Blockchain Technologies

Hash Function and Application

MESSAGE AUTHENTICATION

- Goal: having received a message one would like to make sure that the message has not been altered on the way (data integrity)
  - Produce a short sequence of bits that depends on the message and on a secret key
  - To authenticate the message, the partner will compute the same bit pattern, assuming he shares the same secret key
- This does not necessarily include encrypting or signing the message
  - The message can be sent in plain, with the authenticator appended
  - One may encrypt the authenticator with his private key to produce a digital signature
  - One may encrypt both the message and the authenticator

AUTHENTICATION FUNCTIONS

- Possible attacks on message authentication:
  - Content modification
  - Sequence modification – modifications to a sequence of messages, including insertion, deletion, reordering
  - Timing modification – delay or replay messages
- Some types of authentication functions exist
  - Message encryption – the ciphertext serves as authenticator
  - Hash function – a public function mapping an arbitrary length message into a fixed- length hash value to serve as authenticator
    - This does not provide a digital signature because there is no key

HASH FUNCTION

- ➢Takes any string as input
- ➢ Fixed-size output (we'll use 256 bits)
- ➢Efficiently computable
- ➢Three mathematically requierments
  - ➢Preimage resistance
  - ➢Second-preimage resistance
  - ➢Collision-resistance
- ➢Security properties
  - ➢Collision-resistance
  - ➢Hiding
  - ➢Puzzle-friendly

HASH FUNCTIONS

- A fixed-length hash value h is generated by a function H that takes as input a message of arbitrary length: $h = H(M)$
  - **A** sends M and $H(M)$
  - **B** authenticates the message by computing $H(M)$ and checking the match

- Basic requirements for a hash function:
  - H can be applied to a message of any size
  - H produces fixed-length output
  - It is easy to compute $H(M)$

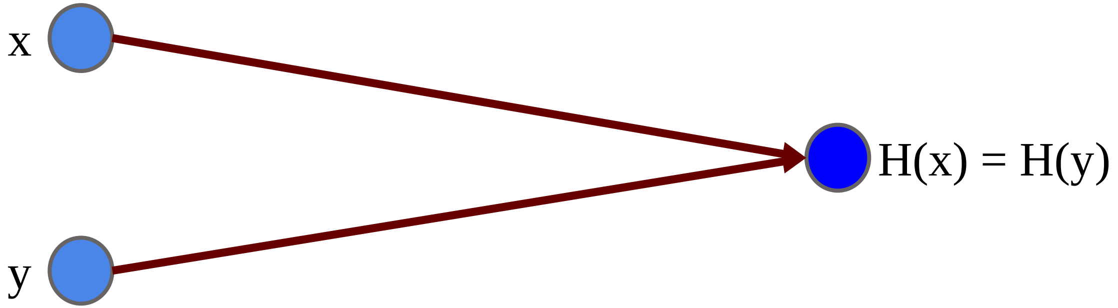
REQUIREMENTS FOR A HASH FUNCTION

- Preimage resistance property: for a given h, it is computationally infeasible to find M such that $H(M) = h$.
- Second-preimage resistance property: for a given M, it is computationally infeasible to find $M' \neq M$ such that $H(M') = H(M)$.
- Collision-resistance property: it is computationally infeasible to find M, M' with $H(M) = H(M')$

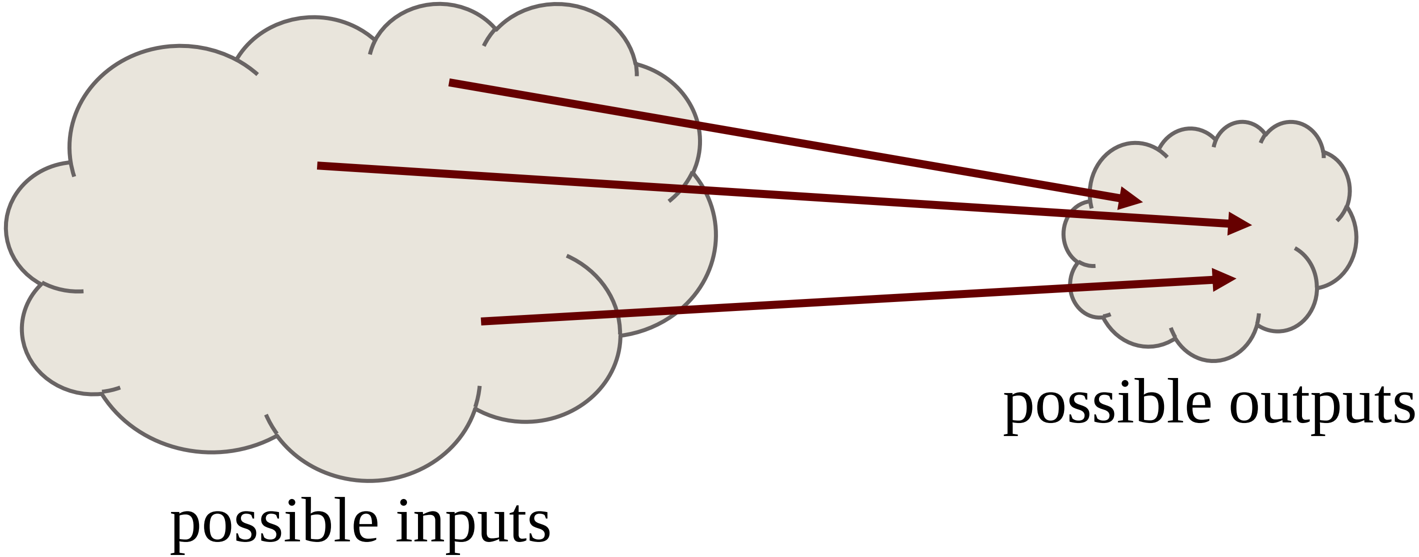
HASH PROPERTY 1: COLLISION-RESISTANCE

Nobody can find x and y such that

x != y and H(x)=H(y)



Collisions do exist ...



… but can anyone find them?

APPLICATION: HASH AS MESSAGE DIGEST

If we know H(x) = H(y),

it’s safe to assume that x = y.

To recognize a file, especialy a large file,

just remember its hash.

Useful because the hash is small.

HASH PROPERTY 2: HIDING

We want something like this:

Given H(x), it is infeasible to find x.

H("heads")

H("tails")

easy to find x!

HASH PROPERTY 2: HIDING

If r is chosen from a probability distribution that has *high min-entropy*, then given H(r | x), it is infeasible to find x.

concatnation

High min-entropy means that the distribution is “very spread out”, so that no particular value is chosen with more than negligible probability.

APPLICATION: COMMITMENT

Want to “seal a value in an envelope”, and
“open the envelope” later.

Commit to a value, reveal it later.

COMMITMENT API

(*com*, *key*) := commit(*msg*)

match := verify(*com*, *key*, *msg*)

To seal *msg* in envelope:

(*com*, *key*) := commit(*msg*) -- then publish *com*

To open envelope:

publish *key*, *msg*

anyone can use verify() to check validity

COMMITMENT API

commit(*msg*) := (H(*key* || *msg*))

where *key* is a random 256-bit value

verify(*com*, *key*, *msg*) := (H(*key* || *msg*) == *com*)

Security properties:

Hiding: Given H(*key* || *msg*), infeasible to find *msg*.

Binding: Infeasible to find *msg* != *msg'* such that

H(*key* || *msg*) == H(*key'* || *msg'*)

Simple Hash Commitment Scheme

- Why are these hash properties useful?

Consider a simple example: Alice and Bob bet $100 on a coin flip

1) Alice calls the outcome of the coin flip
2) Bob flips the coin
3) Alice wins the $100 if her guess was correct

Now, what if Alice and Bob are separated and don't trust one another?

- Alice wants to give Bob a *commitment* to her guess, without revealing her guess before Bob flips the coin, otherwise Bob can cheat!

SIMPLE HASH COMMITMENT SCHEME

- Instead, we can modify our "protocol" to bind Alice's guess with a commitment:

1) Alice chooses a large random number, R.
2) Alice guesses the outcome of the coin flip, B.
3) Alice generates a *commitment* to the coin flip, $C = H(B \parallel R)$
4) Alice sends this commitment to Bob.
5) Bob flips the coin and sends the value to Alice.
6) Alice sends Bob the random number and her guess: (R', B')
7) Bob then checks that $C' = H(B' \parallel R') = C = H(B \parallel R)$, to ensure Alice did not change her guess mid commitment.
8) Both can now agree on who won the \$100.

SIMPLE HASH COMMITMENT SCHEME - CHEATING

- How could Bob cheat Alice?

  1) When Bob receives $C = H(B \parallel R)$, if he can compute $H^{-1}(C) = B \parallel R$, Bob can recover Alice's guess and send her the opposite outcome!

  If our hash function, H, is **preimage resistant**, this shouldn't be possible.

- How could Alice cheat Bob?

  1) Alice sends Bob her commitment $C = H(B \parallel R)$, but reveals the opposite guess, $(!B, R')$. Alice wins if she can pick R' such that $C' = H(!B \parallel R') = C$.

  This fails if our hash function, H, is **second preimage resistant**!

HASH PROPERTY 3: PUZZLE-FRIENDLY

For every possible output set value y,
if k is chosen from a distribution with high min-entropy,
then it is infeasible to find x such that H(k | x) = y.

APPLICATION: SEARCH PUZZLE

Given a "puzzle ID" *id* (from high min-entropy distrib.),

and a target set *Y*:

Try to find a "solution" *x* such that

$$H(id \mid x) \in Y.$$

Puzzle-friendly property implies that no solving strategy is much better than trying random values of *x*.

HASH PUZZLE SCHEME

- Bitcoin mining

A FEW SIMPLE HASH FUNCTIONS

- Bit-by-bit XOR of plaintext blocks: $h = D_1 \oplus D_2 \oplus \cdots \oplus D_N$
  - Provides a parity check for each bit position
  - Not very effective with text files: most significant bit always 0
  - Attack: to send blocks $X_1, X_2, \ldots, X_{N-1}$ choose:

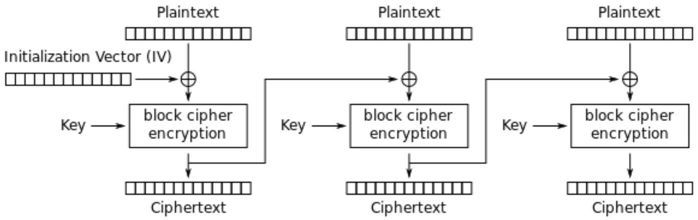
$$X_N = X_1 \oplus X_2 \oplus \cdots \oplus X_{N-1} \oplus h$$

  - Does not satisfy the preimage resistance condition: "computationally infeasible to find M such that $H(M) = h$, for a given h "

- Another example: rotated XOR – before each addition the hash value is rotated to the left with 1 bit
  - Better than the previous hash on text files
  - Similar attack

A FEW SIMPLE HASH FUNCTIONS

- Another method: cipher block chaining technique without a secret key
  - Divide message into blocks $D_1, D_2, \ldots, D_N$ and use them as keys in the encryption method (e.g., DES)
  - H_0 = some initial value, $H_i = E_{D_i}(H_{i-1})$
  - $H = H_N$
  - This can be attacked with the birthday attack if the key is short (as in DES)



Cipher Block Chaining (CBC) mode encryption

- Birthday paradox: Given at least 23 people, the probability of having two people with the same birthday is more then 0.5
- **General case**: Given two sets X, Y each having k elements from the set $\{1,2,\ldots,N\}$, how large should k be so that the probability that X and Y have a common element is more than 0.5?
  - Answer: k should be larger than $\sqrt{N}$
  - If $N = 2^m$, take $k = 2^{m/2}$

BIRTHDAY ATTACK

- Suppose a hash value on 64 bits is used (as the one based on DES)
  - In principle this is secure: given M, to find a message M' with $H(M) = H(M')$, one has to generate in average 2^{63} messages M'.
- A different much more effective attack is possible:
  - **A** is prepared to sign the document by appending its hash value (on m bits) and then encrypting the hash code with its private key
  - **E** (i.e. attacker) will generate $2^{m/2}$ variations of the message M and computes the hash values for all of them.
  - **E** also generates $2^{m/2}$ variations of the message M' that she would really like to have **A** authenticating and computes the hash values for all of them
  - Birthday paradox: the probability that the two sets of hash values have one common element is more than 0.5 - she finds $M \neq M'$ such that $H(M) = H(M')$
  - **E** will offer M to **A** for hashing and then signing and will send instead M' with the signature A has produced
  - **E** breaks the protocol although she does not know **A**'s private key with a level of effort for the hash based on DES: 2^{33}

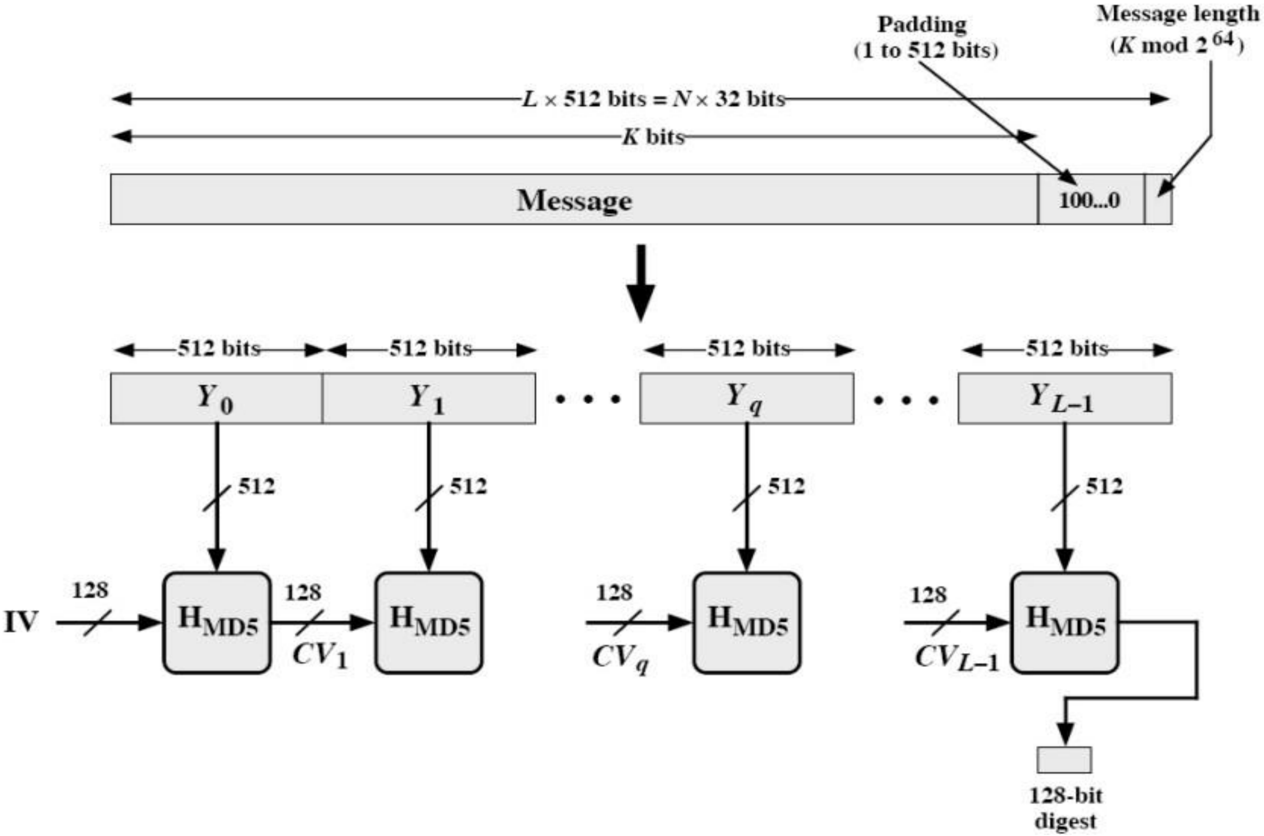
POPULAR HASH ALGORITHMS:

- ➢MD5 (Message Digest 5)
- ➢SHA1 (Secure Hash Algorithm 1)
- ➢SHA2 family: **SHA-224, SHA-256, SHA-384, SHA-512, SHA-512/224, SHA-512/256**
- ➢SHA3 (Secure Hash Algorithm 3)

MD5

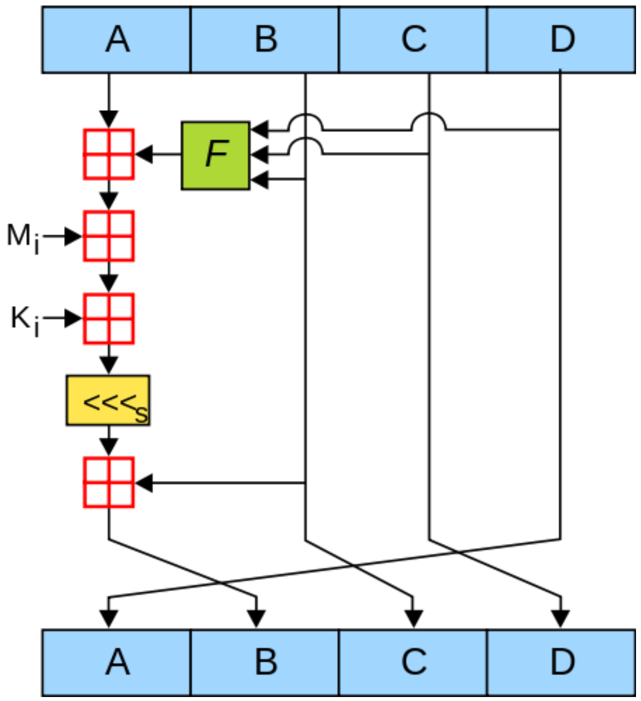
- Most popular hash algorithm until recently: concerns for its security were raised and is replaced by SHA-1, SHA-2, and SHA-3
- Developed by Ron Rivest at MIT in 1991.
- For a message of arbitrary length, it produces an output of 128 bits
  - Processes the input in blocks of 512 bits
- Idea:
  - Start by padding the message to a length of 448 bits modulo 512 – padding is always added even if the message is of required length
    - The length of the message is added on the last 64 bits so that altogether the length is a multiple of 512 bits
  - Several rounds, each round takes a block of 512 bits from the message and mixes it thoroughly with a 128 bit buffer that was the result of the previous round
  - The last content of the buffer is the hash value.

MERKLE-DAMGÅRD STRUCTURE OF MD5



MD5 OPERATIONS

- MD5 consists of 64 of these operations, grouped in four rounds of 16 operations.
- F is a nonlinear function; one function is used in each round.
- M_i denotes a 32-bit block of the message input, and K_i denotes a 32-bit constant, different for each operation.
- $\lll_s$ denotes a left bit rotation by s places; s varies for each operation.
- $\boxplus$ denotes addition modulo 2^{32}.



SECURITY ISSUES OF MD5

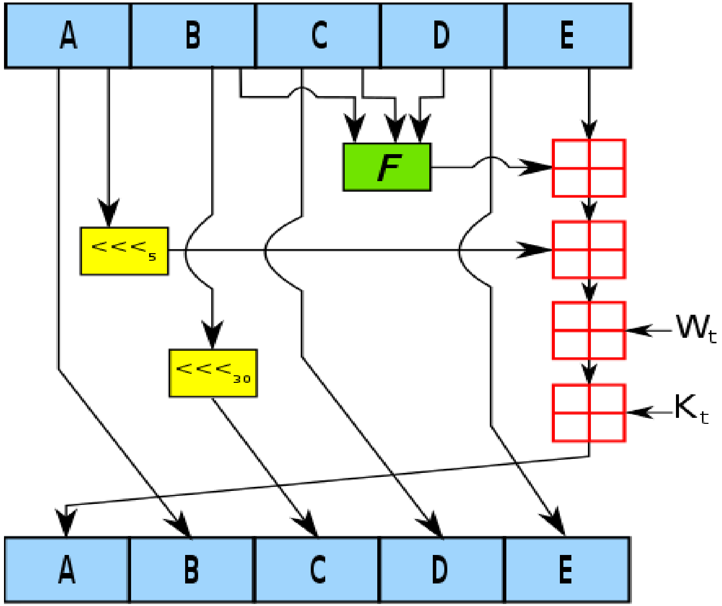
- In 1996 a flaw was found in the design of MD5.
  - While it was not deemed a fatal weakness at the time, cryptographers began recommending the use of other algorithms.
- In 2004 it was shown that MD5 is not collision-resistant.
  - As such, MD5 is not suitable for applications like SSL certificates or digital signatures that rely on this property for digital security.
- In December 2008, a group of researchers used this technique to fake SSL certificate validity.
- In 2012, the Flame malware exploited the weaknesses in MD5 to forge a Windows code-signing certificate.
  - majority of targets in Iran (more than 65%)
- Best known attack: 2013 attack by Xie Tao, Fanbao Liu, and Dengguo Feng breaks MD5 collision resistance in 2^{18} time.
  - This attack runs in less than a second on a regular computer!

SHA-1

- Developed by NSA and adopted by NIST in FIPS 180-1 (1993)
- Part of a family of 3 hashes: SHA-0, SHA-1, SHA-2
  - SHA-1 most widely used
- Design based on MD4 (previous version of MD5)
- Takes as input any message of length up to 2^{64} bits and gives a 160-bit message digest
- Microsoft, Google, Apple and Mozilla have all announced that their respective browsers will stop accepting SHA-1 SSL certificates by 2017.
- On February 23, 2017 CWI Amsterdam and Google announced they had performed a collision attack against SHA-1, publishing two dissimilar PDF files which produce the same SHA-1 hash.

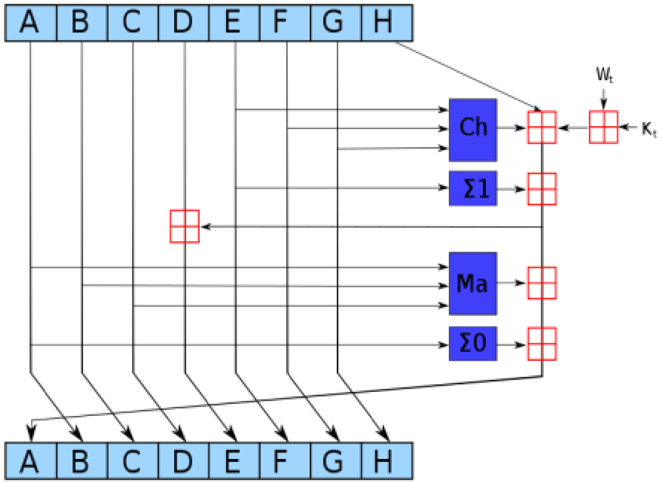
SHA-1 OPERATION

- Structure very similar to MD4 and MD5.
    - Secret design criteria
- Stronger than MD5 because of longer message digest
- Slower than MD5 because of more rounds



SHA-2

- SHA-2 similar to SHA-1, but with different input-output length.
- The algorithms are collectively known as SHA-2, named after their digest lengths: SHA-256, SHA-384, and SHA-512.
- There is no known attack against SHA-2.



$Ch(E, F, G) = (E \land F) \oplus (\neg E \land G)$

$Ma(A, B, C) = (A \land B) \oplus (A \land C) \oplus (B \land C)$

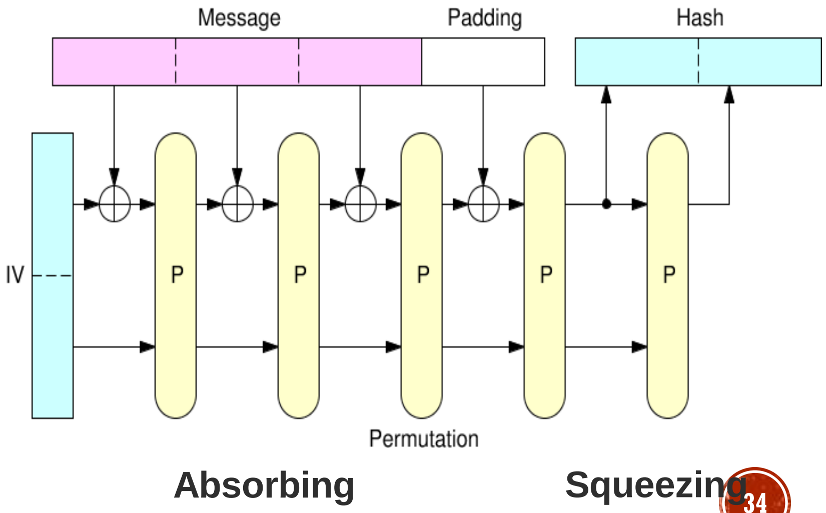
$\Sigma 0(A) = (A \ggg 2) \oplus (A \ggg 13) \oplus (A \ggg 22)$

$\Sigma 1(E) = (E \ggg 6) \oplus (E \ggg 11) \oplus (E \ggg 25)$

SHA-3

- SHA-3 is the latest member of the Secure Hash Algorithm family of standards, released by NIST on 2015 as FIPS 202.
- In 2006 NIST started to organize the NIST hash function competition to create a new hash standard, SHA-3.
  - On October 2, 2012, Keccak was selected as the winner of the competition.

Sponge construction of SHA-3:



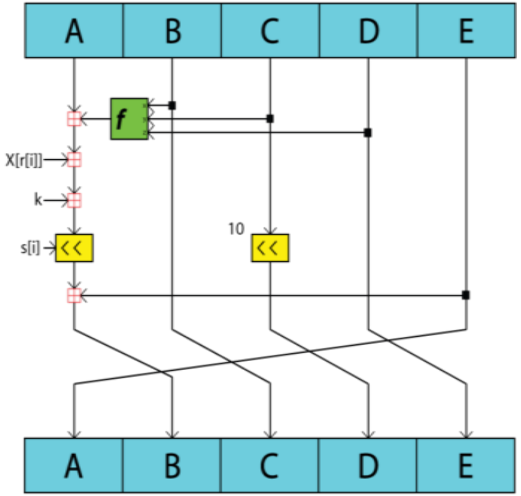
RIPEMD-160

- RIPEMD (RACE Integrity Primitives Evaluation Message Digest) is a family of cryptographic hash functions developed in Katholieke Universiteit Leuven, and first published in 1996.

- RIPEMD-160 is an improved, 160-bit version of the original RIPEMD, and the most common version in the family.

- RIPEMD-160 was designed in the open academic community, in contrast to the NSA- designed SHA-1 and SHA-2 algorithms.

- There is no known attack against RIPEMD-160.



HASH FUNCTIONS IN BITCOIN

A. Producing the public bitcoin address by hashing the public key.

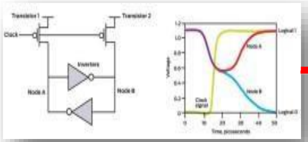
B. Producing a transaction digest for use as the input in signing a transaction.

C. Producing the hash of the previous block to use in the block header in the Blockchain.

D. Producing the Merkle tree root for authenticating the transactions in a block (using hashes all the way up the tree).

E. Producing the double hash of the block (with nonces) to find a block that satisfies the difficulty needed in mining.

A. Generating a Bitcoin Address



256 random bits

generate random private key k

Compute point kG on spec256k1 curve (public key)

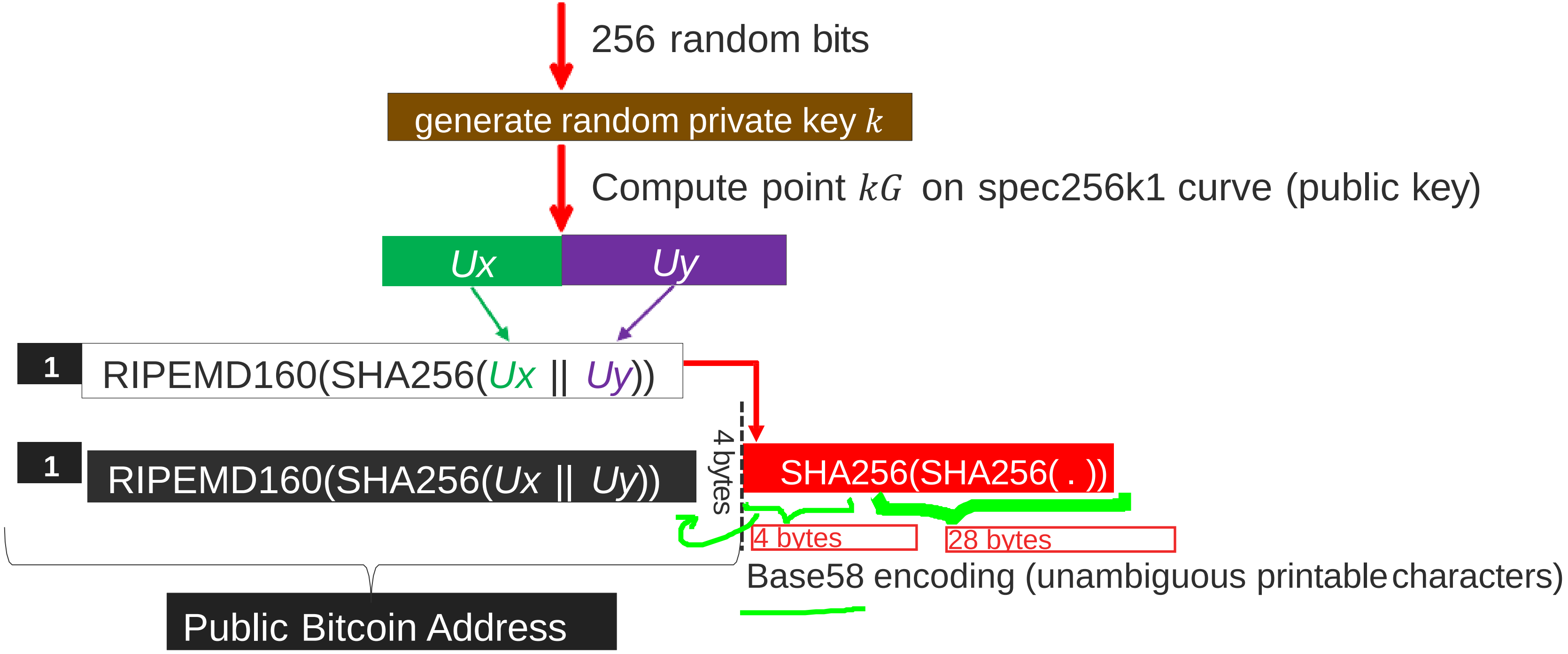
Ux	*Uy*

G = 04 79BE667E F9DCBBAC 55A06295 CE870B07 029BFCDB 2DCE28D9 59F2815B 16F81798 483ADA77 26A3C465 5DA4FBFC 0E1108A8 FD17B448 A6855419 9C47D08F FB10D4B8

1 RIPEMD160(SHA256(*Ux* || *Uy*)) 21 bytes

Add version byte in front of RIPEMD-160 hash (0x00 for public key hash in main network)

A. GENERATING A BITCOIN ADDRESS



e.g. 16UwLL9Risc3QfPqBUvKofHmBQ7wMtjvM

BASE58 ENCODING

- Why base-58 instead of standard base-64 encoding?
  - Don't want 0OIl characters that look the same in some fonts and could be used to create visually identical looking account numbers.
  - A string with non-alphanumeric characters is not as easily accepted as an account number.
  - E-mail usually won't line-break if there's no punctuation to break at.
  - Doubleclicking selects the whole number as one word if it's all alphanumeric.

```
code_string = "123456789ABCDEFGHJKLMNPQRSTUVWXYZabcdefghijkmnopqrstuvwxyz"
x = convert_bytes_to_big_integer(hash_result)  output_string = ""

while(x >0) {
      (x, remainder) = divide(x, 58)
      output_string.append(code_string[remainder])
}
```

More information: https://en.bitcoin.it/wiki/Base58Check_encoding

VANITY ADDRESSES

- Some individuals or merchants like to have an address that starts with some human-meaningful text.
- For example, the gambling website Satoshi Bones has users send money to addresses containing the string "bones" in positions 2-6, such as:

  1bonesEeTcABPjLzAb1VkFgySY6Zqu3sX

- How much work does this take?
  - Since there are 58 possibilities for every character, if you want to find an address which starts with a specific k-character string, you'll need to generate 58^k addresses on average until you get lucky.
  - So finding an address starting with "bones" would have required generating over 650 million addresses!

B. TRANSACTION DIGEST

"I, Alice, hereby pay Bob an amount of 23 mBTC"

- We said that digital signatures work on hash of messages.
- To assure people that a transaction is done by Alice, she signs the hash of transaction.

- A hash of a transaction is a double hash of the binary format of the transaction. Algorithm SHA-256 is applied twice:

```
var hash = function (encodedTransaction)
{ return sha256 (sha256 (encodedTransaction) );}
```

Blockchain Luxembourg S.A [LU] | https://www.blockchain.com/btc/tx/0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

BLOCKCHAIN WALLET DATA API ABOUT BLOCK, HASH, TRANSACTION, ETC... GET A FREE WALLET

Transaction View information about a bitcoin transaction

0627052b6f28912f2703066a912ea577f2ce4da4caa5a5fbd8a57286c345c2f2

1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK	→	1GdK9UzpHBzqzX2A9JFP3Di4weBwqgmoQA	0.015 BTC
		1Cdid9KFAaatwczBwBttQcwXYCpvK8h7FK	0.0845 BTC
			0.0995 BTC

Summary	
Size	258 (bytes)
Weight	1032
Received Time	2013-12-27 23:03:05
Included In Blocks	277316 (2013-12-27 23:11:54 + 9 minutes)
Confirmations	288154
Visualize	View Tree Chart

Inputs and Outputs	
Total Input	0.1 BTC
Total Output	0.0995 BTC
Fees	0.0005 BTC
Fee per byte	193.798 sat/B
Fee per weight unit	48.45 sat/WU
Estimated BTC Transacted	0.015 BTC
Scripts	Show scripts & coinbase

HASH POINTERS

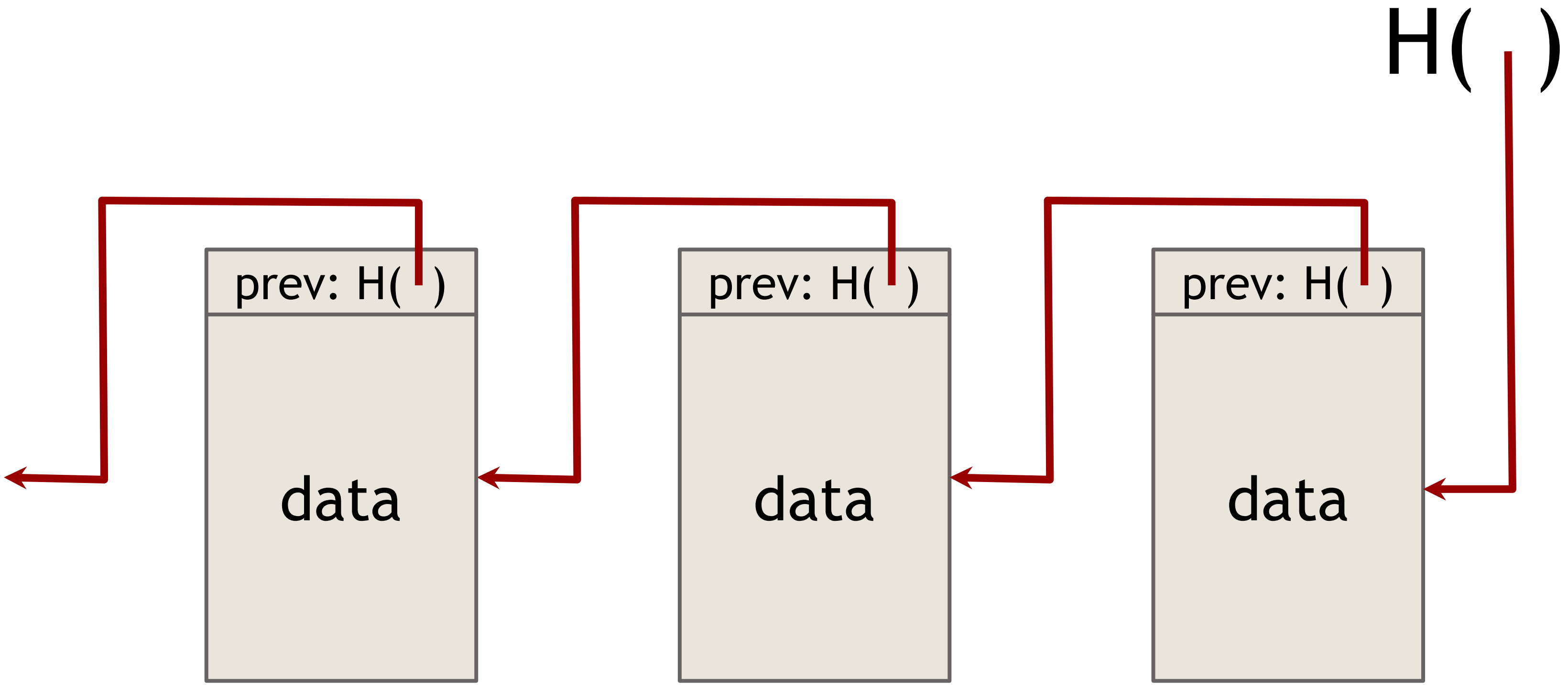
- A hash pointer is a pointer to where data is stored together with a cryptographic hash of the value of that data at some fixed point in time.

- Whereas a regular pointer gives you a way to retrieve the information, a hash pointer also gives you a way to verify that the information hasn't changed.

AUTHENTICATED DATA STRUCTURES

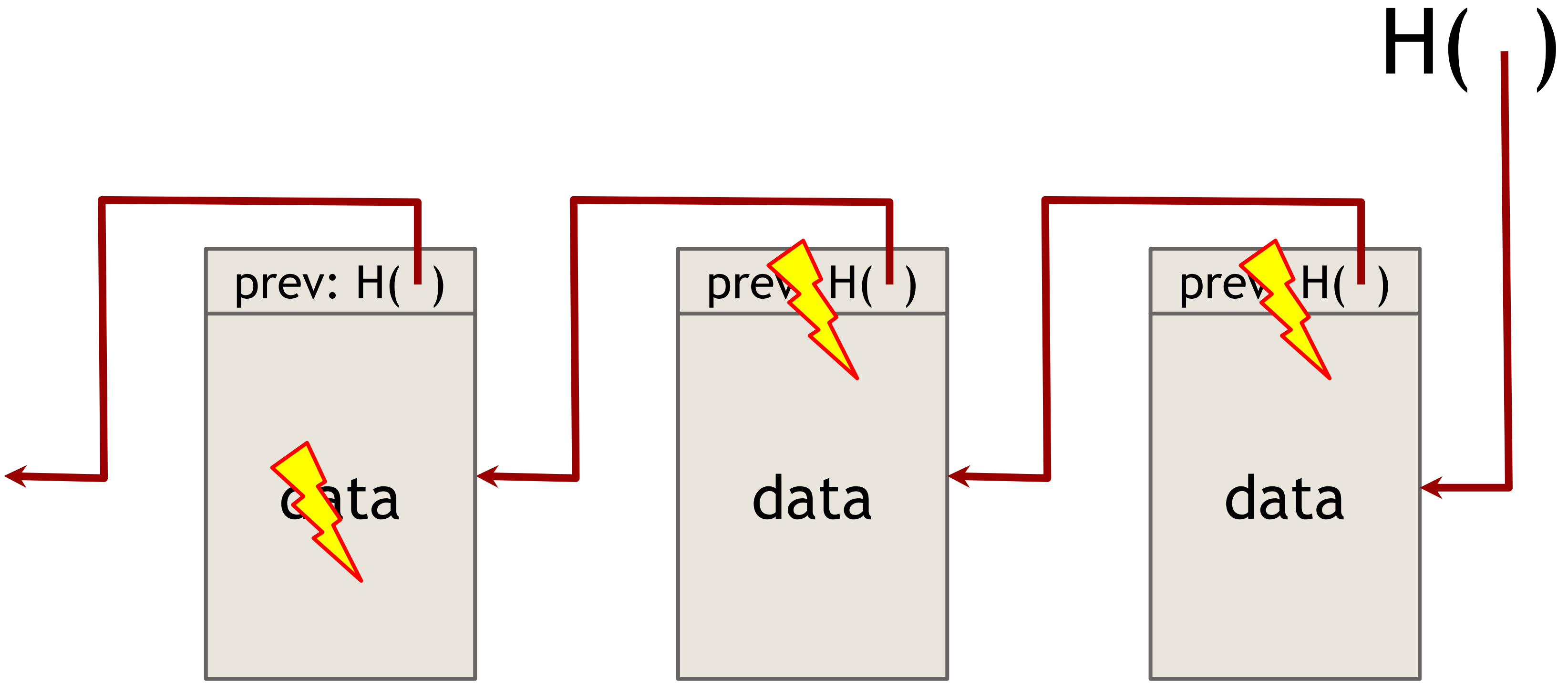
- Key idea:
  1. Take any pointer-based data structure
  2. Replace pointers with cryptographic hashes
- We now have an authenticated data structure

linked list with hash pointers = "block chain"



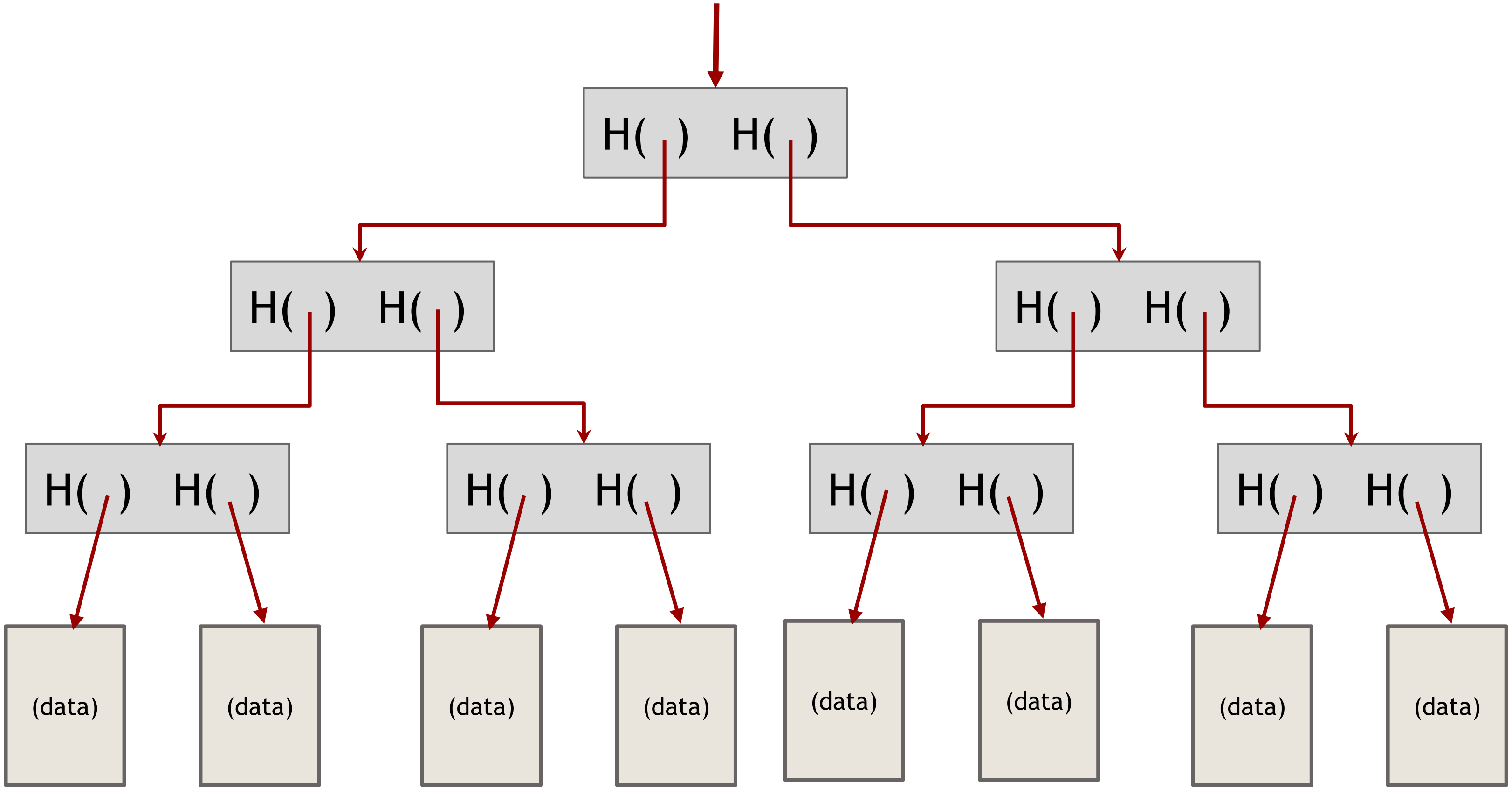
use case: tamper-evident log

detecting tampering



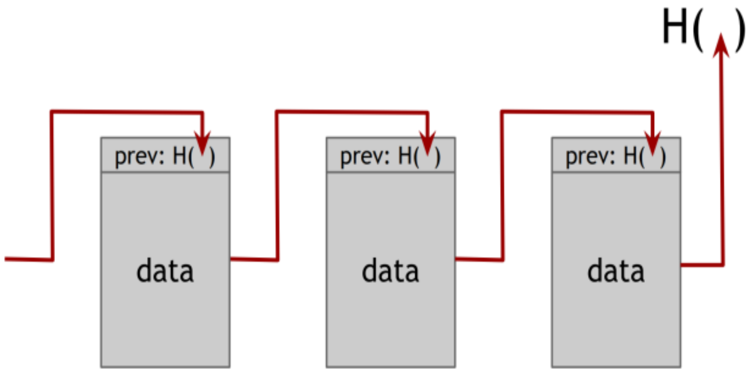
use case: tamper-evident log

binary tree with hash pointers = “Merkle tree”



C. BLOCK CHAIN

- Block chain is a linked list using hash pointers.
- Whereas as in a regular linked list where you have a series of blocks, each block has data as well as a pointer to the previous block in the list, in a block chain the previous block pointer will be replaced with a hash pointer.
- Each block not only tells us where the value of the previous block was, but it also contains a digest of that value that allows us to verify that the value hasn't changed.



BLOCK CHAIN

Blockchain Luxembourg S.A [LU] | https://www.blockchain.com/btc/block/00000000000000000021de9cb414194e2e9a8d4b520970a964a381a659ba32e0

BLOCKCHAIN WALLET DATA API ABOUT BLOCK, HASH, TRANSACTION, ETC... GET A FREE WALLET

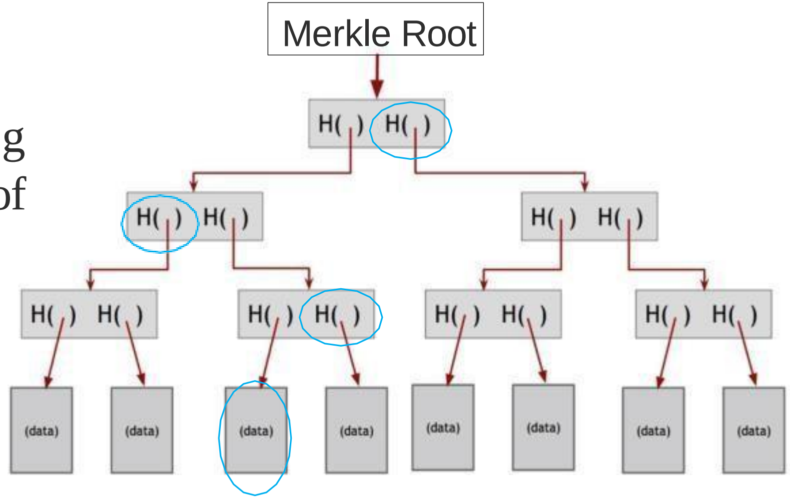
Block #565472

Summary	
Number Of Transactions	3176
Output Total	6,816.05505224 BTC
Estimated Transaction Volume	509.23929063 BTC
Transaction Fees	0.1853349 BTC
Height	565472 (Main Chain)
Timestamp	2019-03-03 11:22:25
Received Time	2019-03-03 11:22:25
Relayed By	F2Pool
Difficulty	6,071,846,049,920.75
Bits	388914000
Size	1168.147 kB
Weight	3998.149 kWU
Version	0x20000000
Nonce	604595516
Block Reward	12.5 BTC

Hashes	
Hash	00000000000000000021de9cb414194e2e9a8d4b520970a964a381a659ba32e0
Previous Block	0000000000000000000f2de9e13f28df2001a7179f6d46289e1f077c3fd1cc6e
Next Block(s)	
Merkle Root	41c7416f554451294732939f20fcd1fd6d6d010ebf1116521721769e6546131f

D. MERKLE TREE

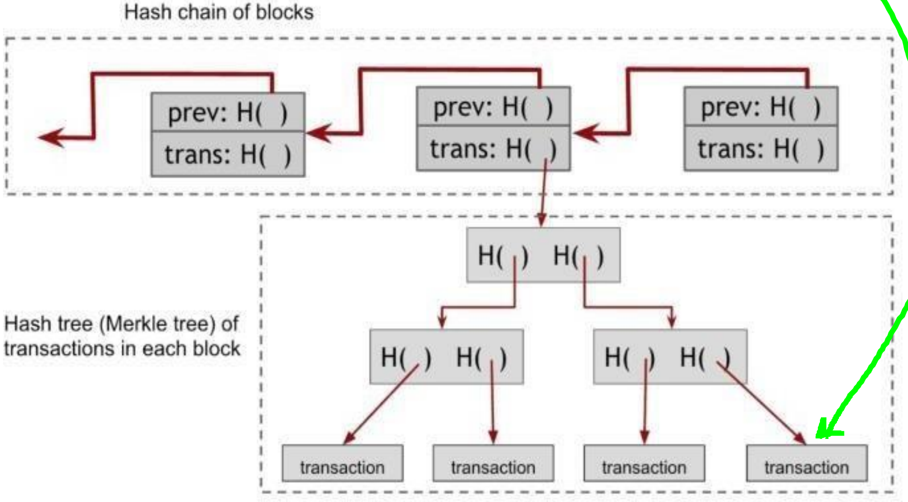
- A binary tree of hash pointers (another authenticated data structure)
  - Hashes are hashed together.
  - If there are n nodes in the tree, only about log (n) items need to be shown as proof of membership.
  - To prove inclusion of data in the Merkle tree, provide root data and intermediate hashes.
  - To fake the proof, one would need to find hash preimages.



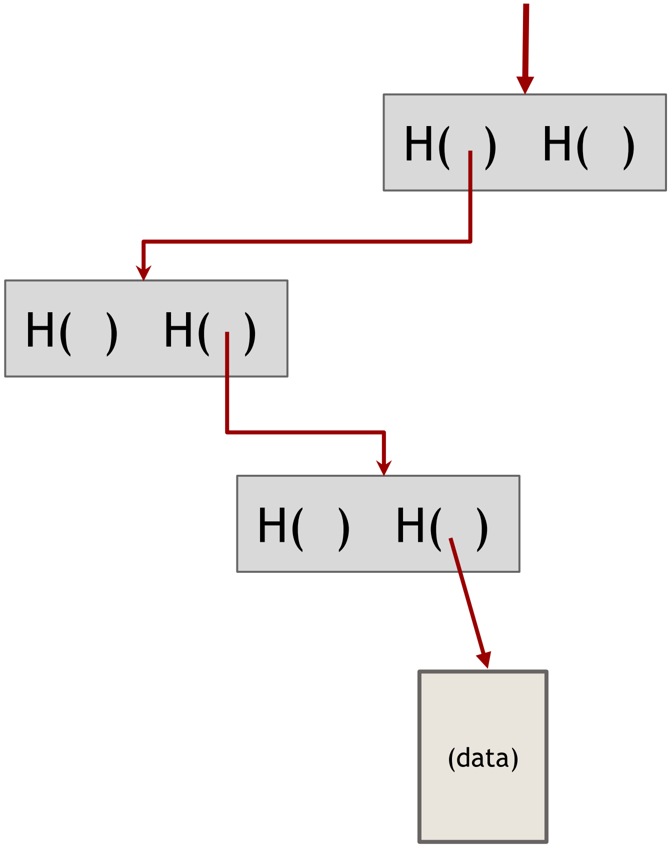
MERKLE TREE - BITCOIN CONSTRUCTION

- Transactions are leaves in the Merkle tree, includes a coinbase transaction

- Two hash structures:
  - Hash chain of blocks.
    - These blocks are linked together and based off of each other.
  - A Merkle tree of transactions, internal to each block.



proving membership in a Merkle tree



show O(log n) items

ADVANTAGES OF MERKLE TREES

Tree holds many items

but just need to remember the root hash

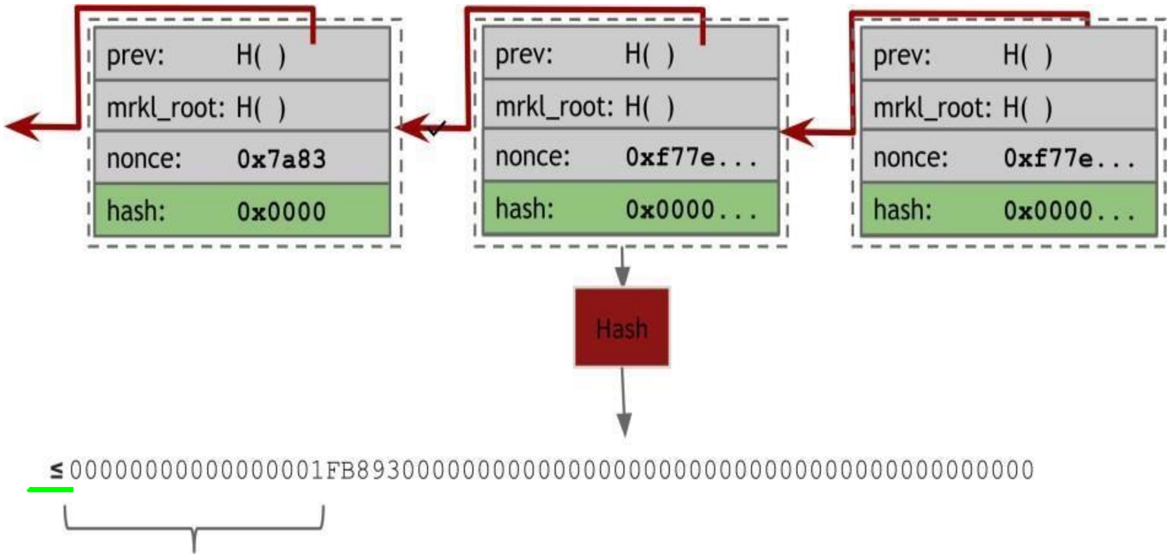
Can verify membership in O(log n) time/space

Variant: sorted Merkle tree

can verify non-membership in O(log n) ?

(show items before and after the missing one)

VALIDITY OF A BLOCK HEADER



76+ leading zeroes required ...

How to reach 76 bits zero a with 32 bit nonce?solutions:1- Reorder or d

VALIDITY OF A BLOCK HEADER

Blockchain Luxembourg S.A [LU] | https://www.blockchain.com/btc/block/00000000000000000021de9cb414194e2e9a8d4b520970a964a381a659ba32e0

BLOCKCHAIN WALLET DATA API ABOUT BLOCK, HASH, TRANSACTION, ETC... GET A FREE WALLET

Block #565472

Summary	
Number Of Transactions	3176
Output Total	6,816.05505224 BTC
Estimated Transaction Volume	509.23929063 BTC
Transaction Fees	0.1853349 BTC
Height	565472 (Main Chain)
Timestamp	2019-03-03 11:22:25
Received Time	2019-03-03 11:22:25
Relayed By	F2Pool
Difficulty	6,071,846,049,920.75
Bits	388914000
Size	1168.147 kB
Weight	3998.149 kWU
Version	0x20000000
Nonce	604595516
Block Reward	12.5 BTC

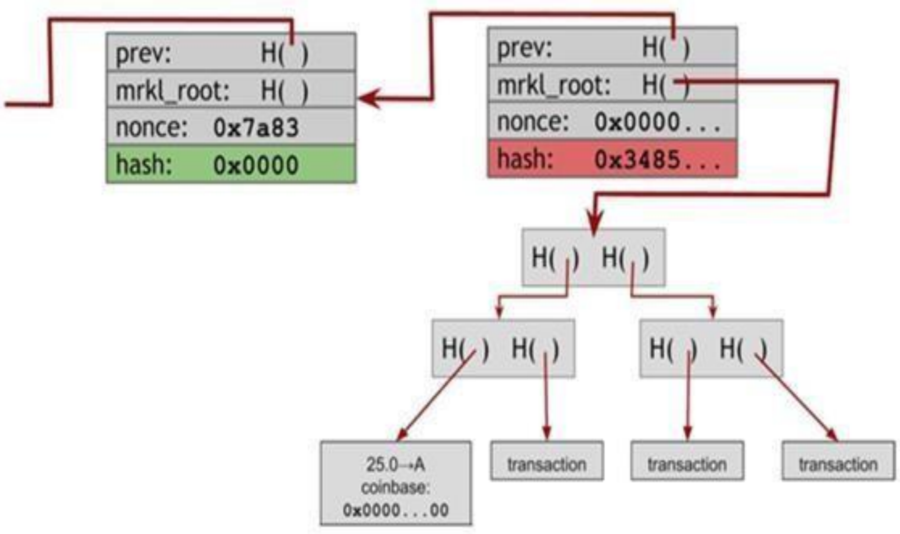
Hashes	
Hash	00000000000000000021de9cb414194e2e9a8d4b520970a964a381a659ba32e0
Previous Block	0000000000000000000f2de9e13f28df2001a7179f6d46289e1f077c3fd1cc6e
Next Block(s)	
Merkle Root	41c7416f554451294732939f20fcd1fd6d6d010ebf1116521721769e6546131f

E. Bitcoin Mining

- Previously, hash of:
  - Merkle Root
  - PrevBlockHas
  - Nonce (varied value)

below some target value.

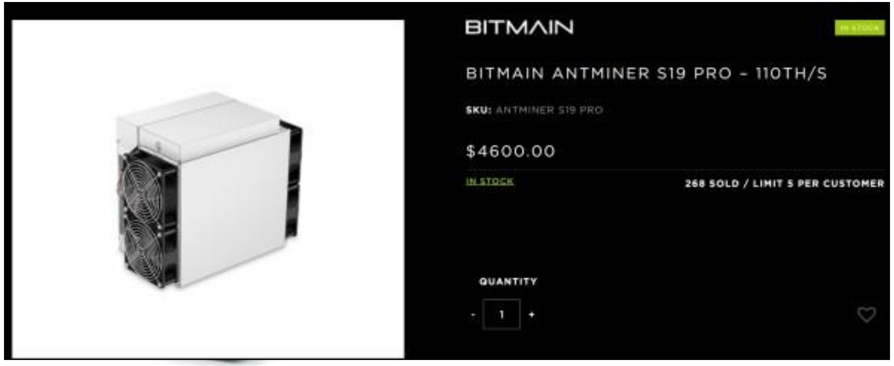
- Actually two nonces:
  - In the block header
  - In the coinbase tx
- Hash of
  - PrevBlockHash
  - Coinbase nonce (varied value)
    - Affects the Merkle Root
  - Block header nonce (varied value)



What if num nodes != 2 ^ k ?

MERKLE TREE - BITCOIN CONSTRUCTION

- What if there is no solution?
  - Block header nonce is 32 bits
    - Antminer S19 pro hashes 110 TH/s
    - How long to try all combinations?
    - 2^{32}/110,000,000,000,000 = 0.000039 sec
    - Exhausted 25600 times per second



- Therefore, must change Merkle root
  - Increment coinbase nonce, then run through block header nonce again.