CS251 Fall 2023

(cs251.stanford.edu)

Final Topics:
Bridging, Account Abstraction, and DAOs

Dan Boneh

Invited talk final lecture. Final exam next Wednesday.

Many more topics to cover …

- Blockchain interoperability (bridging)
- Account abstraction
- Governance: How to decide on updates to, say, Compound?
- Re-staking: using your stake to secure multiple applications
- Shared sequencing: sequencing Tx for multiple L2s at once
- Embedded wallets: simplifying the user experience
- **Many more cryptography techniques** (see slides at end)

More topics …

Where can I learn more?

- **CS255** and **CS355**: Cryptography (Winter and Spring)
- **EE374**: Scaling blockchains with fast consensus
- **Science of blockchain conference** (SBC): Aug. 2024.
- **Stanford blockchain club**

A career in blockchains? Where to start? [link]

Interoperability between blockchain

How to bridge chains

Many L1 blockchains

Bitcoin: Bitcoin scripting language (with Taproot)

Ethereum: EVM. Currently: high Tx fees (better with Rollups)

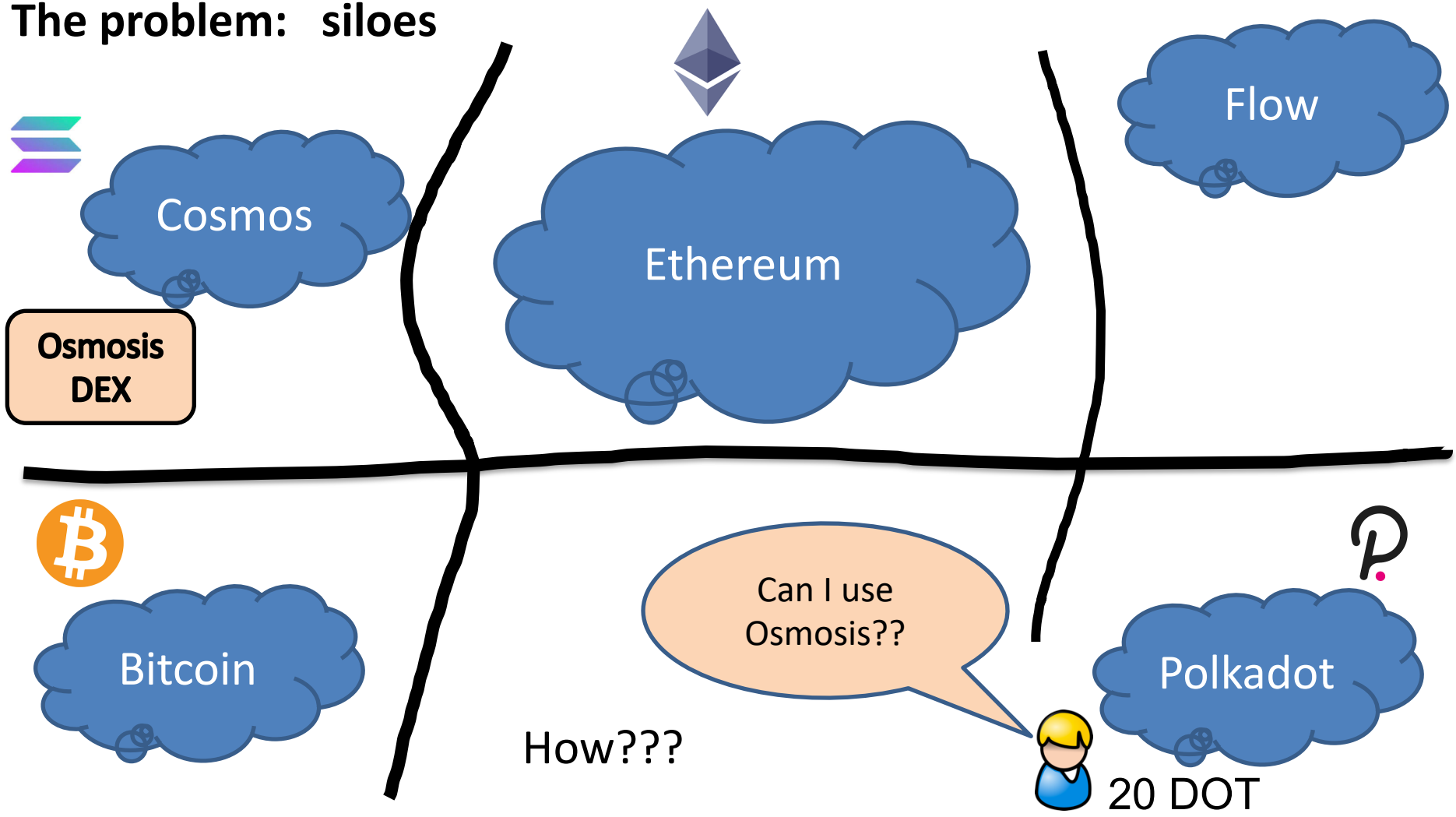
EVM compatible blockchains: **Celo, Avalanche, BSC**, …

- Higher Tx rate $\Longrightarrow$ lower Tx fees
- EVM compatibility $\Longrightarrow$ easy project migration and user support

Other fast non-EVM blockchains: **Solana, Flow, Algorand**, …

- Higher Tx rate $\Longrightarrow$ lower Tx fees

The problem: siloes



Interoperability

Interoperability:

- User owns funds or assets (NFTs) on one blockchain system
  Goal: enable user to move assets to another chain

Composability:

- Enable a DAPP on one chain to call a DAPP on another

Both are easy if the entire world used Ethereum

- In reality: many blockchain systems that need to interoperate
- The solution: **bridges**

A first example: BTC in Ethereum

How to move BTC to Ethereum ?? Goal: enable BTC in DeFi.

$\Longrightarrow$ need new ERC20 on Ethereum pegged to BTC

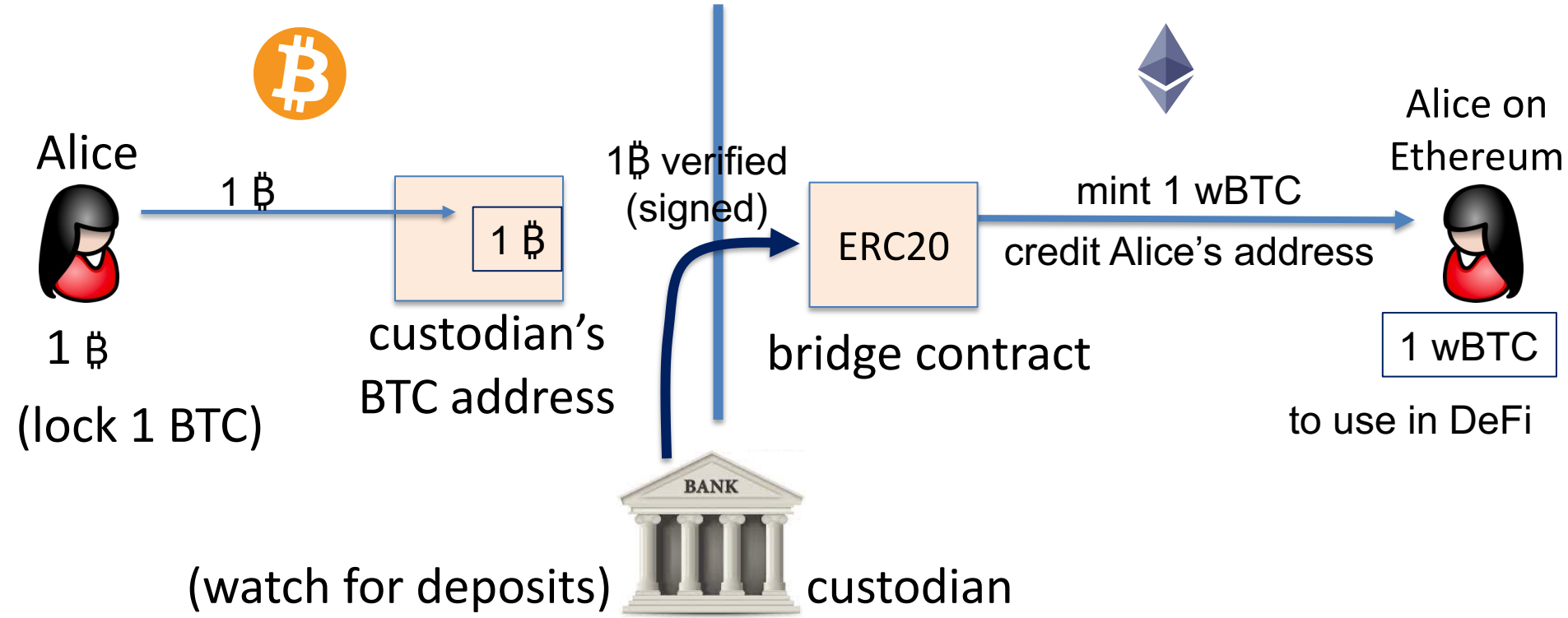
(e.g., use it for providing liquidity in DeFi projects)

The solution: **wrapped coins**

- Asset X on one chain appear as wrapped-X on another chain
- For BTC: several solutions (e.g., wBTC, tBTC, …)

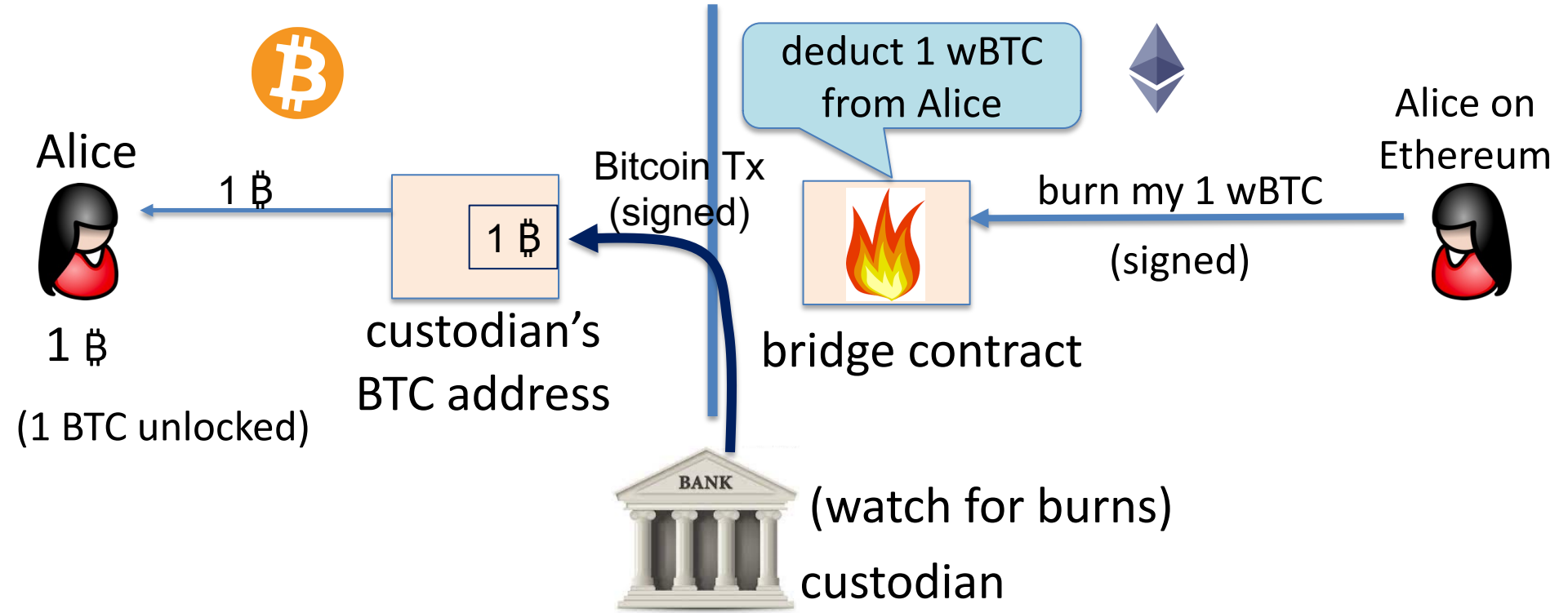
wBTC and tBTC: a lock-and-mint bridge

Let's start with wBTC: **moving 1 BTC to Ethereum**



Alice wants her 1 BTC back

Moving 1 wBTC back to the Bitcoin network:



wBTC

Example BTC → Ethereum:

Nov 26 2021 - 07:36	FUNDS SENT TO CUSTODIAN c605b4f2f0948e7deae0c5d7c27b3256b97120be760e2b81136eb95c819570f6	(Bitcoin Tx: ≈4,000 BTC)
Nov 26 2021 - 09:50	MINT COMPLETED BY CUSTODIAN 0x70475eca8be89b67143f1b52df013fc1df7d254e836c836c8f368fc516aca76b	(Ethereum Tx:)

Why two hours? … make sure no Bitcoin re-org

The problem: trusted custodian

Can we do better?

tBTC: no single point of trust

Alice requests to mint tBTC:

random three registered custodians are selected and they generate P2PKH Bitcoin address for Alice

signing key is 3-out-of-3 secret shared among three

(all three must cooperate to sign a Tx)

Alice sends BTC to P2PKH address, and received tBTC.

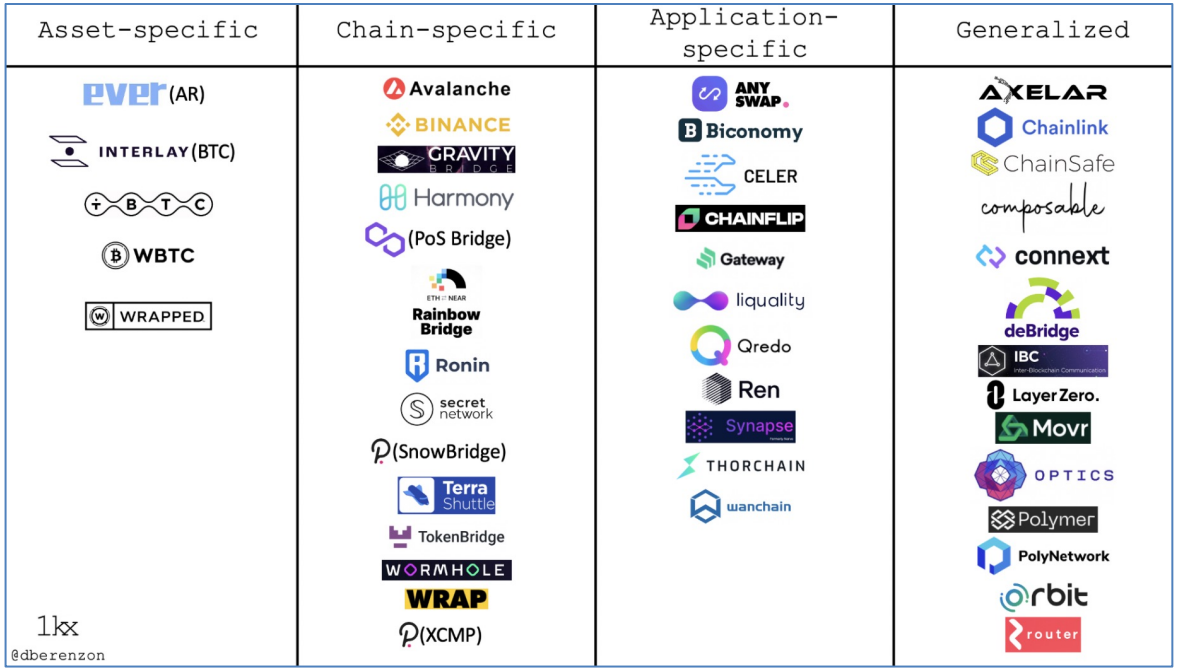
Custodians must lock 1.5x ETH stake for the BTC they manage

- If locked BTC is lost, Alice can claim staked ETH on Ethereum.

Bridging smart chains (with Dapp support)

A very active area:

- Many super interesting ideas



https://medium.com/1kxnetwork/blockchain-bridges-5db6afac44f8

Two types of bridges

Type 1: a lock-and-mint bridge

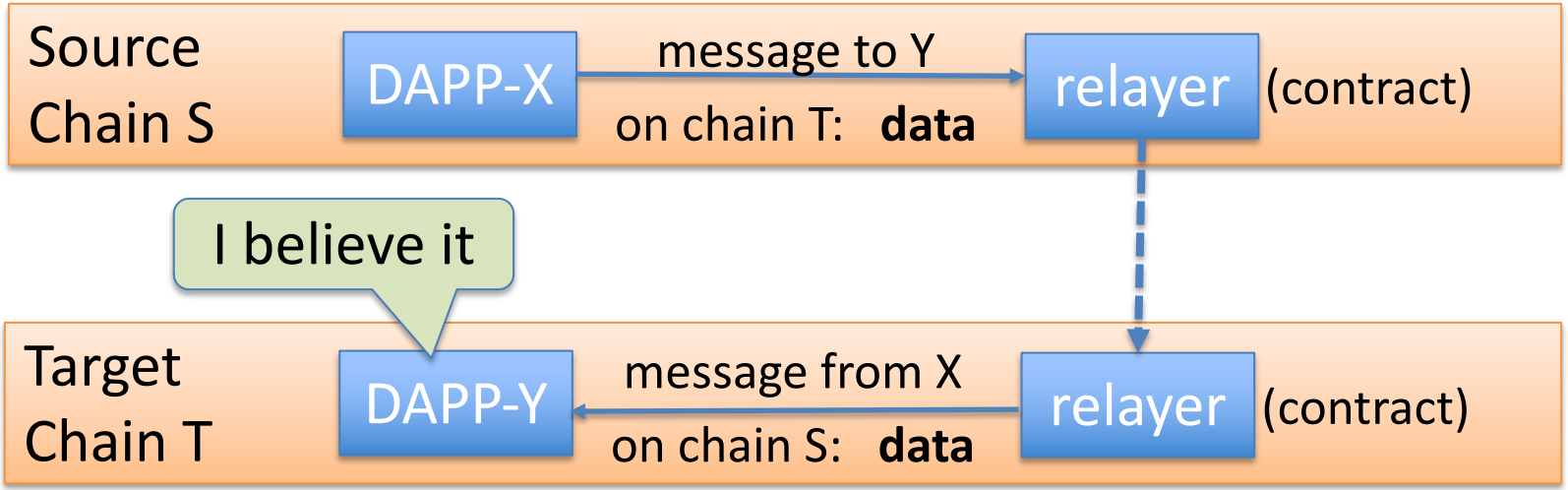
- SRC → DEST: user locks funds on SRC side,
  wrapped tokens are minted on the DEST side
- DEST → SRC: funds are burned on the DEST side,
  and released from lock on the SRC Side

Type 2: a liquidity pool bridge

- Liquidity providers provide liquidity on both sides
- SRC → DEST: user sends funds on SRC side,
  equivalent amount released from pool on DEST side

Bridging smart chains (with Dapp support)

Step 1 (hard): a secure cross-chain messaging system



Step 2 (easier): build a bridge using messaging system

Bridging smart chains (with Dapp support)

Step 1 (hard): a secure cross-chain messaging system

Step 2 (easier): build a bridge using messaging system

- DAPP-X → DAPP-Y: "I received 3 CELO, ok to mint 3 wCELO"
- DAPP-Y → DAPP-X: "I burned 3 wCELO, ok to release 3 CELO"

If messaging system is secure, no one can steal locked funds at S

Primarily two types of messaging systems

(1) **Externally verified**: external parties verify message on chain S

RelayerT dispatches only if all trustees signed

$\Longrightarrow$ **if** DAPP-Y trusts trustees, it knows DAPP-X sent message

Primarily two types of messaging systems

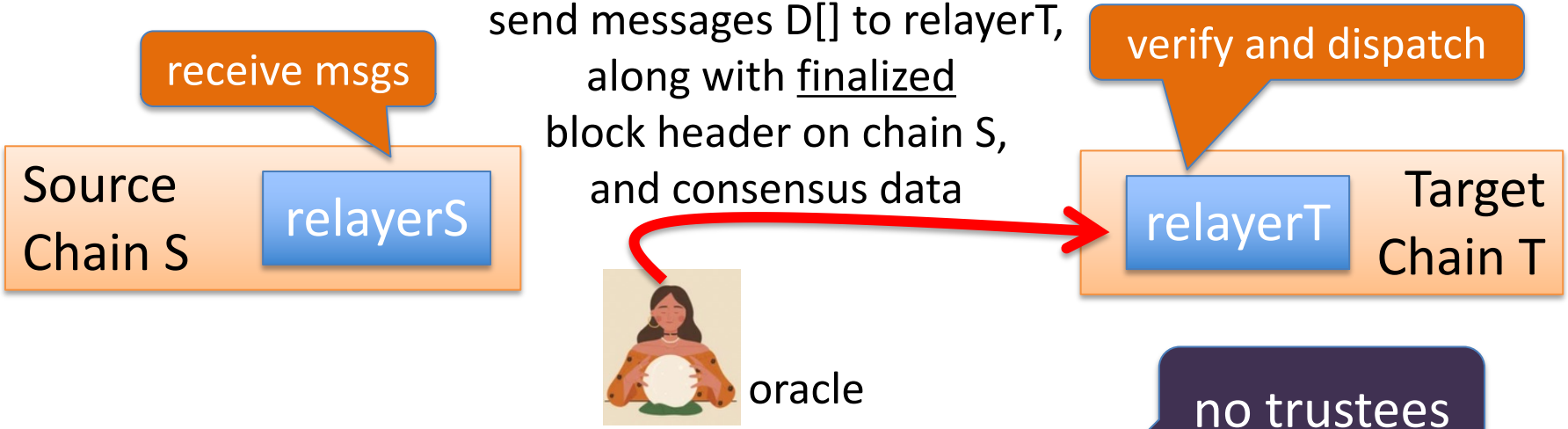
(1) **Externally verified**: external parties verify message on chain S

What if trustees sign and post a fake message to relayerT?

- anyone can send trustee's signature to relayerS $\Longrightarrow$ trustee slashed on S

Primarily two types of messaging systems

(2) **On-chain verified**: chain T verifies block header of chain S

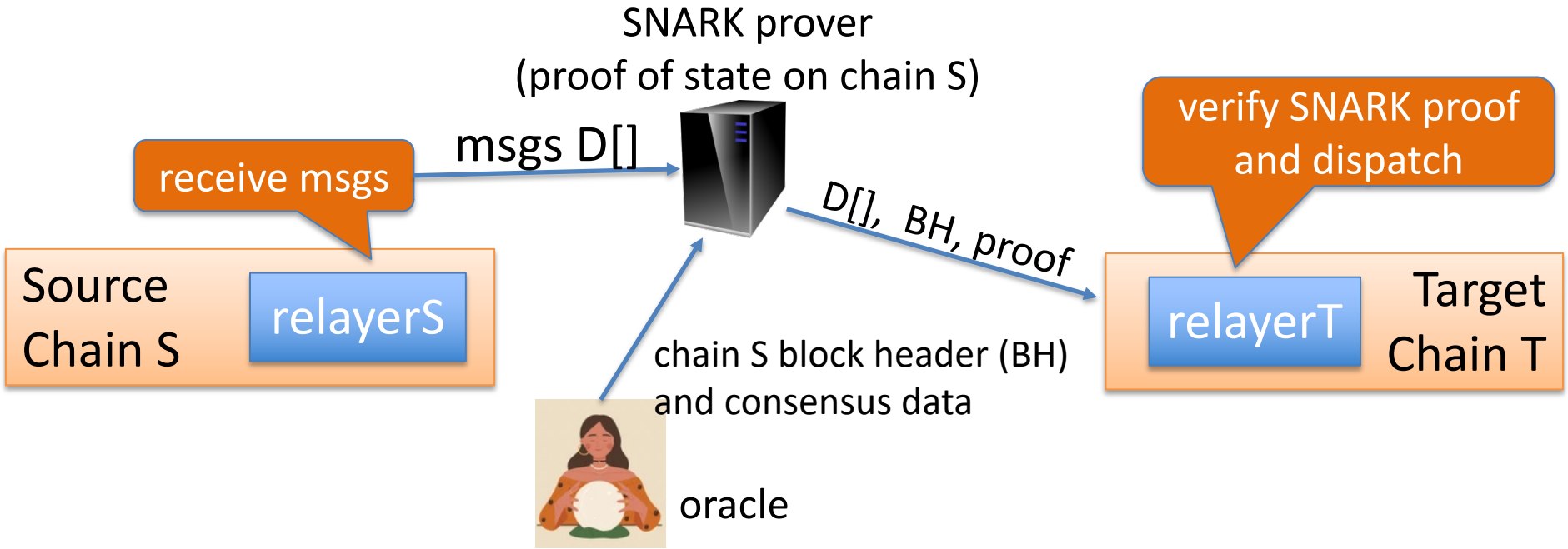


relayerT runs a light-client for chain S to verify that relayerS received messages D[]

no trustees

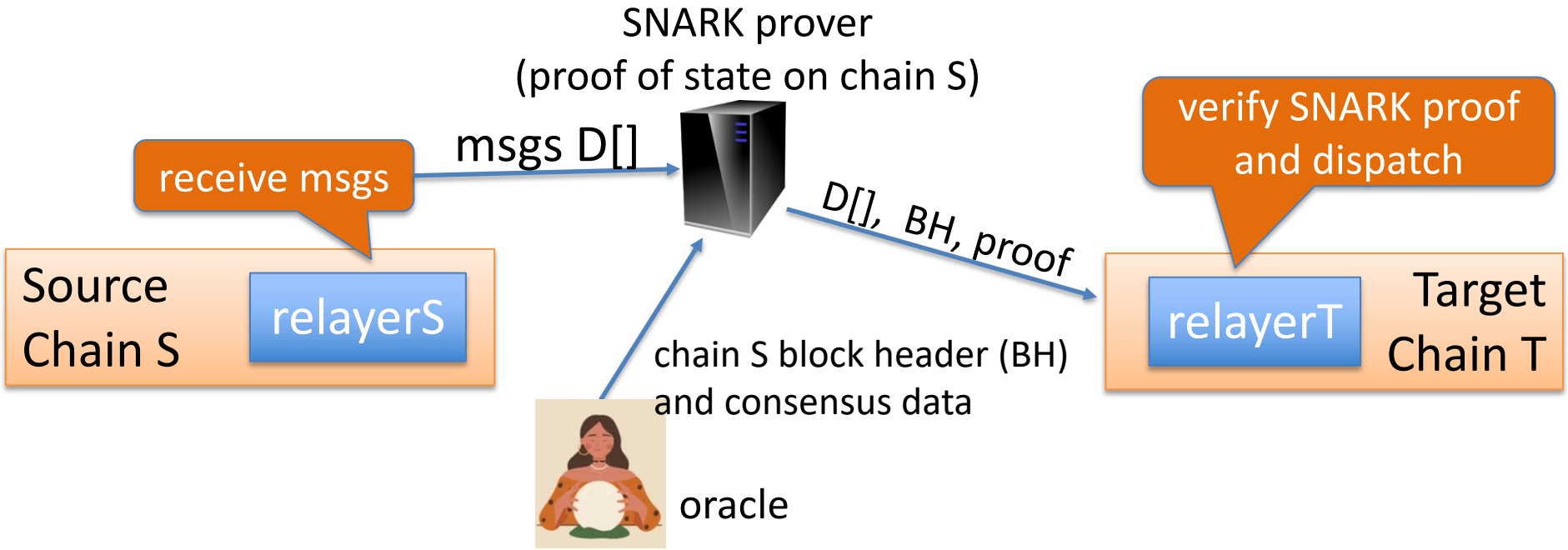
assumes security of light client

Primarily two types of messaging systems



Problem: high gas costs on chain T to verify state of source chain S.
Solution: zkBridge: use SNARK to reduce work for relayerT

Primarily two types of messaging systems



… being built by Succinct Labs

Bridging: the future vision

User can hold assets on any chain

- Assets move cheaply and quickly from chain to chain
- A project's liquidity is available on all chains
- Users and projects choose the chain that is best suited for their application and asset type

We are not there yet …

Account abstraction

… as designed in ERC-4337

The Goals of Account Abstraction (AA)

Recall: an asset held in an Externally Owned Account (EOA) is completely controlled by the EOA's signing key

- What if I want two keys to control the asset? Or rate limits? …

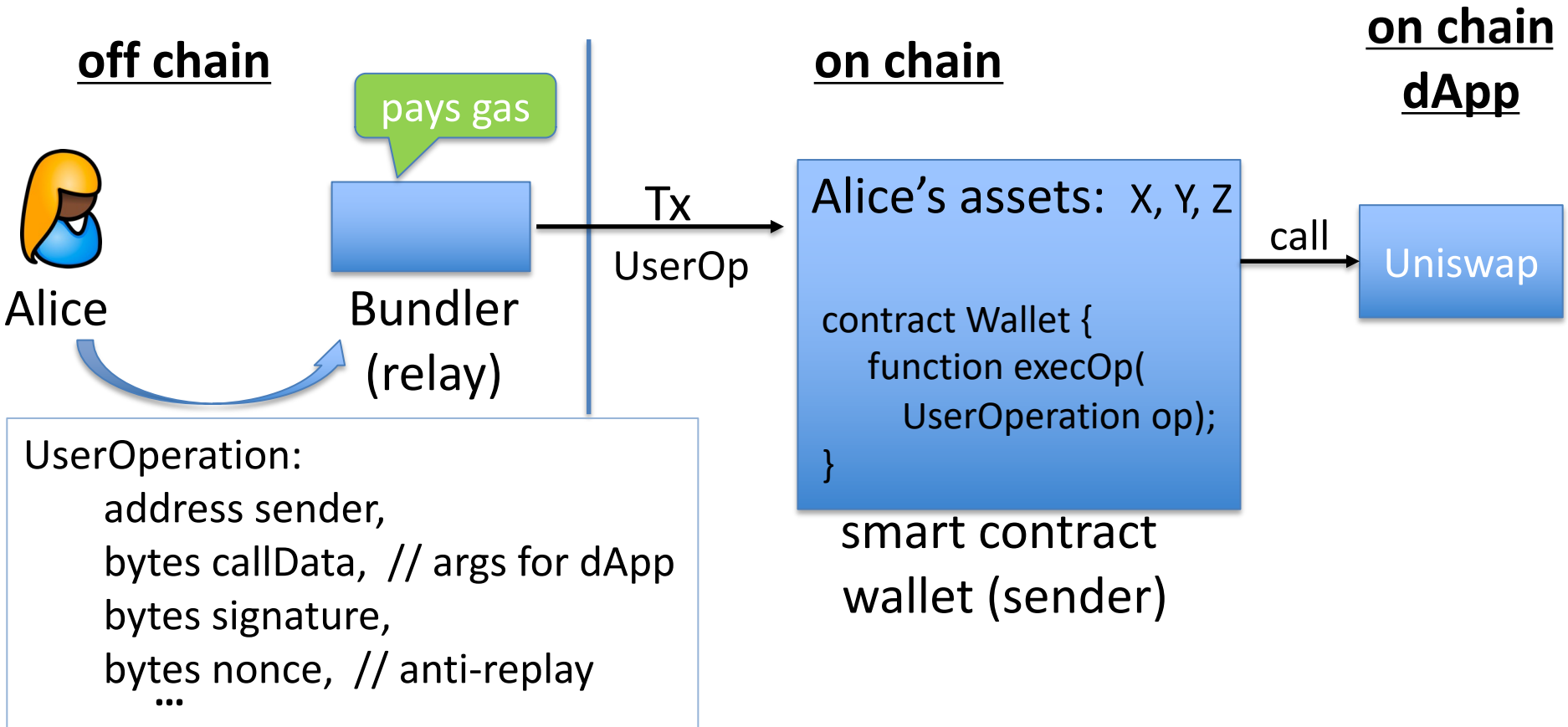
⇒ asset canot be held by an EOA

Two primary goals for AA:

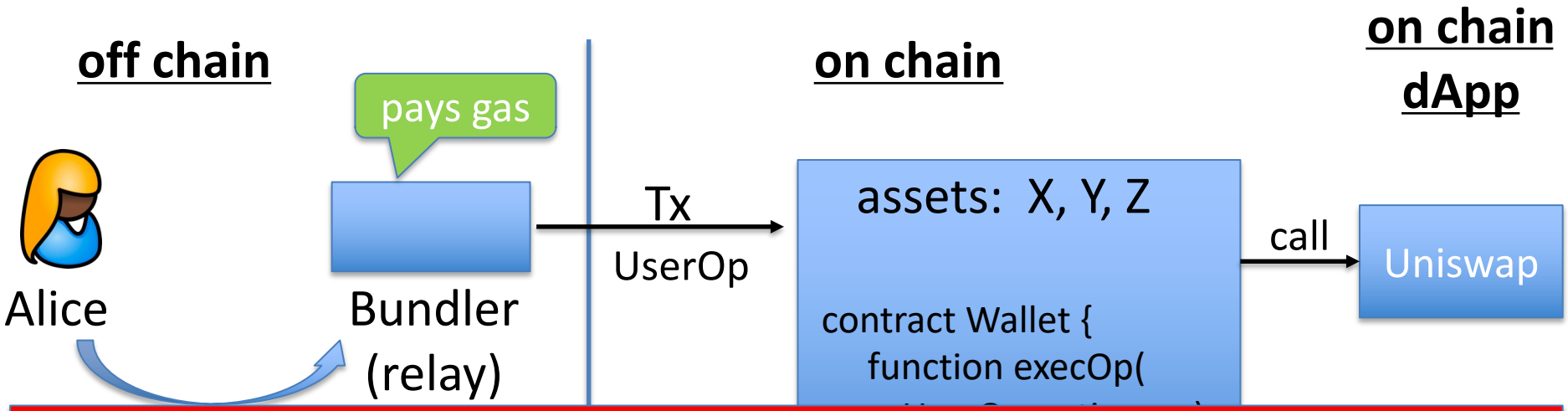
- Goal 1: arbitrary complex policies for controlling an asset
- Goal 2: a 3rd party (bundler, paymaster) pays Tx gas for users

⇒ improves user experience – no need to get ETH

Assets are held by a smart contract wallet



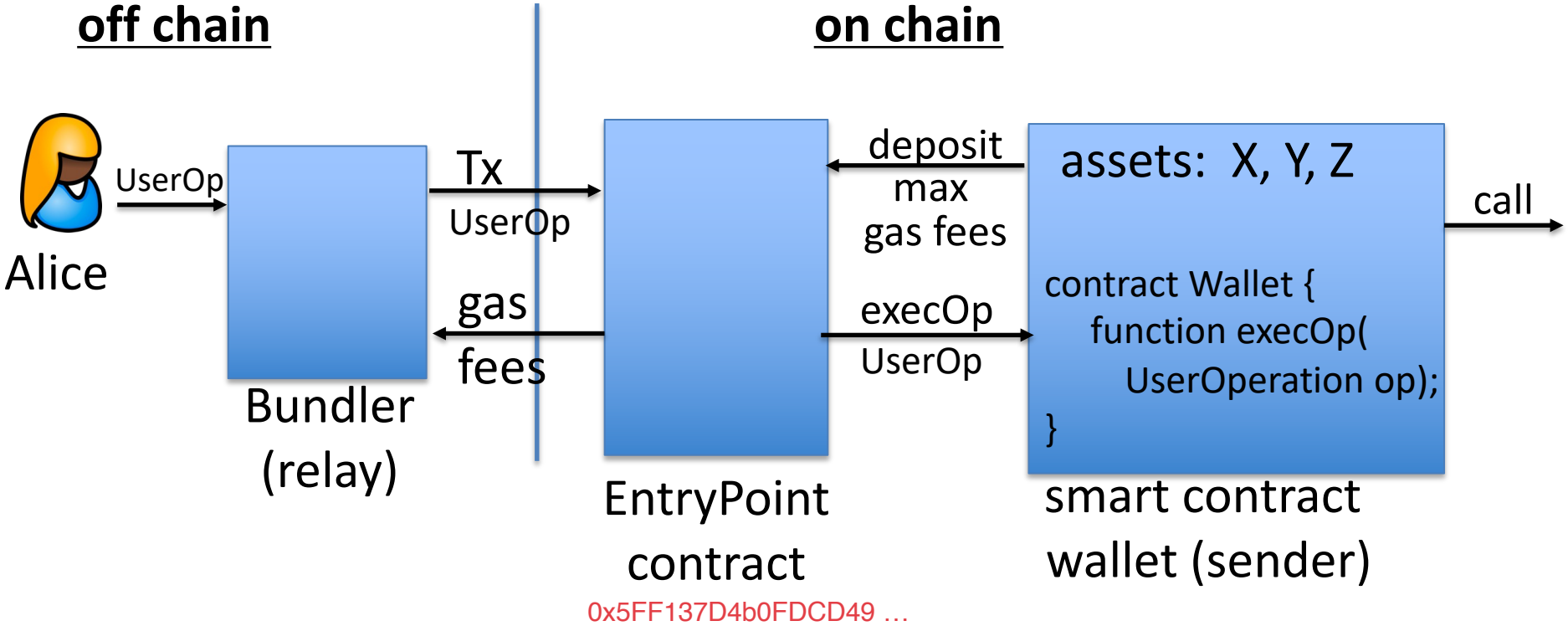
Smart contract wallet



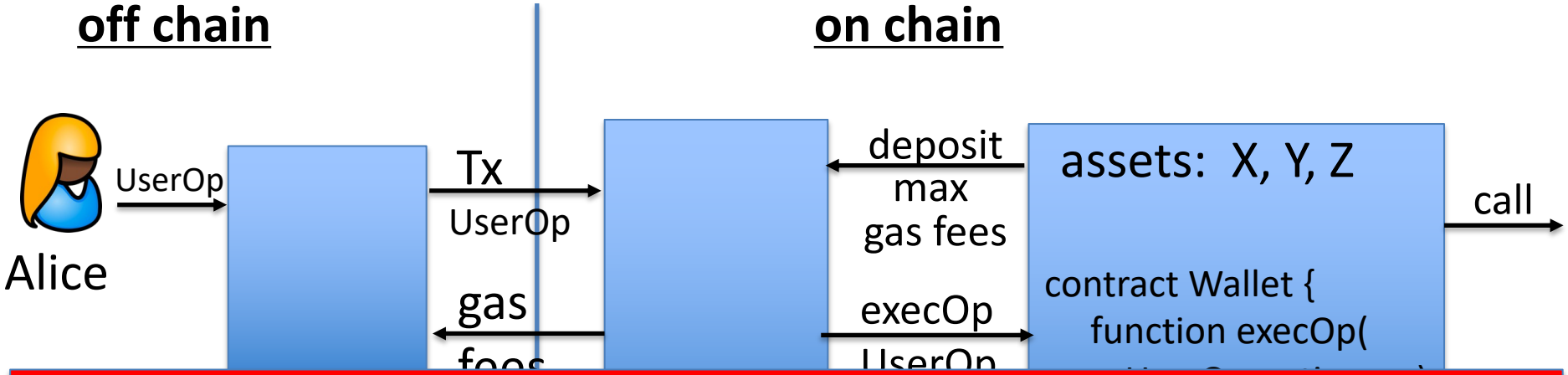
Problems:

- How to reimburse the bundler for the gas it paid?
- Bundler doesn't want to call (potentially malicious) wallet code

The EntryPoint trusted contract



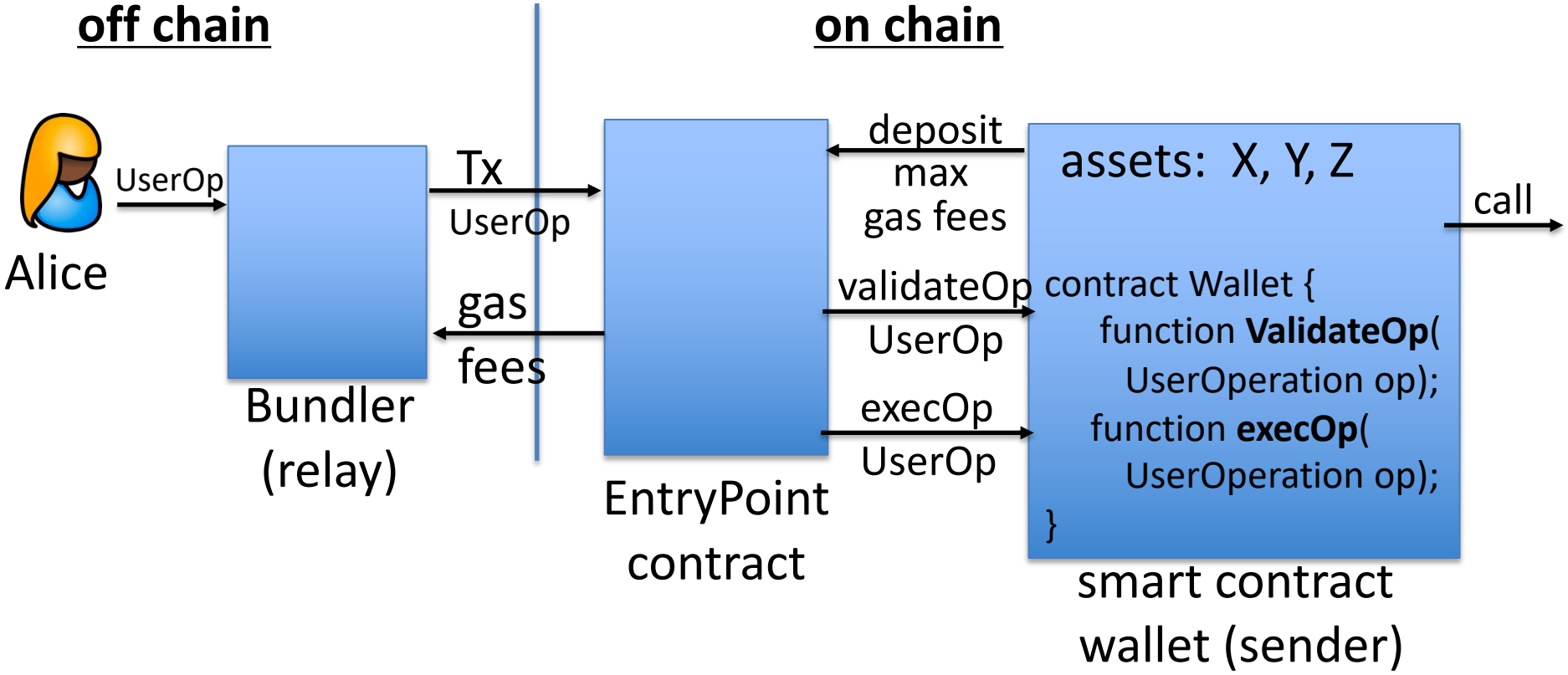
The EntryPoint trusted contract



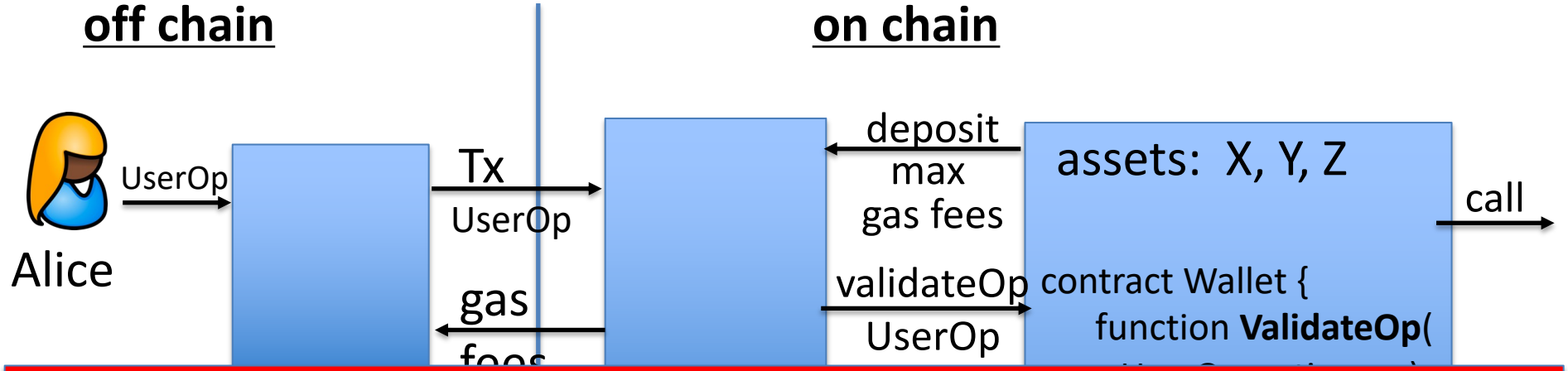
More problems:

- DoS: Anyone can issue invalid UserOp (e.g., with a bad sig) to EntryPoint, causing Wallet to keep paying gas to bundler.

Solution: validateOp



Solution: validateOp



EntryPoint steps: (simplified)

step 1: call **validateOp(UserOp)**

step 2: if fail, stop and Bundler pays the gas

step 3: call **ExecOp(UserOp),** and Wallet reimburses Bundler for gas

More problems …

1. Need to ensure that Wallet does not execute op in **validateOp**

   ⇒ Bundler will be stuck with execution fees

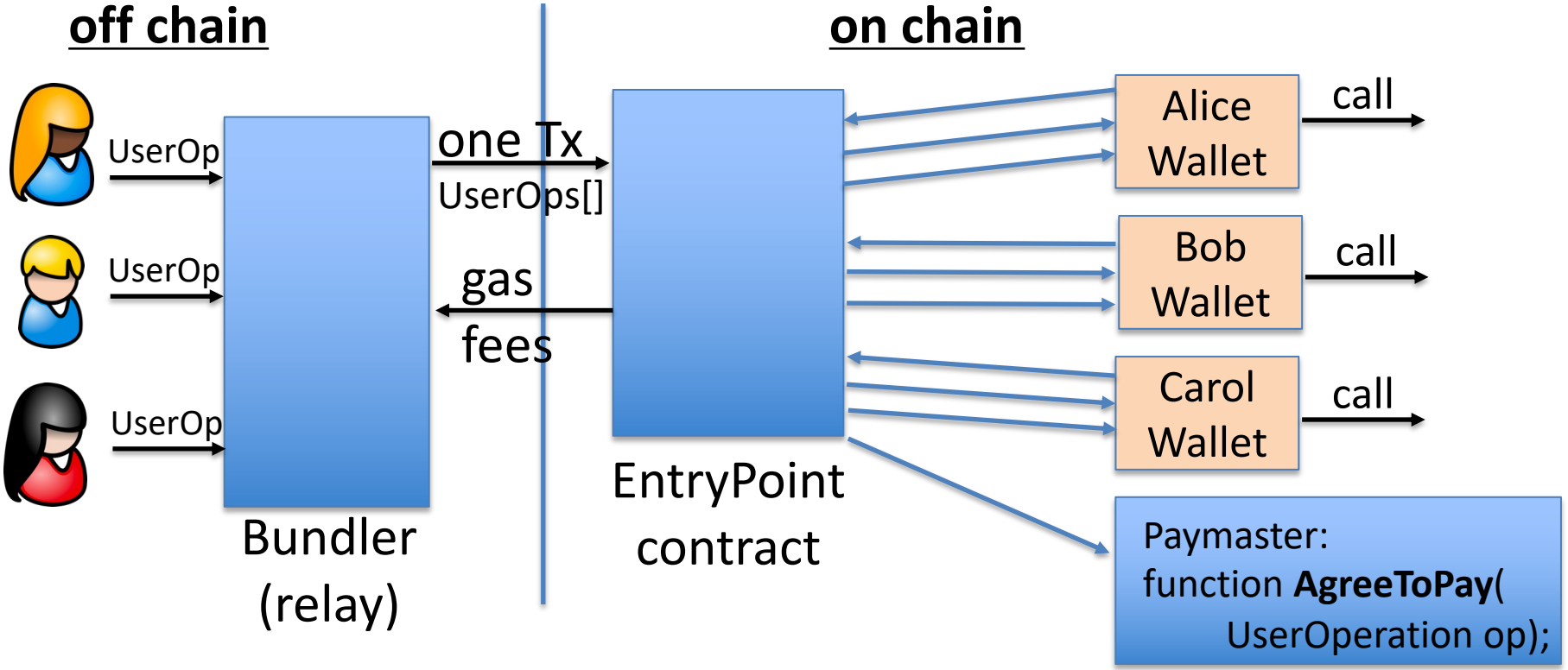
2. Before issuing Tx, Bundler needs to run **validateOp** in its head (a.k.a simulates validateOp) to ensure that UserOp is valid

   ⇒ need to ensure that simulation and on-chain execution of **validateOp** produce the same output

Solution: **validateOp** is restricted in what EVM opcodes it can use

[**execOp** is not restricted because Wallet is paying gas fees]

Bundling and Paymasters



As usual, Users pay Bundler a fee to be placed in Tx

Very complex … is this real?

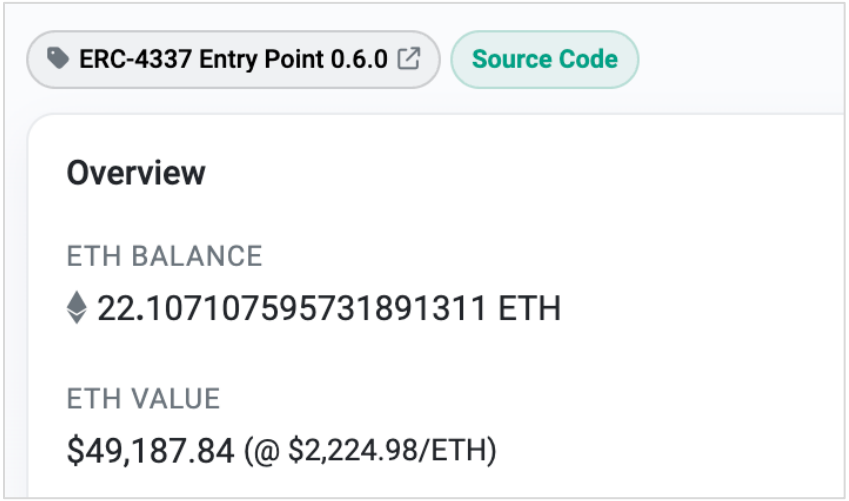
- Bundler's already exists, such as Pimlico

- Simple Tx now consume a lot of gas:
  - Simple ETH transfer from Alice⇾Bob: 4X more expensive
  - ERC20 transfer from Alice⇾Bob: 2X more expensive
  - Recall: gas on L2 is not as expensive as gas on Ethereum

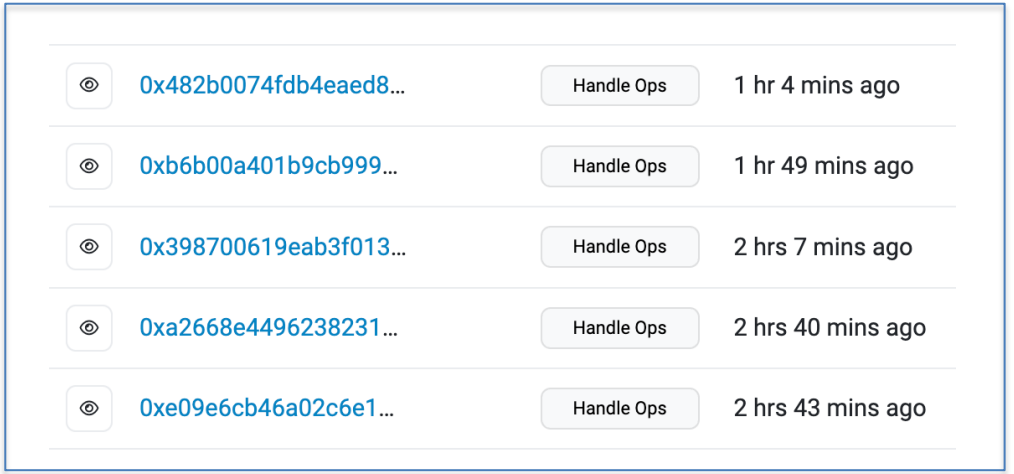
Main reason for adoption:

- Can be deployed now with no changes to dApps !!

An example



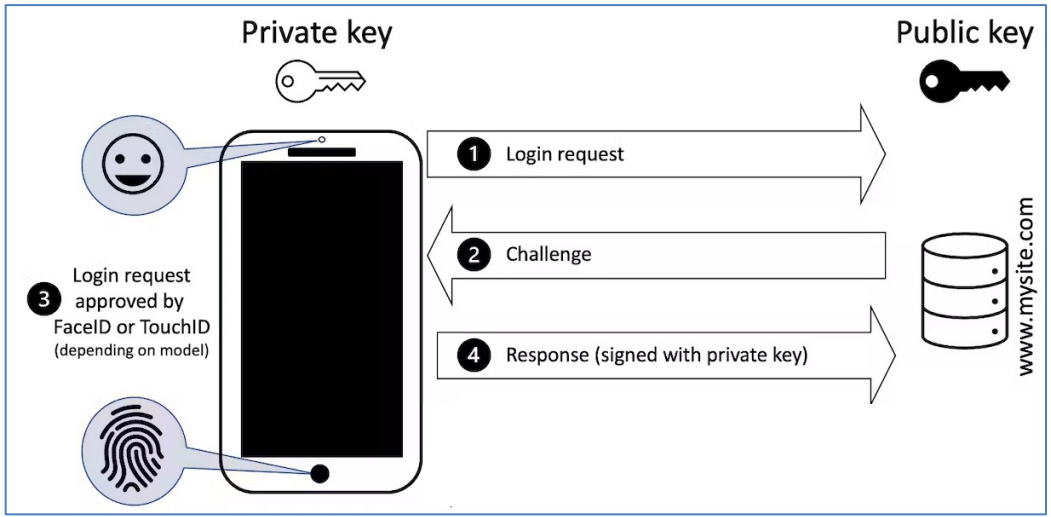
The EntryPoint contract on Ethereum



Some Tx from bundlers

An interesting application of AA: Passkey Tx

User authenticates Tx to AA wallet using TouchID



Good user experience:

- Signing key is backed up to iCloud
- Signing key is available on all user devices

The challenge: Passkeys uses sig. scheme not supported in the EVM

⇒ need to use a SNARK to prove valid sig to on-chain wallet.

Blockchain Governance (DAOs)

Decentralized orgs (DAO)

What is a DAO?

- A Dapp deployed on-chain at a specific address
- Anyone (globally) can send funds to DAO treasury

- Anyone can submit a proposal to DAO

  $\Longrightarrow$ participants vote

  $\Longrightarrow$ approved $\rightarrow$ proposal executes

snapshot.org

(SafeSnap: trustless on-chain execution of off-chain votes)

Examples of DAOs

There are currently about 6500 DAOs managed on Snapshot

- **Collector DAOs**: PleasrDAO, flamingoDAO, ConstitutionDAO, …

  (see art collection at https://gallery.so/pleasrdao)

PleasrDAO: 103 members.

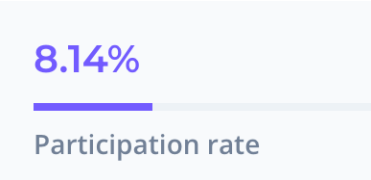
- Manages a treasury, has full time employees.
- Deliberations over what to acquire over telegram.

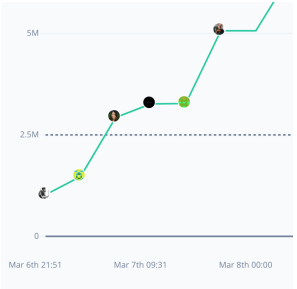
Examples of DAOs

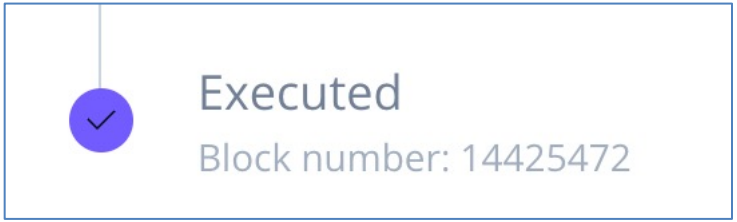
There are currently about 6500 DAOs managed on Snapshot

- Collector DAOs: PleasrDAO, flamingoDAO, ConstitutionDAO, …
- **Charity DAO**: gitcoin (42K members), …

Proposal ID 21: This proposal looks to ratify the allocation of 30,000 GTC from the Community Treasury to the MMM workstream.







(tally.com)

Examples of DAOs

There are currently about 6500 DAOs managed on Snapshot

- Collector DAOs: PleasrDAO, flamingoDAO, ConstitutionDAO, …
- Charity DAO: gitcoin, …
- **Protocol DAO**: manages operation of a specific protocol
  Uniswap DAO (29K members), Compound (4K members), …
- **Social DAO**: FWB, …
- **Investment DAO**: many

Example: Uniswap proposals

Add 1 Basis Point Fee Tier executed

TLDR: Uniswap should add a 1bps fee tier with 1 tick spacing. This change is straightforward from a

Upgrade Governance Contract to Compound's Governor Bravo executed

Previous Discussion: [Temperature Check](https://gov.uniswap.org/t/temperature-check-upgrade-gove...

Community-Enabled Analytics canceled

Past discussion: [Temperature Check](https://gov.uniswap.org/t/temperature-check-larger-grant-pro

DeFi Education Fund executed

(Previously known as: DeFi Political Defense Fund) Past discussion: [Temperature Check](http

Reduce the UNI proposal submission threshold to 2.5M executed

This proposal lowers the UNI proposal submission threshold from 10M UNI to 2.5M UNI. Uniswap's gove

Many DAO governance experiments

Who can vote? How to vote? What voting mechanism?

Lightspeed Democracy: What web3 organizations can learn from the history of governance

by Andrew Hall and Porter Smith

June 29, 2022

DAOs: a platform for experimenting with governance mechanisms

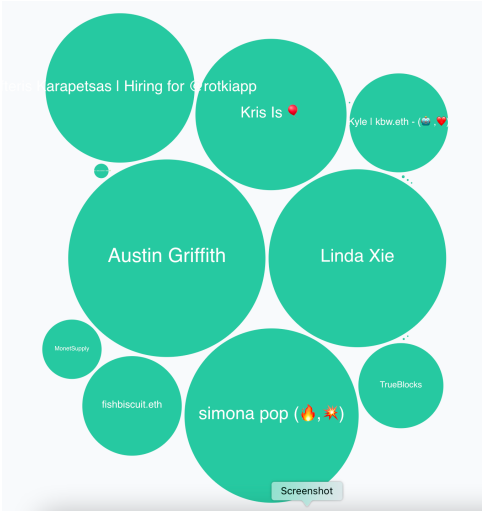
Governance methods

One token one vote: (most common)

- Members receive tokens based on their contribution.
- Everyone can vote.

Frequently implemented using one of
OpenZeppelin's Governor contracts (Solidity code)

_castVote(proposalID, voter, support, reason);



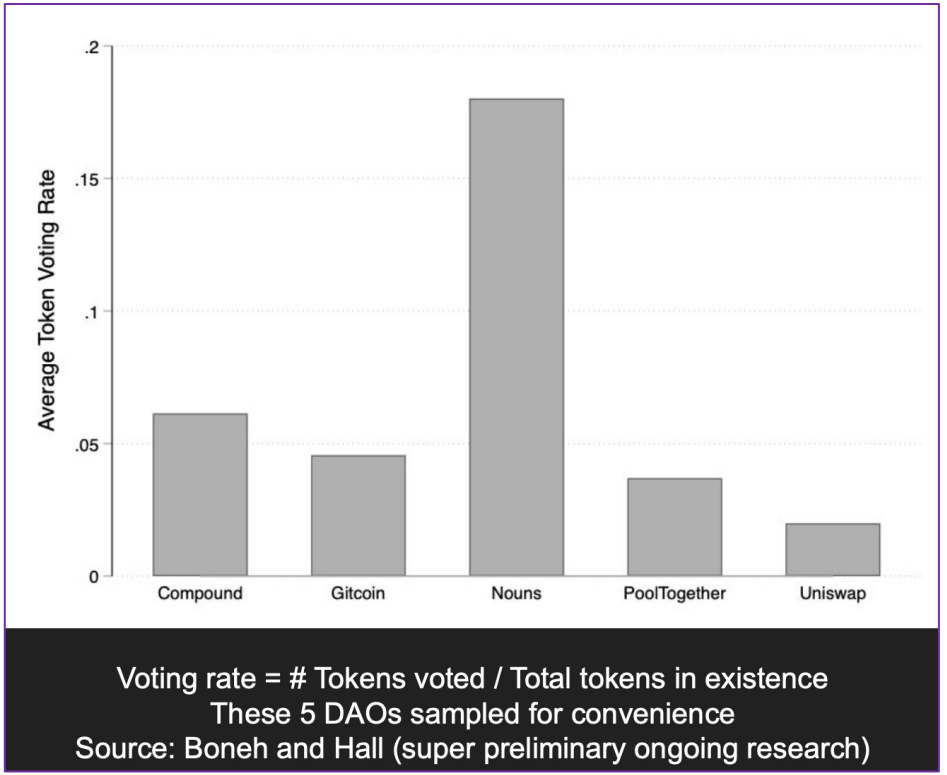
proposal 21 (tally.com)

Problem: direct democracy does not scale.

Poor participation rate

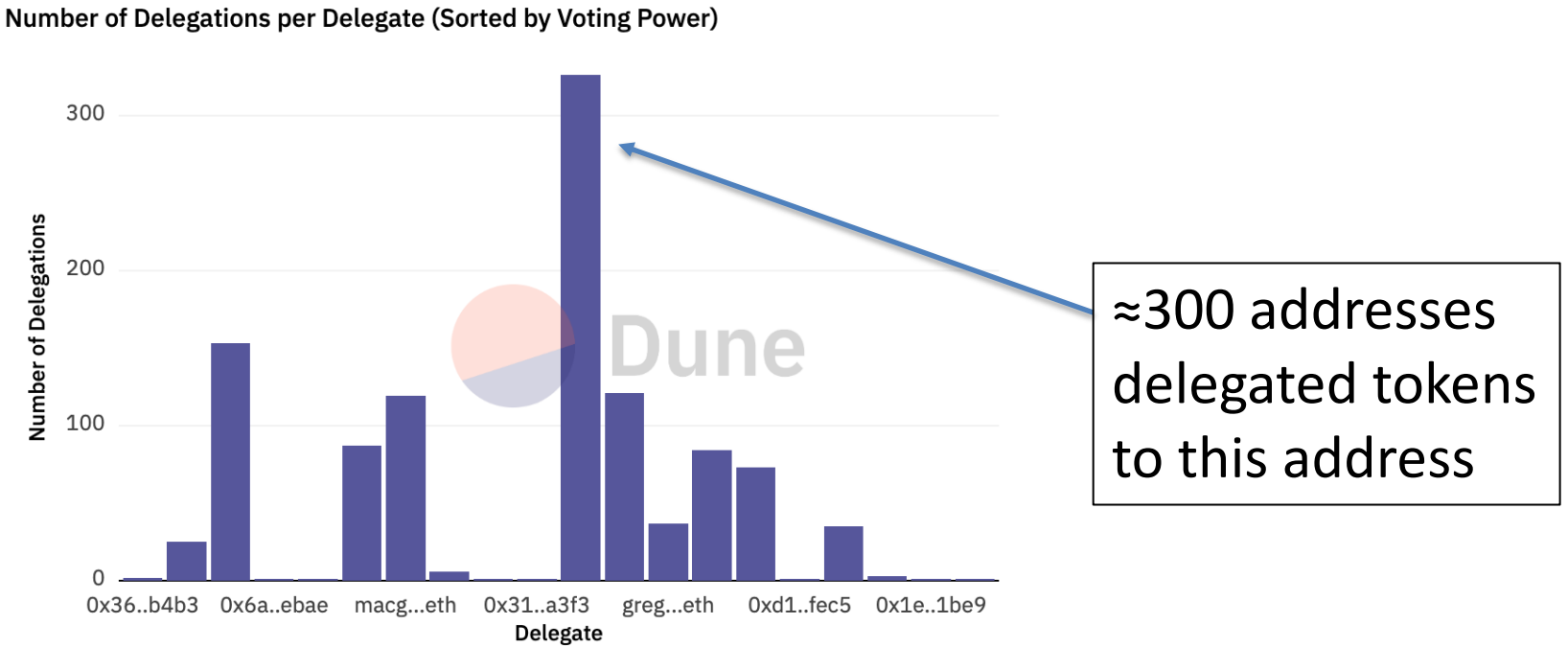
For all but one project:
participation rate < 5%

What to do? **delegation**
Supported in Governor contract



Voting rate = # Tokens voted / Total tokens in existence
These 5 DAOs sampled for convenience
Source: Boneh and Hall (super preliminary ongoing research)

Delegation example: element



Private DAO treasury

2021: an auction for a physical copy of the constitution.

(Sotheby's auction house)

ConstitutionDAO:

- Formed in Nov. 2021 to participate in auction.
- Raised $46.3M from about 20K participants worldwide
- Lost to another bidder who bid $43M

bidder knew that ConstitutionDAO could not outbid it

How to participate in an auction when everyone knows your treasury??

Private DAO treasury

[Dunaif, Boneh, 2021]

The design:

One DAO platform manages many DAOs:
a single Ethereum contract (e.g., JuiceBox)

DAO manager: sets up a DAO by publishing a DAO public key (pk)

Contributor: sends funds to platform with a "blinded DAO-pk"

Contract records contribution

⇒ an observer learns nothing about which DAO received the funds
⇒ only learns total amount stored on the platform as a whole

DAO manager can later use its secret key to claim funds sent to its DAO

Many other DAO privacy questions …

- **Private DAO participation**: keep participant list private
- **Private voting**: keep who voted how on each proposal private
- **Private delegations**

… while complying with all relevant laws.

Some of these questions are solved by
general privacy platforms such as **Aztec**, **Aleo**, and others.

Governance failures

DAO hacks:

- Attack on Beanstalk governance: $182M
- Attack on Yam Finance governance: thwarted
- Attack on Build Finance governance $1.5M
- Attack on Mango governance: $115M
- Steem governance drama

What happened? What can we learn?

(1) Beanstalk (Apr. 2022)

An Ethereum-based stablecoin **Bean**. Governance token **Stalk.**

The problem:

- an attacker can buy Stalk tokens,
- submit a proposal,
- vote on proposal, and
- have proposal execute

all in a single transaction

Beanstalk: the attack

Review: **flashloan** (has many applications)

- a loan that is taken out and repaid back in a single transaction
- No risk to lender ⇒ unbounded amount can be borrowed

The attack:

- Attacker took a huge flashloan from Aave, bought lots of Stalk,
- Passed a proposal to pay $80M to the attacker from the treasury,
- Sold the Stalk, and repaid the flashloan,
- Sent the proceeds to Tornado cash (donated $250K to Ukraine)

The lesson

The protocol relaunched four months later, and is still around.

The lesson: governance requires delays

- Require token holders to hold token for a long period of time
- Build a delay between proposal genesis, voting, and execution

(2) Yam Finance (July 2022)

- A DeFi project running on Ethereum. Governed token **YAM**.

What happened? (within a single day)

- Attacker bought 224739 YAM with borrowed funds
- Attacker submitted a governance proposal
  granting it control of project reserves ($3M USD)
- Voted on proposal with borrowed 224739 YAM, hitting quorum
- Borrowed YAM exchanged for Eth and paid back

What happened next?

Yam Finance team: noticed the proposal shortly after it hit quorum

⇒ retained **cancel** power, and canceled the proposal.

Lesson:

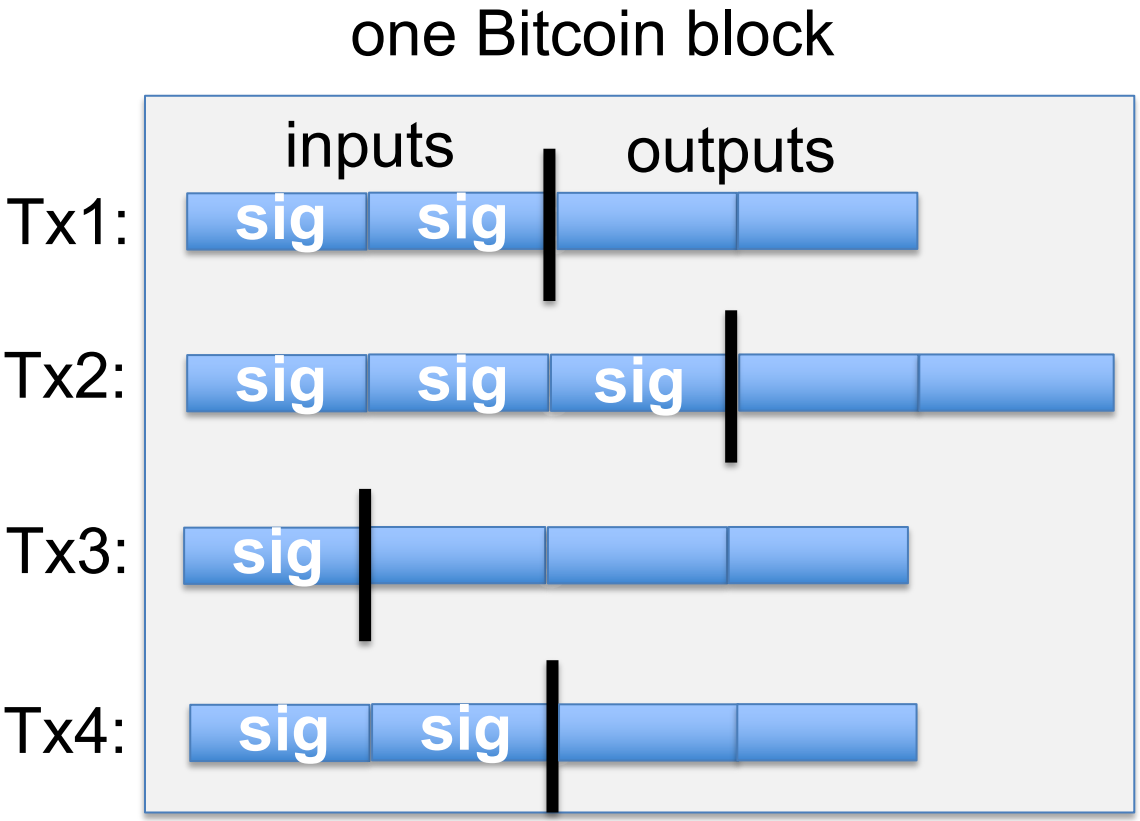
- Illustrates the positive power of a veto
- The problem: power can never be taken away
  (unless voluntarily given away)

END OF LECTURE

Next lecture: super cool final guest lecture

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
- $H: M \times G \rightarrow G$ a hash function (e.g., based on SHA256)

BLS Signatures

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

output sk = α , pk = g^{α} $\in$ G

Sign(sk, m): output sig = $H(m, \mathrm{pk})^{\alpha}$ $\in$ G

Verify(pk, m, sig): output accept if $\log_g(\mathrm{pk}) = \log_{H(m,\mathrm{pk})}(\mathrm{sig})$

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

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

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

if and only if $e(g, sig) = e(pk, H(m,pk))$

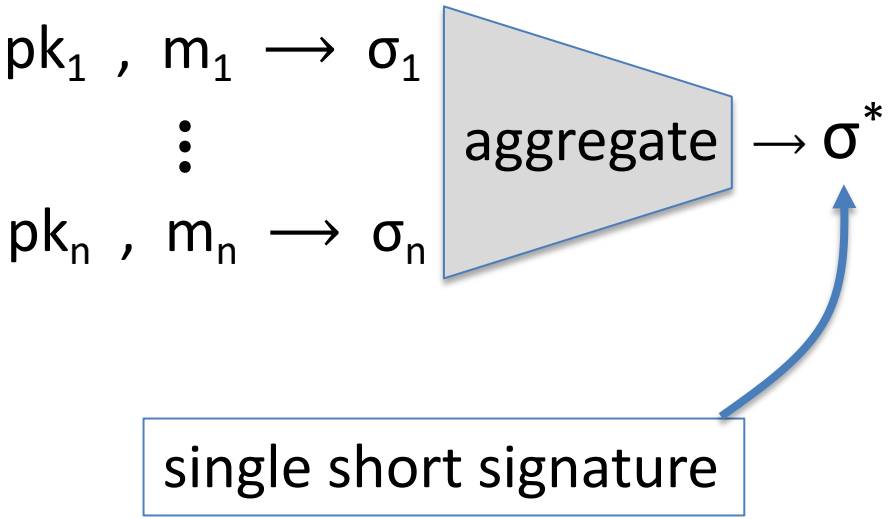
$\|$ $\|$

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

verify test

Properties: signature aggregation [BGLS'03]

Anyone can compress n signatures into one



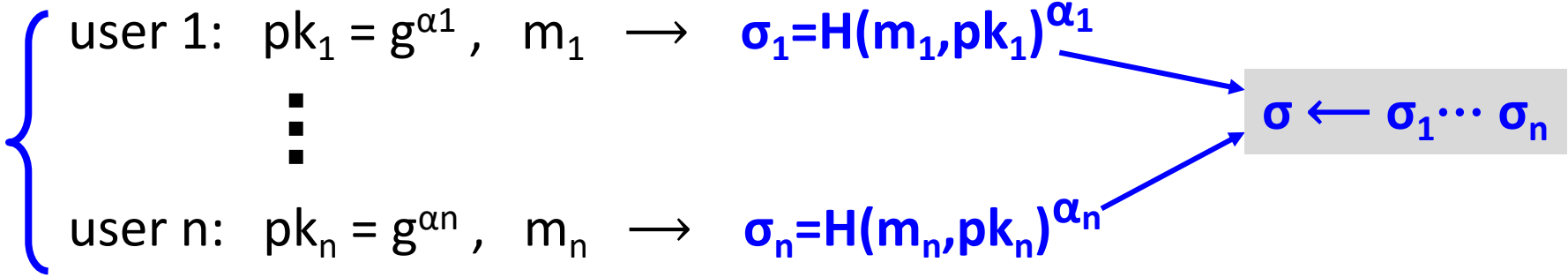
Verify($\overline{\mathbf{pk}}$, $\overline{\mathbf{m}}$, σ^*) = "accept"

convinces verifier that

for i=1,…,n:

user i signed msg m_i

Aggregation: how

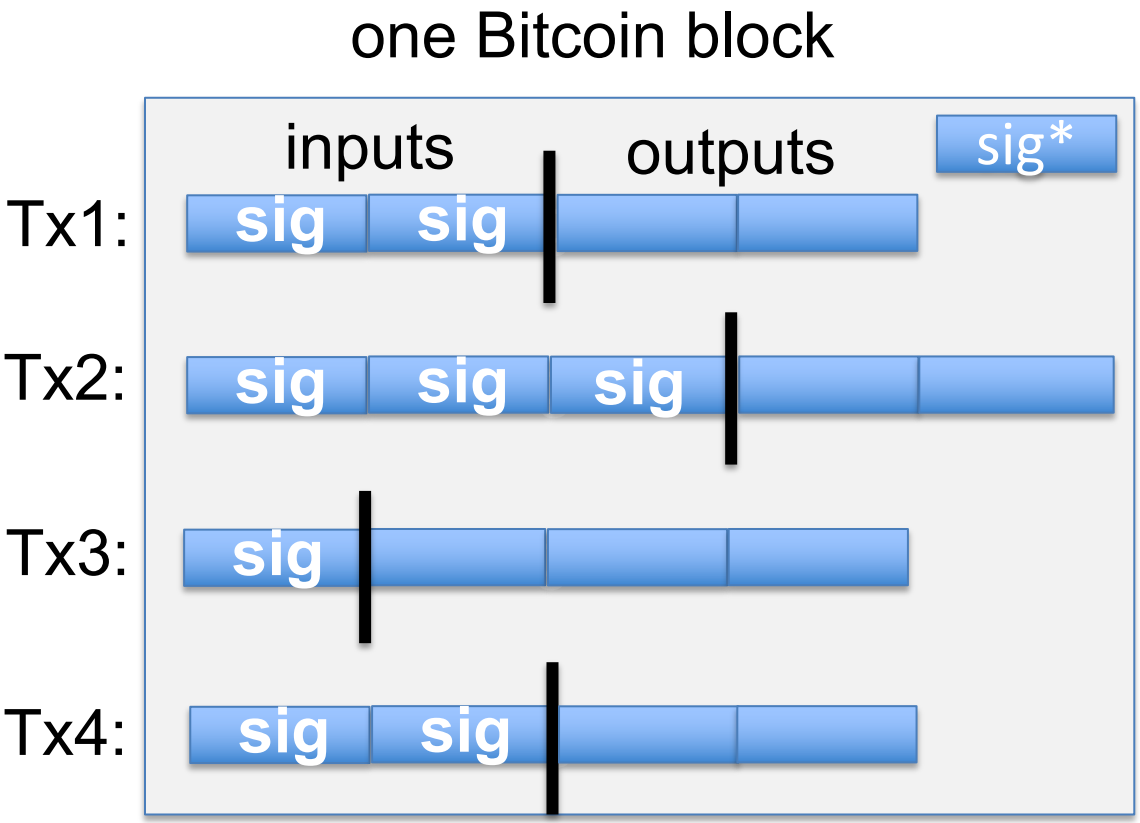


Verifying an aggregate signature: (incomplete)

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

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

Compressing the blockchain with BLS



if needed:

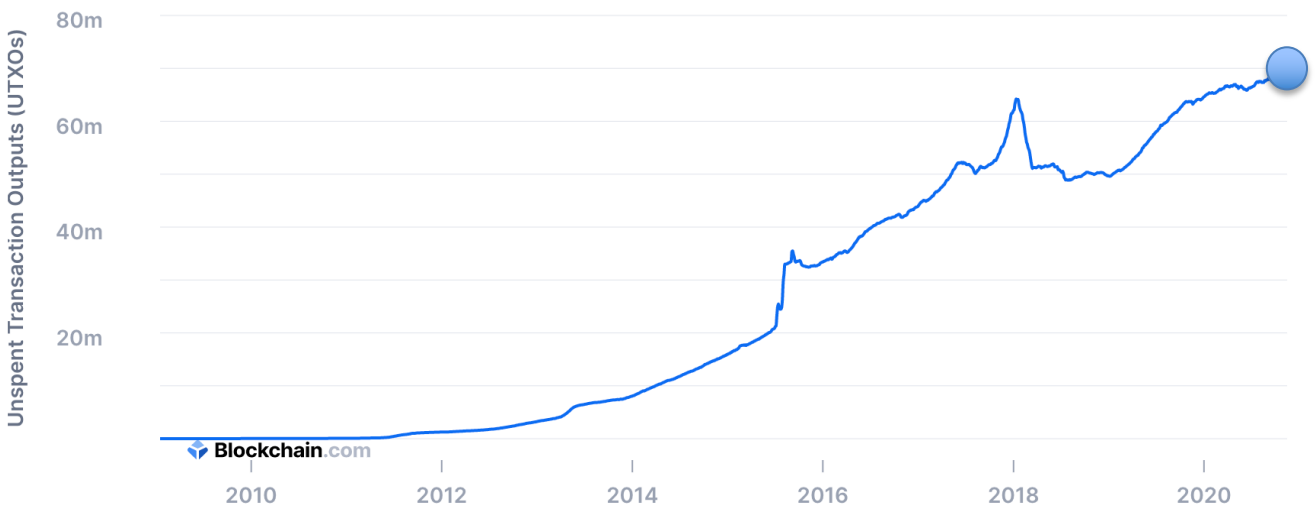
compress all signatures in a block into a single aggregate signatures

⇒ shrink block

or: aggregate in smaller batches

Reducing Miner State

UTXO set size



≈70M UTXOs

Miners need to keep all UTXOs in memory to validate Txs

Can we do better?

Recall: polynomial commitments

- *commit*(*pp*, f, r) $\rightarrow$ $\boldsymbol{com_f}$ commitment to f $\in \mathbb{F}_p^{(\leq d)}[X]$

- *eval*: goal: for a given $\boldsymbol{com_f}$ and x, y $\in \mathbb{F}_p$,

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

Homomorphic polynomial commitment

A polynomial commitment is **homomorphic** if

there are efficient algorithms such that:

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

Then:

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

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

Committing to a set (of UTXOs)

Let $S = \{U_1, \ldots, Un\} \in \mathbb{F}_p$ be a set of UTXOs **(accumulator)**

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: $\boldsymbol{com_f} \rightarrow \boldsymbol{com_{X*f-U*f}}$ ⇐ 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

} ≤ 1KB

com_T, com_S

Tx format:

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

  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

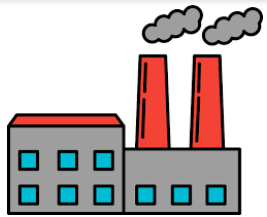
⇒ user needs large memory and compute time

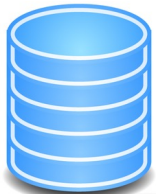
⇒ … can be outsourced to an untrusted 3rd party

UTXO U , fee

proof π

spend U





polynomials S and T

The proof factory

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 here …

Bitcoin's long-anticipated Taproot upgrade is activated

November 14, 2021, 12:49AM EST · 1 min read

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} $\in$ G for α in $\{1, \ldots, q\}$

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

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

  $\Rightarrow$ an interactive protocol between Alice and Bob

  (note: much simpler with BLS)

  $\Rightarrow$ 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

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

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