MIT Digital Currency Initiative and the University of Brasilia presents

Cryptocurrency Design and Engineering

Lecture 4: Cryptographic commitments and digital signatures
Taught by: Ethan Heilman
Date: September 9, 2025
MAS.S62

Hash functions: Uses

Uses for hash functions

- As a secure pointer to some information
- For PoW as we saw in the last class
- To hide passwords:

Bob h(“sekrit p4ssw0rd”) →8be37df…

user=bob, password hash= 8be37df… →

User	Hash of password
root	47a9e23…
alice	ee7497b…
bob	8be37df…

We can do much better things for passwords than just hashing

Commitments

Commitment scheme allow a person to commit to a value and then later reveal the committed value.

- **Hiding** - no one can determine the value from the commitment
- **Binding** - no one can open the commitment to a diff. value

Why is the following scheme not secure?

$c \leftarrow$ Commit(x): h(x)

$x \leftarrow$ Reveal(x): x

$x \leftarrow$ CheckReveal(x, c): c = h(x)?

If x is guessable someone can break hiding by guessing x

Commitments

Commitment scheme allow a person to commit to a value and then later reveal the committed value.

- **Hiding** - no one can determine the value from the commitment
- **Binding** - no one can open the commitment to a diff. value

Ensure there is enough entropy (unguessability)

c ← Commit(x, k): h(x, k)

x ← Reveal(x): x, k

x ← CheckReveal(x, k, c): c = h(x, k)?

Merkle trees

Hash functions are useful compare if two values are the same:

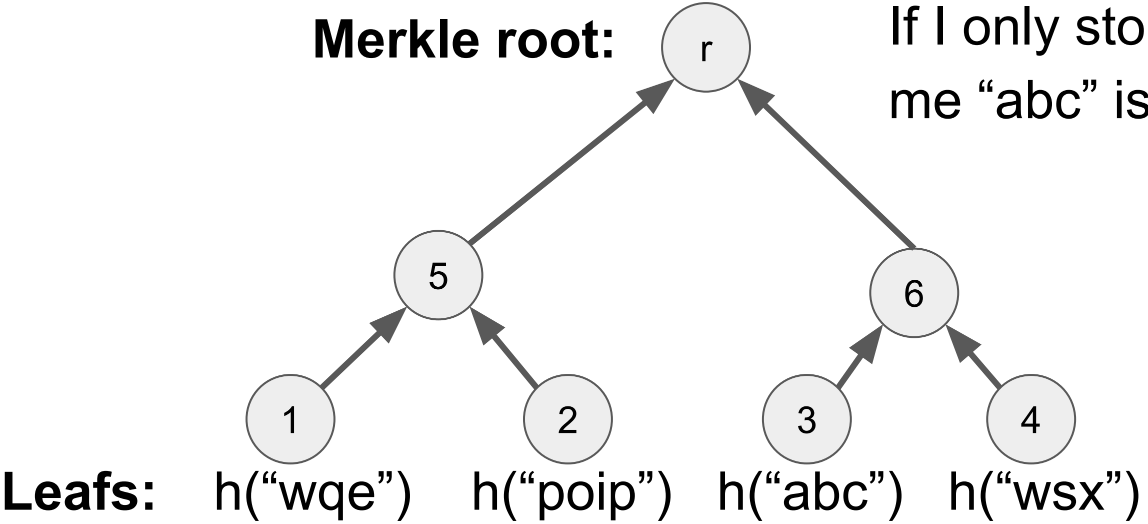
H(x) = H(x’)?

How to show inclusion? “abc” in the set [“wqe”, “poip”, “abc”, “wsx”]

If you hash everything, you still have to keep a hash per object

h(“abc”) in [h(“wqe”), h(“poip”), h(“abc”), h(“wsx”)]

Merkle trees



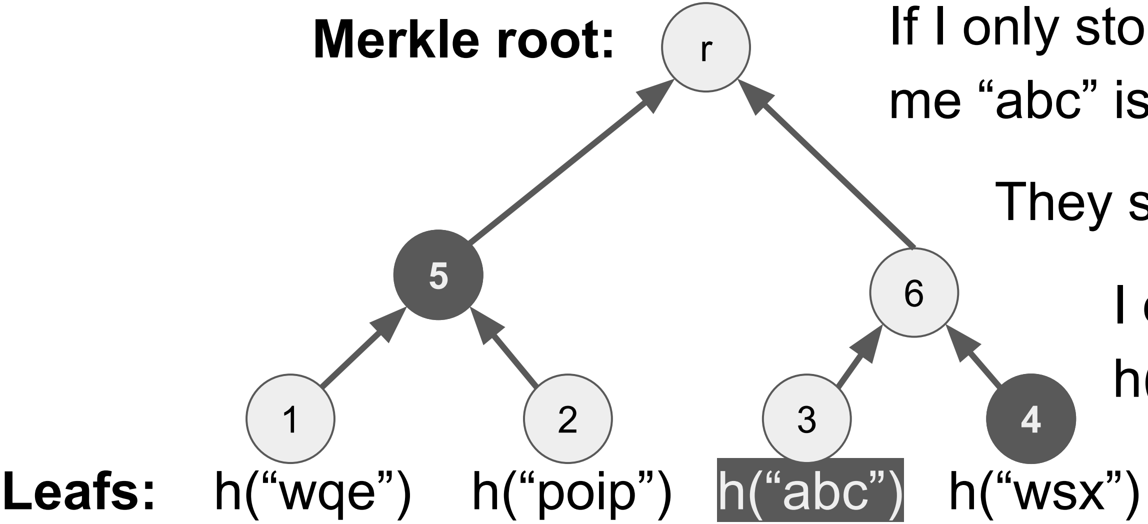
If I only store **r** someone can convince me "abc" is included in the set

h("abc") → 3

h(3, 4) → 6

h(h("abc"),h("wsx)) → 6

Merkle trees: inclusion



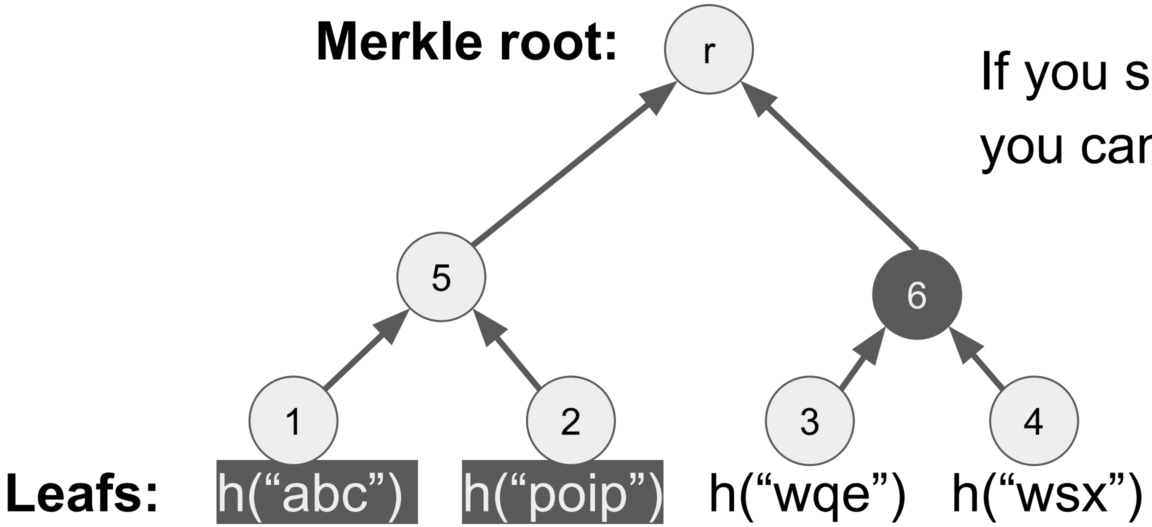
If I only store **r** someone can convince me "abc" is included in the set

They send me: "abc", 4, 5

I check

h(5, h(h("abc"), 4)) == **r?**

Merkle trees: exclusion



If you sort the leafs of the tree, you can prove exclusion

If “abz” were in the set it would be between “abc” and “poip”

…since it isn’t, “abz” isn’t in the set

Merkle trees

Building Merkle tree:
n (storage) or ~2n hashes

Adding a new element: ~log2 n

Proving membership: ~log2 n

Elements in tree	**Data to prove**
8	3
1024	10
1 million	~20
1 trillion	~40

Scales well

Digital signatures

(pk, sk) ← **KeyGen**()

sig ← **SIGN**(m, sk)

VERIFY(pk, sig, m) → (True, False)

Security is defined as NOT being able to:

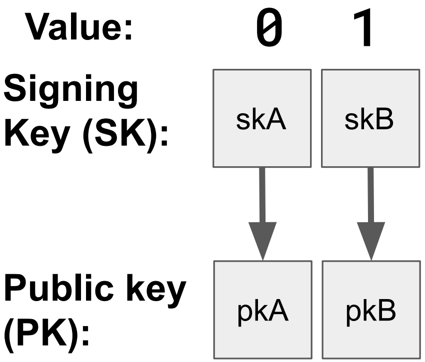
1. create a signature for that passes verify
2. for a message you have not yet seen
3. when you don’t already known the signing key

This is a very informal definition

Digital signatures: One-time

Lamport Signatures - a digital signature from a hash function

m is a single bit: either be 0 or 1



SIGN(sk, m):

IF m == 0: sig=skA

IF m == 1: sig=skB

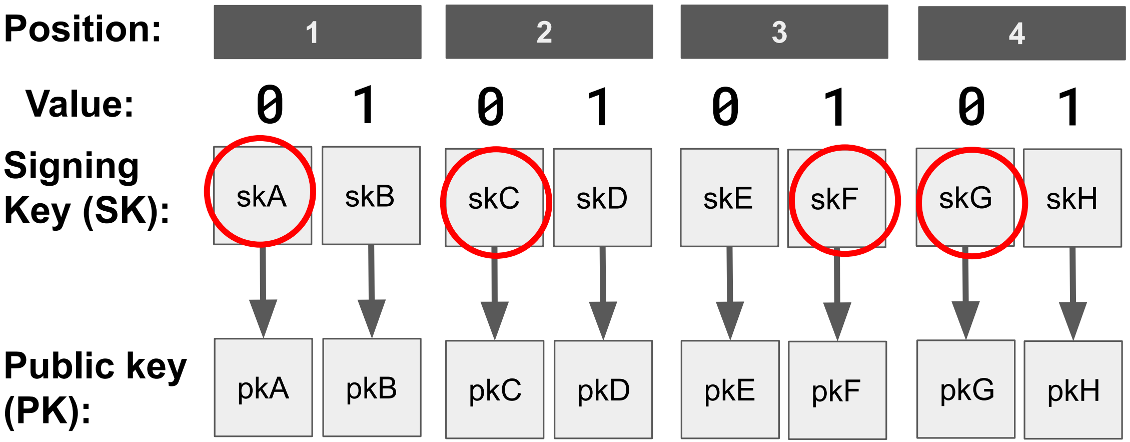
VERIFY(pk, sig, m):

IF m == 0: Check if pkA == h(sig)

IF m == 1: Check if pkB == h(sig)

Digital signatures: Lamport

What if we want to sign messages longer than a single bit?



To sign 0010?

Create signature is A, C, F, G

Drawbacks big public keys/signing keys/big signatures and public keys can NOT be reused

Digital signatures: Lamport

Lamport Signatures are One-Time Signatures:

If you sign two different messages,
anyone that sees those signatures can forge signatures.

We want signatures that do not have this limitation

Digital signatures: Schnorr

Schnorr signatures, unlike Lamport signatures let’s you sign many times without a loss of security.

Let’s take a quick brief look at Schnorr signatures

Digital signatures: Schnorr

KEYGEN:

Signing key (SK): x ← random()

Public key (PK): y ← g^(-x)

SIGN(SK=x, m):

k ← random()

r ← g^k

s ← k + x*h(r,m)

return (r, s)

VERIFY(PK=y, sig=(r,s), m):

Check: g^s * y^h(r,m) == r?

Why?

g^(k + x*h(r,x)) * y^h(r,s)

= g^k * g^x*h(r,x) * g^(-x * h(r,s))

= r * g^(x*h(r,x) + (-x * h(r,s)) = r * g^1 = r

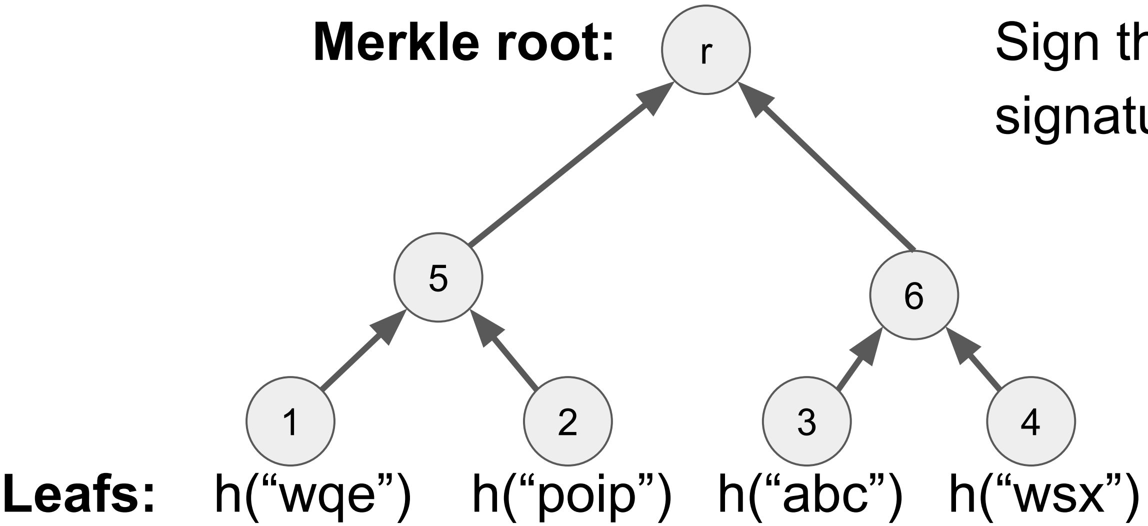
Security Assumption is the Discrete Log Problem (DLP):

Easy to compute: g^x if given x

Hard to compute: x if given g^x

…also hash function is a Random Oracle

Digital signatures:Merkle tree



Sign the root of a Merkle tree signatures covers all the leafs

Multi Signatures

Multi Signature:

Alice
Bob → Signature …can be verified by a single public key
Carol

Threshold Signature (n-of-m): only requires a subset

Alice
Bob → Signature …can be verified by a single public key
Carol

Multi Signatures

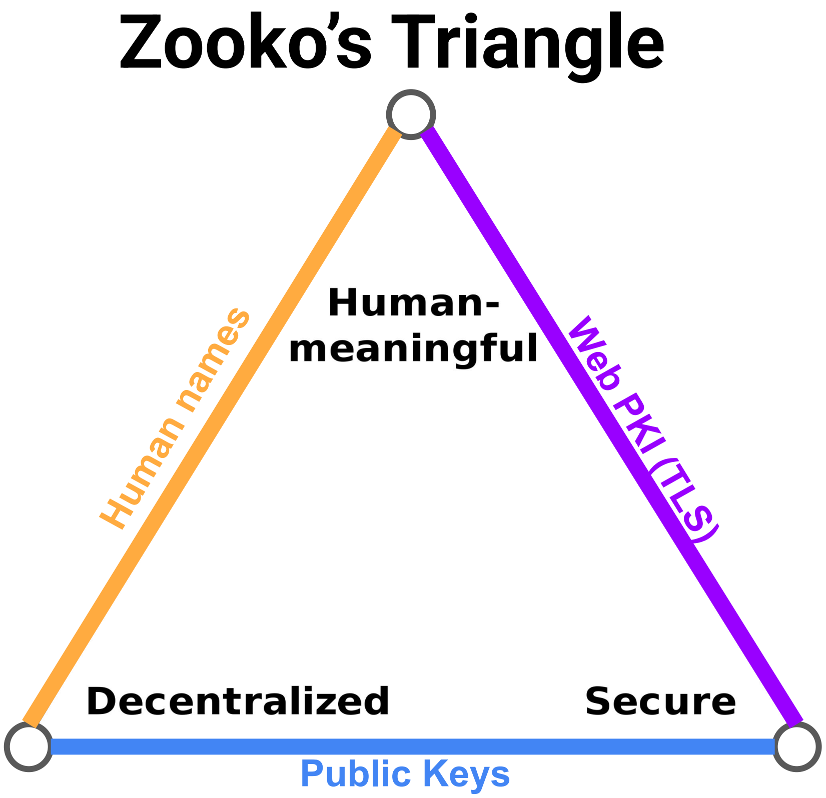
One solution is to simply define the public key as a list of public lists
…and the signature as a list of signatures.

This works, but the public key and signature grows in size with the number of participants, consider hundreds or millions of signers?

Can we have a small constant sized public key and signature that captures the desired requirements?

Yes, the readings present one such scheme (musig2)

Signatures & Identity



Choose a side (Pick 2):

1. Decentralized and Meaningful (Not Secure)
   a. Human names → Alice, Bob, Carol
2. Meaningful and Secure (Not Decentralized)
   a. Web PKI (TLS) → trust a Certificate Authority https://google.com
   b. OpenID Connect → trust an OpenID Provider alice@gmail.com
3. Decentralized and Secure (Not Meaningful)
   a. Public Keys → a string of random numbers "0Wi6ZXc54…"

Reminders & Next Week & Questions

- Reminder:
  - Sign up for the class Discord
  - Register for the class if you haven't yet
  - Source of truth https://github.com/mit-dci/cde-2025
- Next week
  - State model
  - Consensus
- Questions?

The End