CS251 Fall 2023

(cs251.stanford.edu)

Using zk-SNARKs for Privacy on the Blockchain

Dan Boneh

The need for privacy in the financial system

Supply chain privacy:

- A manufacturer does not want to reveal how much it pays its supplier for parts.

Payment privacy:

- A company that pays its employees in crypto wants to keep list of employees and salaries private.
- Endusers need privacy for rent, donations, purchases

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

Previous 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

This lecture: general tools for privacy on the blockchain

What is a zk-SNARK?

Succinct zero knowledge proofs:
an important tool for privacy on the blockchain

What is a zk-SNARK ? (intuition)

SNARK: a succinct proof that a certain statement is true

Example statement: "I know an m such that $\text{SHA256}(m) = 0$"

- **SNARK**: the proof is "**short**" and "**fast**" to verify

  [if m is 1GB then the trivial proof (the message m) is neither]

- **zk-SNARK**: the proof "reveals nothing" about m

Commercial interest in SNARKs

Many more building applications that use SNARKs

Blockchain Applications I

Outsourcing computation: (no need for zero knowledge)

L1 chain quickly verifies the work of an off-chain service

To minimize gas: need a short proof, fast to verify

Examples:

- **Scalability:** proof-based Rollups (zkRollup)

  off-chain service processes a batch of Tx;

  L1 chain verifies a succinct proof that Tx were processed correctly

- **Bridging blockchains:** proof of consensus (zkBridge)

  Chain A produces a succinct proof about its state. Chain B verifies.

Blockchain Applications II

Some applications require zero knowledge (privacy):

- **Private Tx on a public blockchain**:
  - zk proof that a private Tx is valid (Tornado cash, Zcash, IronFish, Aleo)
- **Compliance:**
  - Proof that a private Tx is compliant with banking laws (Espresso)
  - Proof that an exchange is solvent in zero-knowledge (Proven)

More on these blockchain applications in a minute

Many non-blockchain applications

Blockchains drive the development of SNARKs

… but **many** non-blockchain applications benefit

Why is all this possible now?

The breakthrough: new fast SNARK provers

- Proof generation time is linear (or quasilinear) in computation size
- **Many** beautiful ideas … next lecture

a large bibliography: a16zcrypto.com/zero-knowledge-canon

What is a SNARK?

Review: arithmetic circuits

Fix a finite field $\mathbb{F} = \{0, \dots, p-1\}$ for some prime p>2.

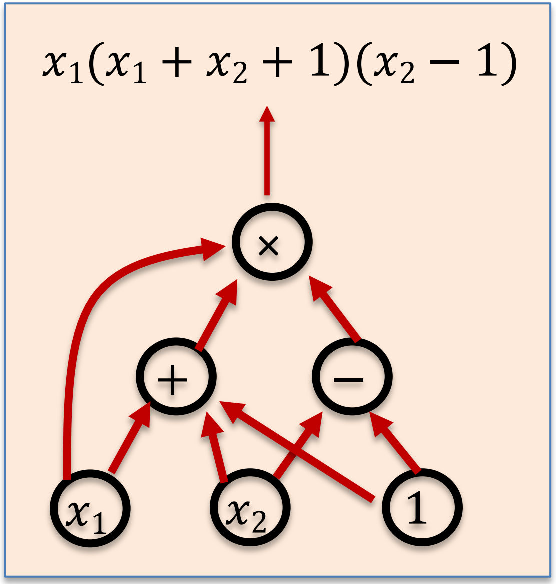
Arithmetic circuit: $C\colon \mathbb{F}^n \to \mathbb{F}$

- directed acyclic graph (DAG) where

  internal nodes are labeled $+, -,$ or $\times$

  inputs are labeled $1, x_1, \dots, x_n$
- defines an n-variate polynomial with an evaluation recipe

$|C| = \#$ gates in C



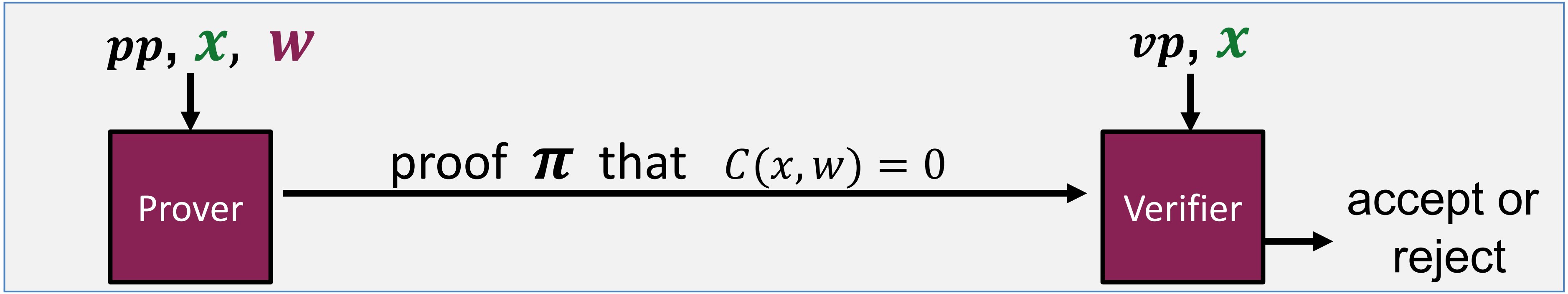
(preprocessing) NARK: Non-interactive ARgument of Knowledge

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

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

secret witness in $\mathbb{F}^m$

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



(preprocessing) NARK: Non-interactive ARgument of Knowledge

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

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

all algs. and adversary have access to a random oracle

NARK: requirements (informal)

Prover P(pp, $\boldsymbol{x}$, $\boldsymbol{w}$) Verifier V (vp, $\boldsymbol{x}$, $\boldsymbol{\pi}$)

$\xrightarrow{\text{proof } \pi}$ accept or reject

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

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

(an extractor E can extract a valid $\boldsymbol{w}$ from P)

Optional: **Zero knowledge**: $(C, pp, vp, \boldsymbol{x}, \pi)$ "reveal nothing new" about $\boldsymbol{w}$

(witness exists $\Rightarrow$ can simulate the proof)

SNARK: a Succinct ARgument of Knowledge

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

- **S**(C) → public parameters (pp, vp) for prover and verifier
- **P**$(pp, \boldsymbol{x}, \boldsymbol{w})$ → **short** proof π ; $\boxed{\text{len}(\pi) = O_\lambda(\,\textbf{polylog}(|\boldsymbol{C}|))}$
- **V**$(vp, \boldsymbol{x}, \boldsymbol{\pi})$ **fast to verify** ; $\boxed{\text{time(V)} = O_\lambda(|x|,\ \textbf{polylog}(|\boldsymbol{C}|))}$

short "summary" of circuit

V has no time to read C !!

[for some SNARKs, $\text{len}(\pi) = \text{time}(V) = O_\lambda(1)$]

SNARK: a Succinct ARgument of Knowledge

SNARK: a NARC (complete and knowledge sound) that is **succinct**

zk-SNARK: a SNARK that is also **zero knowledge**

Types of preprocessing Setup

Recall setup for circuit C: **S**$(C; r)$ $\rightarrow$ public parameters (pp, vp)

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 $\Rightarrow$ can prove false statements

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

$\boldsymbol{S} = (S_{init}, S_{index})$: $\underbrace{S_{init}(\lambda; r) \rightarrow gp,}_{\text{one-time}}$ $\underbrace{S_{index}(gp, C) \rightarrow (pp, vp)}_{\text{no secret data from prover}}$

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

Significant progress in recent years (partial list)

	size of proof π	verifier time	Setup	post-quantum?
Groth'16	≈ 200 Bytes $O_\lambda(1)$	$\approx$ 1.5 ms $O_\lambda(1)$	trusted per circuit	no
Plonk / Marlin	≈ 400 Bytes $O_\lambda(1)$	$\approx$ 3 ms $O_\lambda(1)$	universal trusted setup	no

(for a circuit with 2^{20} gates)

Significant progress in recent years (partial list)

	size of proof π	verifier time	setup	post-quantum?
Groth'16	$\approx$ 200 Bytes $O_\lambda(1)$	$\approx$ 1.5 ms $O_\lambda(1)$	trusted per circuit	no
Plonk / Marlin	$\approx$ 400 Bytes $O_\lambda(1)$	$\approx$ 3 ms $O_\lambda(1)$	universal trusted setup	no
Bulletproofs	$\approx$ 1.5 KB $O_\lambda(\log \lvert C \rvert)$	$\approx$ 3 sec $O_\lambda(\lvert C \rvert)$	transparent	no
STARK	$\approx$ 100 KB $O_\lambda(\log^2 \lvert C \rvert)$	$\approx$ 10 ms $O_\lambda(\log \lvert C \rvert)$	transparent	yes

⋮ (for a circuit with 2^{20} gates) ⋮

Significant progress in recent years (partial list)

	size of proof π	verifier time	setup	post-quantum?
Groth'16				
Plonk / Marlin				
Bulletproofs				
STARK	$O_\lambda(\log^2 \lvert C \rvert)$	$O_\lambda(\log \lvert C \rvert)$		

⋮ (for a circuit with 2^{20} gates) ⋮

Prover time is almost linear in $|C|$

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

Definitions: (1) knowledge sound (simplified)

<u>Formally</u>: a universal SNARK (S, P, V) is **knowledge sound** if

for every poly. time adversary $A = (A_0, A_1)$ there exists a poly. time **extractor** Ext (that uses A as a black box) s.t.

$$\text{if} \quad gp \leftarrow S_{init}(\,), \quad (C, x, \text{state}) \leftarrow A_0(gp), \quad (pp, vp) \leftarrow S_{index}(gp, C),$$

$$\pi \leftarrow A_1(pp, x, \text{state}), \qquad w \leftarrow Ext(gp, C, x)$$

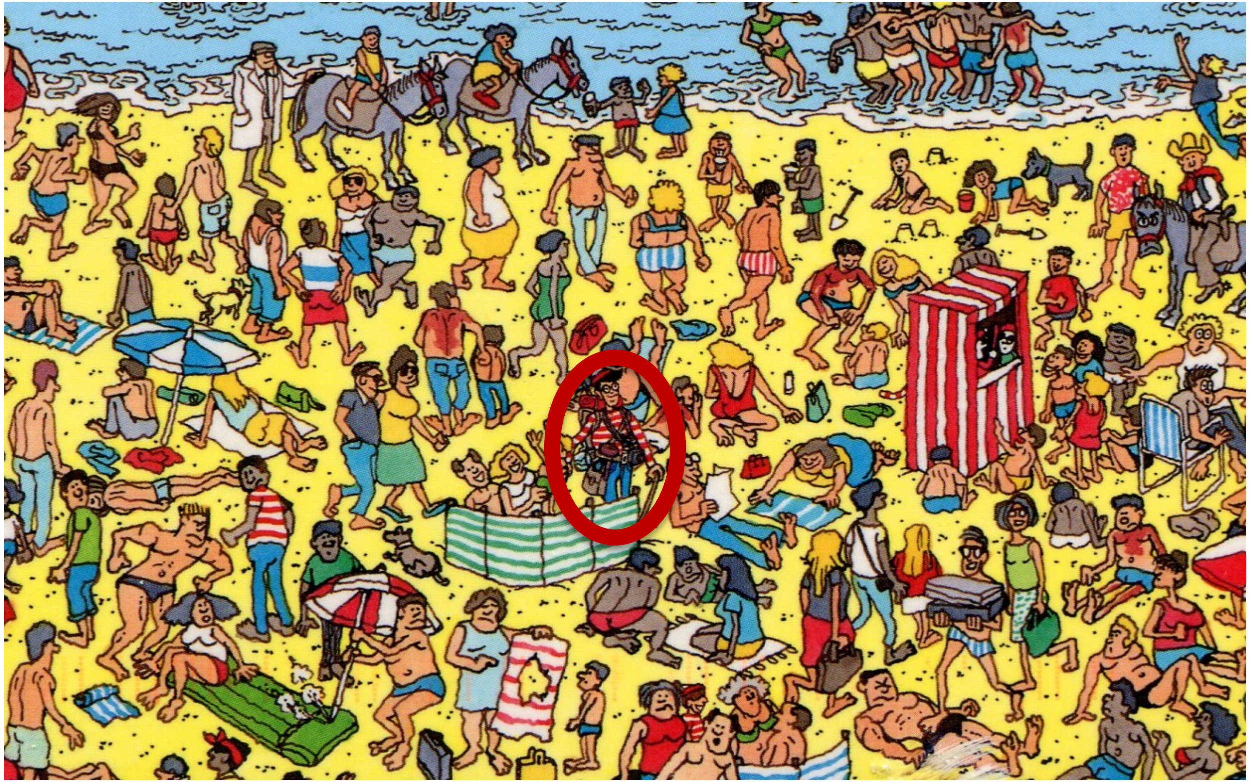
extracted witness

Then

$$\Pr\big[\, V(vp, x, \pi) = \text{accept} \;\Rightarrow\; C(x, w) = 0 \,\big] \geq 1 - \epsilon \qquad (\text{for a negl. } \epsilon)$$

Definitions: (2) Zero knowledge

Where is
Waldo?



Definitions: (2) Zero knowledge (simplified)

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

proof π "reveals nothing" about $\boldsymbol{w}$, other than its existence

What does it mean to "reveal nothing" ??

Informal def: π "reveals nothing" about $\boldsymbol{w}$ 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. $(pp, vp, \pi) \leftarrow$ ***Sim***(C, x) "look like" the real pp,vp and π.

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

Definitions: (2) Zero knowledge (simplified)

<u>**Formally**</u>: (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:

(C, pp, vp, x, π): where $(pp, vp) \leftarrow$ S(C), $\pi \leftarrow$ P($pp, x, \boldsymbol{w}$)

is indistinguishable from the distribution:

(C, pp, vp, x, π): where $(pp, vp, \pi) \leftarrow$ ***Sim***(C, x)

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

How to build a zk-SNARK?

Recall: prover generates a **short** proof that is **fast** to verify

How to build a zk-SNARK ??

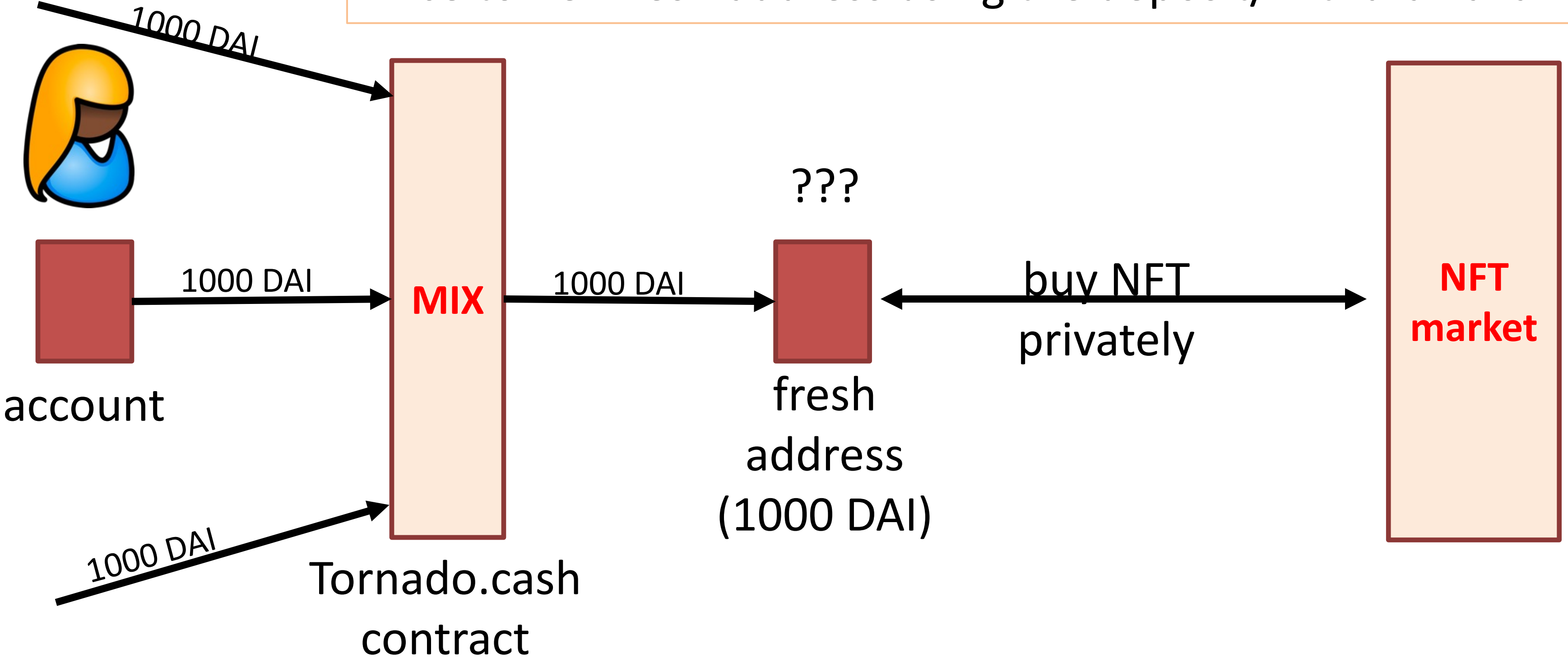
Next lecture

Applications of SNARKs:
(1) Tornado cash: a zk-based mixer

Launched on the Ethereum blockchain on May 2020 (v2)

Tornado Cash: a ZK-mixer

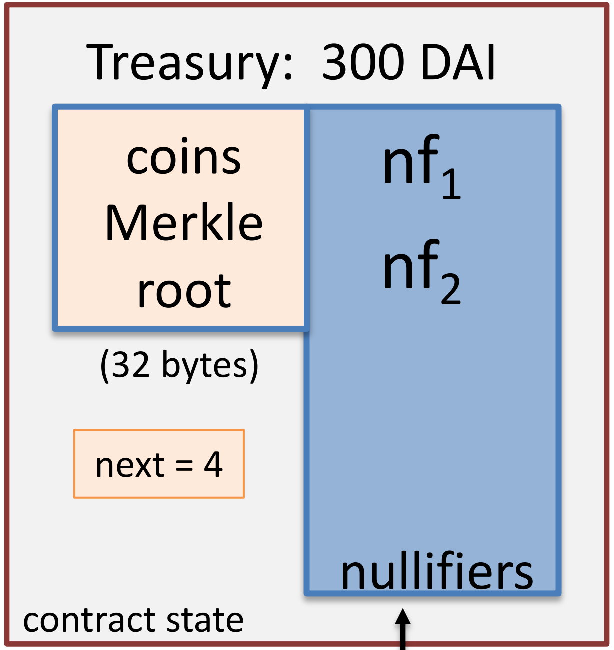
A common denomination (1000 DAI) is needed to prevent linking Alice to her fresh address using the deposit/withdrawal amount



The tornado cash contract (simplified)

100 DAI pool:
each coin = 100 DAI

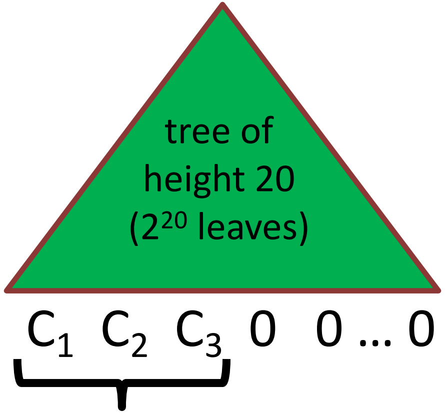
Currently:
- three coins in pool
- contract has 300 DAI
- two nullifiers stored



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

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

Coins Merkle root



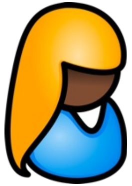
public list of coins

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)

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

choose random k, r in R

set $C_4 = H_1(k, r)$

Treasury: 300 DAI

coins Merkle root

(32 bytes)

next = 4

nf_1

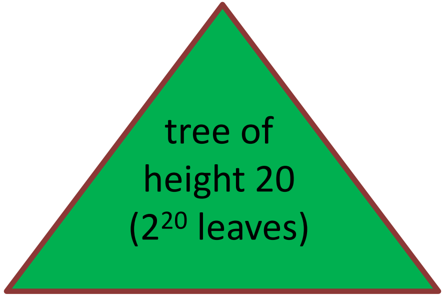
nf_2

nullifiers

contract state

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

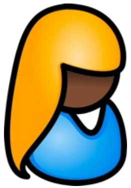
Coins Merkle root



C_1 C_2 C_3 0 0 … 0

public list of coins

Tornado cash: deposit (simplified)



100 DAI →

C_4 , MerkleProof(4)

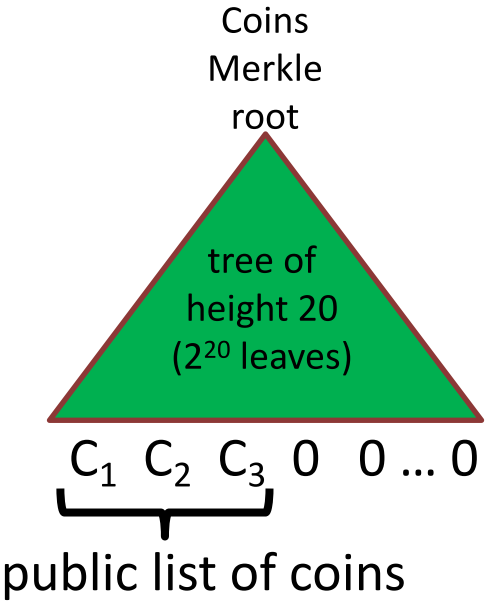
coins Merkle root

(32 bytes)

next = 4

Tornado contract

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



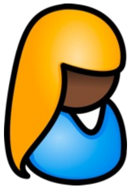
Tornado contract does:

(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

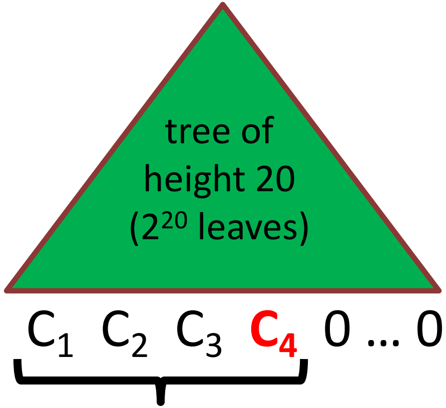
(32 bytes)

next = 4

Tornado contract

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

updated Merkle root



public list of coins

Tornado contract does:

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

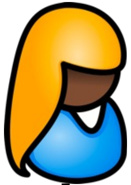
(2) use C_4 and MerkleProof(4) to compute updated Merkle root

(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
(32 bytes)

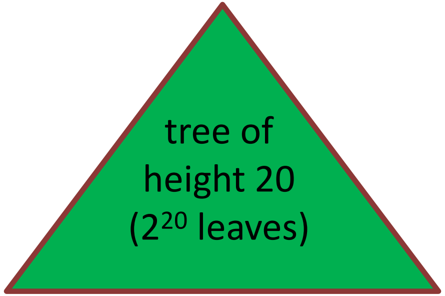
next = **5**

nf_1
nf_2
nullifiers

updated contract state

Every deposit: new Coin added sequentially to tree

updated Merkle root



C_1 C_2 C_3 $\mathbf{C_4}$ 0 … 0

public list of coins

an observer sees who owns which leaves

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

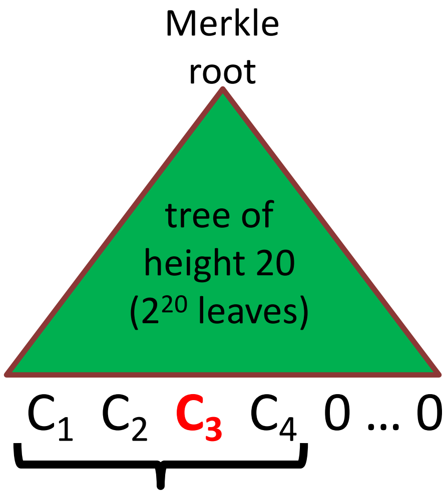
coins Merkle root	nf_1 nf_2 nullifiers

(32 bytes)

next = 5

contract state

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



public list of coins

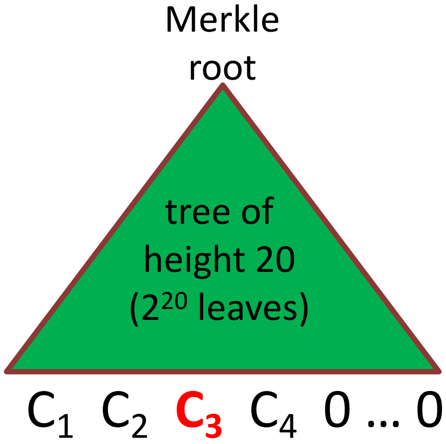
Bob proves "I have a note for some leaf in the coins tree, and its nullifier is **nf**"
(without revealing which coin)

Tornado cash: withdrawal (simplified)

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

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 $w = \left(k', r', C_3, \text{MerkleProof}(C_3)\right)$

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

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')

Bob builds zk-SNARK proof π for

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

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

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

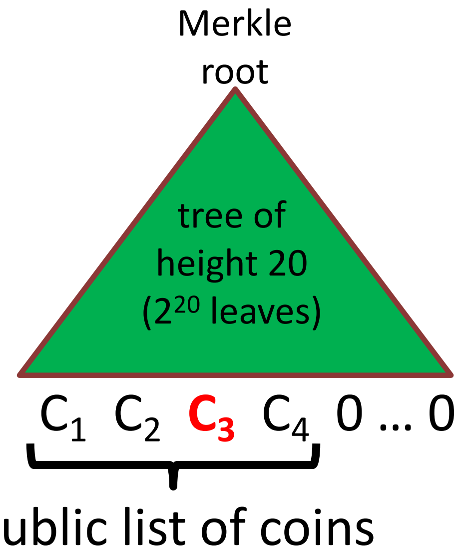
coins Merkle root
(32 bytes)

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
(32 bytes)

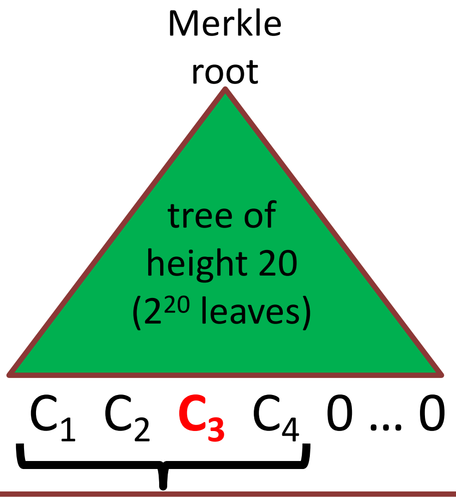
next = 5

nf_1
nf_2
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.

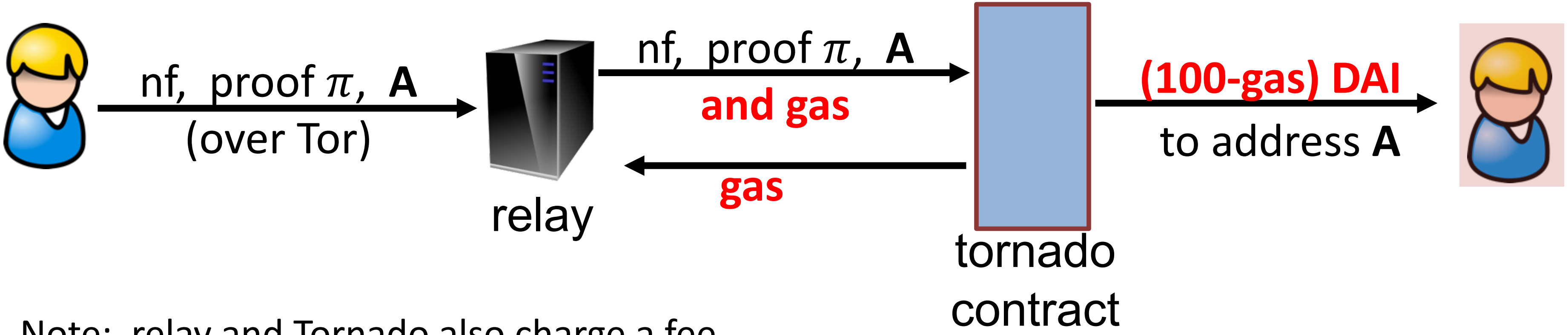
But, coin #3 cannot be spent again, because **nf = $H_2(k')$** is now nullified.

Who pays the withdrawal gas fee?

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

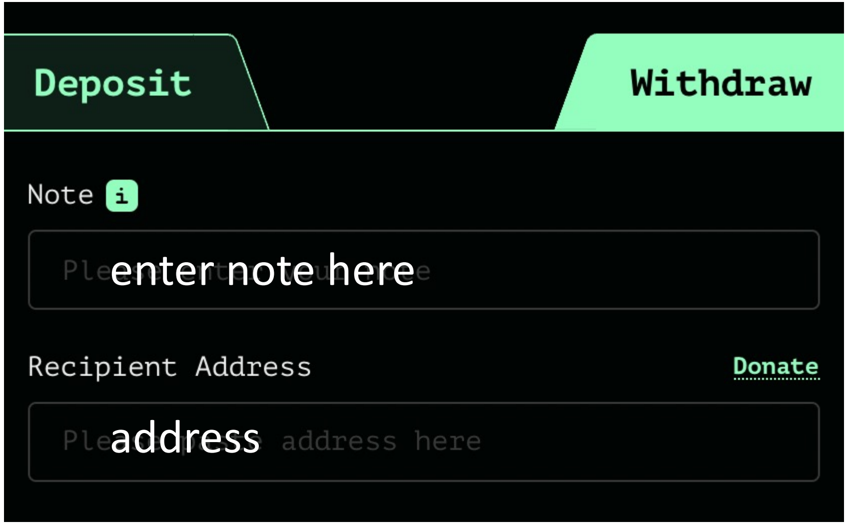
- If paid from Bob's address, then fresh address is linked to Bob

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



Note: relay and Tornado also charge a fee

Tornado Cash: the UI



After deposit: get a note

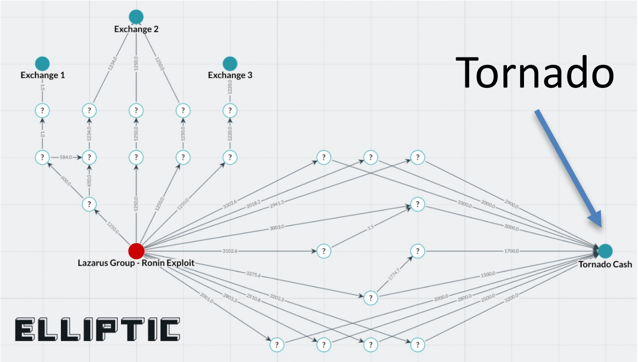
Later, use note to withdraw

(wait before withdrawing)

Tornado trouble … U.S. sanctions

The Ronin-bridge hack (2022):

- In late March: ≈600M USD stolen … $80M USD sent to Tornado
- April: Lazarus Group suspected of hack
- August: "U.S. Treasury Sanctions Virtual Currency Mixer Tornado Cash"
  - Lots of collateral damage … and two lawsuits



The lesson: complete anonymity in the payment system is problematic

Sanctions

"U.S. persons would not be prohibited by U.S. sanctions regulations from copying the open-source code and making it available online for others to view, as well as discussing, **teaching about**, or including open-source code in written publications, such as textbooks, absent additional facts"

U.S. Treasury FAQ, Sep. 2022

Designing a compliant Tornado??

(1) **deposit filtering**: ensure incoming funds are not sanctioned

Chainalysis **SanctionsList** contract:

```
function isSanctioned(address addr) public view returns (bool) {
        return sanctionedAddresses[addr] == true ;
}
```

Reject funds coming from a sanctioned address.

Difficulties: (1) centralization, (2) slow updates

Designing a compliant Tornado??

(2) Withdrawal filtering: at withdrawal, require a ZK proof that the source of funds is not currently on sanctioned list.

How?

- modify the way Tornado computes Merkle leaves during deposit to include **msg.sender**.

  in our example Alice sets: $\color{red}{\mathbf{C_4}}$ = [H_1(k, r), **msg.sender**]

- During withdrawal Bob proves in ZK that **msg.sender** in his leaf is not <u>currently</u> on sanctions list.

Designing a compliant Tornado??

(3) Viewing keys: at withdrawal, require nullifier to include an encryption of deposit msg.sender under government public key.

How? Merkle leaf $\mathbf{C_4}$ is computed as on previous slide.

- During withdrawal Bob sets nullifier **nf** = [H_2(k'), ct, π]
  where (i) ct = Enc(pk, msg.sender) and
  (ii) π is ZK proof that ct is computed correctly

⇒ As needed, government can trace funds through Tornado

- lots of problems with this design …

Other private Tx projects

Zcash / IronFISH: private payments

- L1 blockchains that extend Bitcoin, similar use of Nullifiers.
- Support for any value Tx and in-system transfers.

Aztec / Aleo:

- Support for private Tx interacting with a public smart contract.
- Aleo: an L1 blockchain. Aztec: runs on top of Ethereum.

END OF LECTURE

Next lecture: how to build a SNARK

Further topics

Privately communicating with the blockchain: Nym

- How to privately compensate proxies for relaying traffic

Next lecture: how to build a SNARK