Cryptography background

Cryptography Background

cryptographic hash functions:

An efficiently computable function $H\colon\ M \rightarrow T$

where $|M| \gg |T|$

$$\text{megabytes} \Rightarrow \overbrace{\text{hash value}}^{\text{32 bytes}} \qquad T = \{0,1\}^{256}$$

Collision resistance

Def: a **collision** for $H: M \to T$ is pair $x \neq y \in M$ s.t. $\boxed{H(x) = H(y)}$

$|M| \gg |T|$ implies that many collisions exist

Def: a function $H: M \to T$ is **collision resistant** if it is "hard" to find even a single collision for H (we say H is a CRHF)

Example: **SHA256**: $\{x : \text{len}(x) < 2^{64} \text{ bytes}\} \to \{0,1\}^{256}$

Application: committing to data on a blockchain

Alice has a large file m. She posts $h = H(m)$ (32 bytes)

Bob reads h. Later he learns m' s.t. $H(m') = h$

H is a CRHF $\Rightarrow$ Bob is convinced that $m' = m$
(otherwise, m and m' are a collision for H)

We say that $h = H(m)$ is a **binding commitment** to m
(note: not hiding, h may leak information about m)

Committing to a list (of transactions)

Alice has $S = (m_1, m_2, \ldots, m_n)$

32 bytes

Goal:

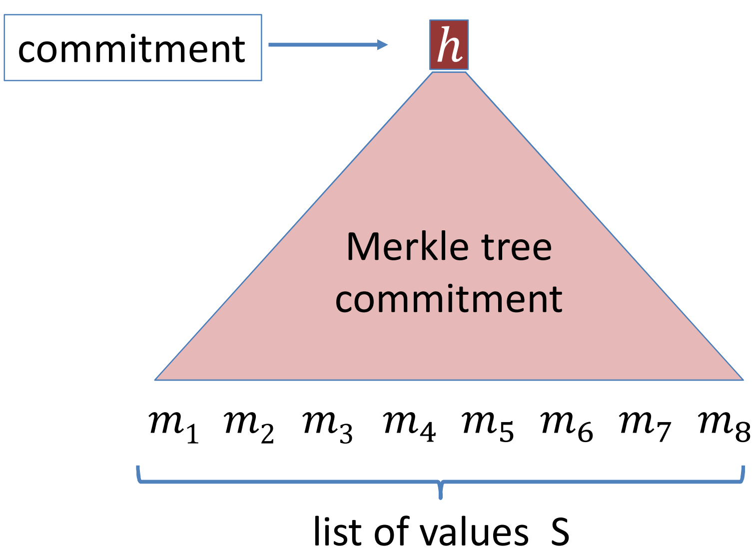
- Alice posts a <u>short</u> binding commitment to S, $h = \text{commit}(S)$
- Bob reads h. Given $(m_i,\ \text{proof}\ \pi_i)$ can check that $S[i] = m_i$

  Bob runs $\text{verify}(h, i, m_i, \pi_i) \rightarrow$ accept/reject

security: adv. cannot find (S, i, m, π) s.t. $m \neq S[i]$ and

$\text{verify}(h, i, m, \pi) = \text{accept}$ where $h = \text{commit}(S)$

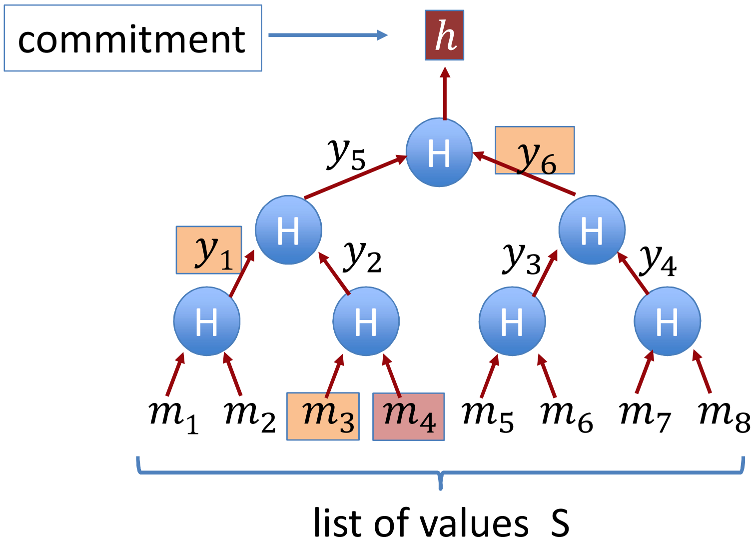
Merkle tree (Merkle 1989)



Goal:

- commit to list S of size n
- Later prove $S[i] = m_i$

Merkle tree (Merkle 1989)



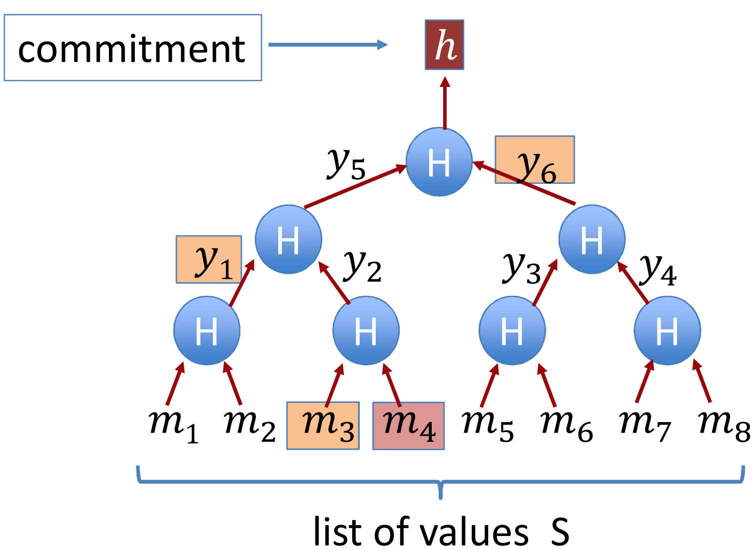
Goal:
- commit to list S of size n
- Later prove $S[i] = m_i$

To prove $S[4] = m_4$,

proof π = (m_3, y_1, y_6)

length of proof: $\log_2 n$

Merkle tree (Merkle 1989)



To prove $S[4] = m_4$,

proof π = (m_3, y_1, y_6)

Bob does:

$y_2 \leftarrow H(m_3, m_4)$

$y_5 \leftarrow H(y_1, y_2)$

$h' \leftarrow H(y_5, y_6)$

accept if $h = h'$

Merkle tree (Merkle 1989)

Thm: H CRHF $\Rightarrow$ adv. cannot find (S, i, m, π) s.t. $m \neq S[i]$ and

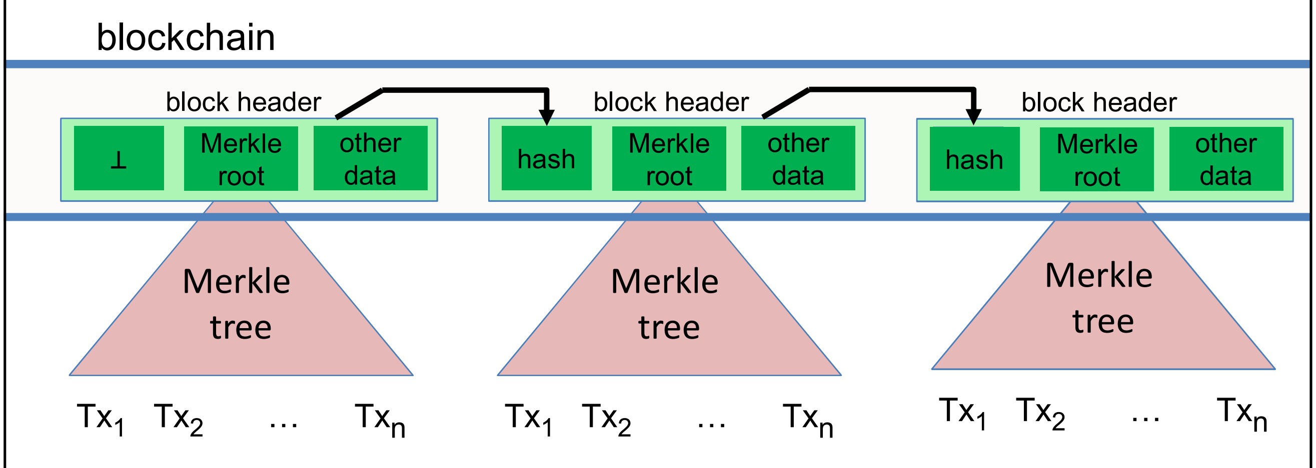
$$\text{verify}(h, i, m, \pi) = \text{accept} \quad \text{where} \quad h = \text{commit}(S)$$

(to prove, prove the contra-positive)

How is this useful? Super useful. Example

- When writing a block of transactions S to the blockchain, suffices to write commit(S) to chain. Keep chain small.
- Later, can prove contents of every Tx.

Abstract block chain



Merkle proofs are used to prove that a Tx is "on the block chain"

Another application: proof of work

Goal: computational problem that

- takes time $\Omega(D)$ to solve, but
- solution takes time O(1) to verify

(D is called the **difficulty**)

How? $H: X \times Y \rightarrow \{0,1,2, \dots, 2^n - 1\}$ e.g. $n = 256$

- puzzle: input $x \in X$, output $y \in Y$ s.t. $H(x,y) < 2^n/D$
- verify(x, y): accept if $\boxed{H(x,y) < 2^n/D}$

Another application: proof of work

Thorem: if H is a "random function" then the best algorithm requires D evaluations of H in expectation.

Note: this is a parallel algorithm

$\Rightarrow$ the more machines have, the faster solve the puzzle.

Proof of work is used in some consensus protocols (e.g., Bitcoin)

Bitcoin uses $H(x,y) = \text{SHA256}(\text{SHA256}(x.y))$

Cryptography background: Digital Signatures

How to authorize a transaction

Cryptography

Cryptography fundamentals which are necessary in blockchain technology

- Public Key Cryptography
- Digital Signatures
- Hash Puzzles
- Hash functions and their properties
- Hash Pointers
- Merkle Tree Data Structures

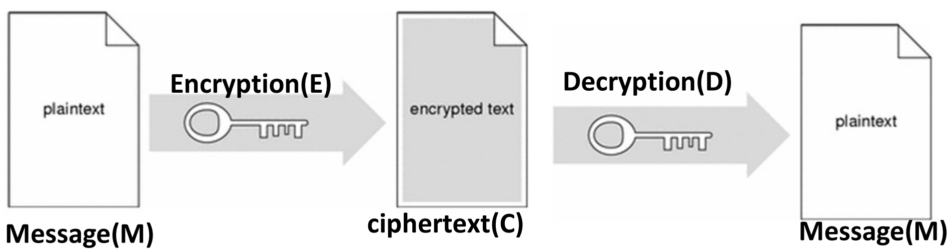
Encryption/ Decryption

Plaintext :The data message

Encryption: Encoding of data message into another form

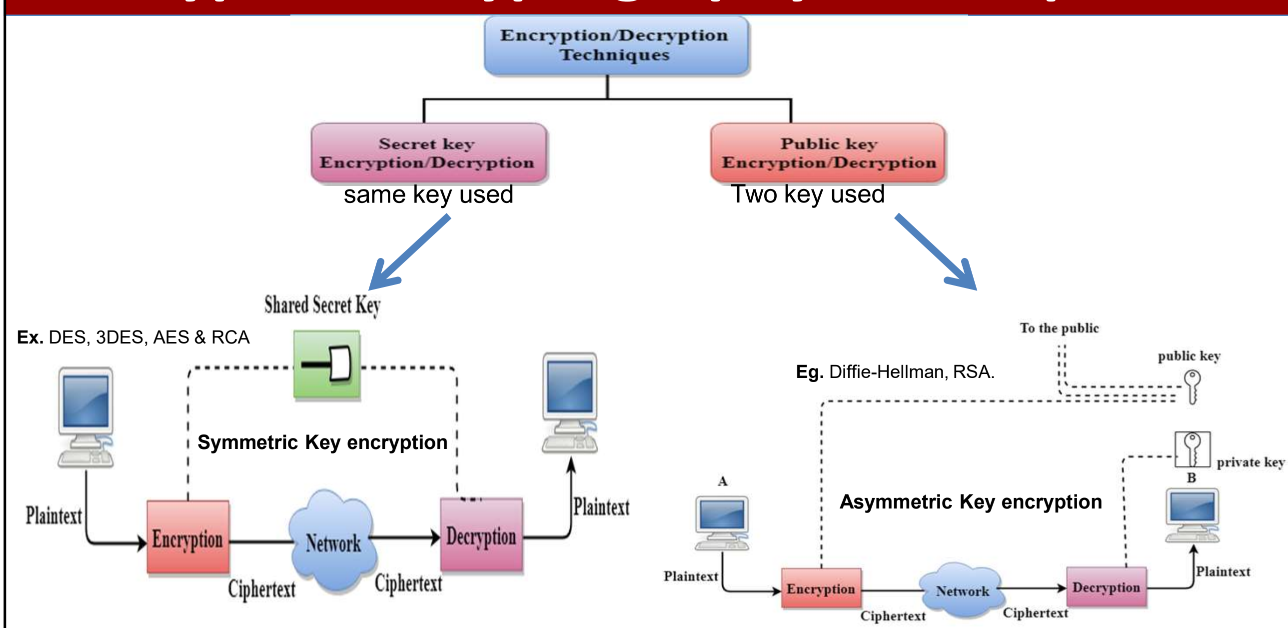
Ciphertext : the encrypted data message.

Decryption :decoding of ciphertext back to original form



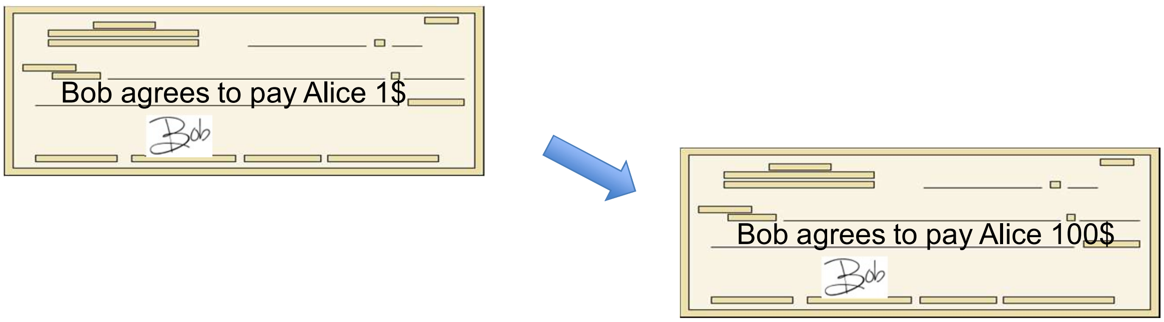
Must hold true: D(C) = M , where C = E(M) and M = D(E(M))

types of Cryptography techniques



Signatures

Physical signatures: bind transaction to author



Problem in the digital world:

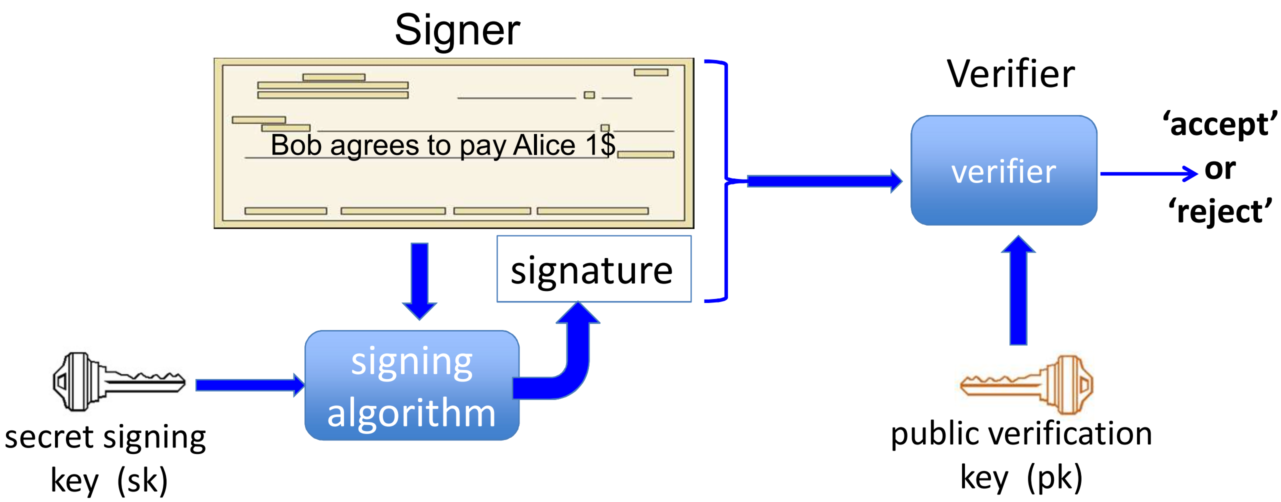
anyone can copy Bob's signature from one doc to another

Solution: digital signature

- a mathematical technique used to validate the authenticity and integrity of a digital document, message or software.
- A digital signature is exactly what it sounds like a modern alternative to handwritten signing documents with paper and pen.
- guarantees that the contents of a message are not altered in transit
- intended to solve the problem of tampering and impersonation in digital communications.
- Digital signatures also provide additional information such as the origin of the message, status, and consent by the signer.

Digital signatures

Solution: make signature depend on document



digital signature

- Using a mathematical algorithm, digital signing solution providers such as Zoho Sign will generate two keys: a public key and a private key. When a signer digitally signs a document, a cryptographic hash is generated for the document.
- That cryptographic hash is then encrypted using the sender's private key, which is stored in a secure HSM (hardware security module) box. It is then appended to the document and sent to the recipients along with the sender's public key.
- recipient can decrypt the encrypted hash with the sender's public key certificate. A cryptographic hash is again generated on the recipient's end.
- Both cryptographic hashes are compared to check its authenticity. If they match, the document hasn't been tampered with and is considered valid.
- Digital signatures are based on public key cryptography, also known as *asymmetric cryptography*.
  - Using a public key algorithm -- such as RSA(Rivest-Shamir-Adleman) -- two keys are generated, creating a mathematically linked pair of keys: one private and one public.

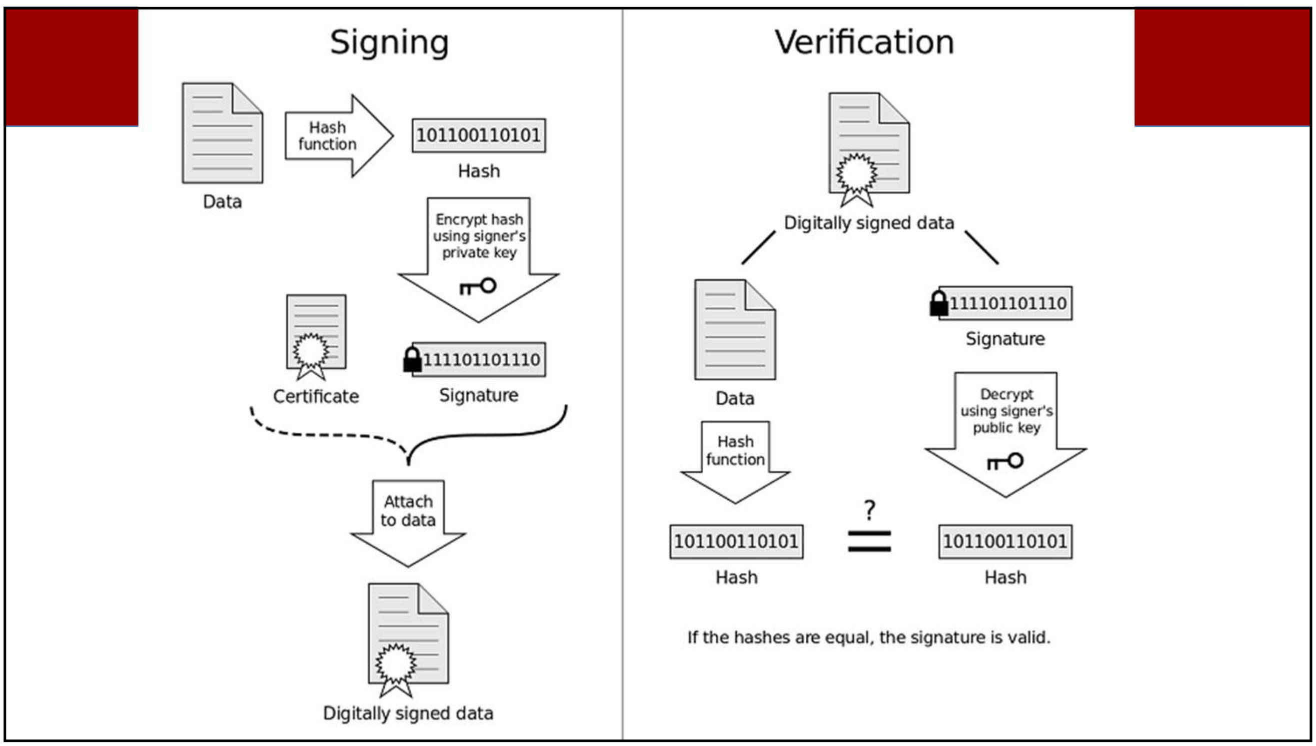
Digital signatures: syntax

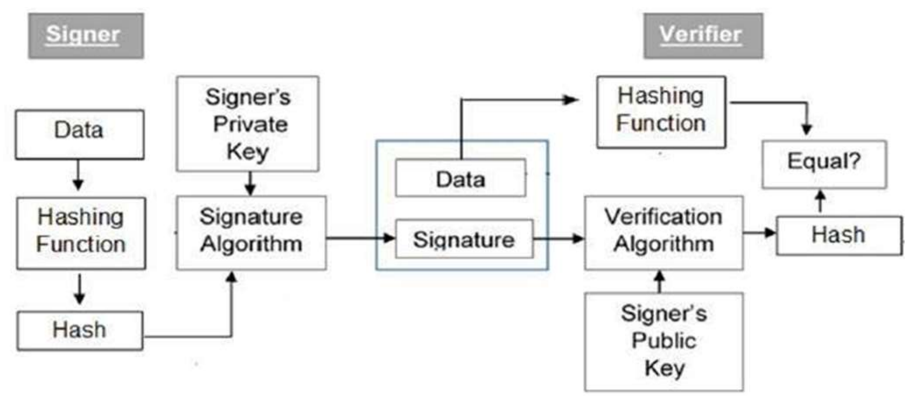
Def: a signature scheme is a triple of algorithms:

- **Gen**(): outputs a key pair (pk, sk) ← to generate two keys
- **Sign**(sk, msg) outputs sig. σ ← to produce signature
- **Verify**(pk, msg, σ) outputs 'accept' or 'reject' ← to verify

Secure signatures: (informal)

Adversary who sees signatures on many messages of his choice, cannot forge a signature on a new message.





Families of signature schemes

1. RSA signatures(Rivest-Shamir-Adleman) (old … not used in blockchains):
    - long sigs and public keys (≥256 bytes), fast to verify
2. Discrete-log signatures: Schnorr and ECDSA (Bitcoin, Ethereum)
    - Elliptic Curve Digital Signature Algorithm(ECDSA)
    - short sigs (48 or 64 bytes) and public key (32 bytes)
3. BLS signatures: 48 bytes, aggregatable, easy threshold (Ethereum 2.0, Chia, Dfinity)
4. Post-quantum signatures: long (≥768 bytes)

signature schemes

- **Elliptic Curve Digital Signature Algorithm (ECDSA)(1992)**
  - uses shorter keys and requires fewer computational requirements than the RSA system, while maintaining strong security
- **SCHNORR SIGNATURES (2008) :**BITCOIN
  - ECDSA lacks one key property is that there is no efficient way to compress and verify signatures together.
  - switched to a new Schnorr signature scheme to improve the cryptocurrency's scalability, efficiency, and privacy
- **BLS SIGNATURES (**Boneh-Lynn-Shacham) **:**ETH2
  - when Ethereum moves to Proof of Stake with eth2, ECDSA will no longer support its validation requirements.
  - for efficient signature verification in consensus
  - uses a bilinear pairing for verification, and signatures are elements of an elliptic curve group.

Signatures on the blockchain

Signatures are used everywhere:

- ensure Tx authorization,
- governance votes,
- consensus protocol votes.