocash, bitcoin gold): more natural, well studied problem, yet no lower bound.

"bitcoin gold: make bitcoin decentralized again"

A good and proven proof-of work would be the proverbial silver bullet.

Conclusion

1. SHA256 (bitcoin): actually designed for software/hardwate $\Longrightarrow$ ASIC
2. Scrypt (litecoin): symmetry of memory use for finding/verifying the nounce
3. Ethash (ethereum): asymmetry, yet quite heuristic
4. Equihash (zerocash, bitcoin gold): more natural, well studied problem, yet no lower bound.

"bitcoin gold: make bitcoin decentralized again"

A good and proven proof-of work would be the proverbial silver bullet.

but would not solve other issues (bandwith, latency, energy consumption, etc)