

# Decentralized Finance

## Privacy on the Blockchain

Instructors: Dan Boneh, Arthur Gervais, Andrew Miller, Christine Parlour, Dawn Song



 Stanford  
University



Imperial College  
London



 UNIVERSITY OF  
ILLINOIS  
URBANA-CHAMPAIGN



Berkeley  
UNIVERSITY OF CALIFORNIA



# Can we have private transactions on a public blockchain?

Naïve reasoning:

universal verifiability  $\Rightarrow$  transaction data must be public.

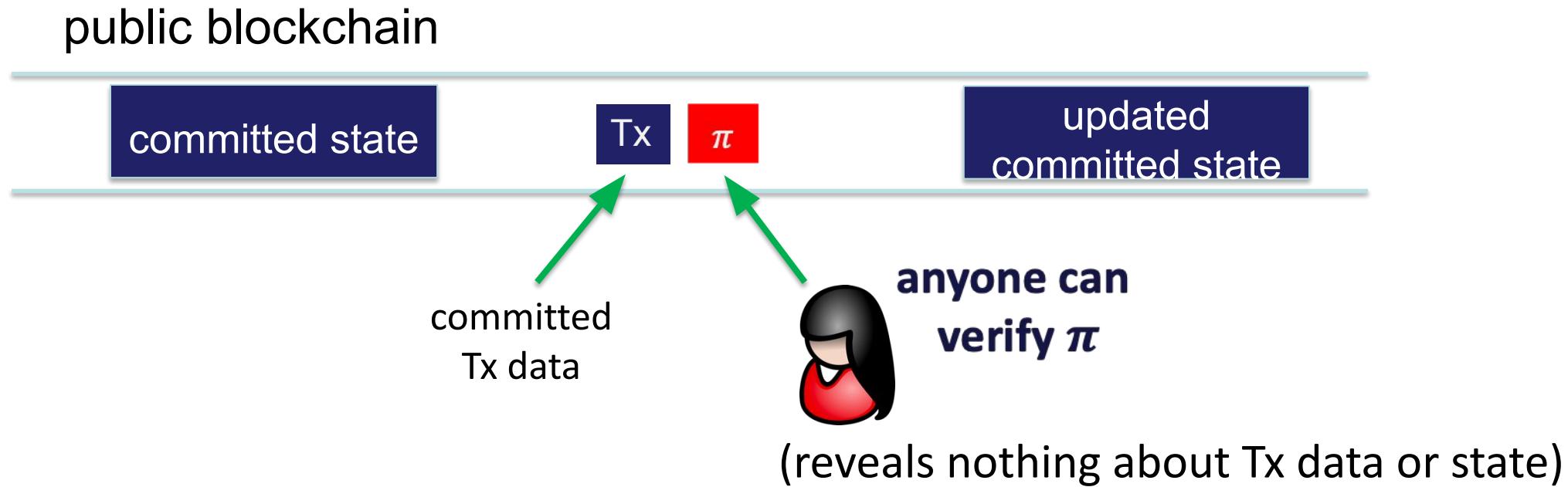
otherwise, how we can verify Tx ??

Goal for this lecture:

crypto magic  $\Rightarrow$  private Tx on a publicly verifiable blockchain  
区块链上的魔法

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

# Private Tx with universal verifiability: how?



**Committed data:** short (hiding) commitment on chain

**Proof  $\pi$ :** succinct *zero-knowledge proof* that

- (1) committed Tx data is consistent with committed current state, and
- (2) committed new state is correct

# 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?

# Types of Privacy

---

- **Pseudonymity: (weak privacy)**
  - One consistent pseudonym (e.g. reddit)
    - Pros: Reputation
    - Cons: Linkable posts: one post linked to you  $\Rightarrow$  all posts linked to you
- **Full anonymity:**
  - User's transactions are unlinkable
    - The system cannot tell if two transactions are from the same person
  - Maintaining reputation is possible but more complex

# Privacy in Ethereum?

- Accounts:

- Every account balance is public
- For Dapps: code and internal state are public
- All account transactions are linked to account

etherscan.io:

Address 0x1654b0c3f62902d7A86237

...

Balance:

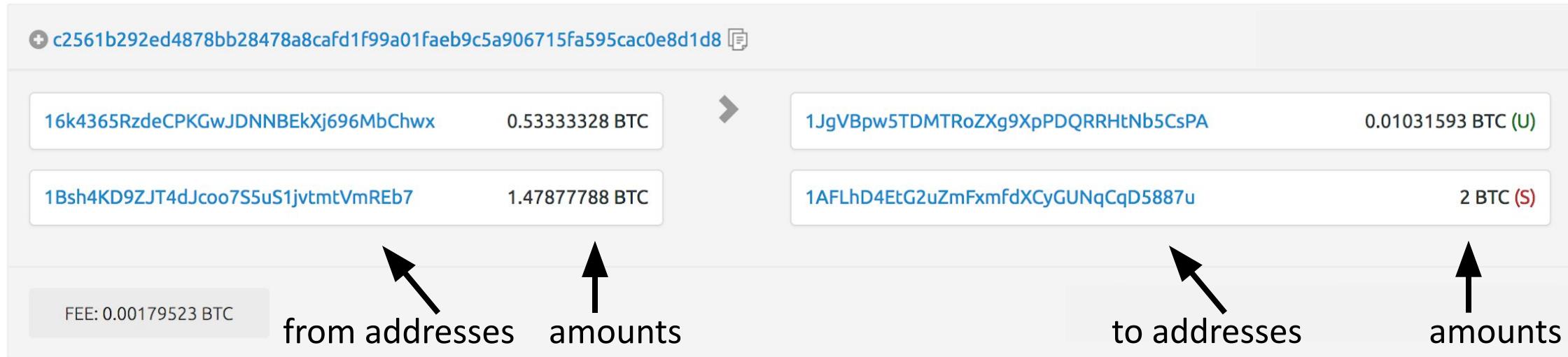
1.114479450024297906 Ether

Ether Value:

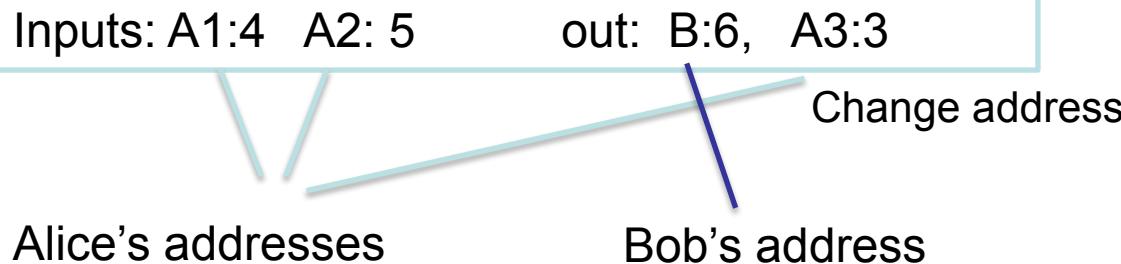
\$4,286.34 (@ \$3,846.05/ETH)

Txn Hash	Method ⓘ	Block
<a href="#">0x0269eff8b4196558c07...</a>	Set Approval For...	13426561
<a href="#">0xa3dacb0e7c579a99cd...</a>	Cancel Order_	13397993
<a href="#">0x73785abcc7ccf030d6a...</a>	Set Approval For...	13387834
<a href="#">0x1463293c495069d61c...</a>	Atomic Match_	13387703

# Privacy in Bitcoin?



Alice can have many addresses (creating address is free)



Transaction data can be used to link addresses to a single owner and to a physical entity

(chainalysis)

# Privacy of Digital Payments

Payments publicly  
visible and linkable



Payments only  
visible to bank



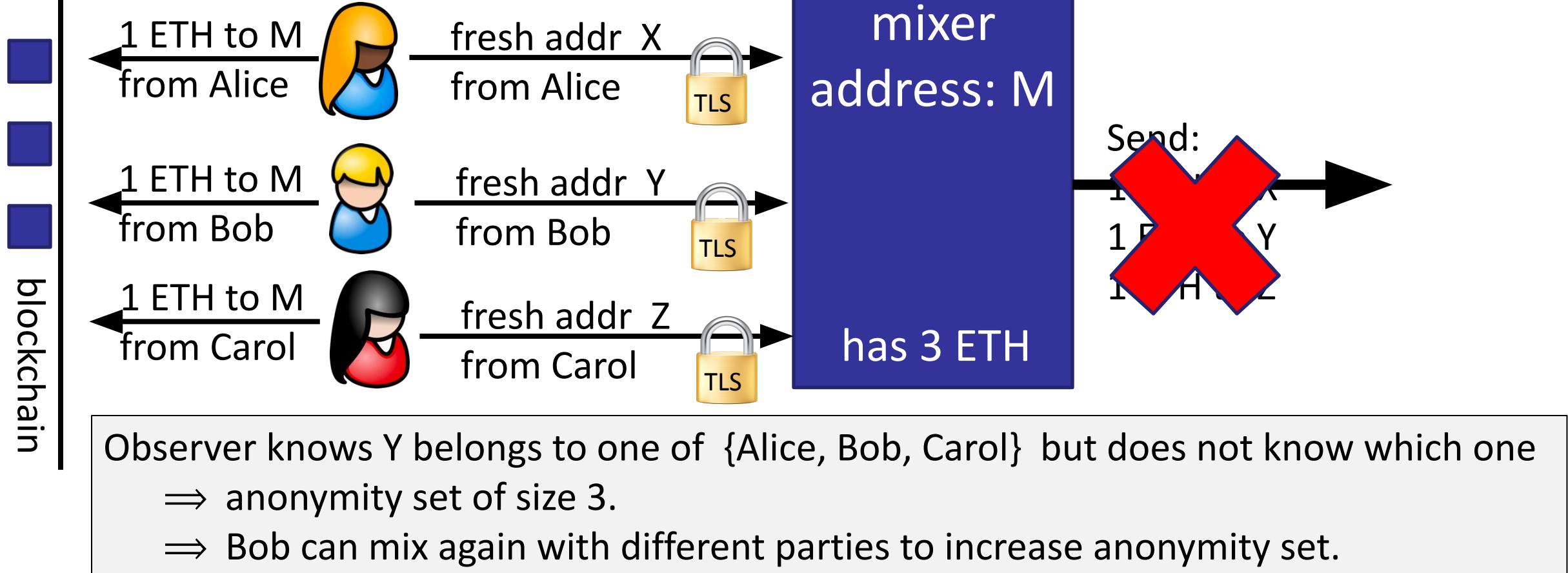
private payments



Less private

More private

# Simple blockchain anonymity via mixing



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**

# Negative aspects of privacy in finance

## Criminal activity:

- Tax evasion, ransomware, ...



Can we support positive applications of private payments, but prevent the negative ones?

- Can we ensure legal compliance while preserving privacy?
- Yes! With proper use of zero knowledge proofs

Next segment: commitments

An important tool

A photograph of Earth taken from space at night. The planet is mostly dark, with numerous glowing city lights visible in the lower half. In the upper portion, the green and blue glow of the aurora borealis or aurora australis is visible against the black void of space.

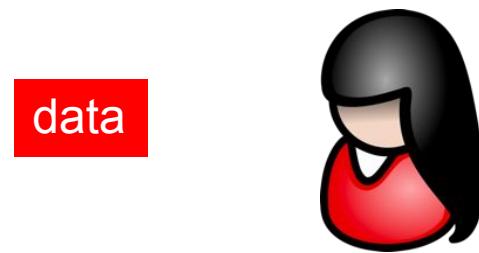
# Cryptographic Commitments

<https://defi-learning.org/>

# Cryptographic commitments

---

Cryptographic commitment: emulates an envelope



Many applications: e.g., a DAPP for a sealed bid auction

- Every participant **commits** to its bid,
- Once all bids are in, everyone opens their commitment

# Cryptographic Commitments

Syntax: a commitment scheme is two algorithms

- commit(*msg, r*) → *com*

secret randomness in  $R$

commitment string

- verify(*msg, com, r*) → accept or reject

anyone can verify that commitment was opened correctly

# Commitments: security properties

---

- **binding**: Bob cannot produce two valid openings for  $\text{com}$ .  
More precisely: no efficient adversary can produce  
 $\text{com}, (m_1, r_1), (m_2, r_2)$   
such that  $\text{verify}(m_1, \text{com}, r_1) = \text{verify}(m_2, \text{com}, r_2) = \text{accept}$   
and  $m_1 \neq m_2$ .

---

- **hiding**:  $\text{com}$  reveals nothing about committed data  
 $\text{commit}(m, r) \rightarrow \text{com}$ , and  $r$  is sampled uniformly in  $R$ ,  
then  $\text{com}$  is statistically independent of  $m$

# Example 1: hash-based commitment

Fix a hash function  $H: M \times R \rightarrow C$  (e.g., SHA256)

where  $H$  is collision resistant, and  $|R| \gg |C|$

- $\text{commit}(m \in M, r \leftarrow R): \quad \text{com} = H(m, r)$
- $\text{verify}(m, \text{com}, r): \quad \text{accept if } \text{com} = H(m, r)$

binding: follows from collision resistance of  $H$

hiding: follows from a mild assumption on  $H$

## Example 2: Pedersen commitment

---

$G$  = finite cyclic group =  $\{1, g, g^2, \dots, g^{q-1}\}$  where  $g^i \cdot g^j = g^{(i+j) \bmod q}$

$q = |G|$  is called the **order** of  $G$ . Assume  $q$  is a prime number.

Fix  $g, h$  in  $G$  and let  $R = \{0, 1, \dots, q-1\}$ . For  $m, r \in R$  define

$$H(m, r) = g^m \cdot h^r \in G$$

**Fact:** for a “cryptographic” group  $G$ , this  $H$  is collision resistant.

⇒ commitment scheme: **commit** and **verify** as in example 1

$$\text{commit}(m \in R, r \leftarrow R) = H(m, r) = g^m \cdot h^r$$

# An interesting “homomorphic” property

---

- $\text{commit}(m \in R, r \leftarrow R) = H(m, r) = g^m \cdot h^r$

Suppose:  $\text{commit}(m_1 \in R, r_1 \leftarrow R) \rightarrow com_1$

$\text{commit}(m_2 \in R, r_2 \leftarrow R) \rightarrow com_2$

Then:  $com_1 \times com_2 = g^{m_1+m_2} \cdot h^{r_1+r_2} = \text{commit}(m_1+m_2, r_1+r_2)$

---

⇒ anyone can sum committed value

Next segment: zero knowledge proofs

An important privacy tool

# What is a zk-SNARK?

Succinct Non-interactive ARgument of Knowledge

# zk-SNARK: Blockchain Applications

## Scalability:

- SNARK Rollup (zk-SNARK for privacy from public)

## Privacy: Private Tx on a public blockchain

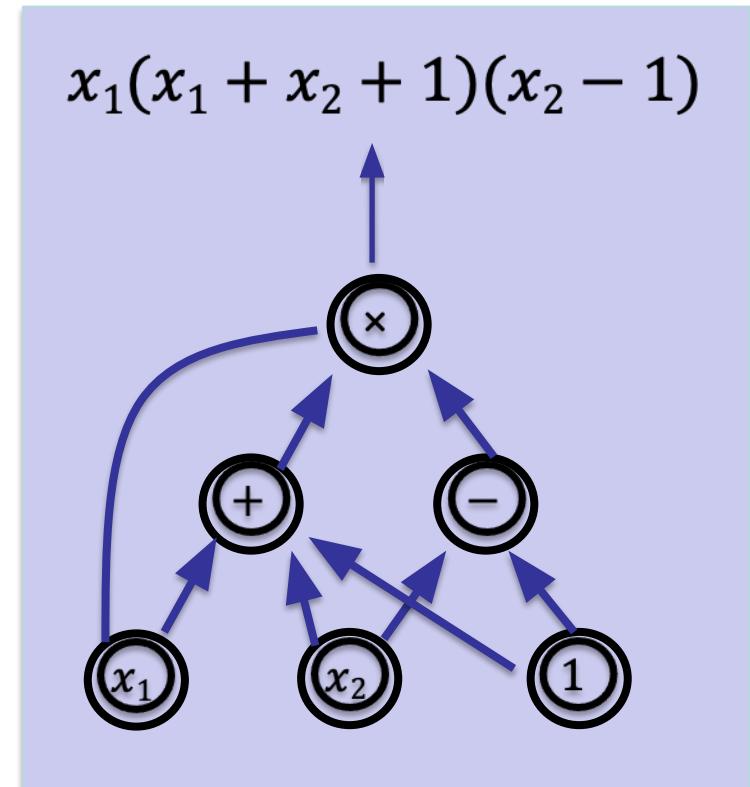
- Confidential transactions
- Tornado cash
- Private Dapps: Aleo

## Compliance:

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

# (1) arithmetic circuits

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



# Interesting arithmetic circuits

---

## Examples:

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

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

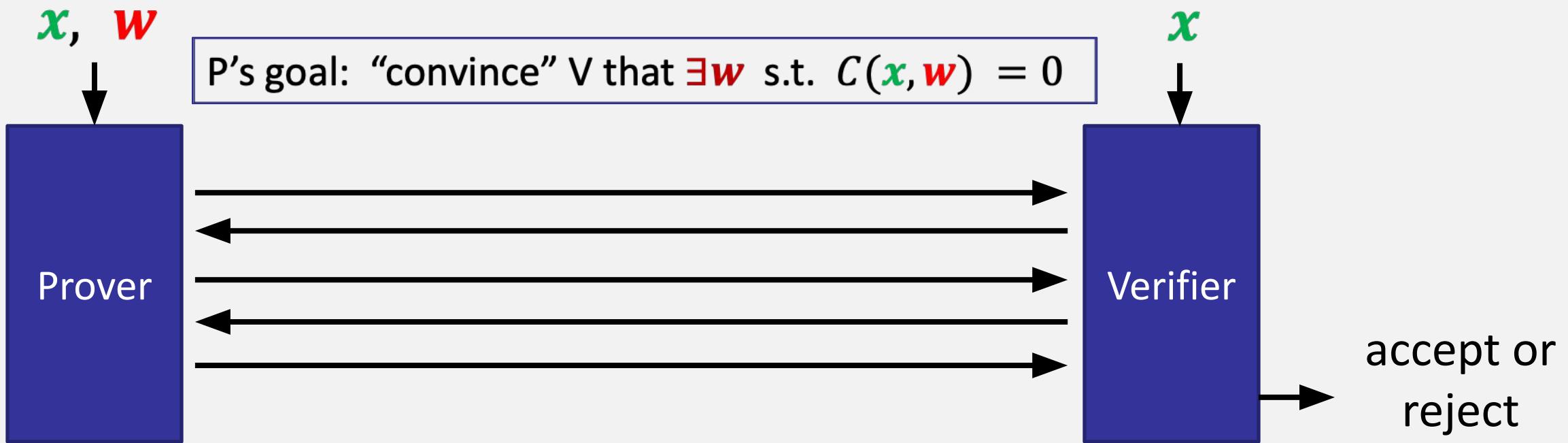
- $C_{\text{sig}}(pk, m, \sigma)$ : outputs 0 if  $\sigma$  is a valid ECDSA signature on  $m$  with respect to  $pk$

# (2) Argument systems

(for NP)

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

public statement in  $\mathbb{F}^n$       secret witness in  $\mathbb{F}^m$



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

Interactive: proof takes multiple  $P \leftrightarrow V$  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

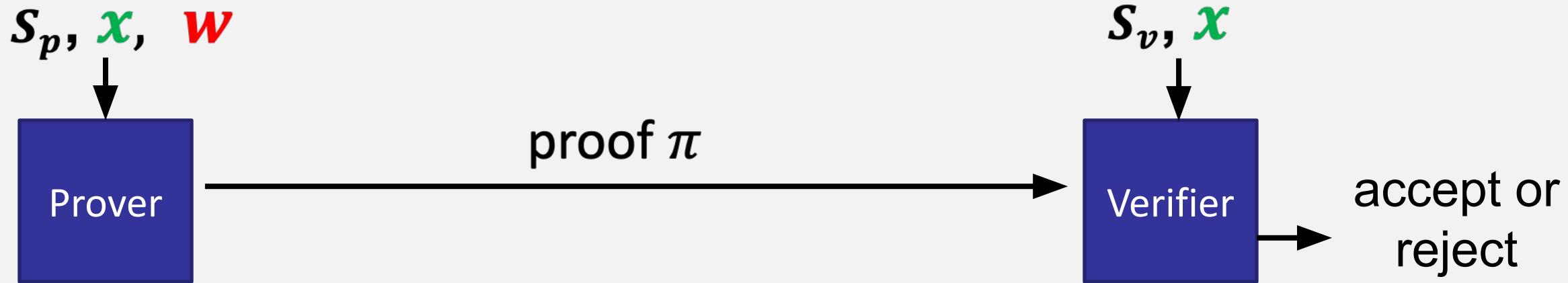
- Used when many verifiers need to verify proof, e.g., Rollup systems
- SNARK: short proof that is fast to verify

# (non-interactive) Preprocessing argument system

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

public statement in  $\mathbb{F}^n$       secret witness in  $\mathbb{F}^m$

Preprocessing (setup):  $S(C) \rightarrow$  public parameters  $(S_p, S_v)$



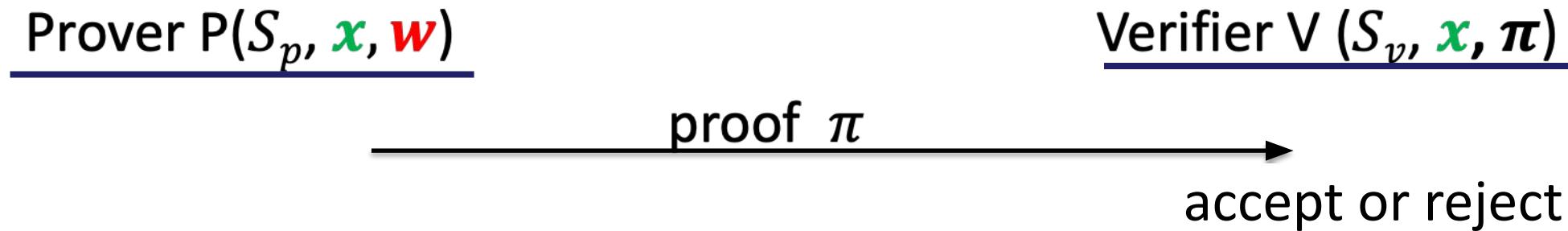
# Preprocessing argument System

---

A **non-interactive argument system** is a triple  $(S, P, V)$ :

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

# Argument system: requirements (informal)



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

**Argument of knowledge:**  $V$  accepts  $\Rightarrow P$  “knows”  $w$  s.t.  $C(\textcolor{green}{x}, \textcolor{red}{w}) = 0$

$P^*$  does not “know”  $w \Rightarrow \Pr[ V(S_v, x, \pi) = \text{accept}] < \text{negligible}$

**Optional: Zero knowledge:**  $(S_v, \textcolor{green}{x}, \pi)$  “reveals nothing” about  $w$

# Preprocessing SNARK

A succinct non-interactive argument system is a triple  $(S, P, V)$ :

- $S(C) \rightarrow$  public parameters  $(S_p, S_v)$  for prover and verifier
  - $P(S_p, \textcolor{green}{x}, \textcolor{red}{w}) \rightarrow$  short proof  $\pi$  ;  $|\pi| = O(\log(|C|), \lambda)$
  - $V(S_v, \textcolor{green}{x}, \boldsymbol{\pi}) \rightarrow$  accept or reject ;  $\text{time}(V) = O(|x|, \log(|C|), \lambda)$
- short “summary” of circuit
- Why preprocess  $C$ ??

# Preprocessing SNARK

A succinct non-interactive argument system is a triple  $(S, P, V)$ :

- $S(C) \rightarrow$  public parameters  $(S_p, S_v)$  for prover and verifier
- $P(S_p, \textcolor{green}{x}, \textcolor{red}{w}) \rightarrow$  short proof  $\pi$  ;  $|\pi| = O(\log(|C|), \lambda)$
- $V(S_v, \textcolor{green}{x}, \boldsymbol{\pi}) \rightarrow$  accept or reject ;  $\text{time}(V) = O(|x|, \log(|C|), \lambda)$

If  $(S, P, V)$  is **succinct** and **zero-knowledge** then we say that it is a **zk-SNARK**

# The trivial argument system

---

- (a) Prover sends  $w$  to verifier,
- (b) Verifier checks if  $C(x, w) = 0$  and accepts if so.

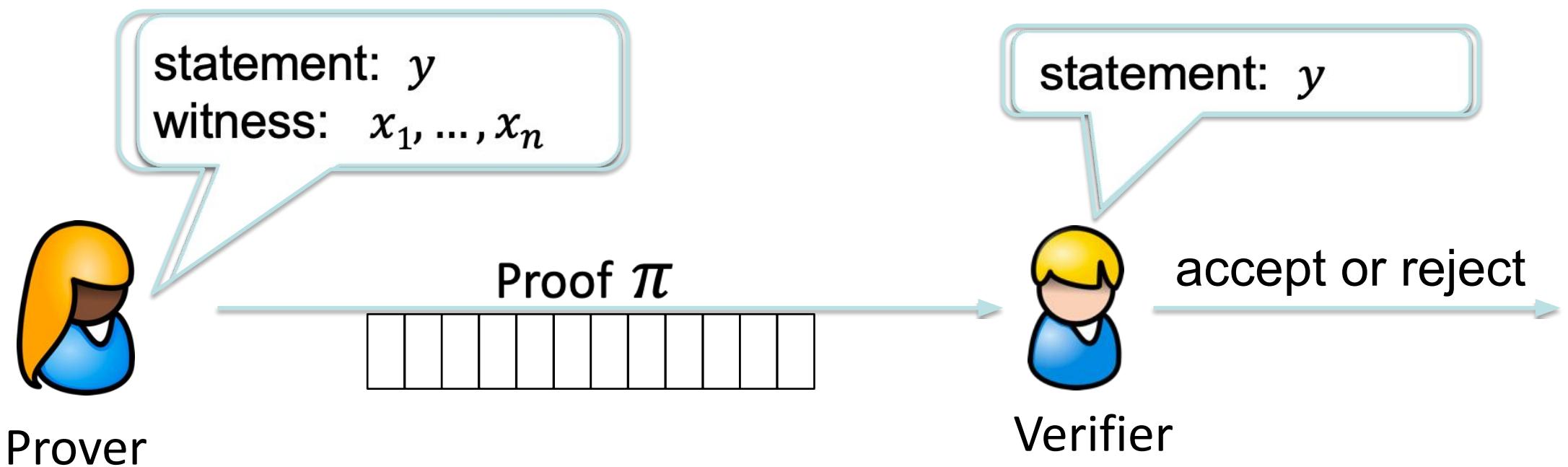
## Problems with this:

- (1)  $w$  might be secret: prover does not want to reveal  $w$  to verifier
- (2)  $w$  might be long: we want a “short” proof
- (3) computing  $C(x, w)$  may be hard: we want a “fast” verifier

# An example

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

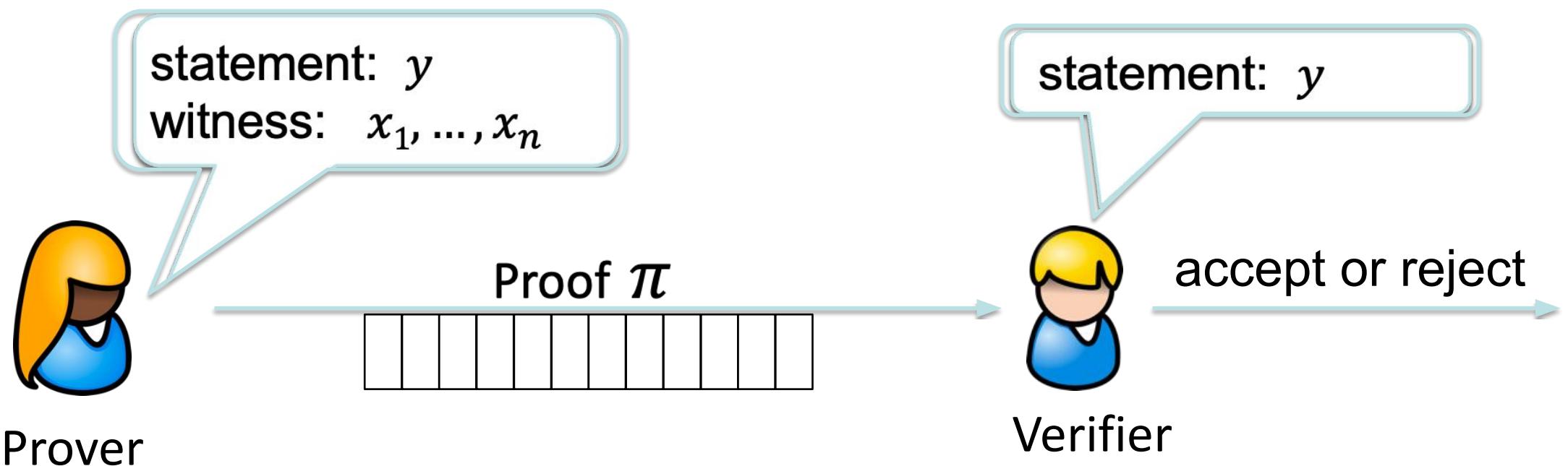
**SNARK:**  $\text{size}(\pi)$  and  $\text{VerifyTime}(\pi)$  is  $O(\log n)$  !!



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

Types of setup:

**trusted setup per circuit:**  $\mathbf{S}(C)$  uses data that must be kept secret

compromised trusted setup  $\Rightarrow$  can prove false statements

**trusted but universal (updatable) setup:** secrets in  $\mathbf{S}(C)$  are independent of  $C$

$$\mathbf{S} = (S_{init}, S_{pre}): \quad \underbrace{S_{init}(\lambda) \rightarrow U}_{\text{one-time}} \quad \underbrace{S_{pre}(U, C) \rightarrow (S_p, S_v)}_{\text{no secret data}}$$

**transparent setup:**  $\mathbf{S}(C)$  does not use secret data (no trusted setup)

better

# 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  $\Rightarrow$  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)

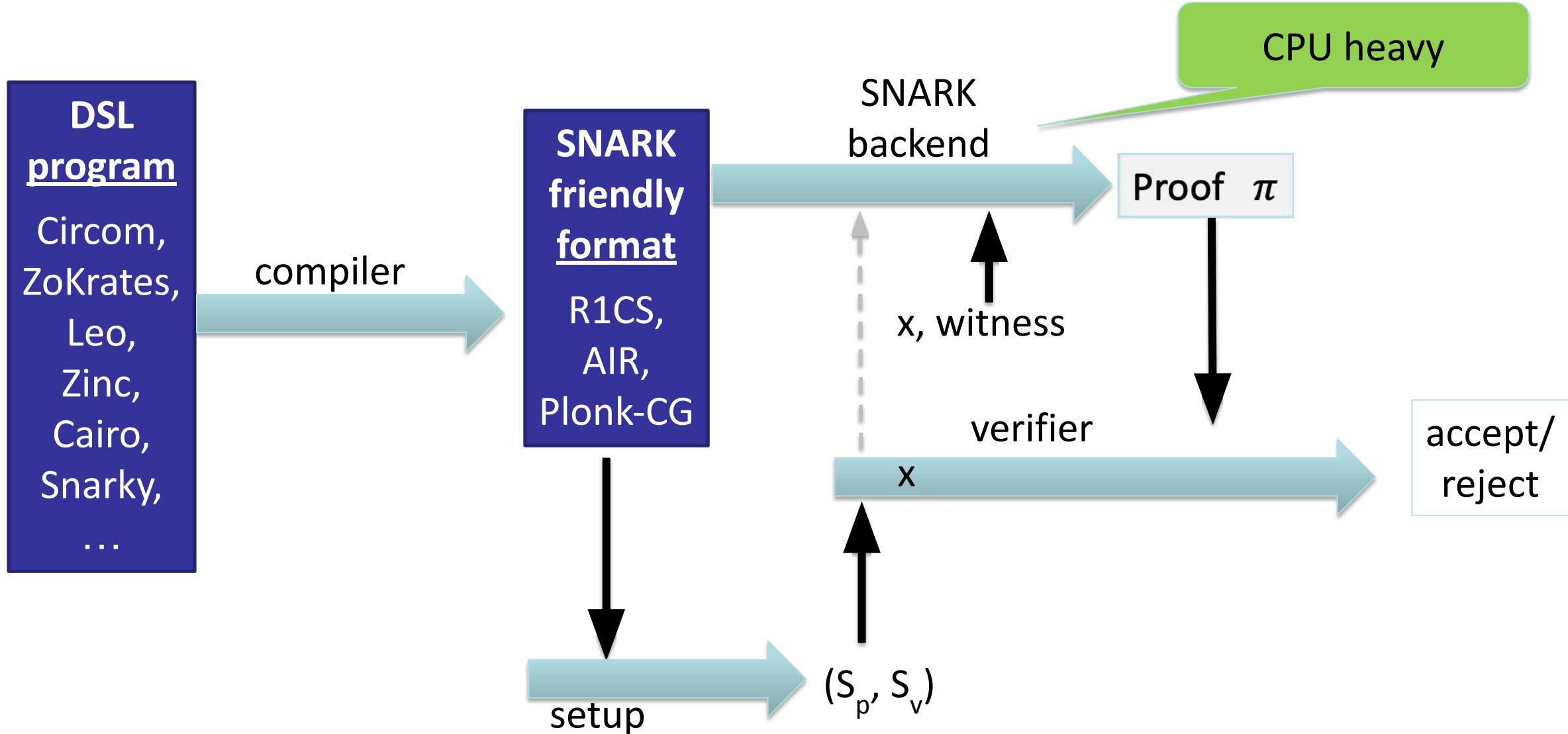
		size of $S_p$	verifier time	trusted setup?
Groth'16	$O(1)$		$O(1)$	yes/per circuit
Plonk/Marlin	$O(1)$		$O(1)$	yes/universal
Bulletproofs		$O(1)$		no
STARK		$O(1)$		no
DARK		$O(1)$		no

::

::

::

# A SNARK software system



# ZoKrates Example

**Goal:** prove knowledge of a hash (SHA256) preimage for a given  $x \in \{0,1\}^{256}$

- For a public  $x$ , prover knows  $w \in \mathbb{F}_p$  such that  $\text{SHA256}(w) = x$
- $\mathbb{F}_p$  is a 254-bit prime field

Compiled into an arithmetic circuits  
(R1CS) over  $\mathbb{F}_p$

```
def main(field x[2], private field w) -> (field):
    h = sha256packed( w )
    h[0] == x[0]    // check top 128 bits
    h[1] == x[1]    // check bottom 128 bits
    return 1
```

How to define “argument of knowledge”  
and “zero knowledge”?

# Definitions: (1) argument of knowledge

---

**Goal:** if  $V$  accepts then  $P$  “knows”  $w$  s.t.  $C(x, w) = 0$

What does it mean to “know”  $w$  ??

**informal def:**  $P$  knows  $w$ , if  $w$  can be “extracted” from  $P$



# Definitions: (1) argument of knowledge

**Formally:**  $(S, P, V)$  is an **argument of knowledge** for a circuit  $C$  if for every poly. time adversary  $A = (A_0, A_1)$  such that

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

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

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

$$S(C) \rightarrow (S_p, S_v), \quad (x, st) \leftarrow A_0(S_p),$$

$$w \leftarrow E^{A_1(S_p, x, st)}(S_p, x):$$

$$\Pr[C(x, w) = 0] > 1/10^6 \quad (\text{non-negligible})$$

If holds for all  $A$ , then  $(S, P, V)$  is a **proof of knowledge**.

# Definitions: (2) Zero knowledge

(against an honest verifier)

$(S, P, V)$  is **zero knowledge** if for every  $x \in \mathbb{F}^n$   
proof  $\pi$  “reveals nothing” about  $w$ , other than its existence

What does it mean to “reveal nothing” ??

**Informal def:**  $\pi$  “reveals nothing” about  $w$  if the verifier can  
generate  $\pi$  **by itself**  $\Rightarrow$  it learned nothing new from  $\pi$

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

Main point:  $\text{Sim}(C, x)$  simulates  $\pi$  without knowledge of  $w$

# 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  $\text{Sim}$  such that

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

$(S_p, S_v, x, \pi)$ : where  $(S_p, S_v) \leftarrow S(C)$ ,  $\pi \leftarrow P(S_p, x, w)$

is indistinguishable from the distribution:

$(S_p, S_v, x, \pi)$ : where  $(S_p, S_v, \pi) \leftarrow \text{Sim}(C, x)$

# 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 ??

Not in this course ...

(see, e.g., cs251)

---

Next segment: confidential transactions

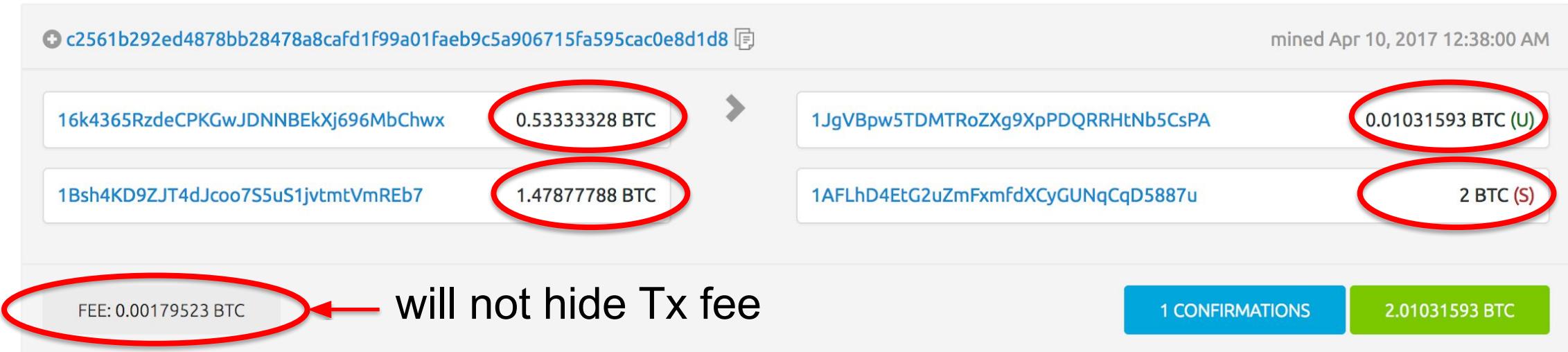
# Private Tx Warmup: Confidential Transactions

<https://defi-learning.org/>

# Confidential Transactions (CT)

Current Bitcoin Tx expose full payment details:

⇒ Businesses cannot use Bitcoin for supply chain payments, salaries, etc.



Goal: hide amounts in Bitcoin transactions.

# Confidential Tx: how?

Bitcoin Tx today: AcmeCo: **30** → Alice: **1**, AcmeCO: **29**

8 bytes

The plan: replace amounts by commitments to amounts

AcmeCo: **com<sub>1</sub>** → Alice: **com<sub>2</sub>**, AcmeCo: **com<sub>3</sub>**

32 bytes

where **com<sub>1</sub>** = commit(30, r<sub>1</sub>), **com<sub>2</sub>** = commit(1, r<sub>2</sub>), **com<sub>3</sub>** = commit(29, r<sub>3</sub>)

# Now blockchain hides amounts

A screenshot of a Bitcoin transaction details page. At the top, the transaction ID is shown as `c2561b292ed4878bb28478a8cafd1f99a01faeb9c5a906715fa595cac0e8d1d8`. To the right, the timestamp is listed as "mined Apr 10, 2017 12:38:00 AM". Below the ID, there are two output sections. The first output shows a sending address `16k4365RzdeCPKGwJDNNBEkjxj696MbChwx` and a receiving address `3bd6e25fqd`. The second output shows a sending address `1Bsh4KD9ZJT4dJcoo7S5uS1jvtmtVmREb7` and a receiving address `8c528ad9fa`. At the bottom left, a red oval highlights the "FEE: 0.00179523 BTC" text. On the right side, there are two buttons: a blue one labeled "1 CONFIRMATIONS" and a green one labeled "2.01031593 BTC".

How much was transferred ???

# The problem: how can miners verify Tx?

AcmeCo:  $\text{com}_1 \rightarrow$  Alice:  $\text{com}_2, \text{AcmeCo: com}_3$

$\text{com}_1 = \text{commit}(m_1=30, r_1), \text{com}_2 = \text{commit}(m_2=1, r_2), \text{com}_3 = \text{commit}(m_3=29, r_3)$

Solution: zk-SNARK (special purpose, optimized for this problem)

- AcmeCo: (1) privately send  $r_2$  to Alice  
(2) construct a proof  $\pi$  for

statement =  $x = (\text{com}_1, \text{com}_2, \text{com}_3, \text{Fees})$   
witness =  $w = (m_1, r_1, m_2, r_2, m_3, r_3)$

where circuit  $C(x,w)$  outputs 0 iff:

CT arithmetic circuit

$\left\{ \begin{array}{ll} \text{(i)} & \text{com}_i = \text{commit}(m_i, r_i) \text{ for } i=1,2,3, \\ \text{(ii)} & m_1 = m_2 + m_3 + \text{Fees}, \\ \text{(iii)} & m_2 \geq 0 \text{ and } m_3 \geq 0 \end{array} \right.$

# The problem: how can miners verify Tx?

---

- AcmeCo:
- (1) privately send  $r_2$  to Alice
  - (2) construct a ZK proof  $\pi$  that Tx is valid
  - (3) embed  $\pi$  in Tx      (need short proof!  $\Rightarrow$  zk-SNARK)

Tx: proof  $\pi$  ,   AcmeCo: **com<sub>1</sub>**    $\rightarrow$    Alice: **com<sub>2</sub>**,   AcmeCo: **com<sub>3</sub>**

Miners: accept Tx if proof  $\pi$  is valid (need fast verification)  
 $\Rightarrow$  learn Tx is valid, but amounts are hidden

# Optimized proof?

circuit  $C(x,w)$  outputs 0 if:

- (i)  $\text{com}_i = \text{commit}(m_i, r_i)$ ,
- ~~(ii)  $m_1 = m_2 + m_3 + \text{Fees}$ ,~~
- (iii)  $m_2 \geq 0$  and  $m_3 \geq 0$

Easy to check with Pedersen commitment:

set  $\text{com} \leftarrow \text{com}_1 / \text{com}_2 \cdot \text{com}_3 \cdot g^{\text{Fees}}$   
(a commitment to  $m_1 - m_2 - m_3 - \text{Fees}$ )

prove that  $\text{com} = \text{commit}(0, r)$

remaining proof is  $\approx 400$  bytes

(CT is the beginning of MimbleWimble implemented in the Grin blockchain)

---

Next segment: anonymous payments

# Anonymous Payments: Tornado Cash and Zcash / IronFish

<https://defi-learning.org/>

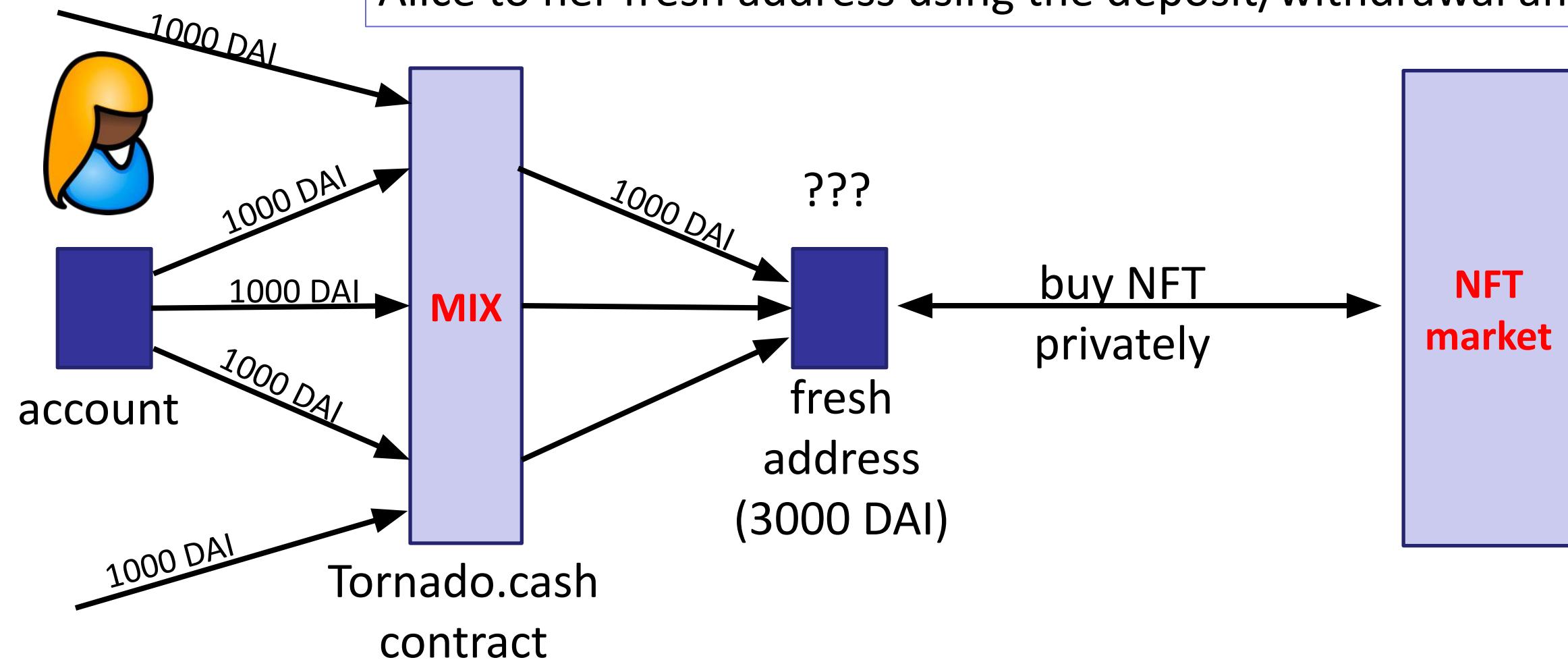
---

# **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



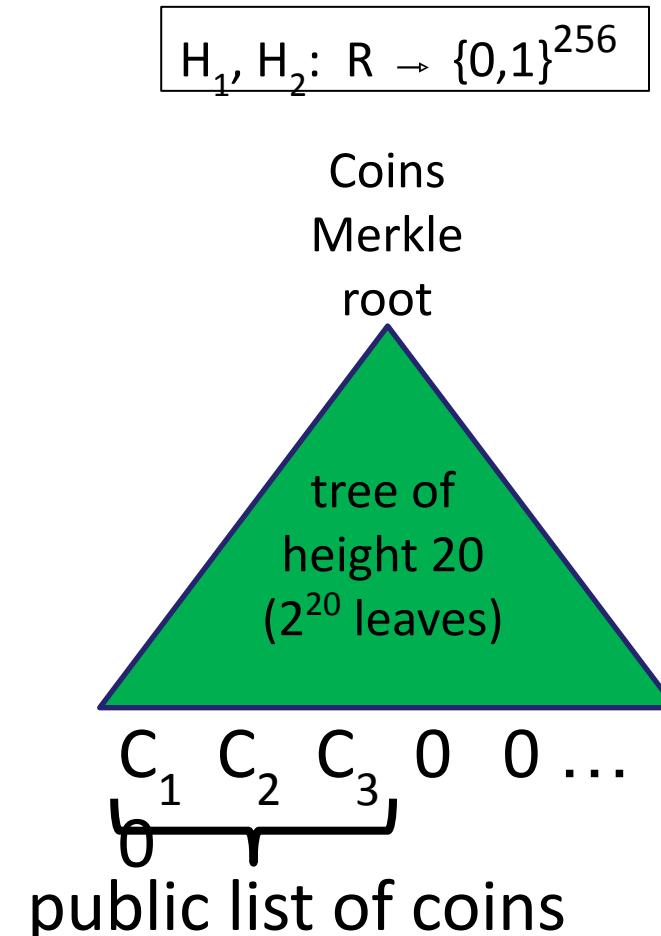
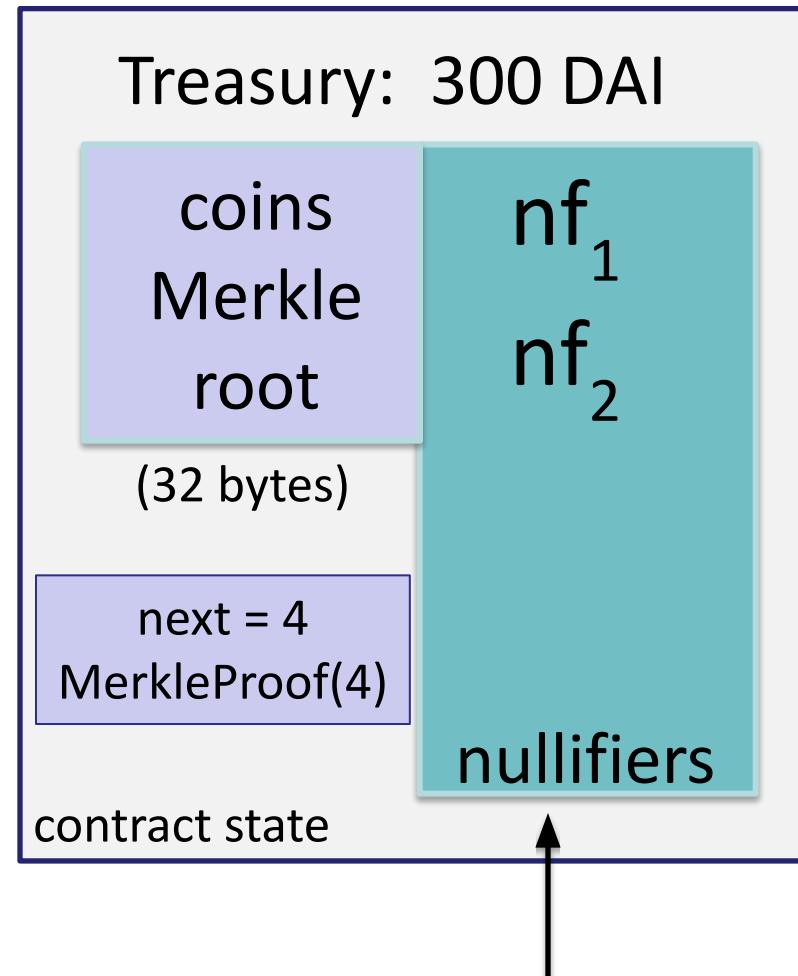
# The tornado cash contract (simplified)

100 DAI pool:

each coin = 100 DAI

Currently:

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



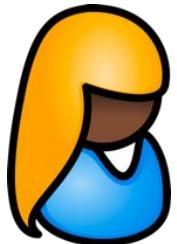
explicit list:  
one entry per **spent coin**

# Tornado cash: deposit (simplified)

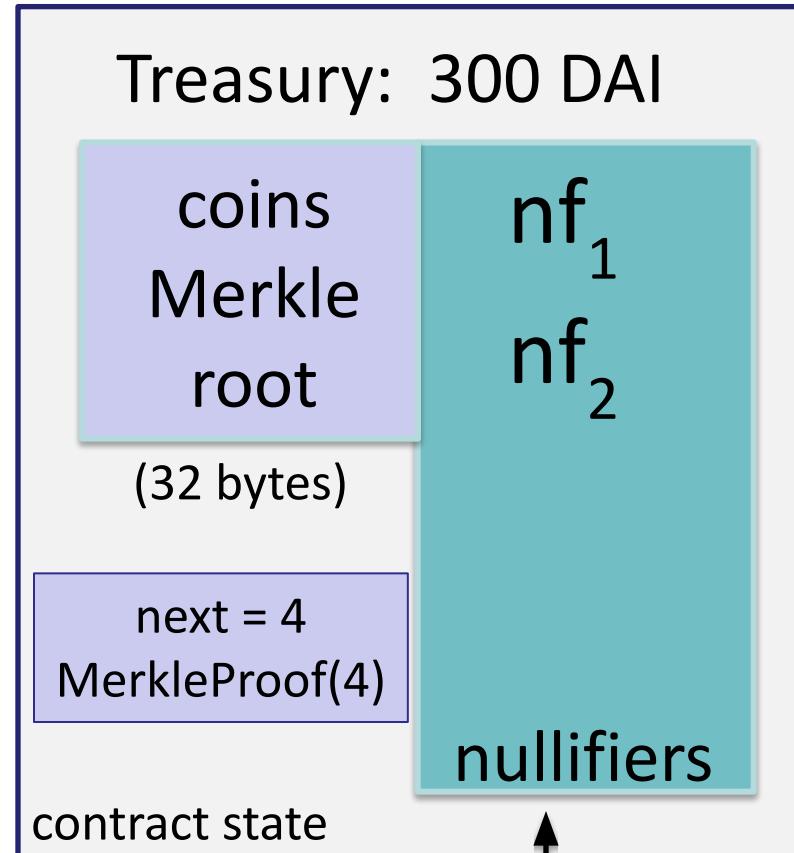
100 DAI pool:

each coin = 100 DAI

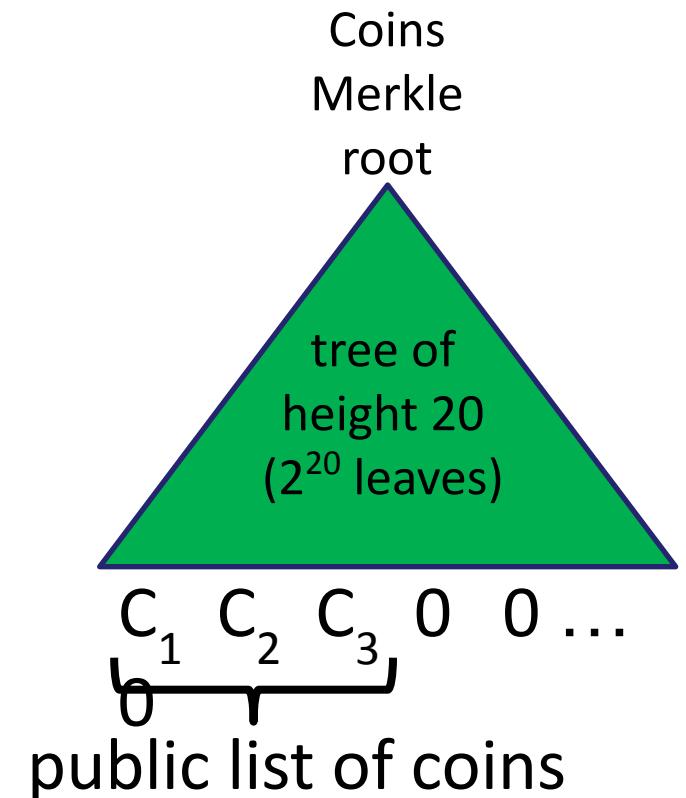
Alice deposits 100 DAI:



choose random  $k, r$  in  $R$   
set  $C_4 = H_1(k, r)$   
write  $C_4$  in leaf #4 in tree  
 $\pi = \text{MerkleProof}(5)$



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

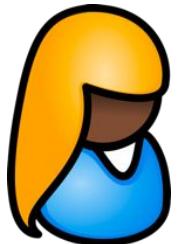


# Tornado cash: deposit (simplified)

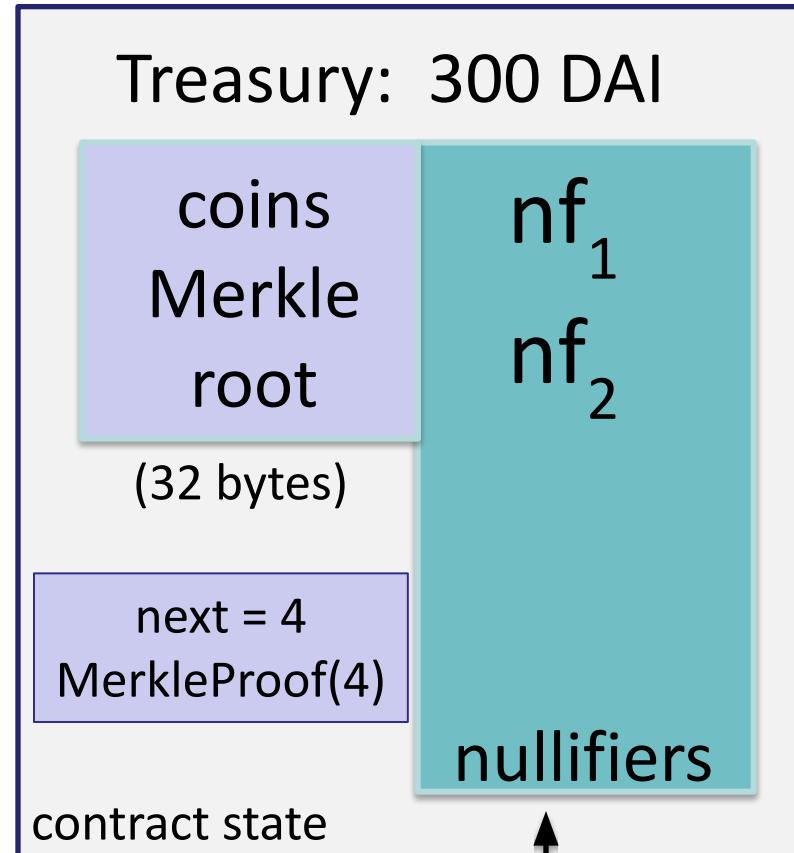
100 DAI pool:

each coin = 100 DAI

Alice deposits 100 DAI:

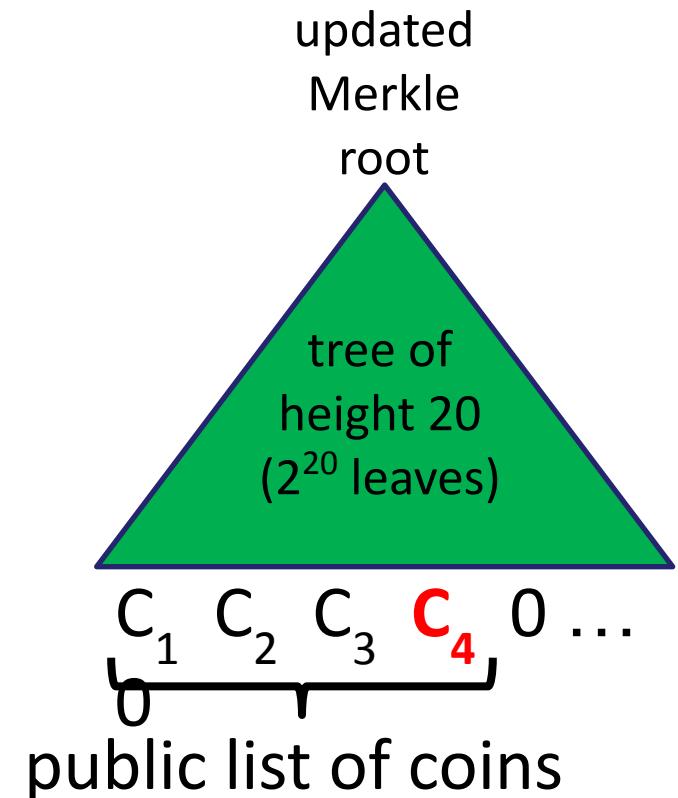


choose random  $k, r$  in  $R$   
set  $C_4 = H_1(k, r)$   
write  $C_4$  in leaf #4 in tree  
 $\pi = \text{MerkleProof}(5)$



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

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

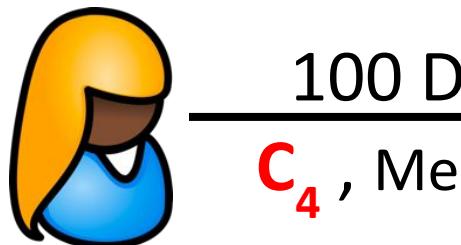


# Tornado cash: deposit (simplified)

100 DAI pool:

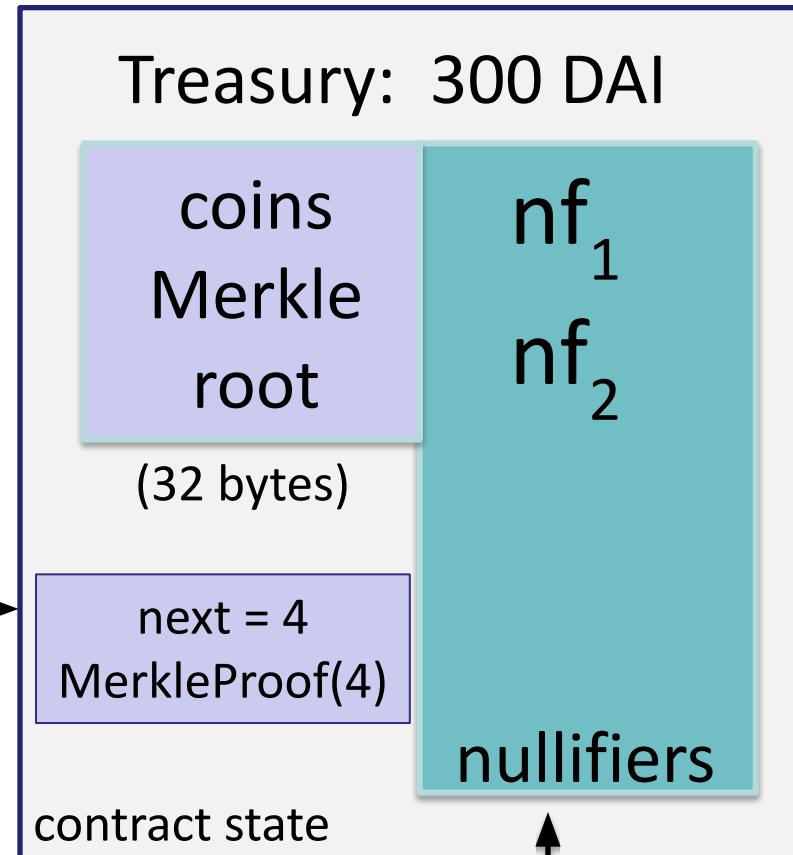
each coin = 100 DAI

Alice deposits 100 DAI:

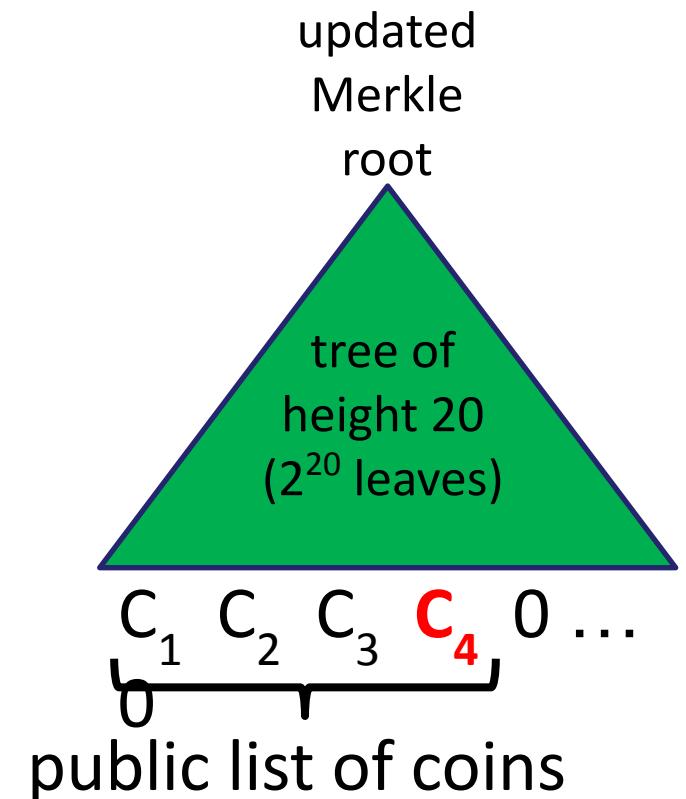


100 DAI  
 $C_4$ , MerkleProof(5)

choose random  $k, r$  in  $R$   
set  $C_4 = H_1(k, r)$   
write  $C_4$  in leaf #4 in tree  
 $\pi = \text{MerkleProof}(5)$

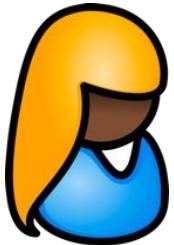


$$H_1, H_2: R \rightarrow \{0,1\}^{256}$$



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

# Tornado cash: deposit (simplified)

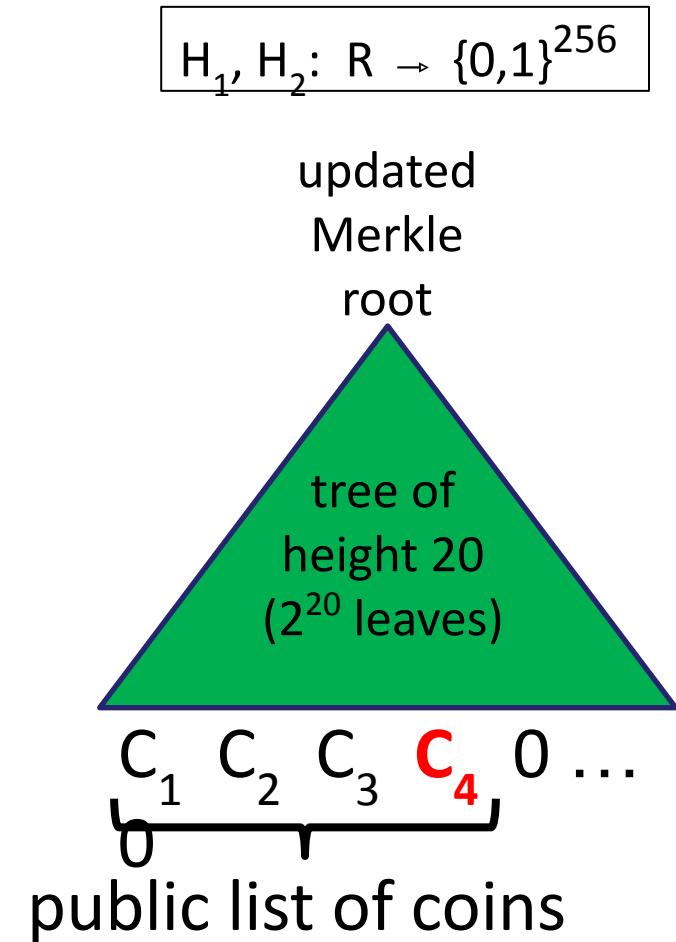
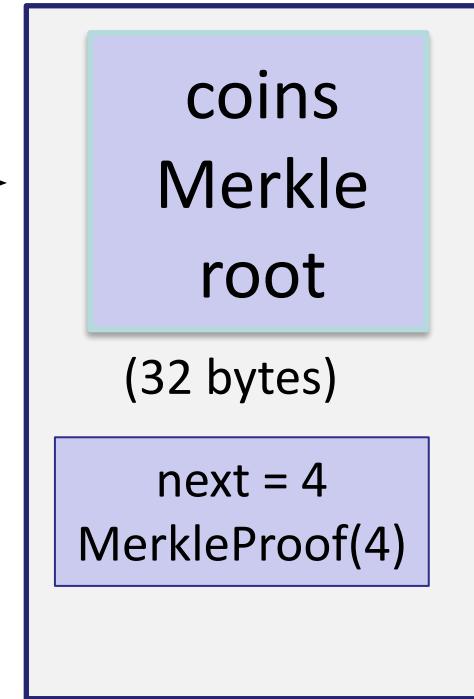


100 DAI

$C_4$ ,  $\pi = \text{MerkleProof}(5)$

## Tornado contract does:

- (1) use  $C_4$  and  $\text{MerkleProof}(4)$  to compute updated Merkle root
- (2) verify  $\pi = \text{MerkleProof}(5)$
- (3) if valid: update state

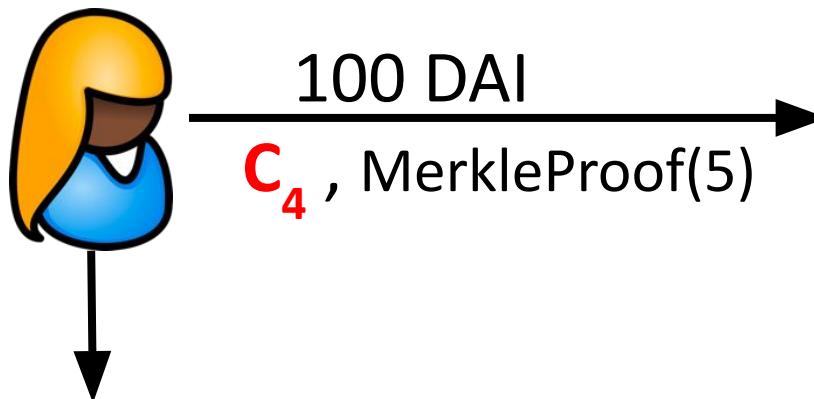


# Tornado cash: deposit (simplified)

100 DAI pool:

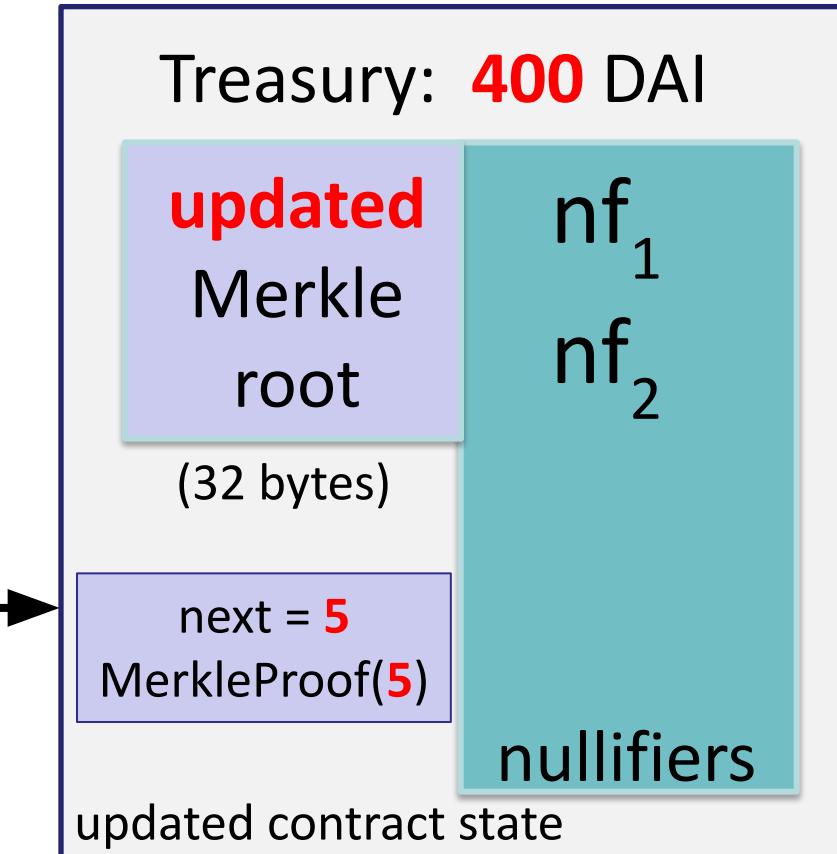
each coin = 100 DAI

Alice deposits 100 DAI:

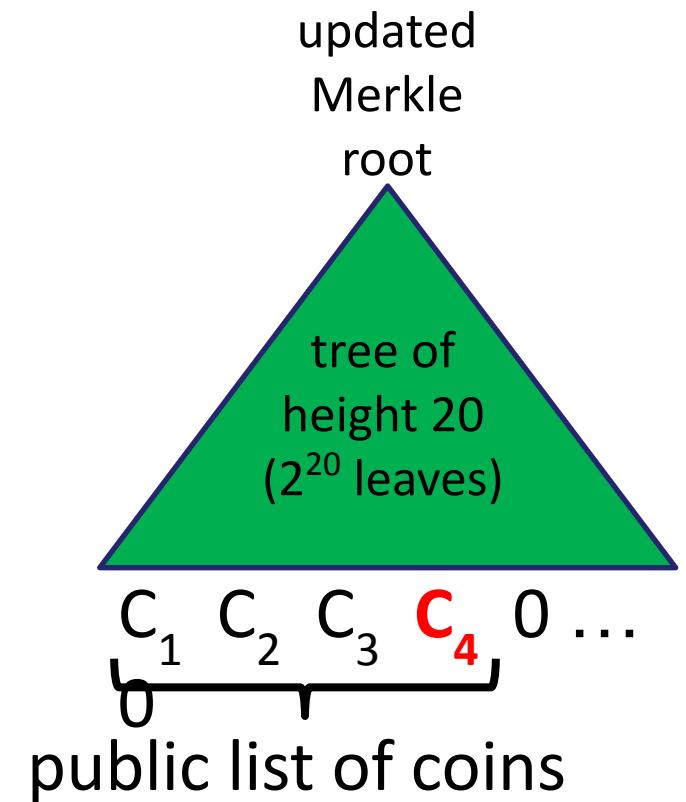


note: (k, r)

Alice keeps secret  
(one note per coin)



Every deposit: new Coin added sequentially to tree



an observer sees who owns which coins

# Tornado cash: withdrawal (simplified)

100 DAI pool:

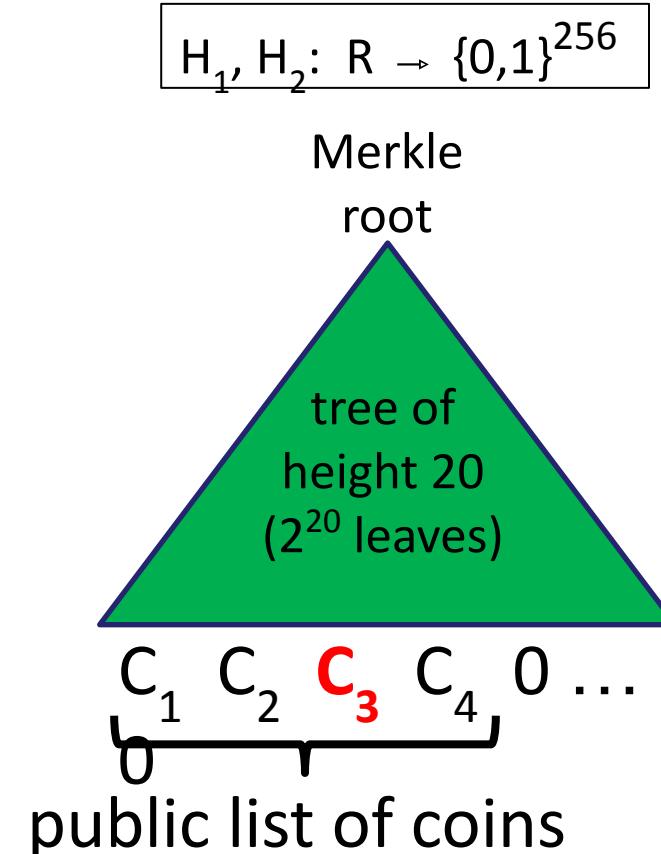
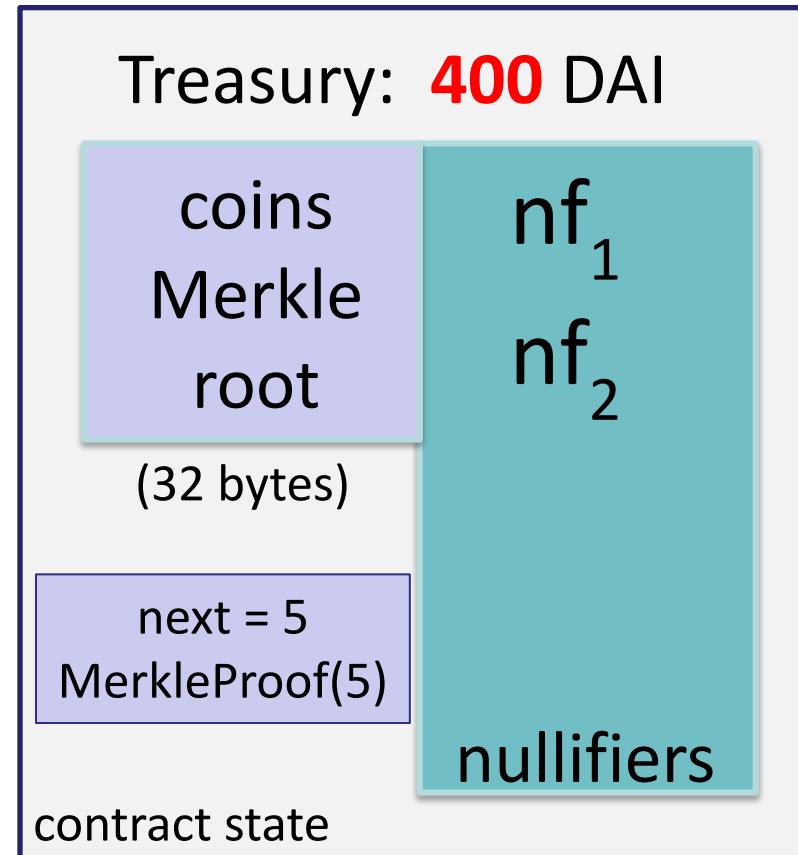
each coin = 100 DAI

Withdraw coin #3  
to addr A:



has note=  $(k', r')$

set  $\text{nf} = H_2(k')$



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)

Withdraw coin #3 to addr A:



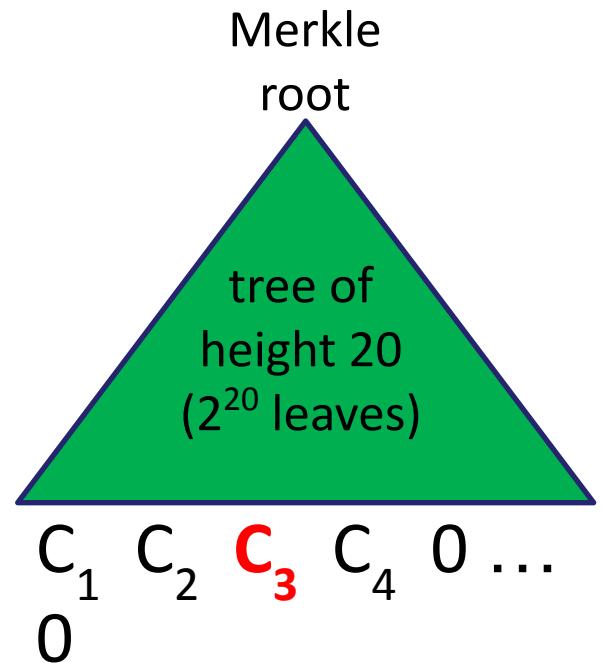
has note=  $(k', r')$       set  $\text{nf} = H_2(k')$

Bob builds zk-SNARK proof  $\pi$  for  
public statement  $x = (\text{root}, \text{nf}, A)$   
secret witness  $w = (k', r', C_3, \text{MerkleProof}(C_3))$

where  $\text{Circuit}(x, w) = 0$  iff:

- (i)  $C_3 = (\text{leaf } \#3 \text{ of } \text{root})$ , i.e.  $\text{MerkleProof}(C_3)$  is valid,
- (ii)  $C_3 = H_1(k', r')$ , and
- (iii)  $\text{nf} = H_2(k')$ .

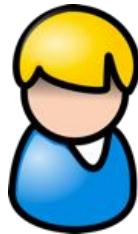
$$H_1, H_2: R \rightarrow \{0,1\}^{256}$$



(address A not used in Circuit)

# Tornado cash: withdrawal (simplified)

## Withdrawal



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  $\pi$  for  $x$  to build a proof  $\pi'$  for some “related”  $x'$  (e.g., where in  $x'$  the address A is replaced by some  $A'$ )

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

Bob builds zk-SNARK proof  $\pi$  for  
public statement  $x = (\text{root}, \text{nf}, A)$   
secret witness  $w = (k', r', C_3, \text{MerkleProof}(C_3))$

$C_1 \ C_2 \ \textcolor{red}{C_3} \ C_4 \ 0 \dots$   
0

# Tornado cash: withdrawal (simplified)

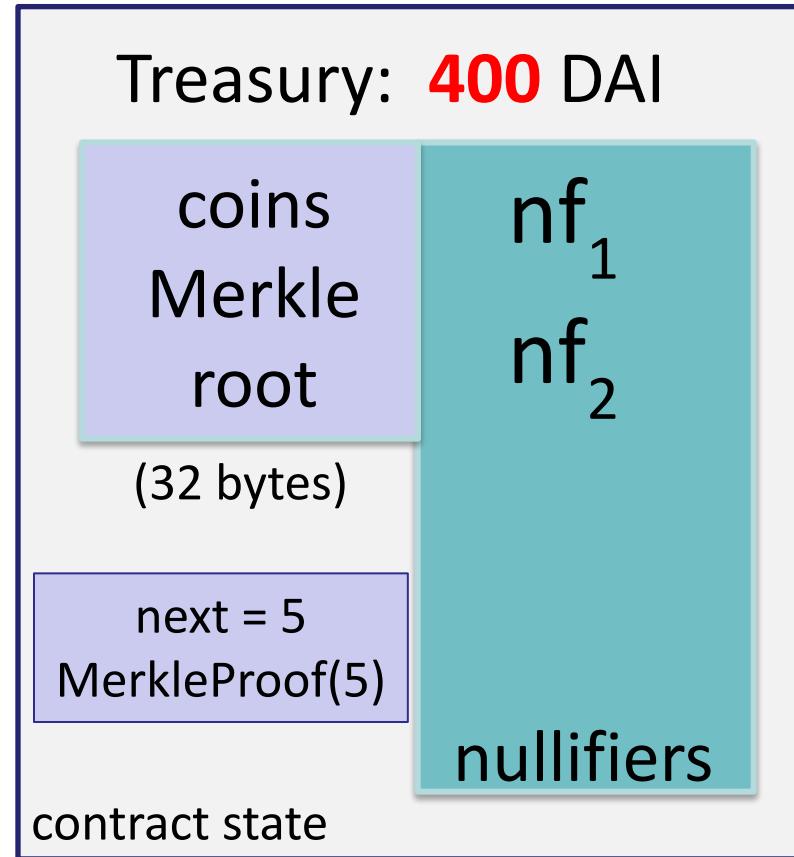
100 DAI pool:

each coin = 100 DAI

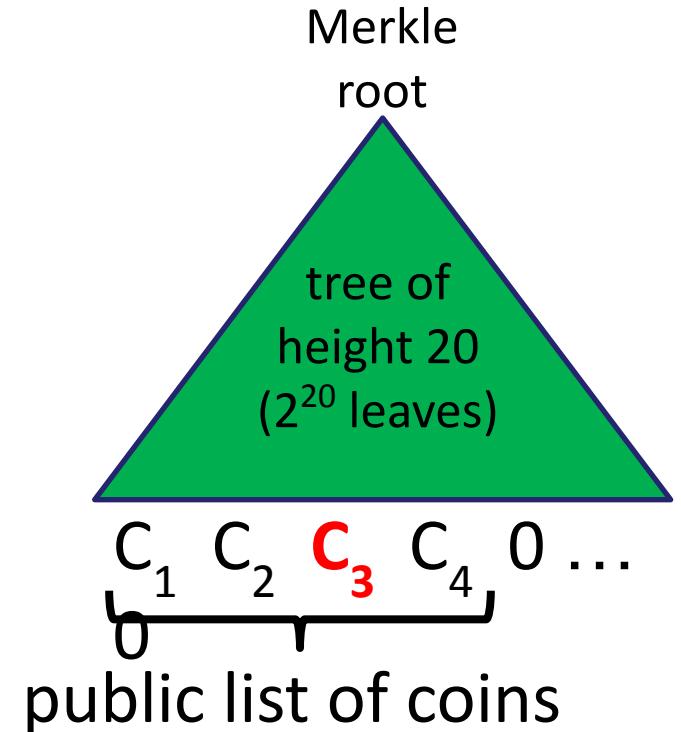
Withdraw coin #3  
to addr A:



Bob's ID and coin  $C_3$  are not revealed



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



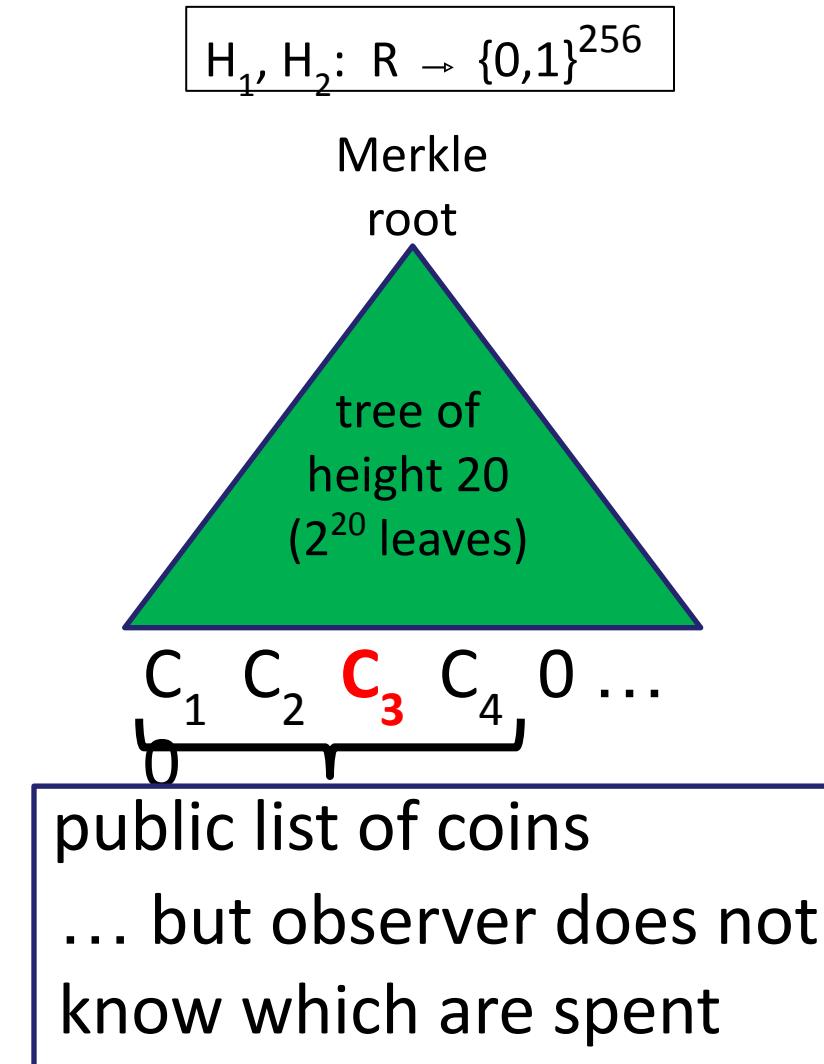
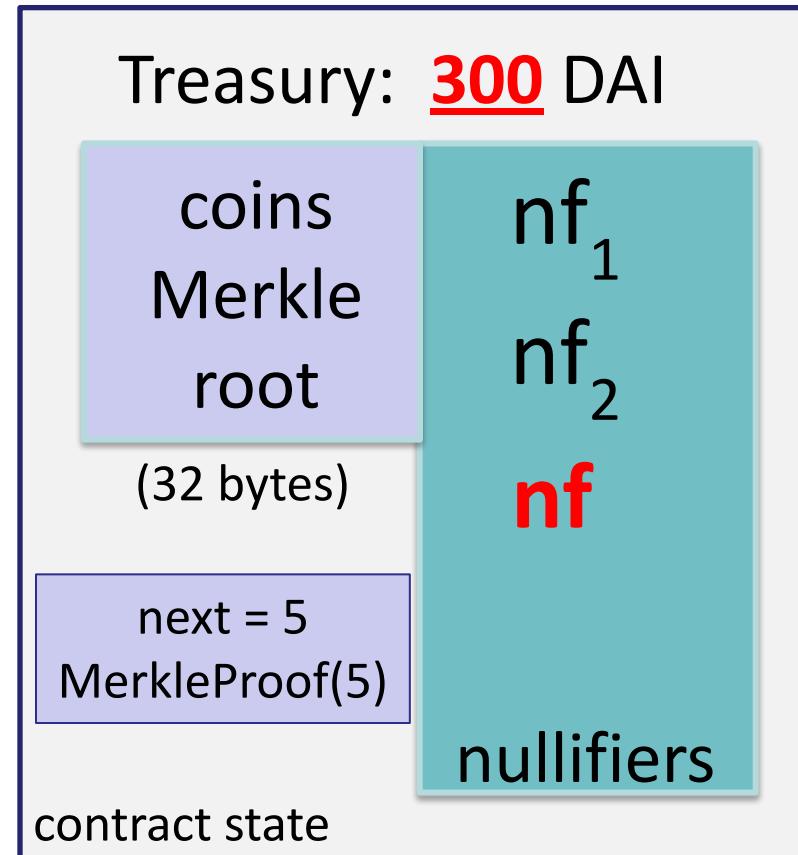
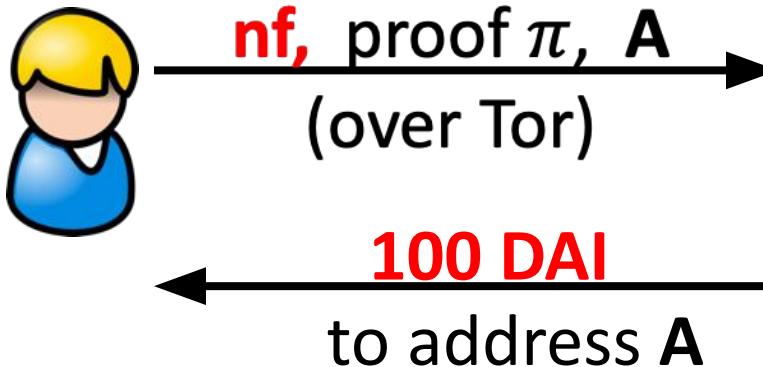
Contract checks (i) proof  $\pi$  is valid for (root, **nf**, A), and  
(ii) **nf** is not in the list of nullifiers

# Tornado cash: withdrawal (simplified)

100 DAI pool:

each coin = 100 DAI

Withdraw coin #3  
to addr A:



**nf** and  $\pi$  reveal nothing about which coin was spent.

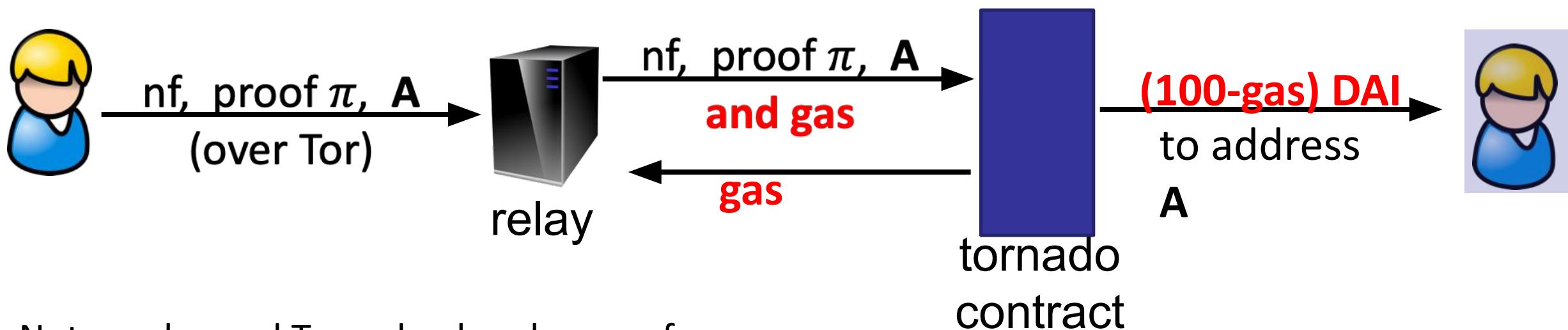
But, coin #3 cannot be spent again, because **nf = H<sub>2</sub>(k')** is now nullified.

# 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**



# Tornado Cash: the UI

**Deposit**      **Withdraw**

Token

DAI

Amount i

100 DAI      1K DAI      10K DAI      100K DAI

After deposit: get a note

(wait before withdrawing)

**Deposit**      **Withdraw**

Note i

enter note here

Recipient Address

address

Donate

Later, use note to withdraw

# Anonymity set

---

**88,036**  
Total deposits

**\$3,798,916,834**  
Total USD deposited

# leaves occupied  
over all pools

# Compliance tool

## Tornado.cash compliance tool

Maintaining financial privacy is essential to preserving our freedoms.

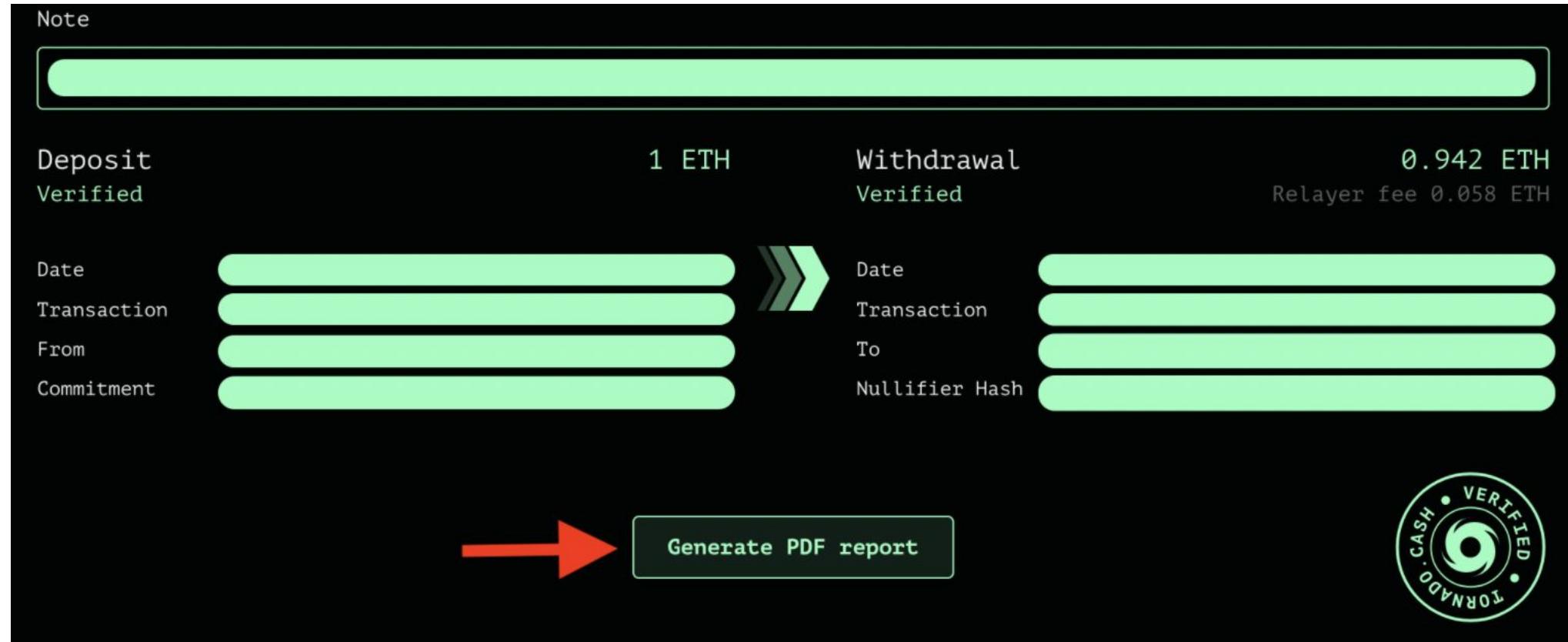
However, it should not come at the cost of non-compliance. With Tornado.cash, you can always provide cryptographically verified proof of transactional history using the Ethereum address you used to deposit or withdraw funds. This might be necessary to show the origin of assets held in your withdrawal 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.

More complicated, but similar use of Nullifiers

# Zcash / IronFish (simplified)

---

**Goal:** fully private payments ... like cash, but across the Internet  
Includes mechanisms to let parties abide by financial regulation

Zcash / IronFish supports two types of TXOs:

- **transparent** (as in Bitcoin)
- **shielded** (anonymized)

a Tx can have both types of inputs, both types of outputs

# Addresses and coins (notes)

$H_1, H_2, H_3$ : cryptographic hash functions.

sk needed to spend note  
for address pk

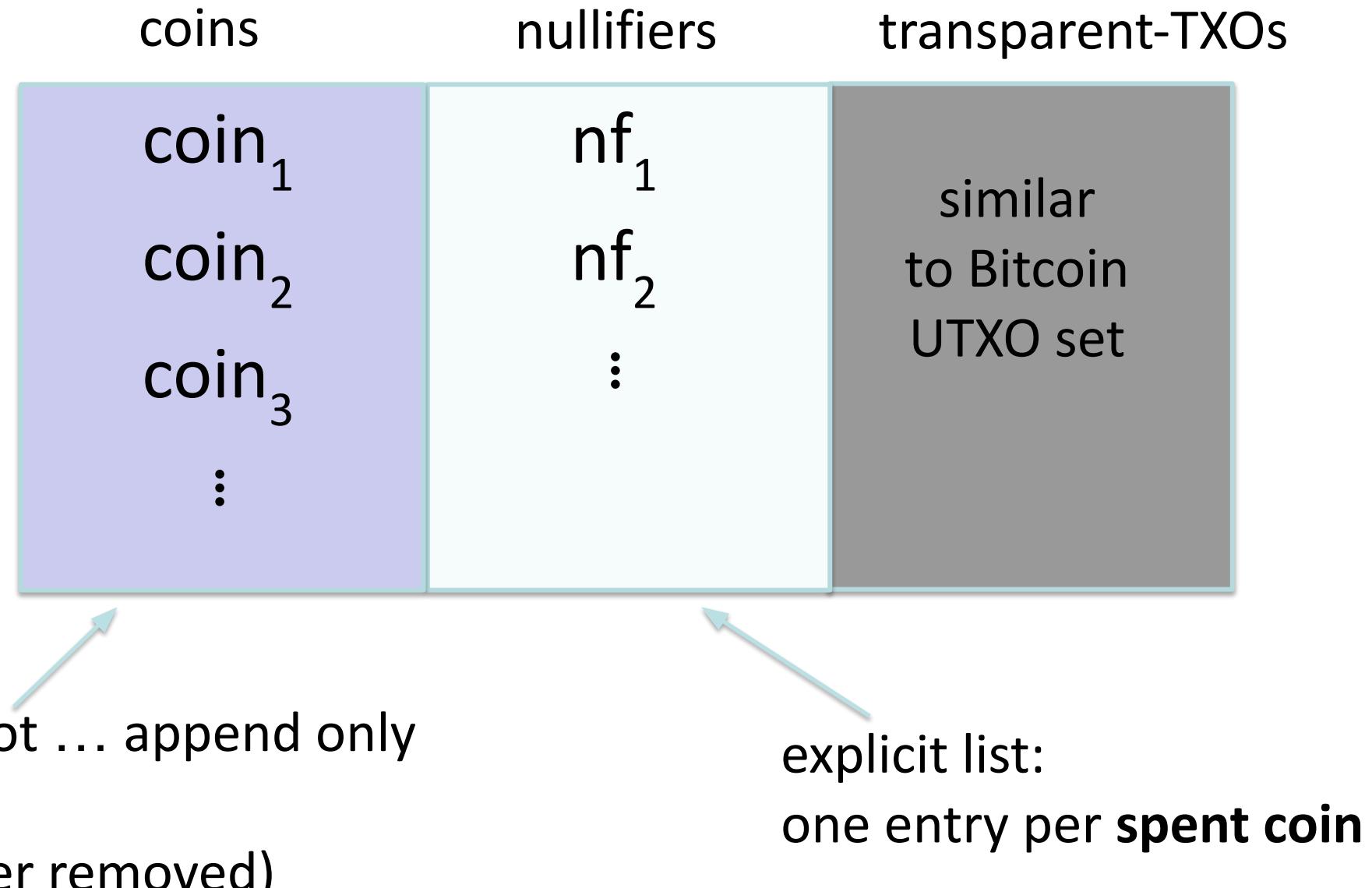
(1) **shielded address:** random  $sk \leftarrow X$ ,  $pk = H_1(sk)$

(2) **shielded coin** owned by address  $pk$ :

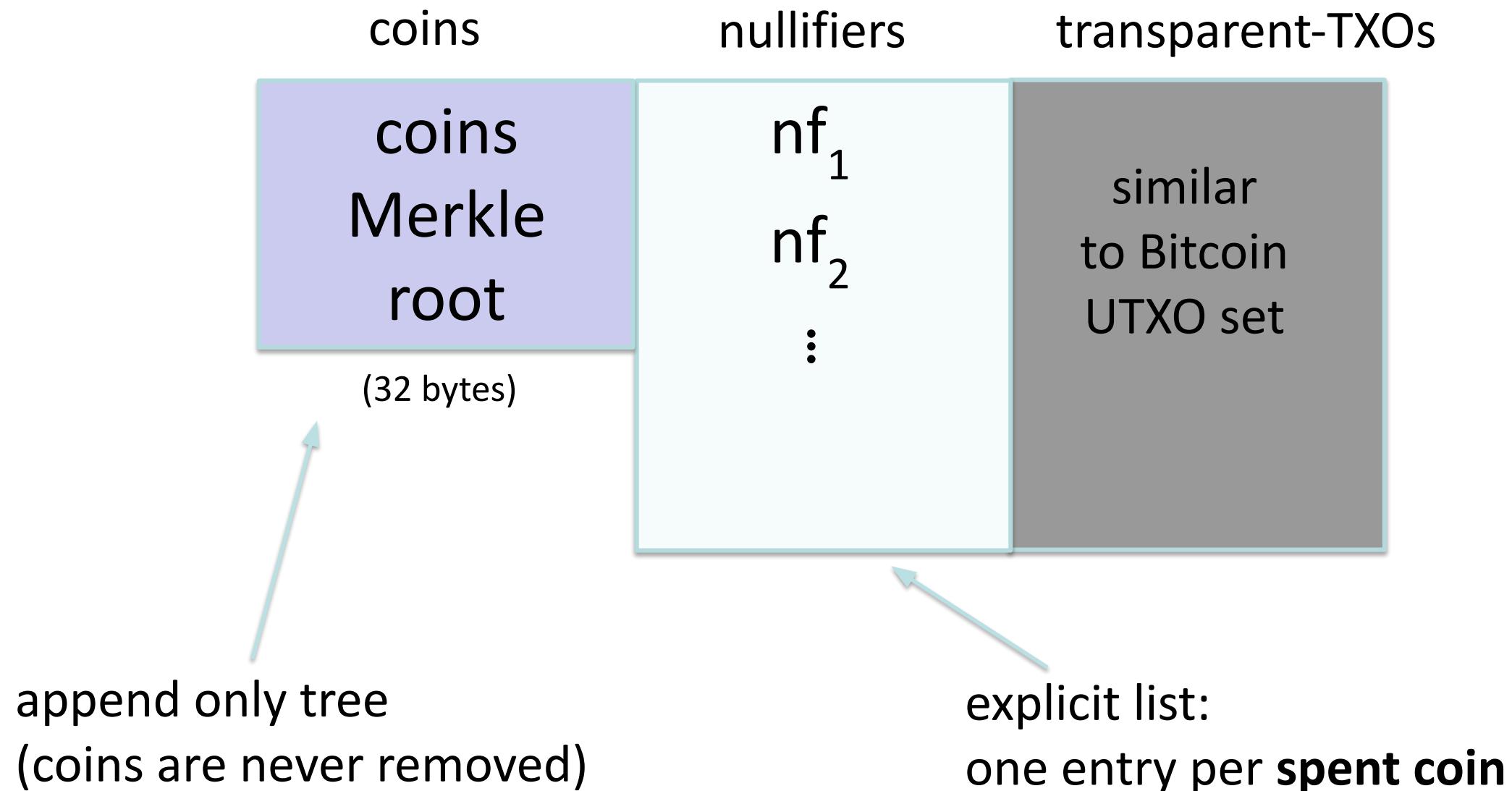
- coin owner has (from payer): value  $v$  and  $r \leftarrow R$
- on blockchain:  $coin = H_2( (pk, v) , r )$  (commitment to  $pk, v$ )

$pk$ : addr. of owner,  $v$ : value of coin,  $r$ : random chosen by payer

# The blockchain



# The blockchain



# Transactions: an example

owner of **coin** =  $H_2((pk, v), r)$

wants to send **coin** value  $v$  to:      shielded  $pk', v'$   
 $(v = v' + v')$  transp.  $pk'', v''$

---

**step 1:** construct new coin: **coin'** =  $H_2((pk', v'), r')$

by choosing random  $r' \leftarrow R$  (and send  $(v', r')$  to owner of  $pk'$ )

**step 2:** compute **nullifier** for spent coin **nf** =  $H_3(sk, \text{index of coin in Merkle tree})$

nullifier **nf** is used to “cancel” **coin** (no double spends)

key point: miners learn that some coin was spent, but not which one!

# Transactions: an example

step 3: construct a zk-SNARK proof  $\pi$  for

statement =  $x = (\text{current Merkle root}, \text{ coin}', \text{ nf}, \text{ v}'' )$

witness =  $w = ( \text{ sk}, \text{ (v, r)}, \text{ (pk', v', r')}, \text{ MerkleProof(coin)} )$

$C(x, w)$  outputs 0 if: compute  $\text{coin} := H_2((\text{pk}=H_1(\text{sk}), \text{v}), \text{r})$  and check

- The Zcash circuit
- (1) MerkleProof(**coin**) is valid,
  - (2)  $\text{coin}' = H_2((\text{pk}', \text{v}'), \text{r}')$
  - (3)  $\text{v} = \text{v}' + \text{v}''$  and  $\text{v}' \geq 0$  and  $\text{v}'' \geq 0$
  - (4)  $\text{nf} = H_3(\text{sk}, \text{index-of-coin-in-Merkle-tree})$

from  
Merkle  
proof

# What is sent to miners

---

**step 4:** send **(coin'**, **nf**, transparent-TXO, proof  $\pi$ ) to miners,  
send **( $v'$  ,  $r'$ )** to owner of  $pk'$

**step 5:** miners verify  
(i) proof  $\pi$  and transparent-TXO  
(ii) verify that **nf** is not in nullifier list (prevent double spending)  
if so, add **coin'** to Merkle tree, **add nf to nullifier list,**  
add transparent-TXO to UTXO set.

# Summary

- Tx hides which coin was spent
    - ⇒ **coin** is never removed from Merkle tree, but cannot be double spent thanks to nullifier

note: prior to spending **coin**, only owner knows **nf**:

$nf = H_3(sk, \text{index of coin in Merkle tree})$

- Tx hides address of **coin'** owner
  - Miners can verify Tx is valid, but learns nothing about Tx details

# End of lecture. Let's do a quick review.

---

## A zk-SNARK for a circuit $C$ :

- Given a public statement  $x$ , prover  $P$  outputs a proof  $\pi$  that “convinces” verifier  $V$  that prover knows  $w$  s.t.  $C(x, w) = 0$ .
- Proof  $\pi$  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

(see, e.g., cs251)

---

- How to build a zk-SNARK?
- Recursive SNARKs:
  - Proving knowledge of a SNARK proof
    - 1-level recursive statement: “I know a proof  $\pi$  that  $\exists w: C(x, w) = 0$ ”
  - Used in systems that keep business logic private
- Privately communicating with the blockchain: Nym
  - And (privately) compensating proxies for relaying traffic

A dark, grainy image of Earth at night, viewed from space. The planet's curvature is visible, and numerous glowing city lights are scattered across the continents, appearing as small yellow and white dots. Some clouds are visible as darker, textured areas.

END OF TOPIC

<https://defi-learning.org/>