Cryptographic hash functions

6 properties of secure hash functions

1. H can be applied to a block of data at any size

2. H produces a fixed length output

3. H(x) is easy to compute for any given x.

4. For any given block x, it is computationally infeasible to find x such that H(x) = h.

5. For any given block x, it is computationally infeasible to find y != x such that H(x) = H(y).

6. It is computationally infeasible to find any pair (x, y) such that H(x) = H(y).

collisions: avoidance vs. resistance

1. Collision avoidance means that given an infinite amount of tries it is impossible to find a h(x) = h(y) where x != y. This is non existing

2. Collision resistance which is what we aim to have, means that it should be computationally infeasible to find a h(x) = h(y) where x != y, but it will still happen.

one-wayness

Hash should be hard to invert

how to use cryptographic hash functions (and public key cryptography) for authentication

1. Asymmetric cryptography means we have 1 key for encryption and 1 key for decryption.

2. In public key cryptography, there are 5 elements: the actual data, sender's public key, sender's private key, receiver's public key and receiver's private key.

3. A public key is announced and known to the world. A private key is stored in the owners node or in a physical/digital safety locker. Private key is otherwise called a secret key.

4. At a given point, neither the sender nor receiver can know each other's private key. Public key of sender and receiver. Private key of their own

5. You can encrypt a piece of data with a public key, but the decryption can be done only with its corresponding private key

6. A sender would always start with the receiver's public key for encryption. The receiver would use its own (receiver) private key for decryption.

7. Receiver still does not know who sent the data. It could have been sent by a hacker. So the sender needs to let the receiver know that the data is indeed sent by sender. This process is called signing. Signing is done by attaching a small piece of addtional data called the signature.

8. The process:

Sender:

\* Sender encrypts the data with sender's private key: sender-privkey(data) = sender-privkey-encryp-data (the signature)

\* Sender the combines the 'sinature' with the receivers public key: recv-pub-key(data + signature) = receiver-pubkey-encrypted-data

\* Send to receiver

Receiver:

\* Reveive message: receiver-pubkey-encrypted-data

\* receiver-privkey(receiver-pubkey-encrypted-data) = data + signature

\* Receiver can now see the data

\* Receiver checks the signature: sender-pubkey(signature) = data1

\* Since data = data1, receiver can confirm sent by sender

Merkle Tree

How to build a Merkle tree

How to use a Merkle tree to compare two data sets, and its time complexity

How to use a Merkle tree to show the integrity of data (that the data has not changed), and its time complexity

Patricia Trie

How is a Trie different from a Patricia Trie

How to build a Patricia Trie

How to insert a new key into a Patricia Trie and its time complexity

How to delete a key in a Patricia Trie and its time complexity

How to use a Patricia Trie to test if a key exists and its time complexity (get0

Transaction support

What is ACID properties

Atomicity: No interleaves in one operation

Coneietency: everyone sees the same data

Isolation: parallel transaction support

Durability: once written, forever written

Example of non-atomic transactions resulting in an inconsistent state

Nakamoto consensus

What does Ethereum require as proof of work?

1. For each block of transactions, miners use computers to repeatedly and very quicky guess answers to a puzzle until one of them wins.

2. More specifically, the miners will run the block's unique header metadata (including timestamp and software version) through a hash function (which will return a fixed-length, scrambled string of numbers and letters that looks random),only changing the 'nonce value', which impacts the resulting hash value.

3. If the miner finds a hash that matches the current target, the miner will be awarded ether and broadcast the block across the network for each node to validate and add to their own copy of the ledger.

4. If a miner, B, finds the hash, another miner A will stop work on the current block and repeat the process for the next block.

What is a fork in blockchain?

1. Temporary Fork: are forks that occur when miners, on cryptocurrency systems discover a block at the same time. The results in two split competing blockchains. Temporary forks are resloved in proof-of-work systems such as Bitcoin when miners select which chain to form subsequent blocks upon. The longest blockchain is viewed as being the 'true' blockchain, and will win out, whilst the shorter chain will be abandoned.

2. Soft forks: This is a method of upgrading the blockchain. A soft fork is software upgrade that is backward compatible with previous sersions of the software. A blockchain fork is essentially a colectively agreed upon software update. An example of a soft fork would be the implementation of a new rule changing. An example of a soft fork would be the implementation of a new rule changing the network lock size from 1MB to 500 KB. Nodes that have not upgraded will continue to see incoming transactions as valid, as these nodes follow the old set of consensus rules as well as the new. However, mining nodes that have not upgraded and attempt to mine new bloks will have these blocks rejected, as it does not conform to the new set of consensus rules (block sizes of 500KB).

How does Ethereum resolve a fork?

We choose the branch that has produced the greatest amount of computing power

When we have two chains, over time, one chain will eventually outpace the other. The longer chain will be considered the main chain.

What is Nakamoto consensus?

Longest chain + Proof of Work = Nakamoto Consensus

Consensus in crash failure model

Example execution where a simple consensus protocol (slide 140 fails to achieve consensus

What is a dangerous chain

1. A dangerous chain is when the last process in the chain is correct, but all others are faulty.

2. If we have f faulty processes, a dangerous chain will have at most f+1 nodes in the chain, spans at most f rounds. It is safe to decide at the end of round f+1.

Prove the minimal number of rounds to achieve consensus when up to f

1. f + 1 is when we are able to achieve consensus.

2. A dangerous chain is having one new value introduced every single round after round 1. Where in round 1 every process sends a valid value to every other round.

3. The last process that can fail is p(f-1). So the dangerous chain ends at p(f).

4. This is the last process that learns of a new value, which is at round f.

5. Since f processes failed/crashed, during round f + 1, p(f) will tell all other processes its final result which allows all processes have the same set.

6. Therefore the longest possible dangerous chain ends at round f.

7. This is worst case scenario. As p(f) goes through the rounds, there is a dangerous chain building up. Up until round f, p(f) will know that the size of the faulty nodes can be from 0->f. When: "number of faulty nodes" < "the number of rounds", then we know the dangerous chain is no longer in progress. If "number of faulty nodes" == "the number of rounds", then the dangerous chain is still in progress.

Early termination protocol

1. Suppose processes can detect the set of processes that have failed by the end od round 1.

2. Where the set of faulty processes are called: faulty(p, i)

3. If faulty(p, i) < i, (faulty process are less than the number of rounds), then there can be no active dangerous chains, and p can safely deliver SF.

Public-Key Cryptography

6 requirements for public-key cryptography

1. Easy generation of a pair: public and private keys

2. Easy for the sender to generate cipher text

3. Easy for receiver to decrypt cipher text using private key

4. Infeasible to determine private key with public key

5. Infeasible to recover message M, knowing public key and cipher text

6. One of the keys is used for encryption and the other for decryption

how to use RSA public key and private key for encryption and decryption

1. de % theta(n) = 1
2. Public key KU = {e, n}
3. Private key KR = {d, n}
4. Encryption: Plaintext M < n, CiperText C = M ^ e (mod n)
5. Decryption: Ciphertext: C, Plaintext M = C^d (mod n)

why is RSA secure? (what assumptions are necessary for RSA to be secure?)

1. The main reason why RSA is secure lies within the difficulty to find the decryption key.

2. Person A = self, Person B = Key holder, Person C = Interceptor

3. A third party that intercepts the message knows that n=pq, the simplest way to find n would be somehow factor n into the exact primes used by Person B in the algorithm.

4. With larger (which are more secure) primes, this turns out to be nearly impossible to do.

5. Now factoring n is basically impossible to do by hand.

6. It would need large amount of computing power to factor n.

how to use public/private key (and cryptographic hash function) for authentication

1. Asymmetric encryption uses two keys to encrypt a plain text.

2. Secret keys are exchanged over the Internet or a large network. It ensures that malicious persons do not misuse the keys.

3. It is important to note that anyone with a secret key can decrypt the message and this is why asymmetrical encryption uses two related keys to boosting security.

4. A public key is made freely available to anyone who might want to send you a message.

5. The second private key is kept a secret so that only you can know.

what is public-key infrastructure (PKI)? (why do we need this?)

The Public key infrastructure (PKI) is the set of hardware, software, policies, processes and procedures required to create, manage, distribute, use, store, and revoke digital certificates and public-keys.

how does Ethereum provides PKI?

Arbitrary failure with message authentication

Example execution where consensus algorithm for crash failure model (slide 30) fails

2 properties of message authentication (signature)

1. Message sent has not been modified while in transit: This ensures that processes cannot send conflicting message to different receivers.

2. Message is signed and so receiver can verify the source of the message: Unforgeable signatures.

How the algorithm in slide 86 achieves consensus in the example execution above

Arbitrary failure model

2 properties of broadcast/accept primitives

1. Correctness: If a correct process p executes broadcast(p, m, i) in round I, then all correct processes will execute accept(p, m, i) in round i.
2. Unforgeability: If a correct process q executes accept(p, m, i) in round j >= i, and p is correct, then p did in fact execute broadcast(p, m, i) in round i.
3. Relay: If a correct process q executes accept(p, m, i) in round j >= i, then all correct processes will execute accept(p, m, i) by round j + 1.

Example execution where consensus algorithm on slide 86 fails to achieve consensus without msg auth

Requirements to become a witness of a message (p, m, r), i.e. when to send echo

A witness sends if either:

1. It receives from p directly.

2. It recives confirmations for (p, m, r) from at least f + 1 process (at least one correct witness)

How the algorithm in slide 94 achieves consensus in the example execution above

Impossibility results

PBFT

BFT = the system achieves the goal in the presence of Byzantine-faulty users using redundancy/replication

PBFT Practical BFT

1. A client sends a request to invoke a service operation  to the primary
2. The primary multicasts the request to the backups
3. Replicas execute the request and send a reply to the  client
4. The client waits for f+1 replies from different  replicas with the same result; this is the result of the operation.
5. Replicated state machine-based BFT
6. Asynchronous system + timeout
7. Every node’s public key is known

How does a client decide the response to a command

1. Pre-prepare: assigns sequence number to request
2. Prepare: Ensures fault-tolerant consistent ordering of requests within views
3. Commit: Ensures fault-tolerant consistent ordering of requests across views.

How many messages are sent for one client's command

O(n ^ 2)

How can backup nodes build a proof of misbehavior of faulty primary

Backups monitor primary’s behavior and trigger view changes to replace faulty primary.