CS251 Fall 2021

(cs251.stanford.edu)

Using zk-SNARKs for Privacy on the Blockchain

Dan Boneh

Can we have private transactions on a public blockchain?

Naïve reasoning:

universal verifiability ⇒ transaction data must be public.

otherwise, how we can the public verify Tx ??

Goal for this lecture:

crypto magic ⇒ private Tx on a publicly verifiable blockchain

Crypto tools: **commitments** and **zero knowledge proofs**

The need for privacy in the financial system

Supply chain privacy:

A car company does not want to reveal
how much it pays its supplier for tires, wipers, etc.

Payment privacy:

- A company that pays its employees in crypto needs to keep list of employees and their salaries private.
- Privacy for rent, donations, purchases

Business logic privacy: Can the code of a smart contract be private?

Last lecture

Neither Bitcoin nor Ethereum are private

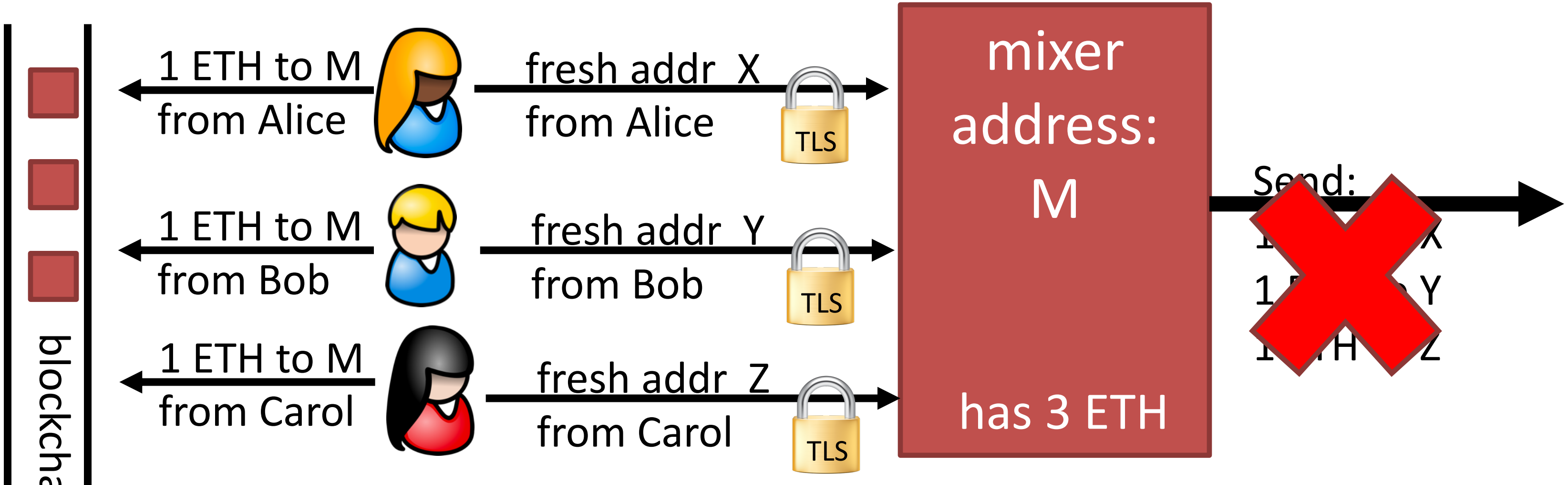
etherscan.io:

Address 0x1654b0c3f62902d7A86237…

Balance:	1.114479450024297906 Ether
Ether Value:	$4,286.34 (@ $3,846.05/ETH)

Txn Hash	Method ⓘ	Block
0x0269eff8b4196558c07…	Set Approval For...	13426561
0xa3dacb0e7c579a99cd…	Cancel Order_	13397993
0x73785abcc7ccf030d6a…	Set Approval For...	13387834
0x1463293c495069d61c…	Atomic Match_	13387703

Simple blockchain anonymity via mixing



Observer knows Y belongs to one of {Alice, Bob, Carol} but does not know which one

$\Longrightarrow$ anonymity set of size 3.

$\Longrightarrow$ Bob can mix again with different parties to increase anonymity set.

Problems: (i) mixer knows all, (ii) mixer can abscond with 3 ETH !!

Mixing without a mixer? on Bitcoin: **CoinJoin** (e.g., Wasabi), on Ethereum: **Tornado cash**

What is a zk-SNARK?

An central tool for
privacy on the blockchain

zk-SNARK: Blockchain Applications

Private Tx on a public blockchain:

- Confidential transactions
- Tornado cash, Zcash, IronFish
- Private Dapps: Aleo

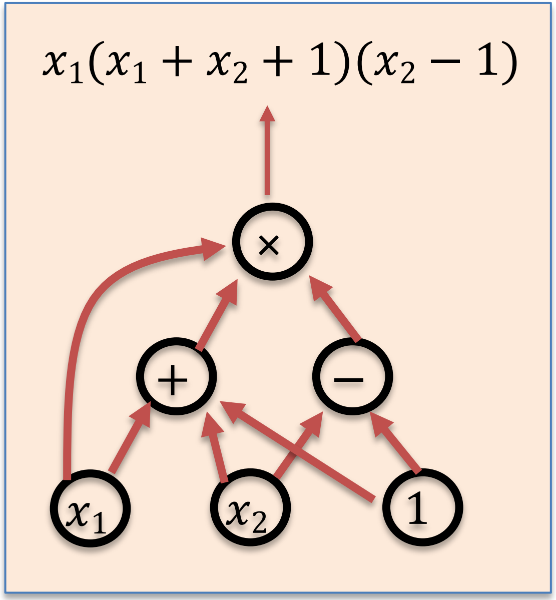
Compliance:

- Proving solvency in zero-knowledge
- Zero-knowledge taxes

Scalability: privacy in zk-SNARK Rollup (next week)

(1) arithmetic circuits

- Fix a finite field $\mathbb{F} = \{0, \dots, p-1\}$ for some prime p>2.
- **Arithmetic circuit:** $C\colon \mathbb{F}^n \to \mathbb{F}$
  - directed acyclic graph (DAG) where
    internal nodes are labeled +, −, or ×
    inputs are labeled $1, x_1, \dots, x_n$
  - defines an n-variate polynomial with an evaluation recipe
- $|C| = \#$ gates in C



Interesting arithmetic circuits

<u>Examples</u>:

- $C_{hash}(h, \mathbf{m})$: outputs 0 if SHA256($\mathbf{m}$) = h , and ≠0 otherwise

  $C_{hash}(h, \mathbf{m}) = (h - \text{SHA256}(\mathbf{m}))$, $|C_{hash}| \approx$ 20K gates

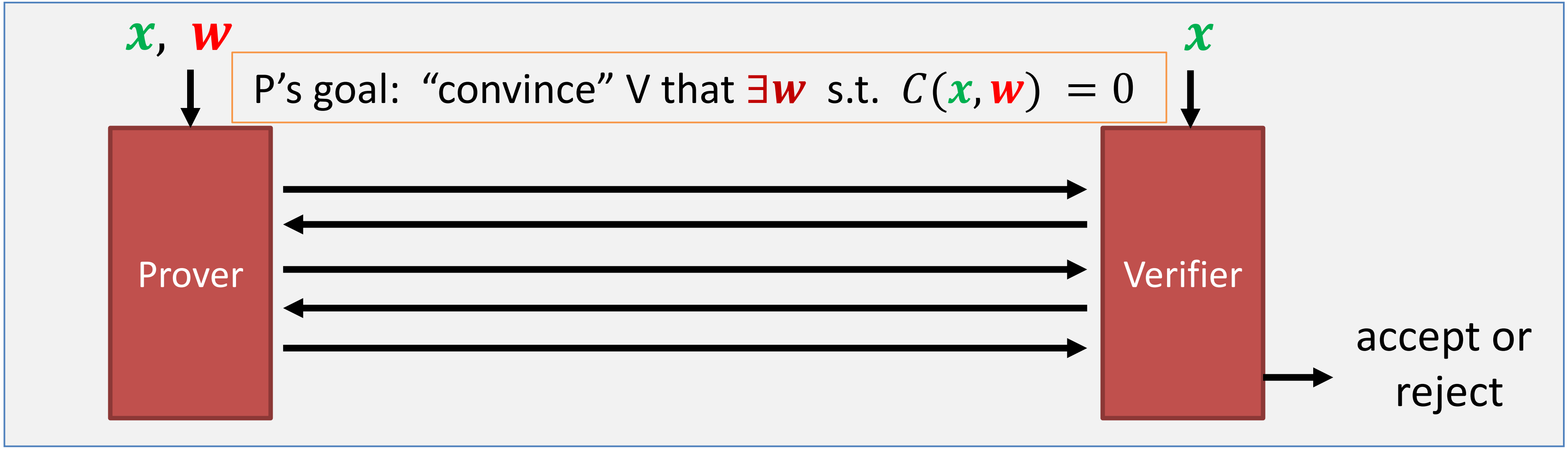
- $C_{sig}(pk, m, \sigma)$: outputs 0 if σ is a valid ECDSA signature on m with respect to pk

(2) Argument systems (for NP)

Public arithmetic circuit: $C(x, w) \rightarrow \mathbb{F}$

public statement in $\mathbb{F}^n$ → x

secret witness in $\mathbb{F}^m$ → w



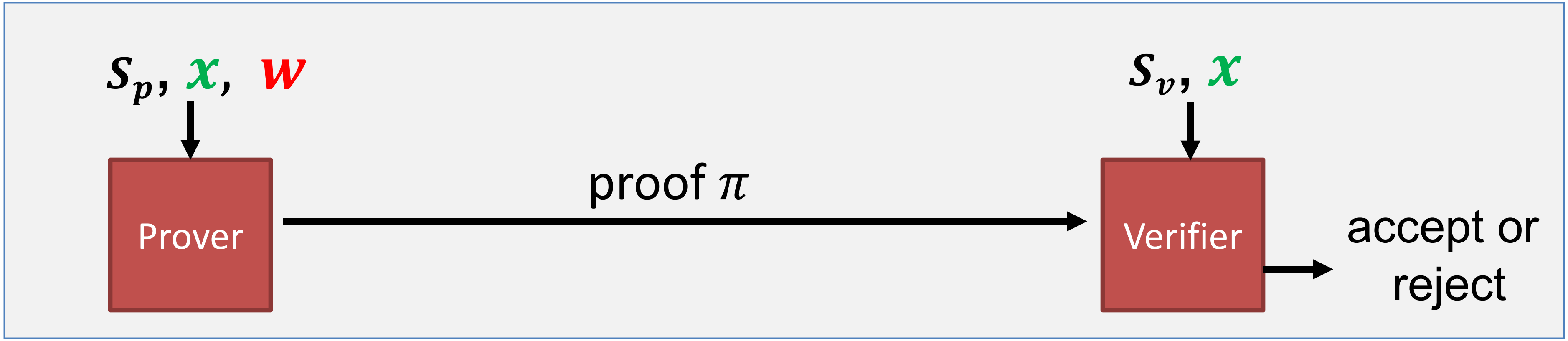
(non-interactive) Preprocessing argument systems

Public arithmetic circuit: $C(\boldsymbol{x}, \boldsymbol{w}) \rightarrow \mathbb{F}$

public statement in $\mathbb{F}^n$ → $\boldsymbol{x}$

secret witness in $\mathbb{F}^m$ → $\boldsymbol{w}$

Preprocessing (setup): $\mathbf{S}(C) \rightarrow$ public parameters $(\boldsymbol{S_p}, \boldsymbol{S_v})$



Preprocessing argument System

A **preprocessing argument system** is a triple (S, P, V):

- **S**(C) → public parameters (S_p, S_v) for prover and verifier
- **P**$(S_p, \boldsymbol{x}, \boldsymbol{w})$ → proof π
- **V**$(S_v, \boldsymbol{x}, \boldsymbol{\pi})$ → accept or reject

Argument system: requirements (informal)

Prover P(S_p, $\boldsymbol{x}$, $\boldsymbol{w}$) — Verifier V (S_v, $\boldsymbol{x}$, $\boldsymbol{\pi}$)

proof π →

accept or reject

Complete: $\forall x, w$: $C(\boldsymbol{x}, \boldsymbol{w}) = 0 \Rightarrow$ Pr[V(S_v, x, P(S_p, $\boldsymbol{x}$, $\boldsymbol{w}$)) = accept] = 1

Knowledge sound: V accepts $\Rightarrow$ P "knows" $\boldsymbol{w}$ s.t. $C(\boldsymbol{x}, \boldsymbol{w}) = 0$

P* does not "know" $\boldsymbol{w}$ $\Rightarrow$ Pr[V(S_v, x, π) = accept] < negligible

Optional: **Zero knowledge**: $(S_v, \boldsymbol{x}, \pi)$ "reveals nothing" about $\boldsymbol{w}$

SNARK: a Succinct ARgument of Knowledge

A **succinct preprocessing argument system** is a triple (S, P, V):

- **S**(C) → public parameters (S_p, S_v) for prover and verifier
- **P**(S_p, $\boldsymbol{x}$, $\boldsymbol{w}$) → **short** proof π ; $|\pi| = O(\log(|C|), \lambda)$
- **V**(S_v, $\boldsymbol{x}$, $\boldsymbol{\pi}$) **fast to verify** ; time(V) = $O(|x|, \log(|C|), \lambda)$

short "summary" of circuit

Why preprocess C??

SNARK: a Succinct ARgument of Knowledge

A **succinct preprocessing argument system** is a triple (S, P, V):

- **S**(C) → public parameters (S_p, S_v) for prover and verifier
- **P**$(S_p, \boldsymbol{x}, \boldsymbol{w})$ → **short** proof π ; $|\pi| = O(\,\mathbf{log}(|C|)\,,\ \lambda)$
- **V**$(S_v, \boldsymbol{x}, \boldsymbol{\pi})$ **fast to verify** ; time(V) = $O(|x|,\ \mathbf{log}(|C|)\,,\ \lambda)$

SNARK: (S, P, V) is **complete**, **knowledge sound**, and **succinct**

zk-SNARK: (S, P, V) is a SNARK and is **zero knowledge**

The trivial argument system

(a) Prover sends $\boldsymbol{w}$ to verifier,

(b) Verifier checks if $C(\boldsymbol{x}, \boldsymbol{w}) = 0$ and accepts if so.

Problems with this:

(1) $\boldsymbol{w}$ might be secret: prover does not want to reveal $\boldsymbol{w}$ to verifier

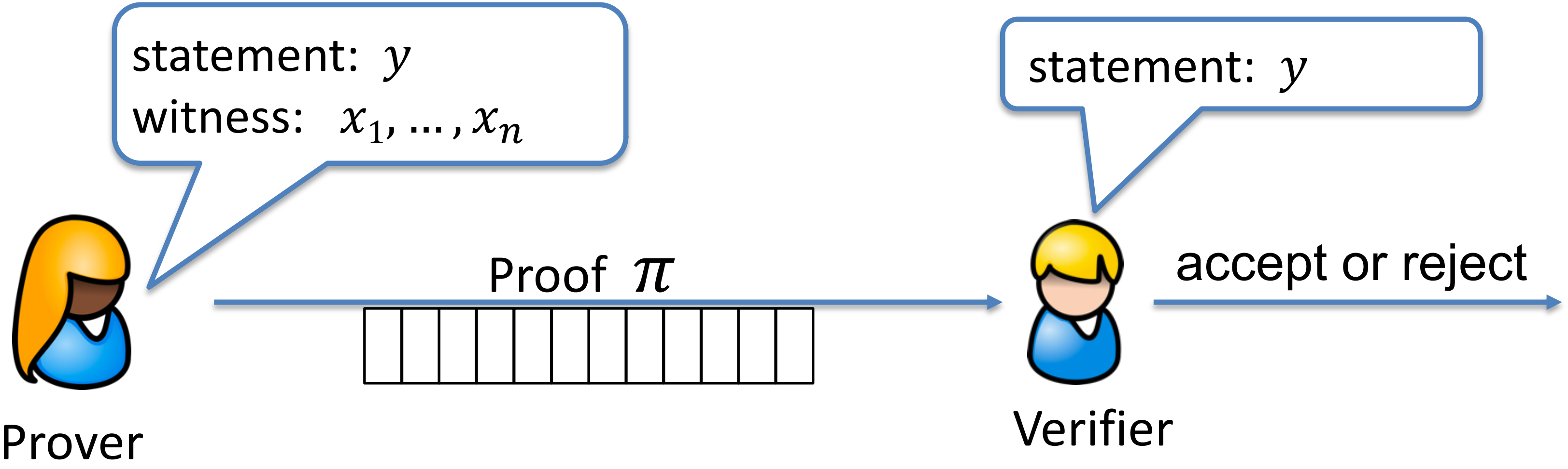
(2) $\boldsymbol{w}$ might be long: we want a "short" proof

(3) computing $C(\boldsymbol{x}, \boldsymbol{w})$ may be hard: we want a "fast" verifier

An example

Prover: I know $(x_1, \ldots, x_n) \in X$ such that $H(x_1, \ldots, x_n) = y$

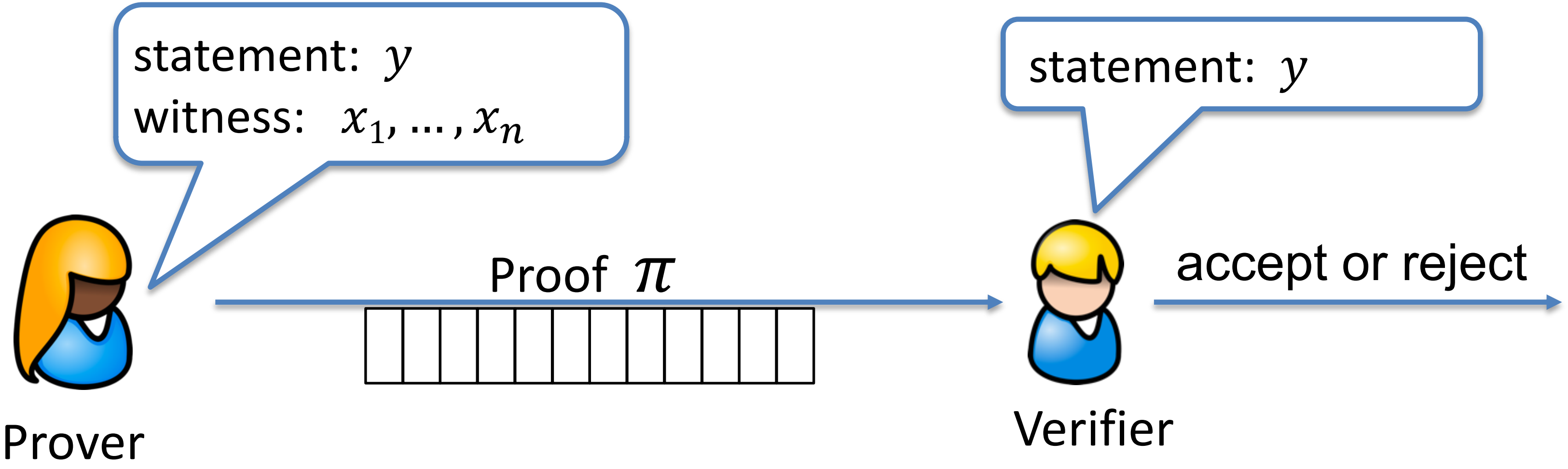
SNARK: $\text{size}(\pi)$ and $\text{VerifyTime}(\pi)$ is $O(\log n)$!!



An example

How is this possible ???

SNARK: $\text{size}(\pi)$ and $\text{VerifyTime}(\pi)$ is $O(\log n)$!!



Types of preprocessing Setup

Recall setup for circuit C: **S**$(C; r)$ → public parameters (S_p, S_v)

random bits

<u>Types of setup</u>:

better

trusted setup per circuit: **S**$(C; r)$ random r must be kept secret from prover

prover learns r ⇒ can prove false statements

trusted but universal (updatable) setup: secret r is independent of C

$\boldsymbol{S} = (S_{init}, S_{index})$: $S_{init}(\lambda; r) \rightarrow pp,$ $S_{index}(pp, C) \rightarrow (S_p, S_v)$

one-time

no secret data from prover

<u>transparent setup</u>: **S**(C) does not use secret data (no trusted setup)

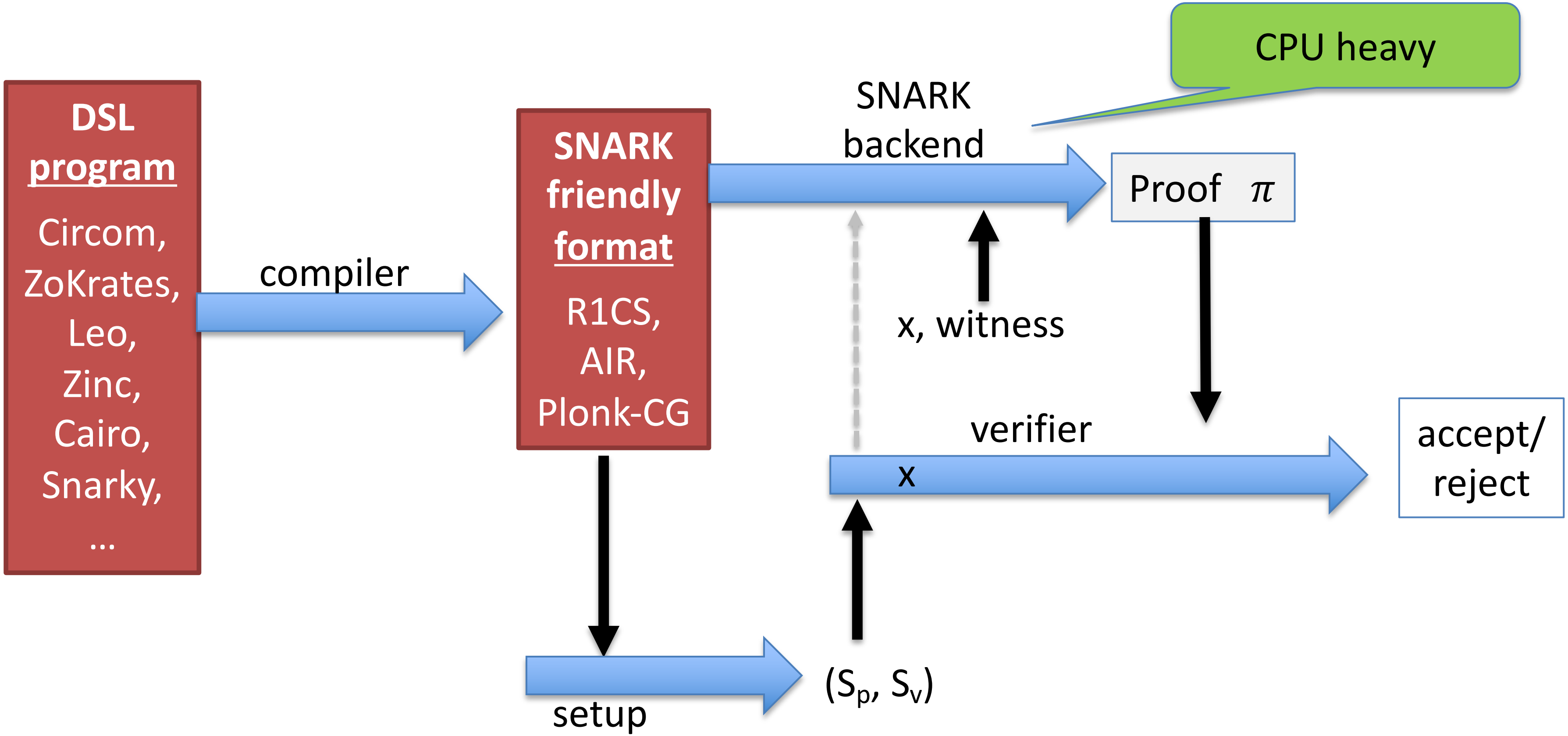
Significant progress in recent years

- **Kilian'92, Micali'94**: succinct transparent arguments from PCP
  - impractical prover time
- **GGPR'13**, **Groth'16, …**: linear prover time, **constant size proof** ($O_\lambda(1)$)
  - **trusted setup per circuit** (setup alg. uses secret randomness)
  - compromised setup ⇒ proofs of false statements
- **Sonic'19, Marlin'19, Plonk'19**, … : universal trusted setup
- **DARK'19, Halo'19, STARK**, … : no trusted setup (transparent)

Types of SNARKs (partial list)

	size of proof π	size of S_p (beyond C)	verifier time	trusted setup?
Groth'16	O(1)	O(|C|)	O(1)	yes/per circuit
Plonk/Marlin	O(1)	O(|C|)	O(1)	yes/universal
Bulletproofs	O(log|C|)	O(1)	O(|C|)	no
STARK	O(log|C|)	O(1)	O(log|C|)	no
DARK	O(log|C|)	O(1)	O(log|C|)	no
⋮	⋮			⋮

A SNARK software system



How to define “knowledge soundness” and “zero knowledge”?

Definitions: (1) knowledge sound

Goal: if V accepts then P "knows" $\boldsymbol{w}$ s.t. $C(\boldsymbol{x},\boldsymbol{w}) = 0$

What does it mean to "know" $\boldsymbol{w}$??

informal def: P knows $\boldsymbol{w}$, if $\boldsymbol{w}$ can be "extracted" from P

P

Definitions: (1) knowledge sound

Formally: (S, P, V) is **knowledge sound** for a circuit C if

for every poly. time adversary $A = (A_0, A_1)$ such that

$S(C) \rightarrow (S_p, S_v)$, $(x, st) \leftarrow A_0(S_p)$, $\pi \leftarrow A_1(S_p, x, st)$:

$\Pr[V(S_v, x, \pi) = \text{accept}] > 1/10^6$ (non-negligible)

there is an efficient **extractor** E (that uses A_1 as a black box) s.t.

$S(C) \rightarrow (S_p, S_v)$, $(x, st) \leftarrow A_0(S_p)$, $w \leftarrow E^{A_1(S_p, x, st)}(S_p, x)$:

$\Pr[C(x, w) = 0] > 1/10^6 - \epsilon$ (for a negligible ϵ)

Definitions: (2) Zero knowledge

A story about the lady sipping tea

Definitions: (2) Zero knowledge (against an honest verifier)

(S, P, V) is **zero knowledge** if for every $x \in \mathbb{F}^n$

proof π "reveals nothing" about ***w***, other than its existence

What does it mean to "reveal nothing" ??

Informal def: π "reveals nothing" about ***w*** if the verifier can generate π **by itself** $\Longrightarrow$ it learned nothing new from π

(S, P, V) is **zero knowledge** if there is an efficient alg. **Sim** s.t. $(S_p, S_v, \pi) \leftarrow$ ***Sim***(C, x) "look like" the real S_p, S_v and π.

Main point: ***Sim***$(C$,x) simulates π without knowledge of ***w***

(but also outputs S_p, S_v)

Definitions: (2) Zero knowledge (against an honest verifier)

Formally: (S, P, V) is (honest verifier) **zero knowledge** for a circuit C

if there is an efficient simulator ***Sim*** such that

for all $x \in \mathbb{F}^n$ s.t. $\exists w: C(x, w) = 0$ the distribution:

> (S_p, S_v, x, π): where $(S_p, S_v) \leftarrow S(C)$, $\pi \leftarrow P(S_p, x, \boldsymbol{w})$

is indistinguishable from the distribution:

> (S_p, S_v, x, π): where $(S_p, S_v, \pi) \leftarrow \textbf{\textit{Sim}}(C, x)$

How to build a zk-SNARK?

Recall: A zero knowledge preprocessing argument system.

Prover generates a **short** proof that is **fast** to verify

How to build a zk-SNARK ??

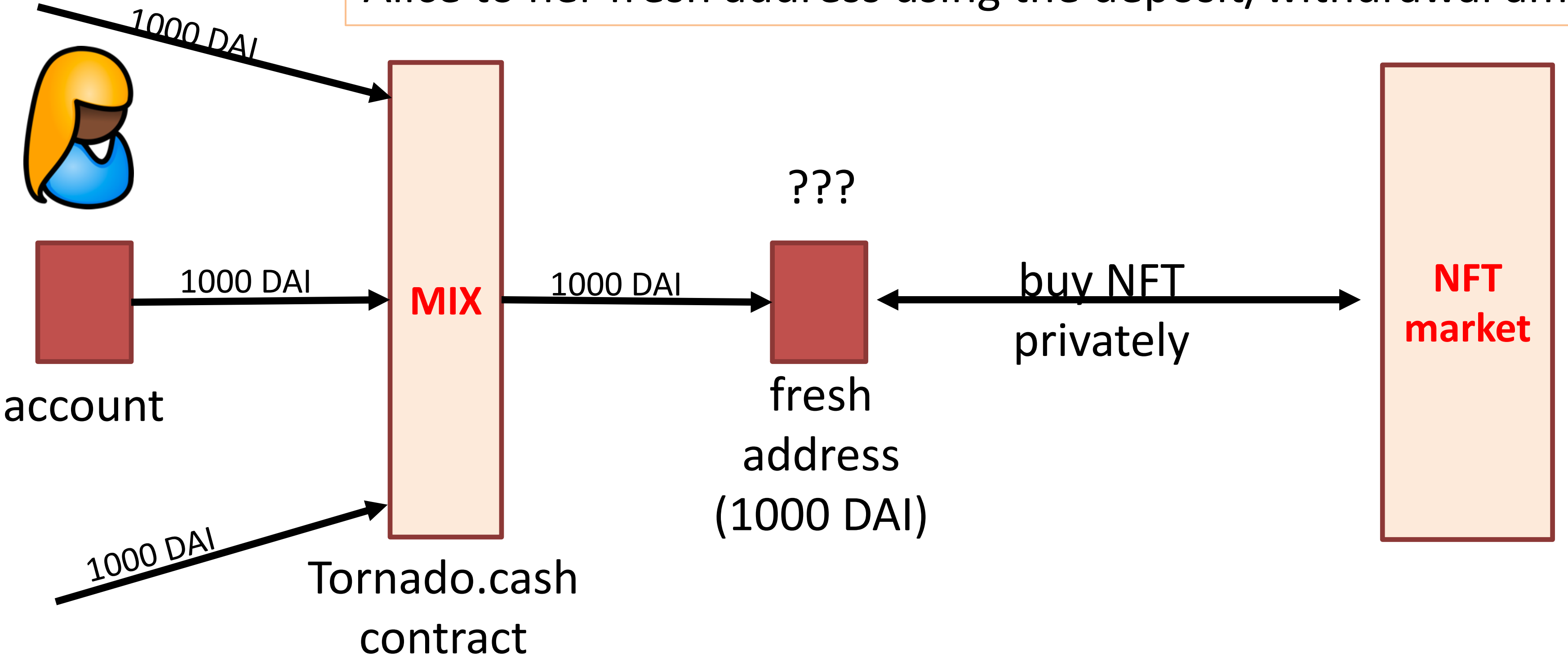
Next lecture

Tornado cash: a zk-based mixer

Launched on the Ethereum blockchain on May 2020 (v2)

Tornado Cash: a ZK-mixer

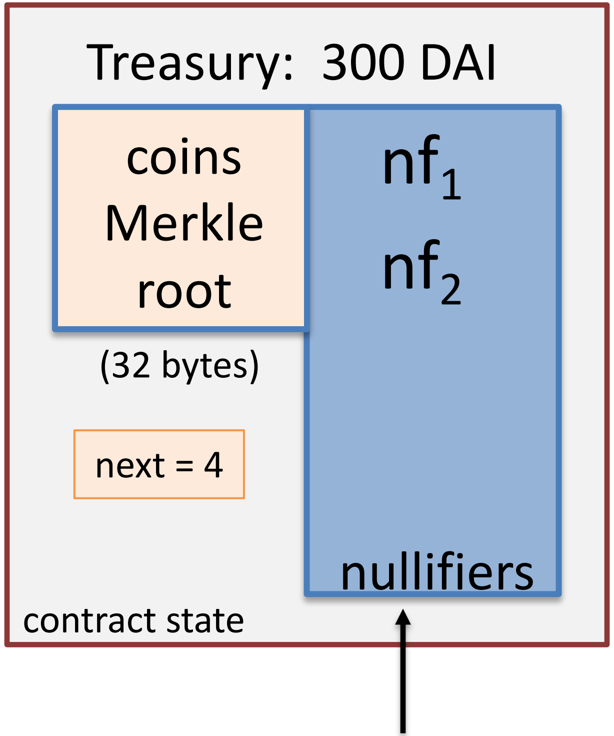
A common denomination (1000 DAI) is needed to prevent linking Alice to her fresh address using the deposit/withdrawal amount



The tornado cash contract (simplified)

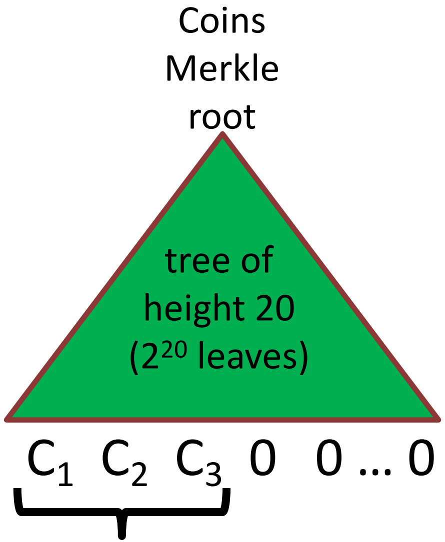
100 DAI pool:
each coin = 100 DAI

Currently:
- three coins in pool
- contract has 300 DAI
- two nullifiers stored



explicit list:
one entry per **spent coin**

H_1, H_2: $R \rightarrow \{0,1\}^{256}$ CRHF



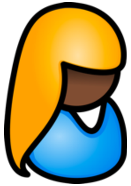
public list of coins

Tornado cash: deposit (simplified)

H_1, H_2: $R \to \{0,1\}^{256}$

100 DAI pool:
each coin = 100 DAI

Alice deposits 100 DAI:



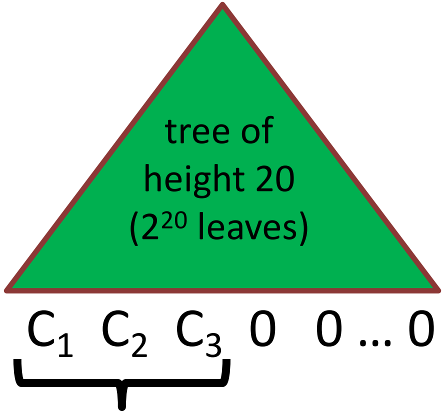
100 DAI →
C_4 , MerkleProof(4)

Treasury: 300 DAI

coins Merkle root	nf_1 nf_2 … nullifiers
(32 bytes)	
next = 4	

contract state

Coins
Merkle
root



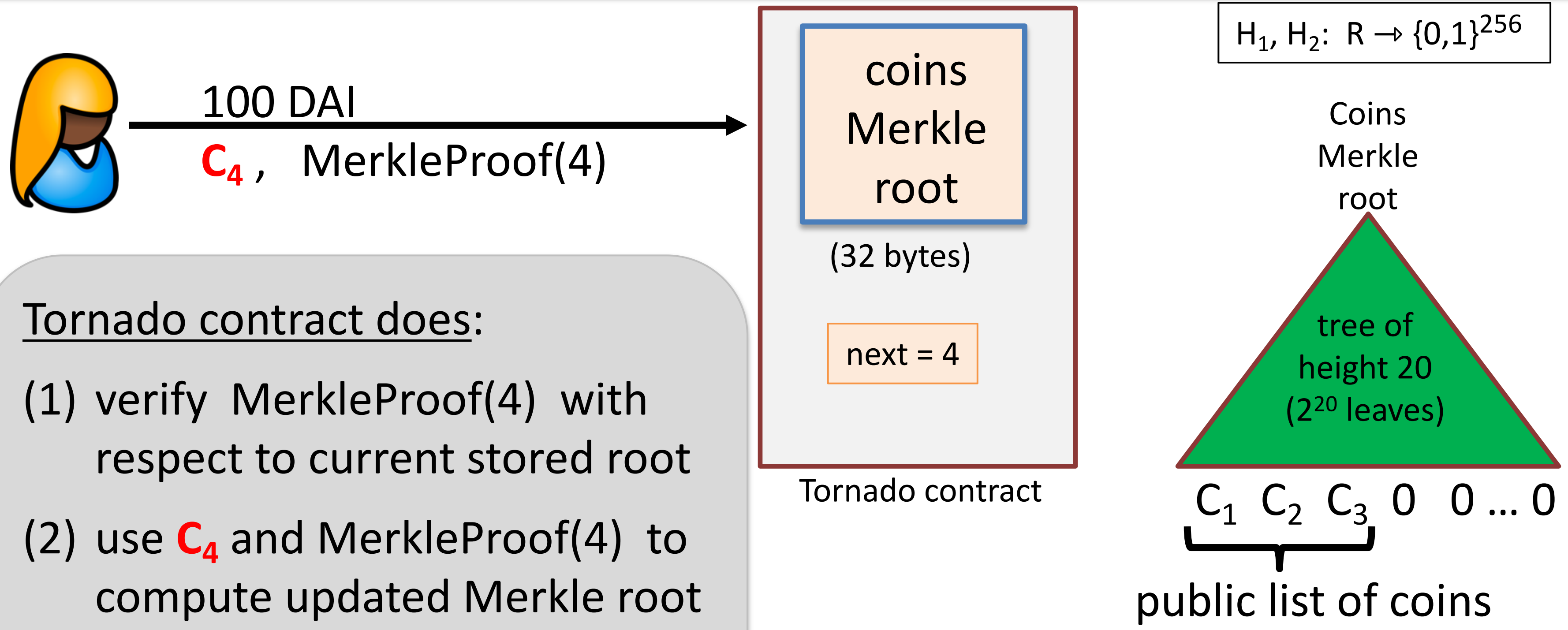
public list of coins

Build Merkle proof for leaf #4:
MerkleProof(4) (leaf=0)

choose random k, r in R
set $C_4 = H_1(k, r)$

explicit list:
one entry per **spent coin**

Tornado cash: deposit (simplified)



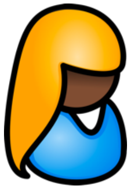
Tornado contract does:

(1) verify MerkleProof(4) with respect to current stored root

(2) use C_4 and MerkleProof(4) to compute updated Merkle root

(3) update state

Tornado cash: deposit (simplified)



100 DAI

C_4 , MerkleProof(4)

coins Merkle root

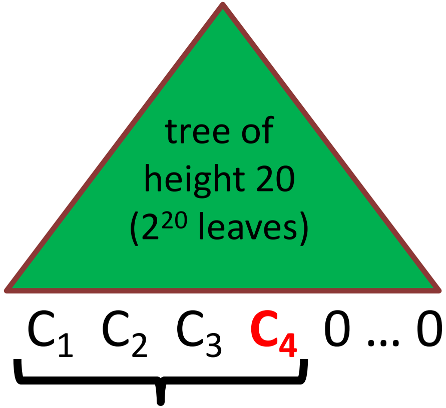
(32 bytes)

next = 4

Tornado contract

$H_1, H_2\colon R \rightarrow \{0,1\}^{256}$

updated Merkle root



public list of coins

Tornado contract does:

(1) verify MerkleProof(4) with respect to current stored root

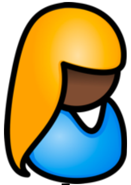
(2) use C_4 and MerkleProof(4) to compute updated Merkle root

(3) update state

Tornado cash: deposit (simplified)

100 DAI pool:
each coin = 100 DAI

Alice deposits 100 DAI:



100 DAI →
C_4 , MerkleProof(4)

note: (k, r)
Alice keeps secret
(one note per coin)

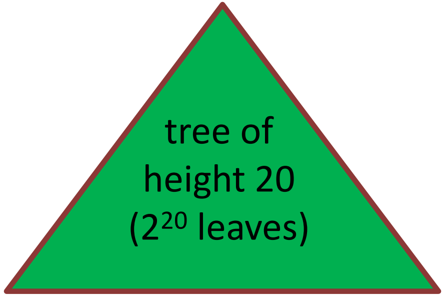
Treasury: **400** DAI

updated Merkle root	nf_1 nf_2
(32 bytes)	
next = **5**	nullifiers

updated contract state

Every deposit: new Coin added sequentially to tree

updated Merkle root



C_1 C_2 C_3 C_4 0 … 0

public list of coins

an observer sees who owns which coins

Tornado cash: withdrawal (simplified)

<u>100 DAI pool</u>:
each coin = 100 DAI

<u>Withdraw coin #3 to addr A:</u>

has note= (k', r')

set **nf** = $H_2(k')$

Treasury: **400** DAI

coins Merkle root

(32 bytes)

next = 5

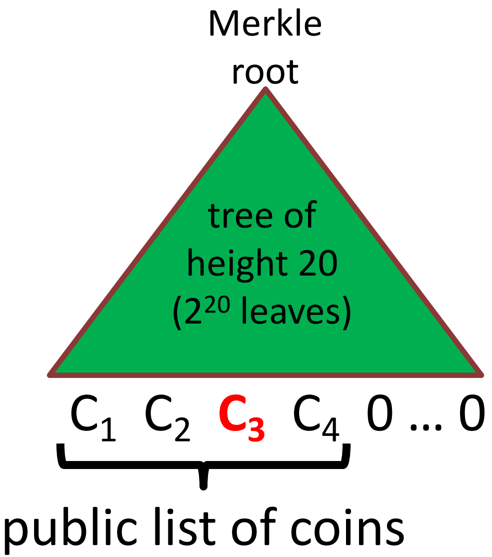
nf_1

nf_2

nullifiers

contract state

H_1, H_2: R → $\{0,1\}^{256}$



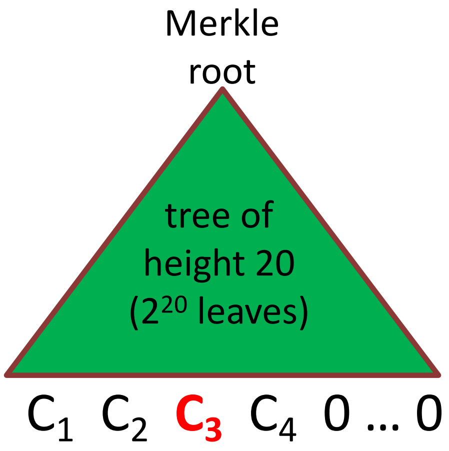
Bob proves "I have a note for some leaf in the coins tree, and its nullifier is **nf**"
(without revealing which coin)

Tornado cash: withdrawal (simplified)

$H_1, H_2: R \rightarrow \{0,1\}^{256}$

Withdraw coin #3 to addr A:

has note= (k', r')   set **nf** = H_2(k')



Bob builds zk-SNARK proof π for

public statement x = (**root**, **nf**, **A**)

secret witness w = (k', r', C_3, MerkleProof(C_3))

where Circuit(x,w)=0 iff:

(i) C_3 = (leaf #3 of **root**), i.e. MerkleProof(C_3) is valid,

(ii) $C_3 = H_1$(k', r'), and

(iii) **nf** = H_2(k').

(address A not used in Circuit)

Tornado cash: withdrawal (simplified)

$H_1, H_2\colon R \to \{0,1\}^{256}$

Withd

The address A is part of the statement to ensure that a miner cannot change A to its own address and steal funds

Assumes the SNARK is **non-malleable**:

adversary cannot use proof π for x to build a proof π' for some "related" x' (e.g., where in x' the address A is replaced by some A')

C_1 C_2 C_3 C_4 0 … 0

Bob builds zk-SNARK proof π for

public statement x = (**root**, **nf**, **A**)

secret witness w = $\big(k', r', C_3, \mathrm{MerkleProof}(C_3)\big)$

Tornado cash: withdrawal (simplified)

100 DAI pool:
each coin = 100 DAI

Withdraw coin #3 to addr A:

nf, proof π, **A**
(over Tor)
Bob's ID and coin C_3 are not revealed

Treasury: **400** DAI

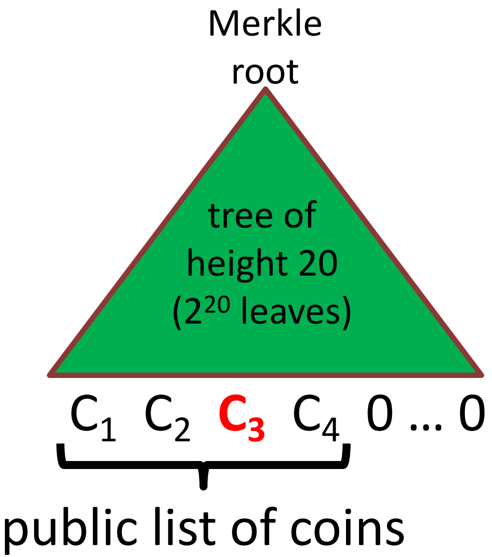
coins Merkle root
(32 bytes)

next = 5

nf_1
nf_2
nullifiers

contract state

H_1, H_2: $R \rightarrow \{0,1\}^{256}$



Contract checks (i) proof π is valid for (root, **nf**, **A**), and (ii) **nf** is not in the list of nullifiers

Tornado cash: withdrawal (simplified)

100 DAI pool:
each coin = 100 DAI

Withdraw coin #3 to addr A:

nf, proof π, **A**
(over Tor)

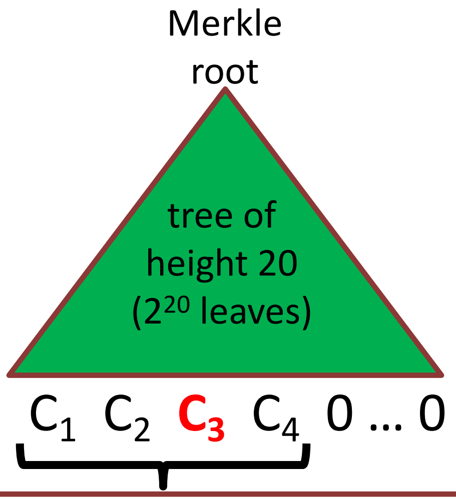
100 DAI
to address **A**

Treasury: **300** DAI

coins Merkle root	nf_1
(32 bytes)	nf_2
next = 5	**nf**
	nullifiers

contract state

H_1, H_2: $R \rightarrow \{0,1\}^{256}$



public list of coins
… but observer does not know which are spent

nf and π reveal nothing about which coin was spent.

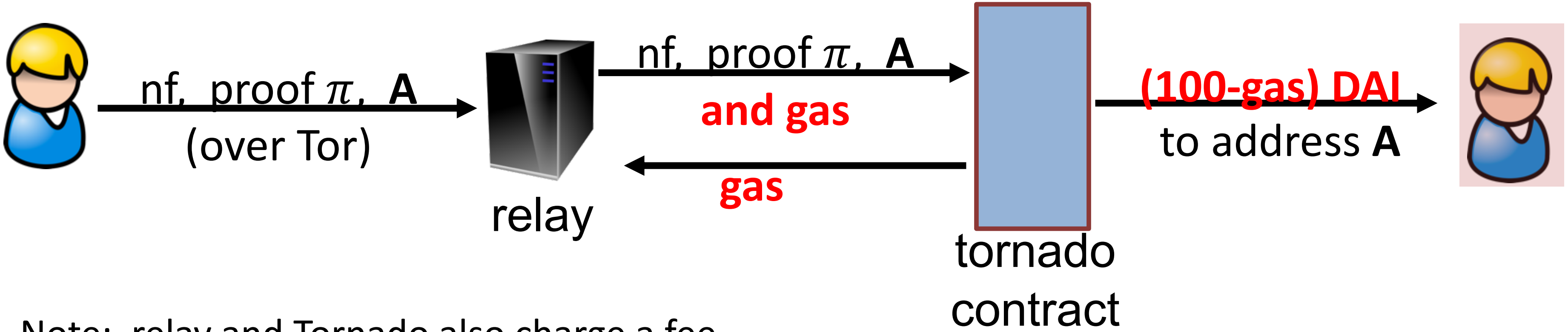
But, coin #3 cannot be spent again, because **nf = $H_2(k')$** is now nullified.

Who pays the withdrawal gas fee?

Problem: how does Bob pay for gas for the withdrawal Tx?

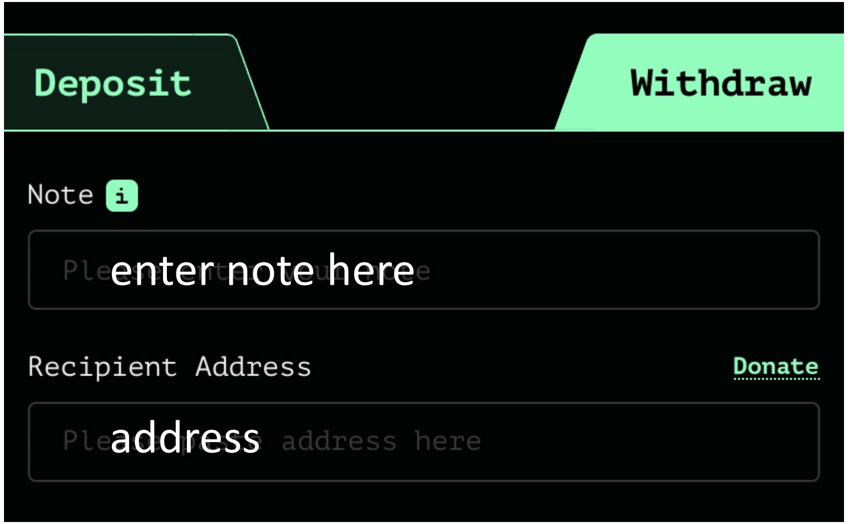
- If paid from Bob's address, then fresh address is linkable to Bob

Tornado's solution: **Bob uses a relay**



Note: relay and Tornado also charge a fee

Tornado Cash: the UI



After deposit: get a note

Later, use note to withdraw

(wait before withdrawing)

Anonymity set

88,036
Total deposits

$3,798,916,834
Total USD deposited

leaves occupied
over all pools

1 ETH pool

30141 equal user deposits

Latest deposits

30141. 4 minutes ago	30136. 3 hours ago
30140. 9 minutes ago	30135. 4 hours ago
30139. 2 hours ago	30134. 5 hours ago
30138. 3 hours ago	30133. 5 hours ago
30137. 3 hours ago	30132. 6 hours ago

Oct. 2021

Compliance tool

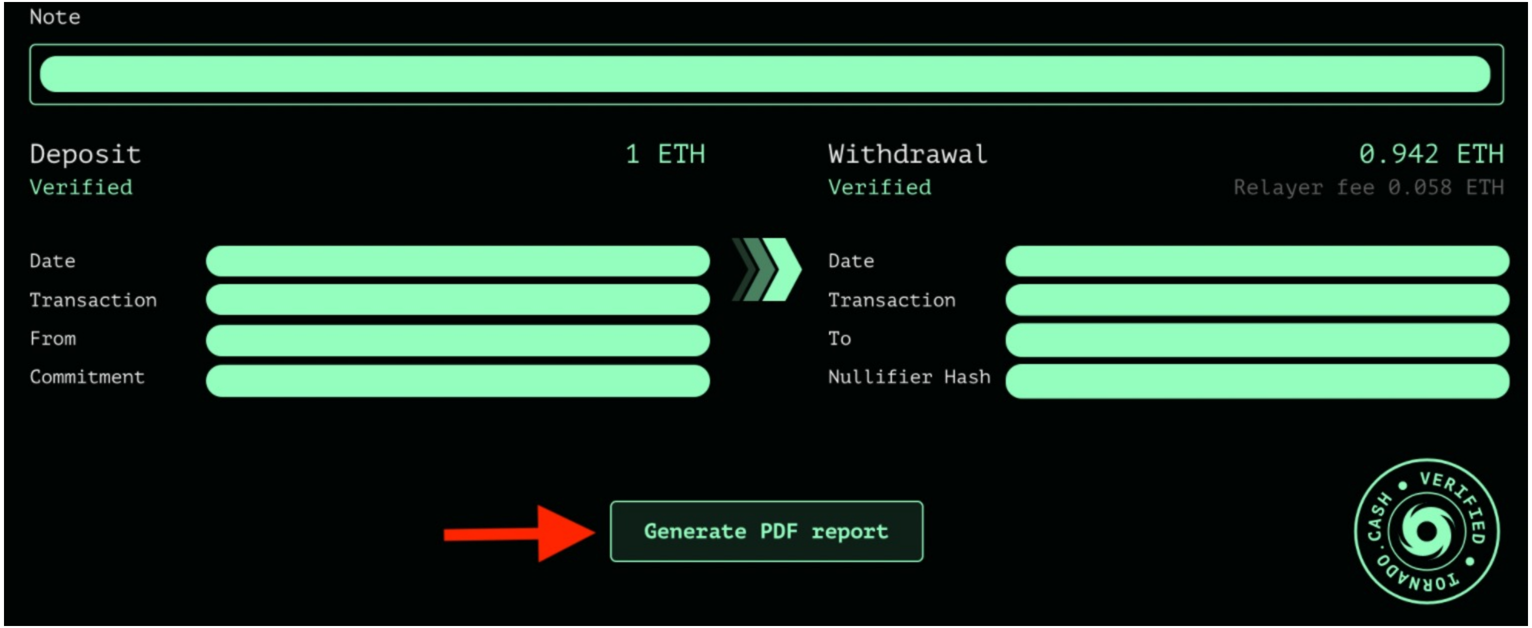
Tornado.cash compliance tool

Maintaining financial privacy is essential to preserving our freedoms.
However, it should not come at the cost of non-compliance. With Tornado.cash, you can always provide cryptographically verified proof of transactional history using the Ethereum address you used to deposit or withdraw funds. This might be necessary to show the origin of assets held in your withdrawal address.
To generate a compliance report, please enter your Tornado.Cash Note below.

Note

enter note here

Compliance tool



Reveals source address and destination address of funds

ZCASH / IRONFISH

Two L1 blockchains that extend Bitcoin.

Sapling (Zcash v2) launched in Aug. 2018.

Similar use of Nullifiers, support for any value Tx.

Quick review

A zk-SNARK for a circuit C:

- For a public statement x, prover outputs a proof that "convinces" verifier that prover knows w s.t. $C(x, w) = 0$.
- Proof is short and fast to verify

What is it good for?

- Private payments and private Dapp business logic (Aleo)
- Private compliance and L2 scalability with privacy

More to think about:

- private DAO participation? private governance?

Further topics

Privately communicating with the blockchain: Nym

- How to privately compensate proxies for relaying traffic

Next lecture: how to build a SNARK

END OF LECTURE

Two types of argument systems: interactive vs. non-interactive

Interactive: proof takes multiple P↔V rounds of interaction

- Useful when there is a single verifier, e.g. a compliance auditor
- Example: zero-knowledge proof of taxes to tax authority

Non-interactive: prover sends a single message (proof) to verifier

- Used when many verifiers need to verify proof
- SNARK: short proof that is fast to verify