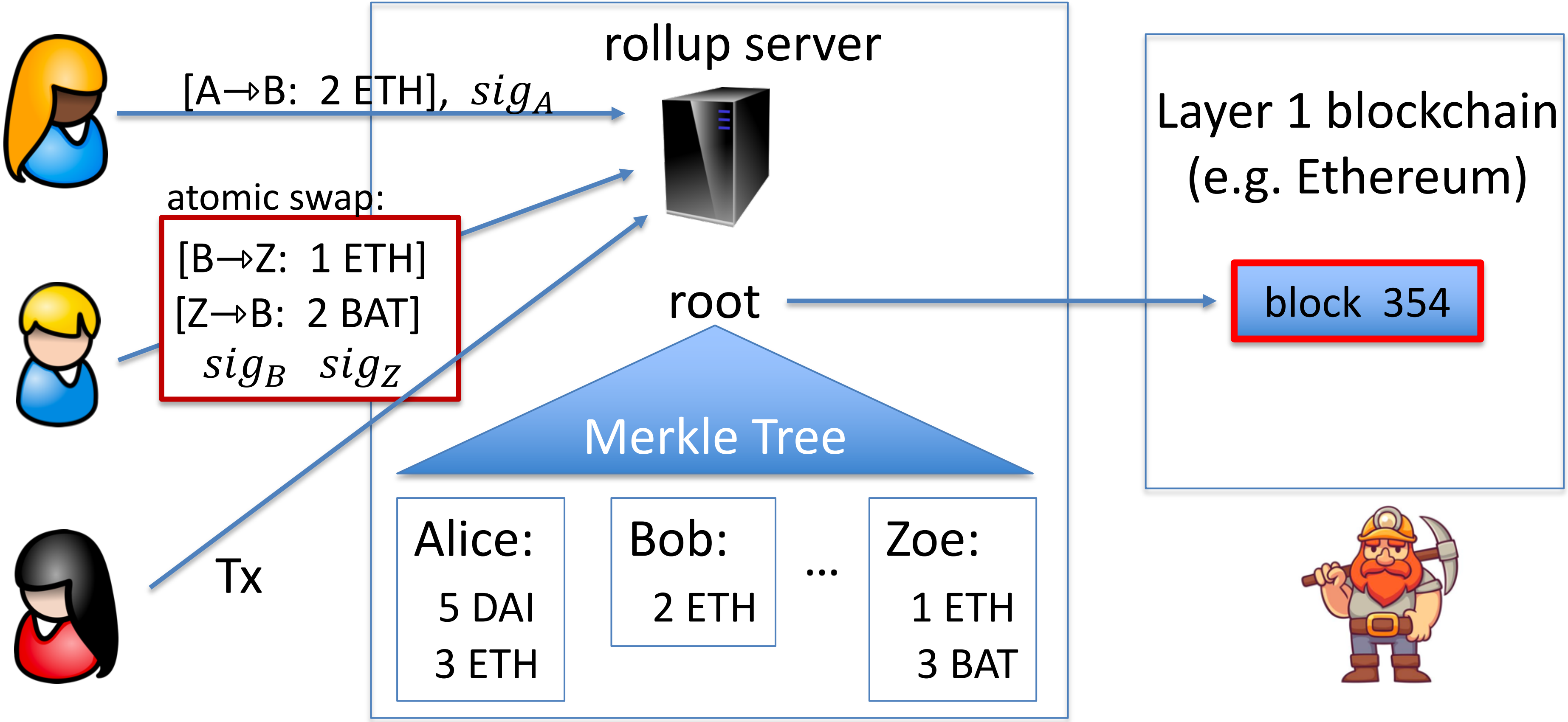
CS251 Fall 2020

(cs251.stanford.edu)

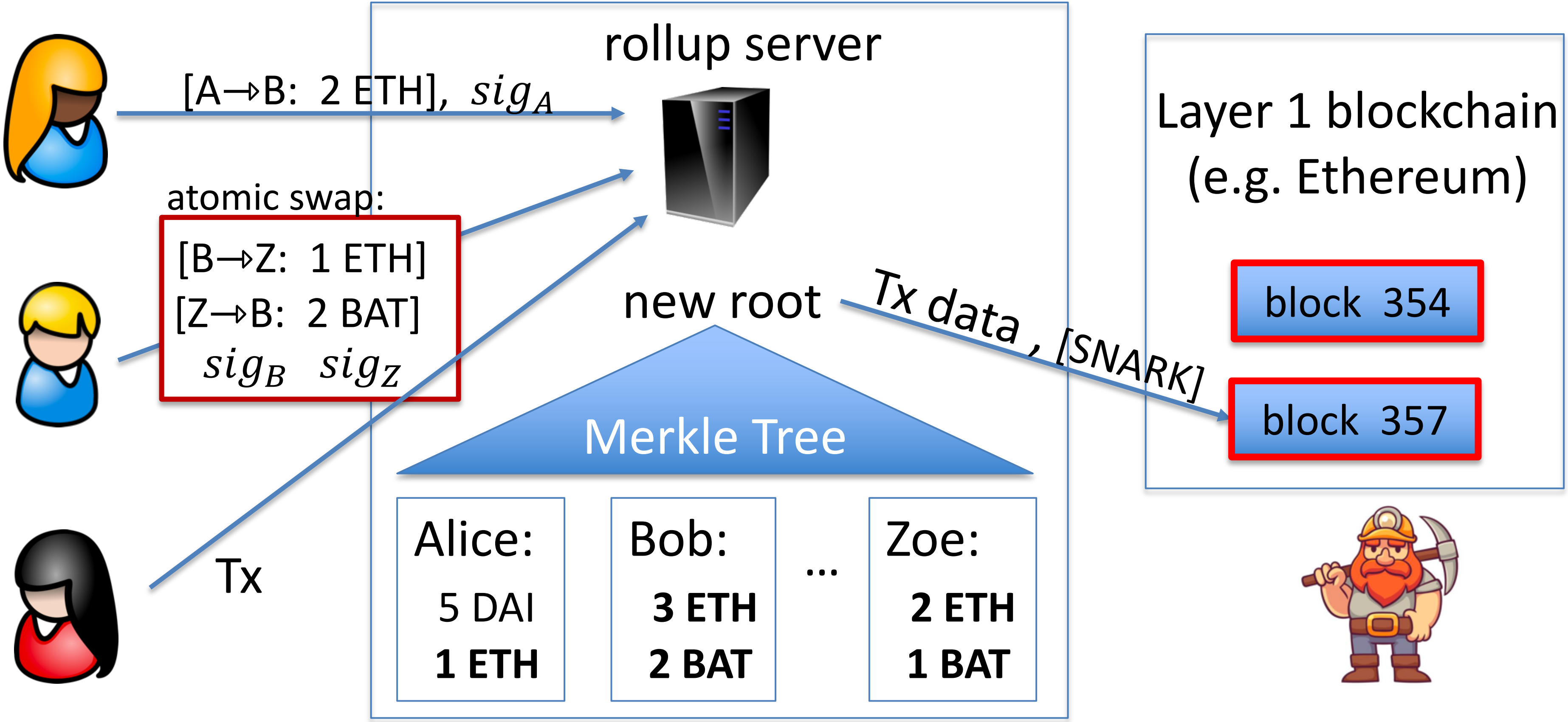
Final topics: Cute Crypto Tricks

Dan Boneh

Quick Recap: Rollup with privacy



Quick Recap: Rollup with privacy



Privacy?

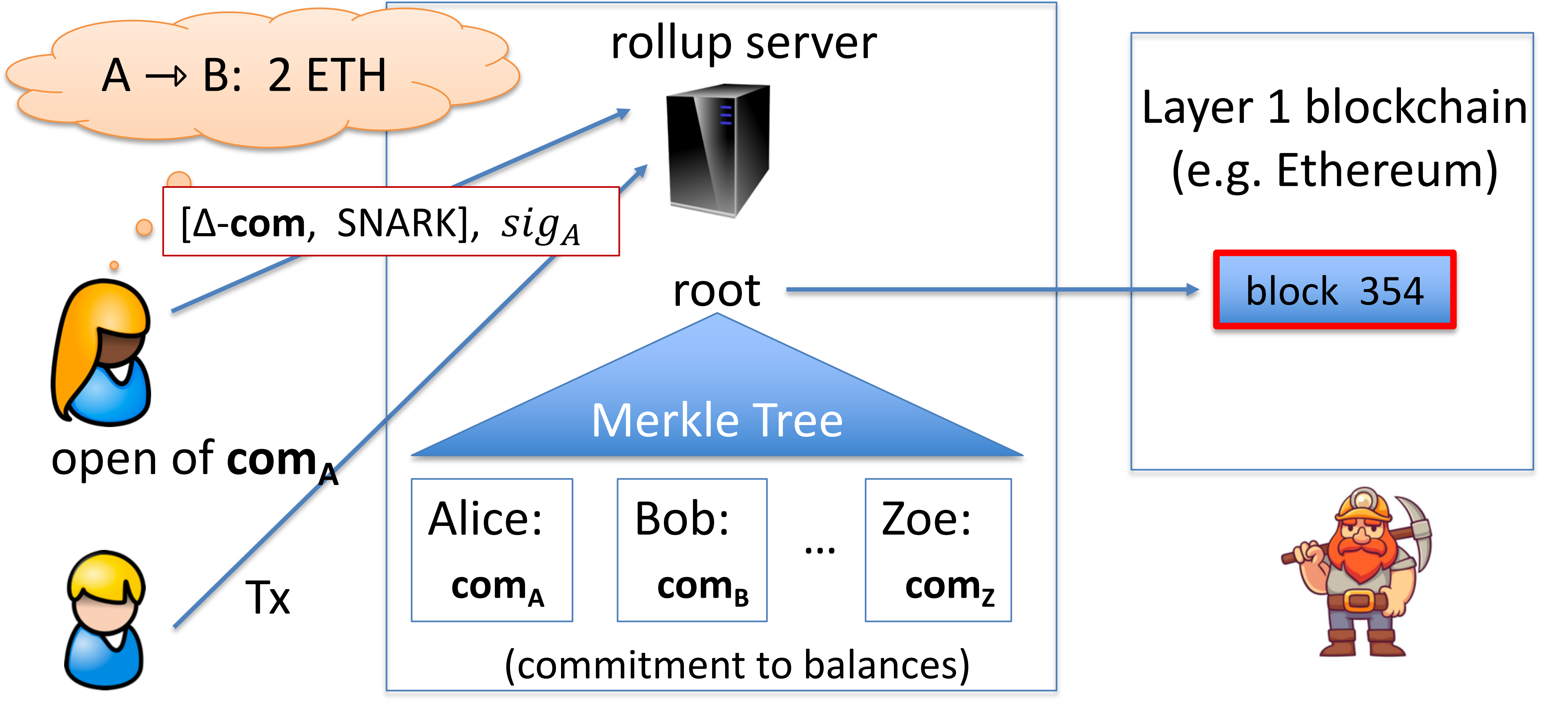
The Rollup server sees:

- all account balances, and
- all transactions.

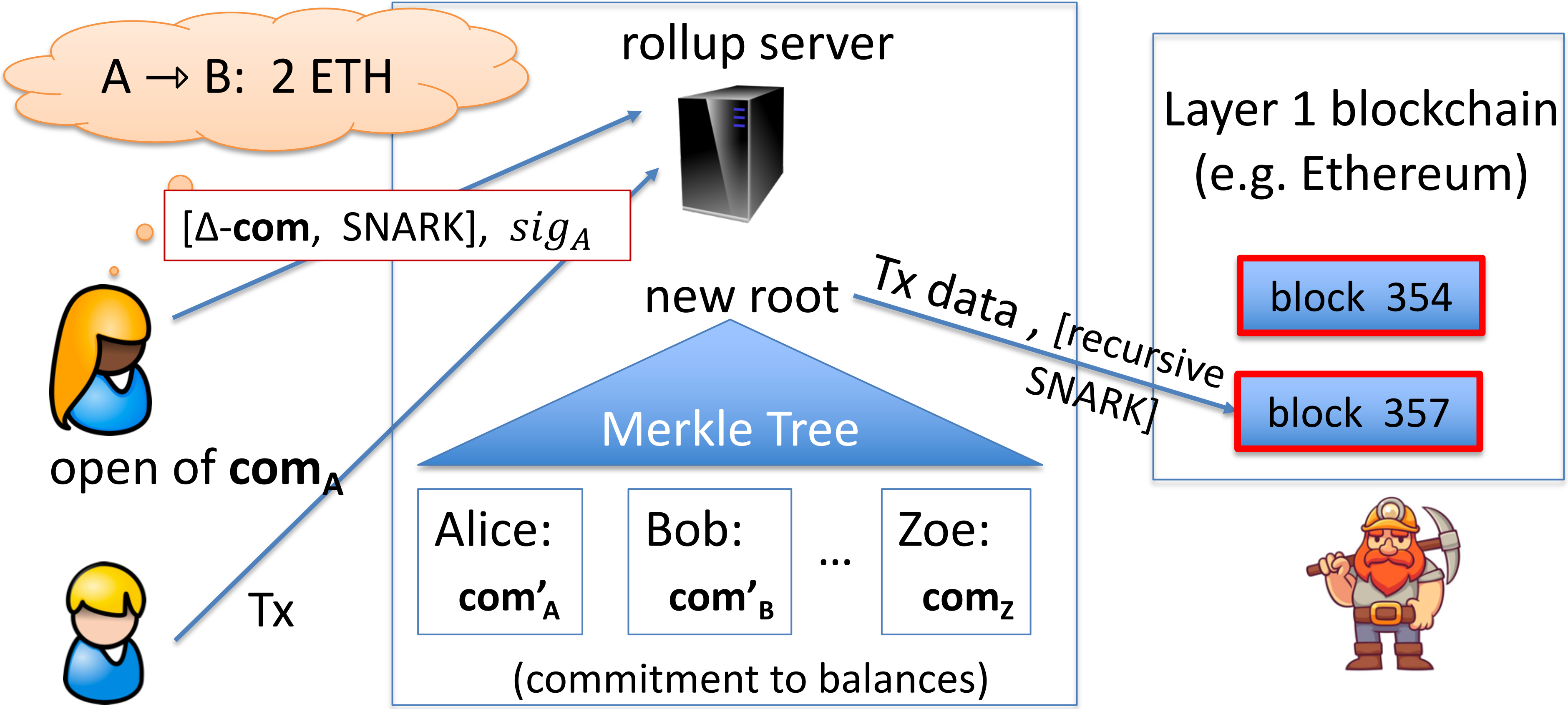
Can we hide this from the Rollup server?

Yes! Using zkSNARKs

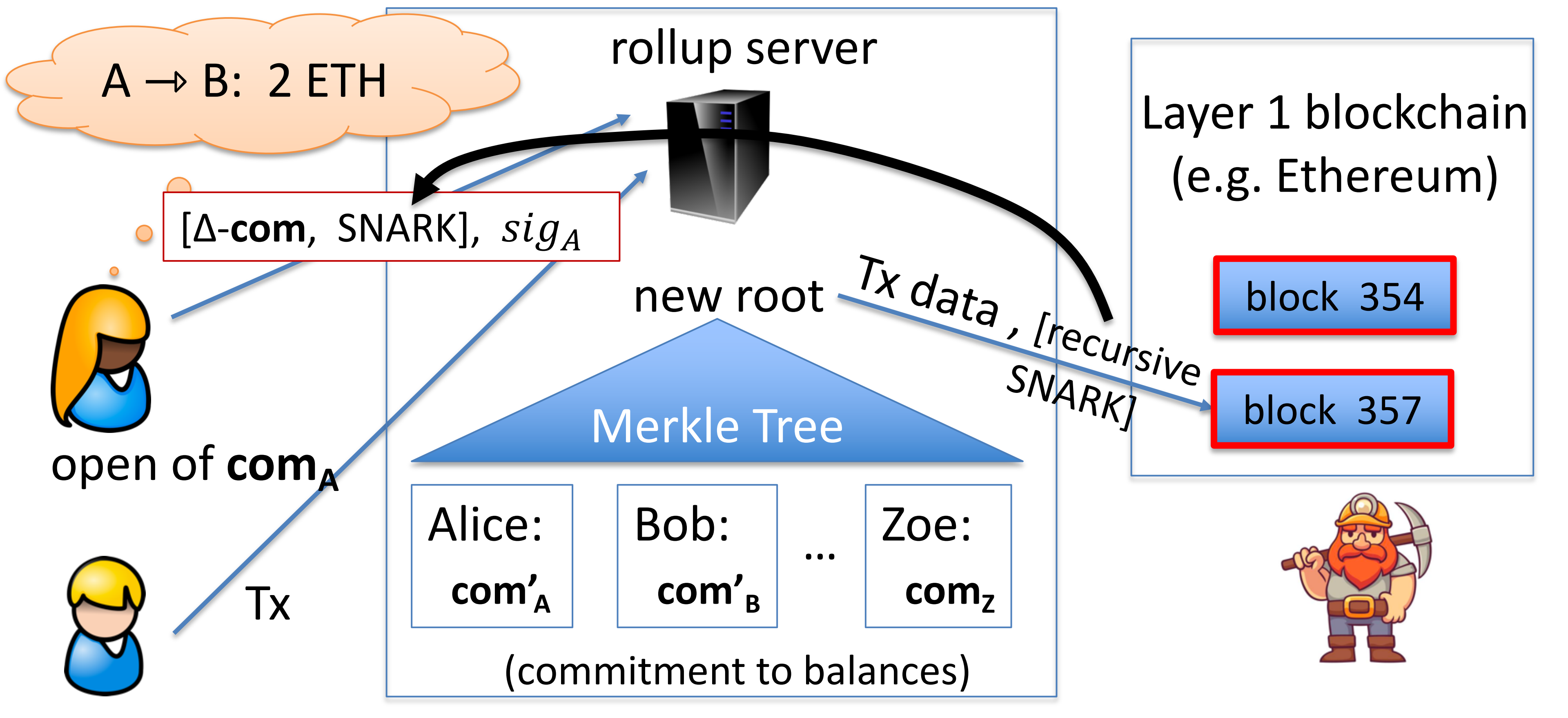
Replace balances with commitments (simplified)



Replace balances with commitments (simplified)



Replace balances with commitments (simplified)



$(zk)^2$-Rollup

Problem: Rollup server sees identity of payer and payee

- Better data structure enables fully private Tx

  … can even run DAPPs privately (e.g., ZEXE)

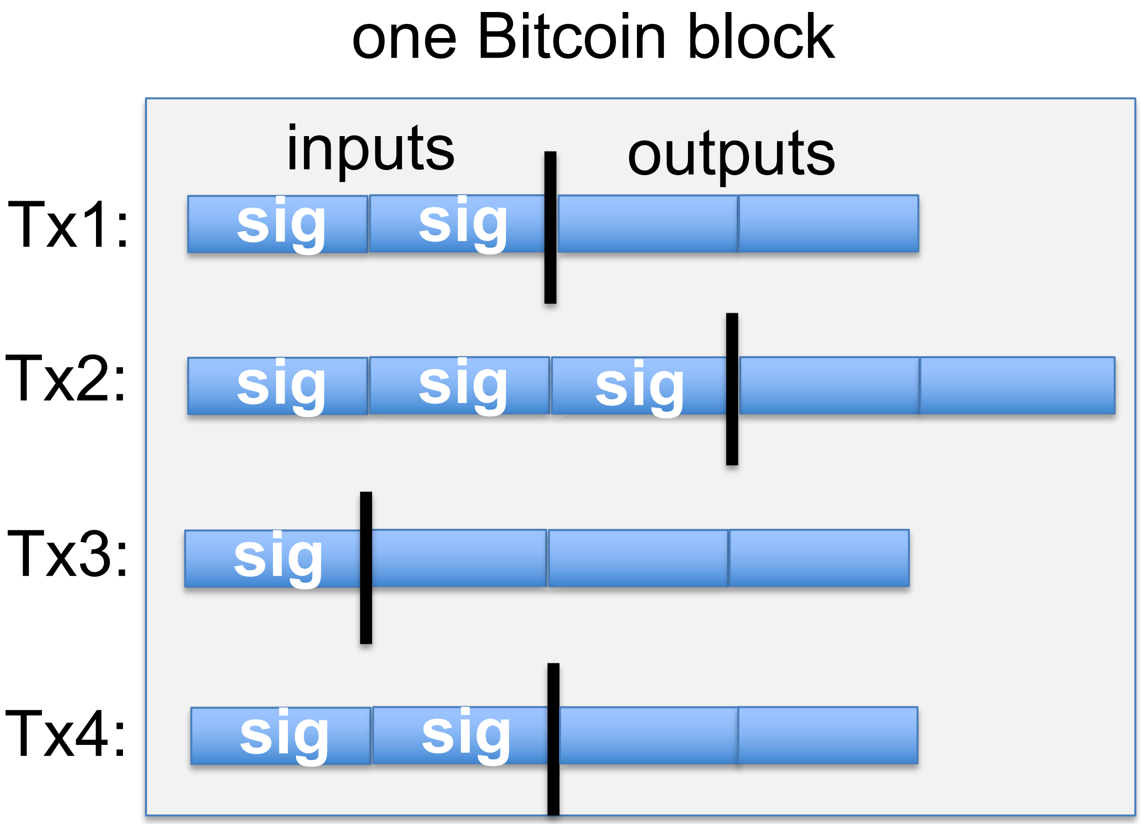
General discussion

Discussion Topics

- Open problems in blockchains
- The future of blockchains and crypto currencies
  - Where is this headed?
- A career in blockchains?
- Where can I learn more?
  - **EE374**: Scaling blockchains (Winter 2021)
  - **CS255** and **CS355**: Cryptography
  - Stanford blockchain conference (SBC)

Fun crypto tricks

BLS signatures



Signatures make up most of Tx data.

Can we compress signatures?

- Yes: aggregation!
- not possible for ECDSA

BLS Signatures

Used in modern blockchains: Ehtereum 2.0, Dfinity, Chia, etc.

The setup:

- $G = \{1, g, \ldots, g^{q-1}\}$ a cyclic group of prime order q
- $H: M \times G \to G$ a hash function (e.g., based on SHA256)

BLS Signatures

<u>KeyGen</u>(): choose random α in $\{1, \ldots, q\}$

output $\boxed{\text{sk} = \alpha \ , \quad \text{pk} = g^{\alpha} \ \in \text{G}}$

<u>Sign</u>(sk, m): output $\boxed{\text{sig} = H(m, \text{pk})^{\alpha} \ \in \text{G}}$

<u>Verify</u>(pk, m, sig): output accept if $\log_g(\text{pk}) = \log_{H(m,\text{pk})}(\text{sig})$

Note: signature on m is unique! (no malleability)

How does verify work?

A pairing: an efficiently computable function $e: G \times G \to G'$

such that $\boxed{e(g^{\alpha}, g^{\beta}) = e(g, g)^{\alpha\beta}}$ for all $\alpha, \beta \in \{1, \ldots q\}$

and is not degenerate: $e(g, g) \neq 1$

Observe: $\log_g(pk) = \log_{H(m,pk)}(sig)$

verify test

if and only if $\boxed{e(g, sig) = e(pk, H(m,pk))}$

$$e(g, sig) = e(g, H(m,pk)^{\alpha})$$

$$e(pk, H(m,pk)) = e(g^{\alpha}, H(m,pk))$$

$$e(g, H(m,pk)^{\alpha}) = e(g^{\alpha}, H(m,pk))$$

Properties: signature aggregation [BGLS'03]

Anyone can compress n signatures into one

pk_1 , m_1 $\longrightarrow$ σ_1

⋮

pk_n , m_n $\longrightarrow$ σ_n

aggregate $\rightarrow \sigma^*$

single short signature

Verify($\overline{\mathbf{pk}}$, $\overline{\mathbf{m}}$, σ^*) = "accept"

convinces verifier that

for i=1,…,n:

user i signed msg m_i

Aggregation: how

$$\begin{cases} \text{user 1:} \quad pk_1 = g^{\alpha 1}, \quad m_1 \longrightarrow \quad \sigma_1 = H(m_1, pk_1)^{\alpha_1} \\ \quad\vdots \\ \text{user n:} \quad pk_n = g^{\alpha n}, \quad m_n \longrightarrow \quad \sigma_n = H(m_n, pk_n)^{\alpha_n} \end{cases}$$

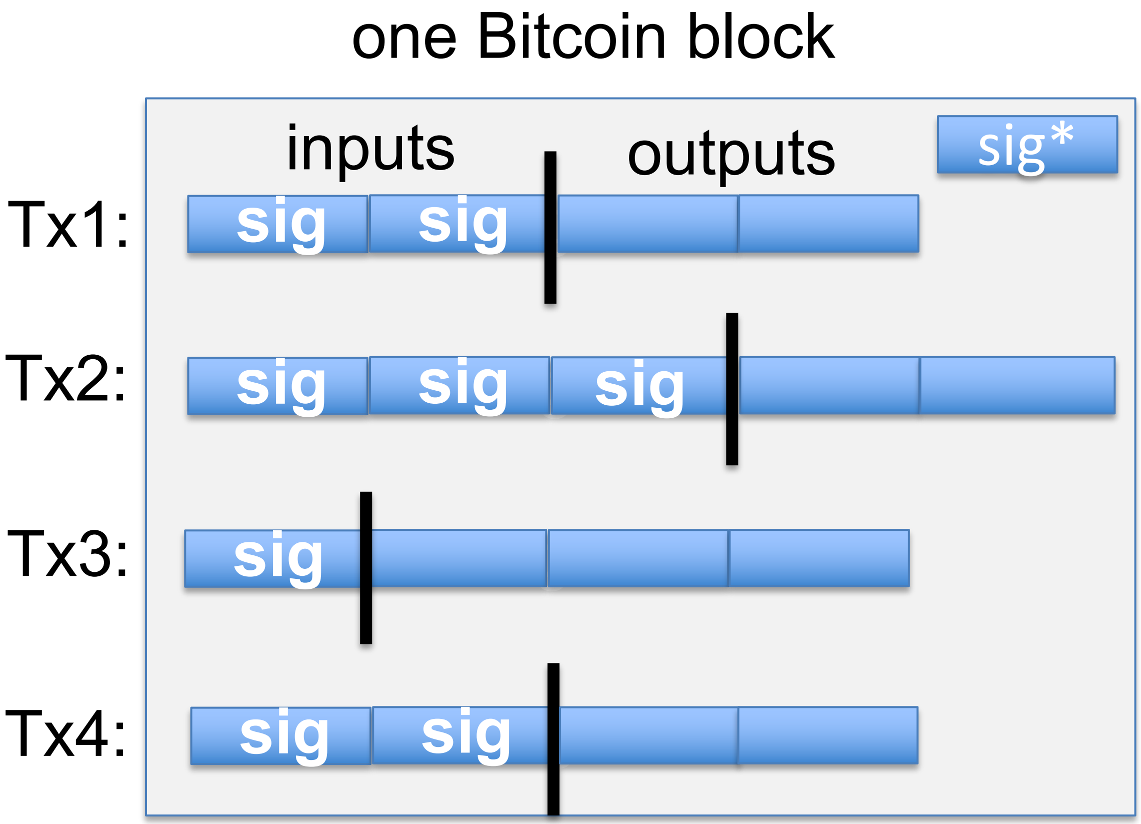
$$\boxed{\sigma \longleftarrow \sigma_1 \cdots \sigma_n}$$

Verifying an aggregate signature: (incomplete)

$$\prod_{i=1}^{n} e(H(m_i, pk_i), g^{\alpha_i}) \stackrel{?}{=} e(\sigma, g)$$

$$\prod_{i=1} e(H(m_i, pk_i)^{\alpha_i}, g) = e(\prod_{i=1} H(m_i, pk_i)^{\alpha_i}, g)$$

Compressing the blockchain with BLS



<u>if needed</u>:

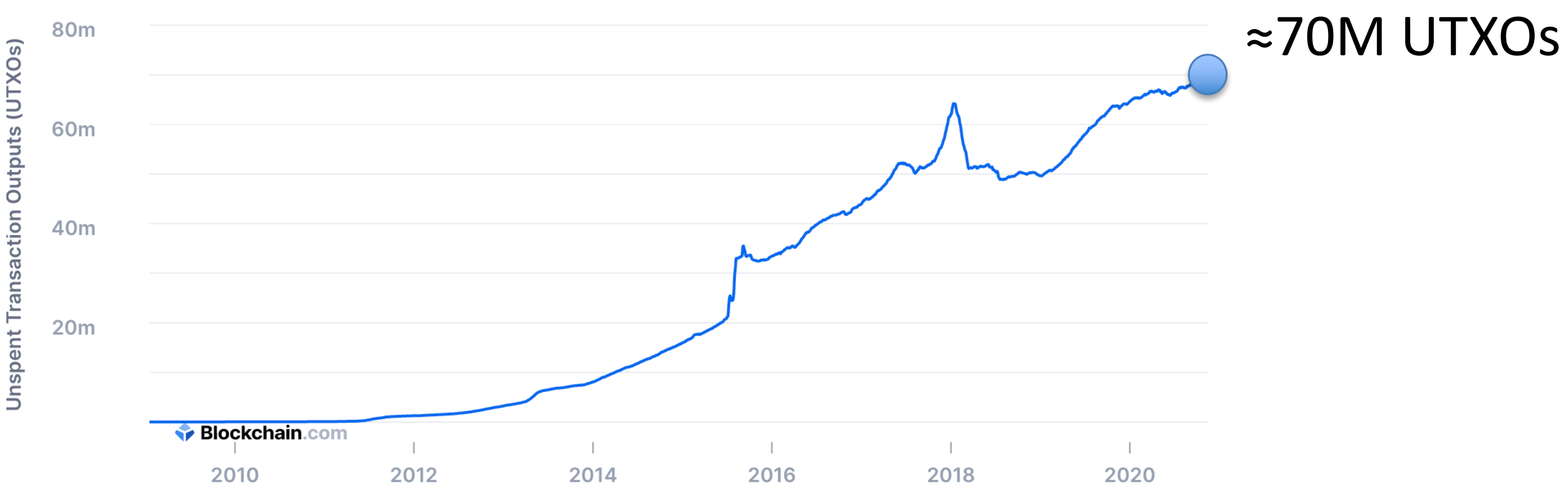
compress all signatures in a block into a single aggregate signatures

⇒ shrink block

or: aggregate in smaller batches

Reducing Miner State

UTXO set size



Miners need to keep all UTXOs in memory to validate Txs

Can we do better?

Recall: polynomial commitments

- <u>*commit*</u>(*pp*, f, r) → ***com$_f$*** commitment to f ∈ $\mathbb{F}_p^{(\leq d)}[X]$

- <u>*eval*</u>: goal: for a given ***com$_f$*** and x, y ∈ $\mathbb{F}_p$,

  construct a SNARK to prove that f(x) = y.

Homomorphic polynomial commitment

A polynomial commitment is **homomorphic** if

there are efficient algorithms such that:

- *commit*$(pp, f_1, r_1) \to$ $\boldsymbol{com_{f1}}$ *commit*$(pp, f_2, r_2) \to$ $\boldsymbol{com_{f2}}$

Then:

(i) for all $a, b \in \mathbb{F}_p$: $\boldsymbol{com_{f1}}$, $\boldsymbol{com_{f2}}$ $\to$ $\boldsymbol{com_{a*f1+b*f2}}$

(ii) $\boldsymbol{com_{f1}}$ $\to$ $\boldsymbol{com_{X*f1}}$

Committing to a set (of UTXOs)

(accumulator)

Let $S = \{U_1, \dots, Un\} \in \mathbb{F}_p$ be a set of UTXOs

Define: $f(X) = (X - U_1) \cdots (X - Un) \quad \in \mathbb{F}_p^{(\leq n)}[X]$

Set: $\boldsymbol{com_f}$ = commit(pp, f, r) ⇽ short commitment to S

For $U \in \mathbb{F}_p$: $U \in S$ if and only if $f(U) = 0$

To add U to S: $\boldsymbol{com_f}$ → $\boldsymbol{com_{X*f-U*f}}$ ⇽ short commitment to $S \cup \{U\}$

How does this help?

Miners maintain two commitments:

(i) commitment to set T of all UTXOs

(ii) commitment to set S of spent TXOs

$\}$ $\leq$ 1KB

com_T, com_S

Tx format:

- every input U includes a proof $(U \in T \;\; \&\& \;\; \mathrm{U} \notin S)$

  Two eval proofs: $T(U) = 0 \;\; \&\& \;\; S(U) \neq 0$ (short)

Tx processing: miners check eval proofs, and if valid,

add inputs to set S and outputs to set T. That's it!

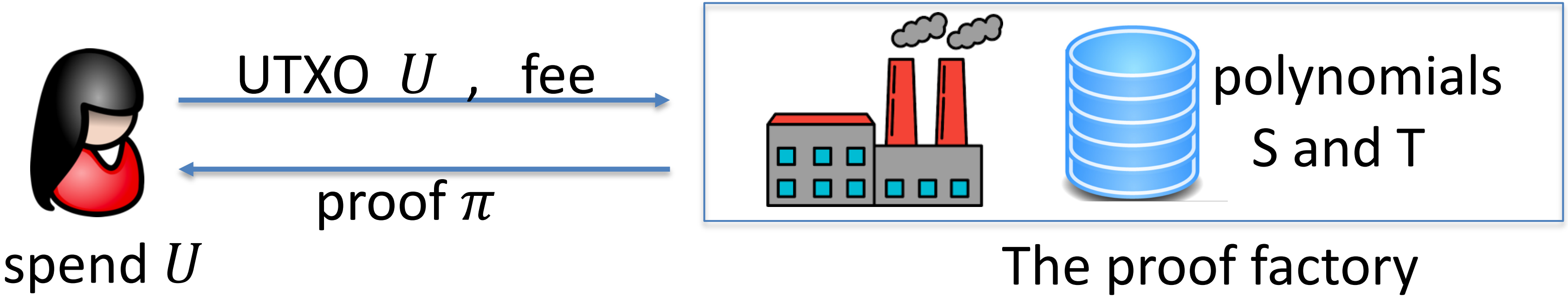
Does this work ??

Problem: how does a user prove that her UTXO U satisfies

$$T(U) = 0 \quad \&\& \quad S(U) \neq 0 \qquad ???$$

This requires knowledge of the entire blockchain

- $\Rightarrow$ user needs large memory and compute time
- $\Rightarrow$ … can be outsourced to an untrusted 3rd party



Is this practical?

Not quite …

- Problem: the factory's work per proof is linear in the number of UTXOs ever created

- Many variations on this design:
  - can reduce factory's work to $\log_2$(# current UTXOs) per proof
  - Factory's memory is linear in (# current UTXOs)

End result: outsource memory requirements to a small number of 3rd party service providers

Taproot: semi-private scripts in Bitcoin

Taproot is coming …

Taproot Has Been Merged Into Bitcoin Core: Here’s What That Means

Oct 15, 2020 at 7:48 a.m. PDT ▪ Updated Oct 15, 2020 at 9:14 a.m. PDT

Script privacy

Currently: Bitcoin scripts must be fully revealed in spending Tx

Can we keep the script secret?

Answer: Yes, easily! when all goes well …

How?

ECDSA and Schnorr public keys:

- **KeyGen**(): sk = α , pk = g^{α} ∈ G for α in $\{1, \ldots, q\}$

Suppose $sk_A = \alpha$, $sk_B = \beta$.

- Alice and Bob can sign with respect to pk $= pk_A \cdot pk_B = g^{\alpha+\beta}$

  ⇒ an interactive protocol between Alice and Bob

  (note: much simpler with BLS)

  ⇒ Alice & Bob can imply consent to Tx by signing with pk = $g^{\alpha+\beta}$

How?

S: Bitcoin script that must be satisfied to spend a UTXO U

S involves only Alice and Bob. Let $pk_{AB} = pk_A \cdot pk_B$

Goal: keep S secret when possible.

How: modify S so that a signature with respect to

pk = $pk_{AB} \cdot g^{H(pk_{AB}\,, S)}$

is sufficient to spend UTXO, without revealing S !!

The main point

- If parties agree to spend UTXO,

  $\Rightarrow$ sign with respect to pk_{AB} and spend while keeping S secret

- If disagreement, Alice can reveal S

  and spend UTXO by proving that she can satisfy S.

Taproot pk compactly supports both ways to spend the UTXO

END OF LECTURE

Next lecture: super cool final guest lecture