Blockchains & Cryptocurrencies

Crypto Background - II

Instructor: Abhishek Jain
Johns Hopkins University - Spring 2021

*Some slides from NBFMG

This lecture

Crypto background (part II)

hash functions (contd.)

digital signatures

secret sharing

threshold and multi-signatures

Recap: Hash Functions

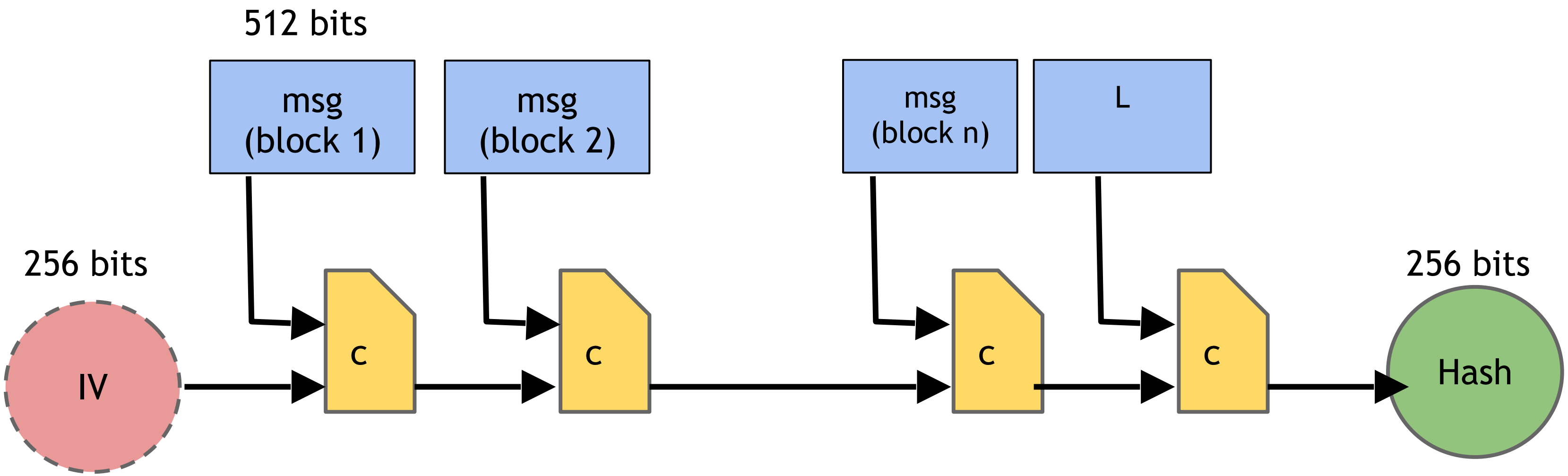
- Take as input an arbitrary length string
- Output a short fixed-length string

Cryptographic hash function security:

- Collision-resistance
- Pre-image resistance
- "Random Oracle" like (sometimes)

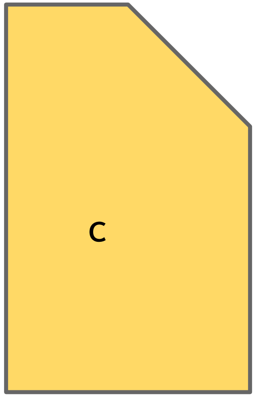
Recap: SHA-256 hash function

Suppose msg is of length L s.t. L is a multiple of 512 (pad with 0s otherwise)



<u>Theorem [Merkle-Damgard]</u>: If c is collision-resistant, then SHA-256 is collision-resistant.

Recap: SHA-256 hash function



Q: What the heck is inside of c?

Theorem [Merkle-Damgard]: If c is collision-resistant, then SHA-256 is collision-resistant.

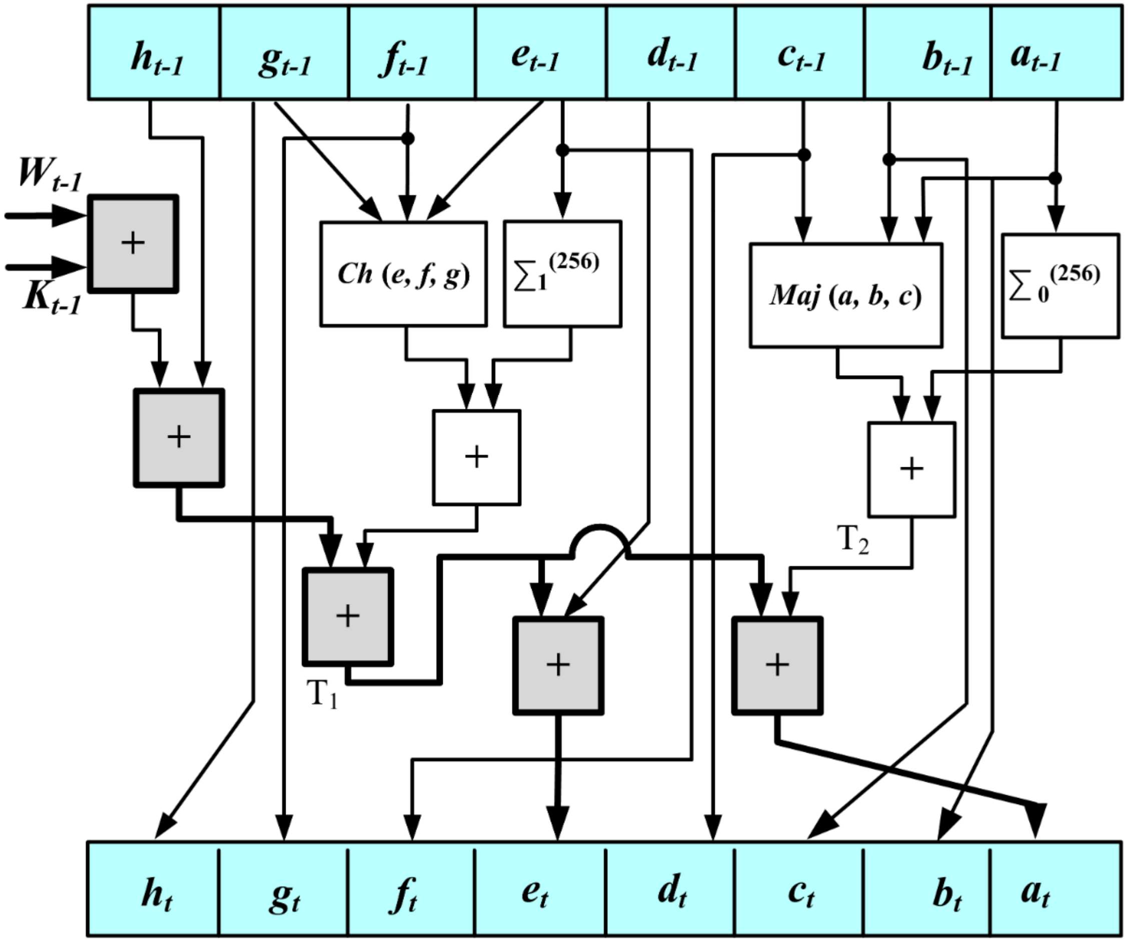


Fig. 3. SHA-256 hash function. Base transformation round

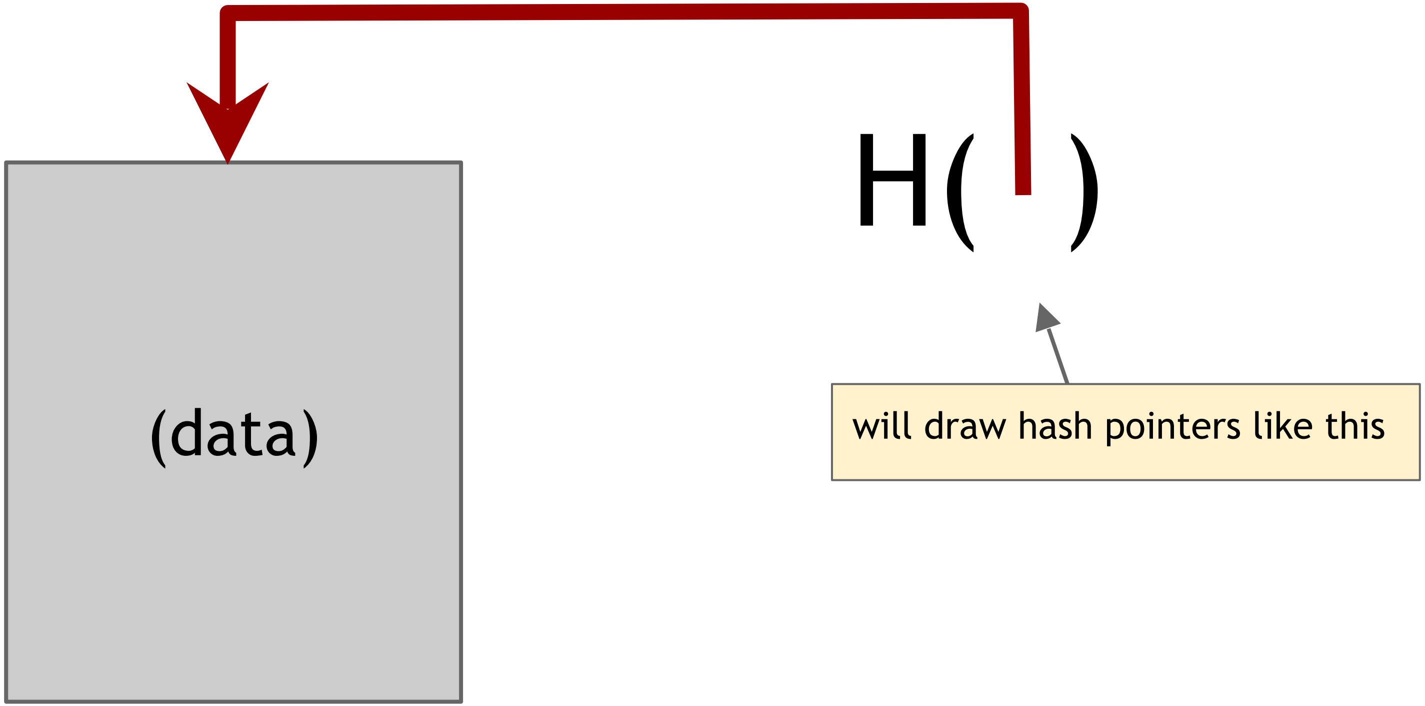
Hash Pointers and Data Structures

Hash pointer

- pointer to where some info is stored, ***and***
- cryptographic hash of the info

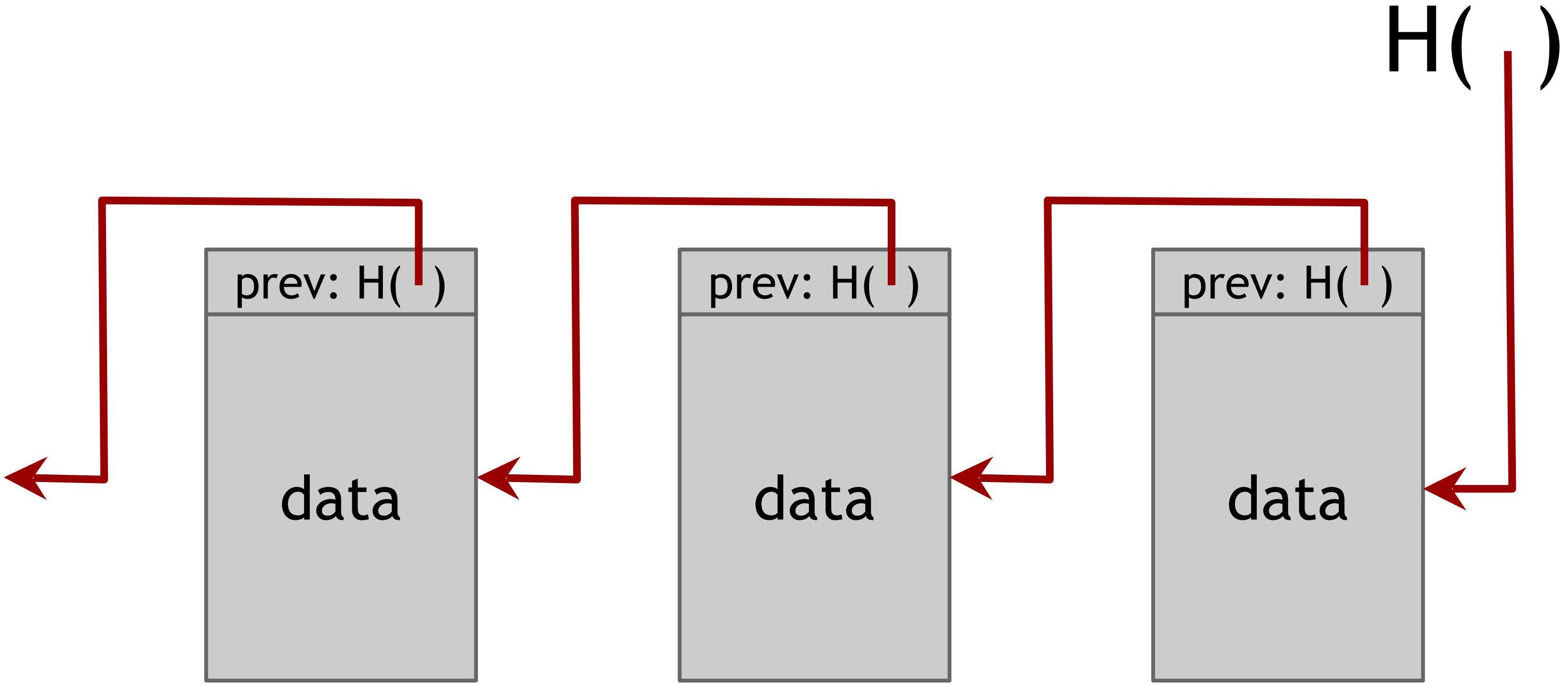
If we have a hash pointer, we can

- ask to get the info back, and
- verify that it hasn't changed


H( )
(data)
will draw hash pointers like this

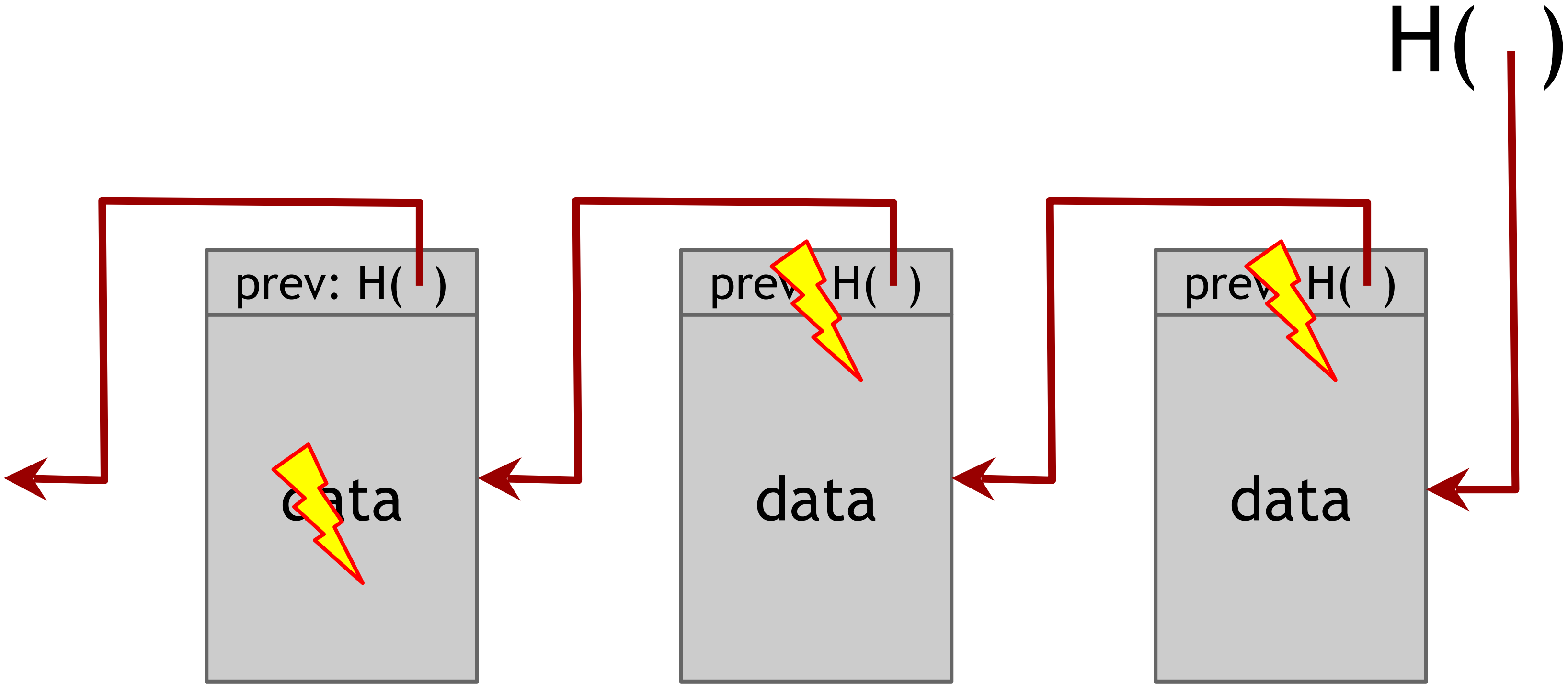
Building data structures with hash pointers

Linked list with hash pointers = "Blockchain"



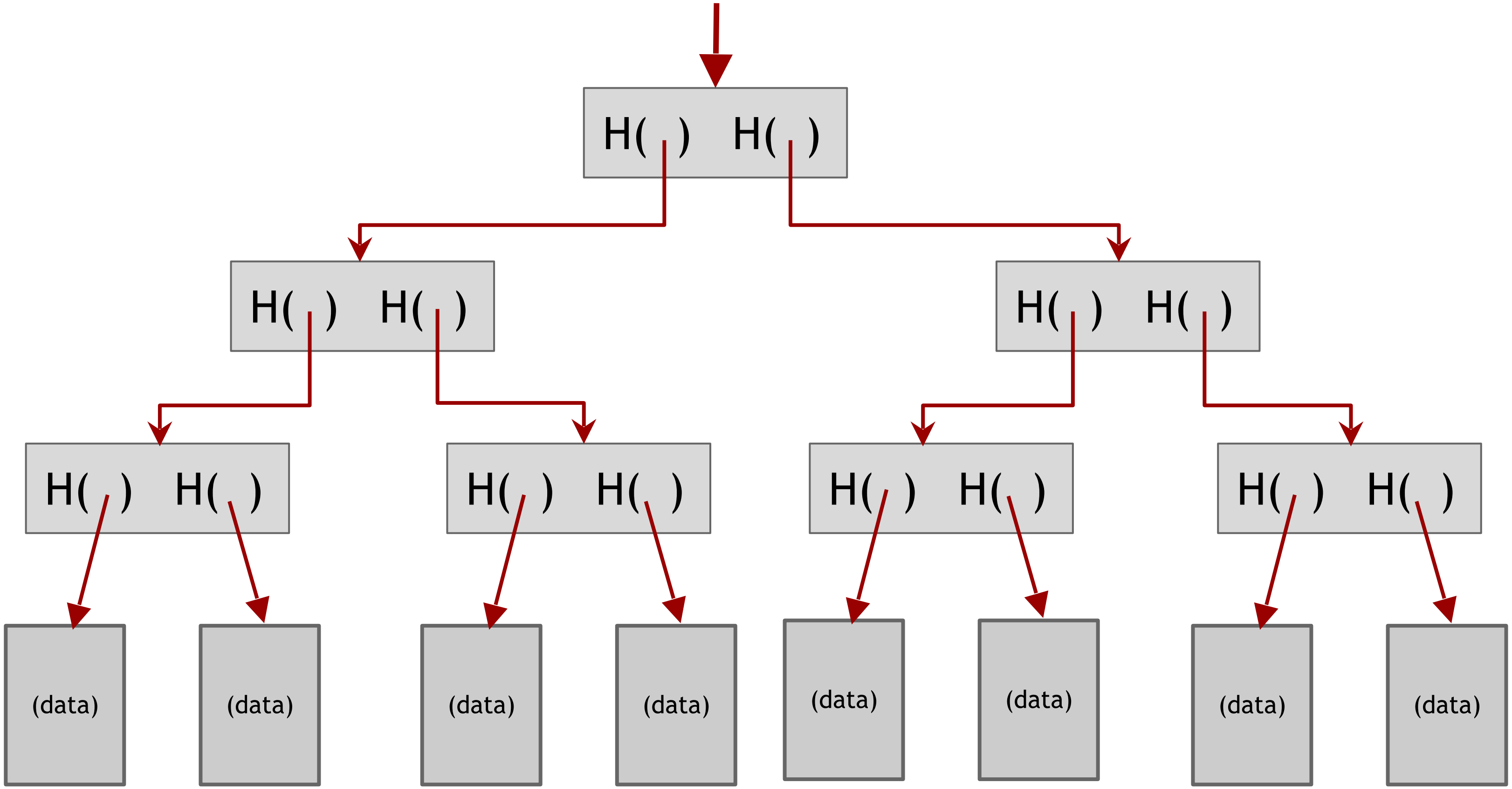
use case: tamper-evident log

Detecting Tampering

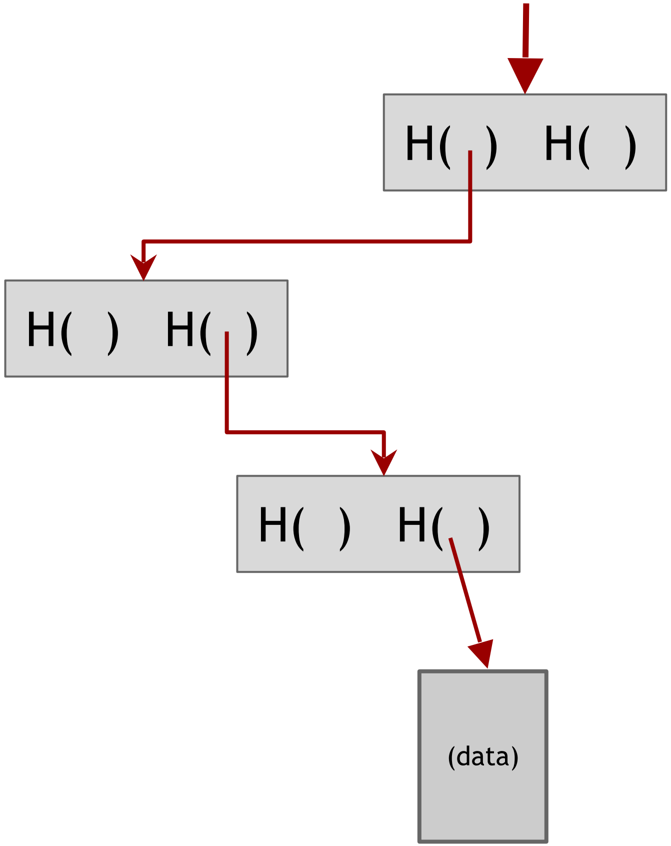


use case: tamper-evident log

Binary tree with Hash pointers = "Merkle tree"



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, after the missing one

General Notion: ***Vector Commitments***

- Commit to a vector of elements “compactly”
- Compact proofs of membership and non-membership
- **Variant:** *Subvector commitments*, that support compact proofs of membership of sub-vectors
- Number-theoretic constructions known where commitments and proofs are constant size
- Very active area of research due to applications to blockchains (good topic for group project!)

Digital Signatures

What we want from signatures

- Only you can sign, but anyone can verify
- Signature is tied to a particular document

  (***can't be cut-and-pasted to another doc)***
- Even if one can see your signature on some documents, he cannot "forge" it

Digital signatures

randomness

- (sk, pk) ← Gen(r)

  sk: secret signing key

  pk: public verification key

randomized algorithm

- sig ← Sign(sk, message)

Typically randomized

- isValid ← Verify(pk, message, sig)

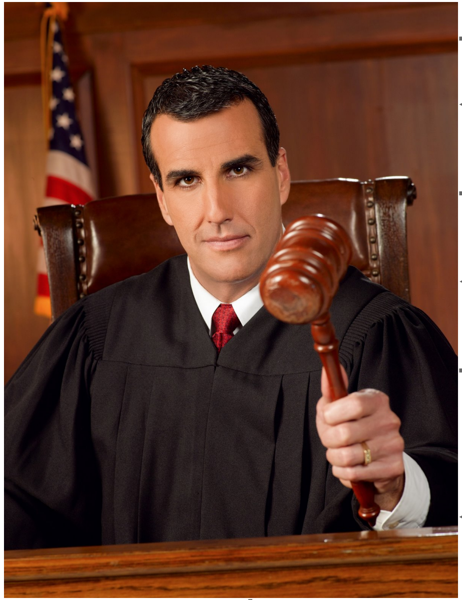
Requirements for signatures

- Correctness: “valid signatures verify”
  - Verify(pk, message, Sign(sk, message)) == true

- Unforgeability under chosen-message attacks (UF-CMA): “can’t forge signatures”
  - adversary who knows pk, and gets to see signatures on messages of his choice, can’t produce a verifiable signature on another message

UF-CMA Security

(sk, pk) ← Gen(1^k)



pk

m_0

Sign(sk, m_0)

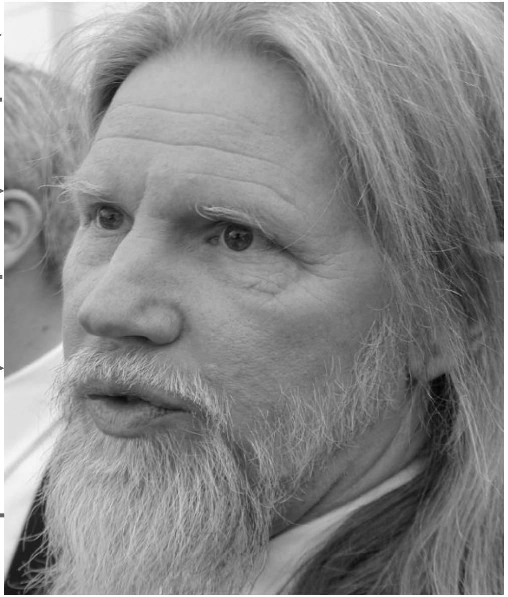
m_1

Sign(sk, m_1)

• • •

M, sig

M not in { m_0, m_1, … }



Challenger

Verify(pk, M, sig)

ifValid, Adversary wins

Adversary

Definition: A signature scheme (Gen,Sign,Verify) is UF-CMA secure if for every PPT adversary A, Pr[A wins in above game] = “very small”

Elliptic-Curve Digital Signature Algorithm

- Signature scheme used in Bitcoin and Ethereum
- Also used in many online systems such as TLS, DNSSEC
- Based on the Digital Signature Algorithm (DSA) by Kravitz: Use of *elliptic curve groups* for shorter key sizes

ECDSA: Formal Description

- H is a hash function
- **Exercise:** What happens if you use the same *k* to sign two different messages?

Algorithm 1. $\mathsf{Gen}(1^\kappa)$**:**

1) Uniformly choose a secret key $\mathsf{sk} \leftarrow \mathbb{Z}_q$.
2) Calculate the public key as $\mathsf{pk} := \mathsf{sk} \cdot G$.
3) Output $(\mathsf{pk}, \mathsf{sk})$.

Algorithm 2. $\mathsf{Sign}(\mathsf{sk} \in \mathbb{Z}_q, m \in \{0,1\}^*)$**:**

1) Uniformly choose an instance key $k \leftarrow \mathbb{Z}_q$.
2) Calculate $(r_x, r_y) = R := k \cdot G$.
3) Calculate
$$\mathsf{sig} := \frac{H(m) + \mathsf{sk} \cdot r_x}{k}$$
4) Output $\sigma := (\mathsf{sig} \mod q, r_x \mod q)$.

Algorithm 3. $\mathsf{Verify}(\mathsf{pk} \in \mathbb{G}, m, \sigma \in (\mathbb{Z}_q, \mathbb{Z}_q))$**:**

1) Parse σ as (sig, r_x).
2) Calculate
$$(r'_x, r'_y) = R' := \frac{H(m) \cdot G + r_x \cdot \mathsf{pk}}{\mathsf{sig}}$$
3) Output 1 if and only if $(r'_x \mod q) = (r_x \mod q)$.

ECDSA: Notes

- Security known in "generic group model"
- **Insecure against randomness reuse:** Known attacks on PS3, Android Apps
- ECDSA popularity (perhaps) largely due to efficiency reasons; many other signature schemes (with better security proofs and other features) known.

Motivating Scenario

- Alice, Bob, Charlie, and David are co-founders of a company.
- To sign a contract, three of them *must* sign.
- Signature is "valid" if and only if *at least* three of them signed
- How can we implement this?

Secret sharing (or *How to share a secret*) [Shamir]

(k,n)-secret sharing: Divide a secret value S into n shares $S_1,\ldots,S_n$ such that:

- Correctness: Any k shares can be used to reconstruct S
- Privacy: S is hidden given at most k-1 shares

Secret sharing [Shamir]

- **Share(S):** Output a tuple $S_1,\ldots,S_n$
- **Reconstruct($x_1,\ldots,x_k$):** Output a value S*

k-Privacy: For any (S,S'), and any subset X of < k indices, the following two distributions are statistically close:

$$\{(S_1, \ldots, S_n) \leftarrow Share(S) : (S_i | i \in X)\},$$

$$\{(S'_1, \ldots, S'_n) \leftarrow Share(S') : (S'_i | i \in X)\}.$$

Example: n=2, k=2

- p = a large prime
- S = secret in $[0, p)$
- R = random in $[0, p)$

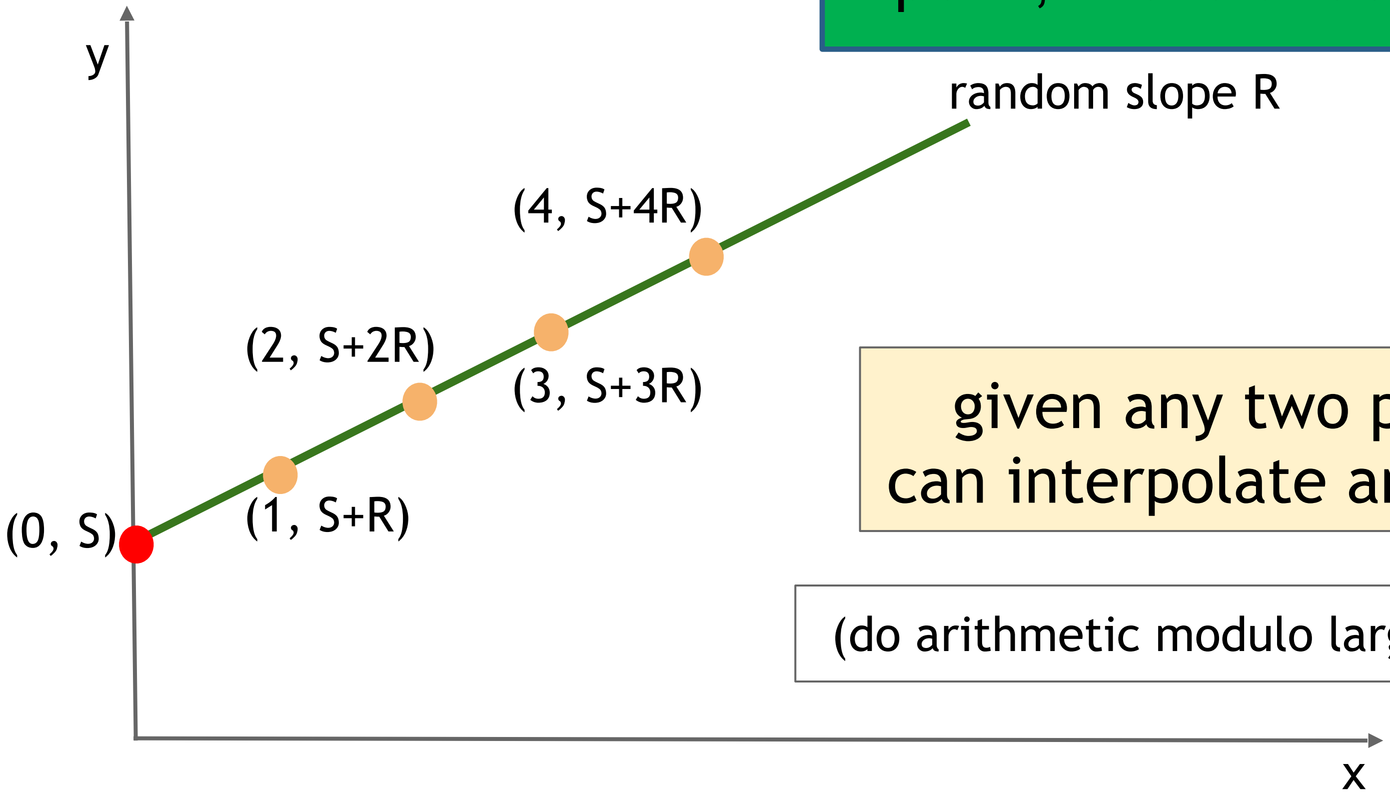
Share(S):
$x_1 = (S+R) \bmod p$ $\quad$ $x_2 = (S+2R) \bmod p$

Reconstruct(x_1,x_2):
$(2x_1 - x_2) \bmod p = S$

2-Privacy: each x_i has uniform distribution over $[0,p)$; independent of S

Example: k = 2, n > 2

k-Privacy: Given only one point, line is undetermined



given any two points, can interpolate and find S

(do arithmetic modulo large prime p)

Going Beyond k = 2

Equation	Random parameters	Points needed to recover S
$(S + RX) \bmod p$	R	2
$(S + R_1X + R_2X^2) \bmod p$	R_1, R_2	3
$(S + R_1X + R_2X^2 + R_3X^3) \bmod p$	R_1, R_2, R_3	4
etc.		

support K-out-of-N sharing, for any K, N

Threshold Signatures

- <u>(k,n)-Threshold Signatures</u>: A signing key can be "divided" amongst n signers such that any subset of k signers can jointly produce a signature, but any subset of <k signers cannot
  - TSetup: Each party learns PK. Party i additionally learns Sk_i
  - TSign(m): Parties run a protocol to compute a signature **sig** on m
  - TVerify(PK,m,**sig**): Output 0/1

Threshold Signatures (contd.)

- Threshold policy enforced "within" the signature scheme
- Can typically be constructed generically from any signature scheme using "secure multiparty computation (MPC)"
- Direct constructions are preferred for improved efficiency: fewer rounds of interaction, smaller signatures
- Constructing Threshold Signatures for ECDSA an active area of research due to its peculiar structure! (Good topic for group project!)

Multisignatures

- Each party samples a key pair independently
- **Main Feature:** Can “aggregate” signatures of multiple parties on same message
- To verify, need the list of signers and their public keys
- Very useful when bandwidth is a concern. Active area of research!
- Bitcoin provides “built-in” multisig support. However, signatures simply concatenated, hence non-compact