

CS251 Fall 2025
(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

Review: 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

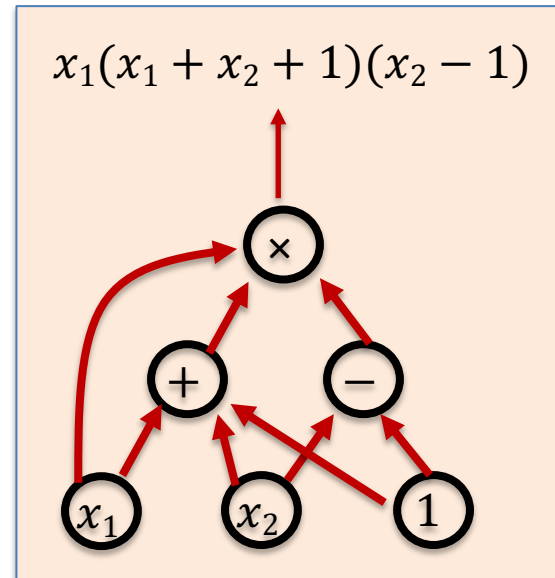
Review: arithmetic circuits

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

$|C| = \# \text{ gates in } C$



(preprocessing) **NARK: Non-interactive ARgument of Knowledge**

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

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

all algs. and adversary have
access to a random oracle

NARK: requirements (informal)

Prover $P(pp, \mathbf{x}, \mathbf{w})$

Verifier $V(vp, \mathbf{x}, \pi)$

————— proof π —————→ accept or reject

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

Adaptively knowledge sound: V accepts $\Rightarrow P$ “knows” \mathbf{w} s.t. $C(\mathbf{x}, \mathbf{w}) = 0$
(an extractor E can extract a valid \mathbf{w} from P)

Optional: Zero knowledge: $(C, pp, vp, \mathbf{x}, \pi)$ “reveal nothing new” about \mathbf{w}
(witness exists \Rightarrow can simulate the proof)

SNARK: a Succinct ARgument of Knowledge

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

$$\left\{ \begin{array}{l} \text{short: } \text{len}(\pi) = O_{\lambda}(\text{polylog}(|C|)) \\ \text{fast: } \text{time}(V) = O_{\lambda}(|x|, \text{polylog}(|C|)) \end{array} \right.$$

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


Types of setup:

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

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

transparent setup: $S(C)$ does not use secret data (no trusted setup)

better


Significant progress in recent years (partial list)

	size of proof π	verifier time	Setup	post- quantum?
Groth'16	≈ 200 Bytes $O_\lambda(1)$	≈ 1.5 ms $O_\lambda(1)$	trusted per circuit	no
Plonk	≈ 400 Bytes $O_\lambda(1)$	≈ 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	≈ 200 Bytes $O_\lambda(1)$	≈ 1.5 ms $O_\lambda(1)$	trusted per circuit	no
Plonk	≈ 400 Bytes $O_\lambda(1)$	≈ 3 ms $O_\lambda(1)$	universal trusted setup	no
Bulletproofs	≈ 1.5 KB $O_\lambda(\log C)$	≈ 3 sec $O_\lambda(C)$	transparent	no
Hash-based proofs (e.g., STARK)	≈ 100 KB $O_\lambda(\log^2 C)$	≈ 10 ms $O_\lambda(\log^2 C)$	transparent	yes

⋮

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

⋮

Significant progress in recent years (partial list)

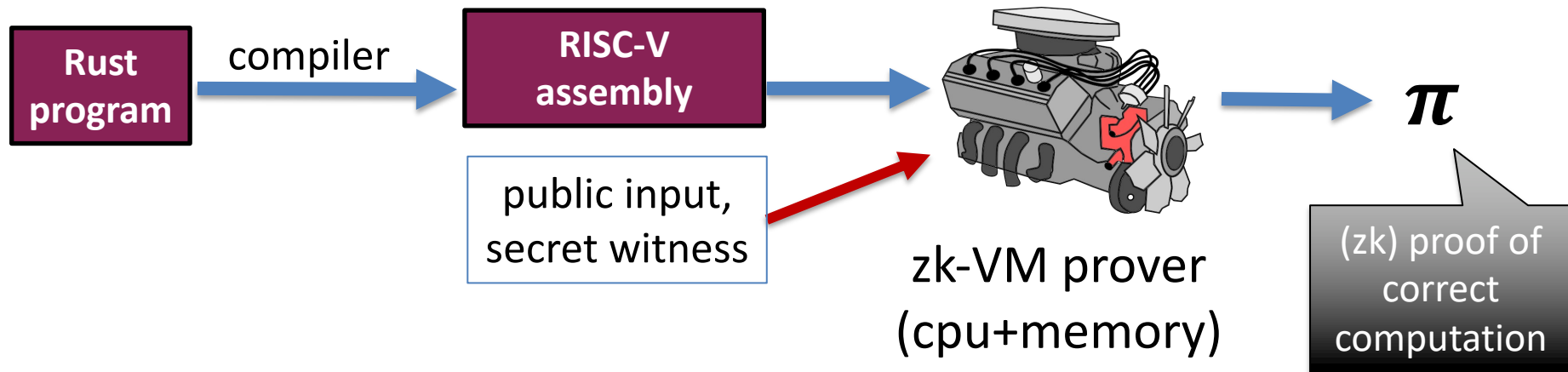
	size of proof π	verifier time	setup	post- quantum?
Groth'16	<div>Prover time is almost linear in \mathcal{C}</div>			
Plonk				
Bulletproofs				
Hash-based proofs (e.g., STARK)				
	$O_\lambda(\log^2 \mathcal{C})$	$O_\lambda(\log \mathcal{C})$		

⋮

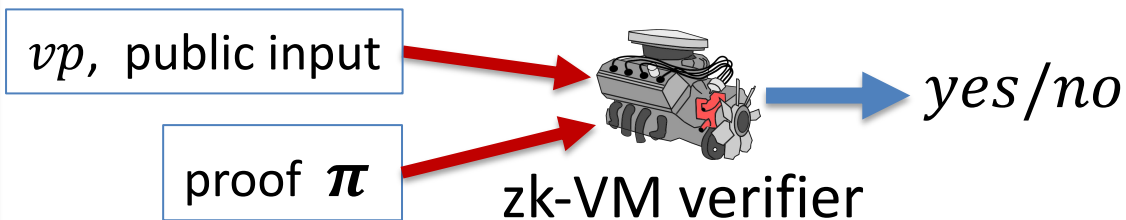
(for a circuit with 2^{20} gates)

⋮

More generally: proving computation (zk-VM)



Current proving speeds:
 ≈ 30 M RISC-V / sec
(on a single GPU)



Why is 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

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

Definitions: (1) knowledge sound

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) knowledge sound (simplified)

Formally: 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.

if $gp \leftarrow S_{init}()$, $(C, x, \text{state}) \leftarrow A_0(gp)$, $(pp, vp) \leftarrow S(gp, C)$,
 $\pi \leftarrow A_1(pp, x, \text{state})$, $w \leftarrow Ext(gp, C, x)$

 extracted witness

Then

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

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 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** \Rightarrow it learned nothing new from π

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

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

Definitions: (2) Zero knowledge (simplified)

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:

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

is indistinguishable from the distribution:

(C, pp, vp, x, π) : where $(pp, vp, \pi) \leftarrow \mathbf{Sim}(C, x)$

Main point: ***Sim*** (C, x) simulates π without knowledge of \mathbf{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

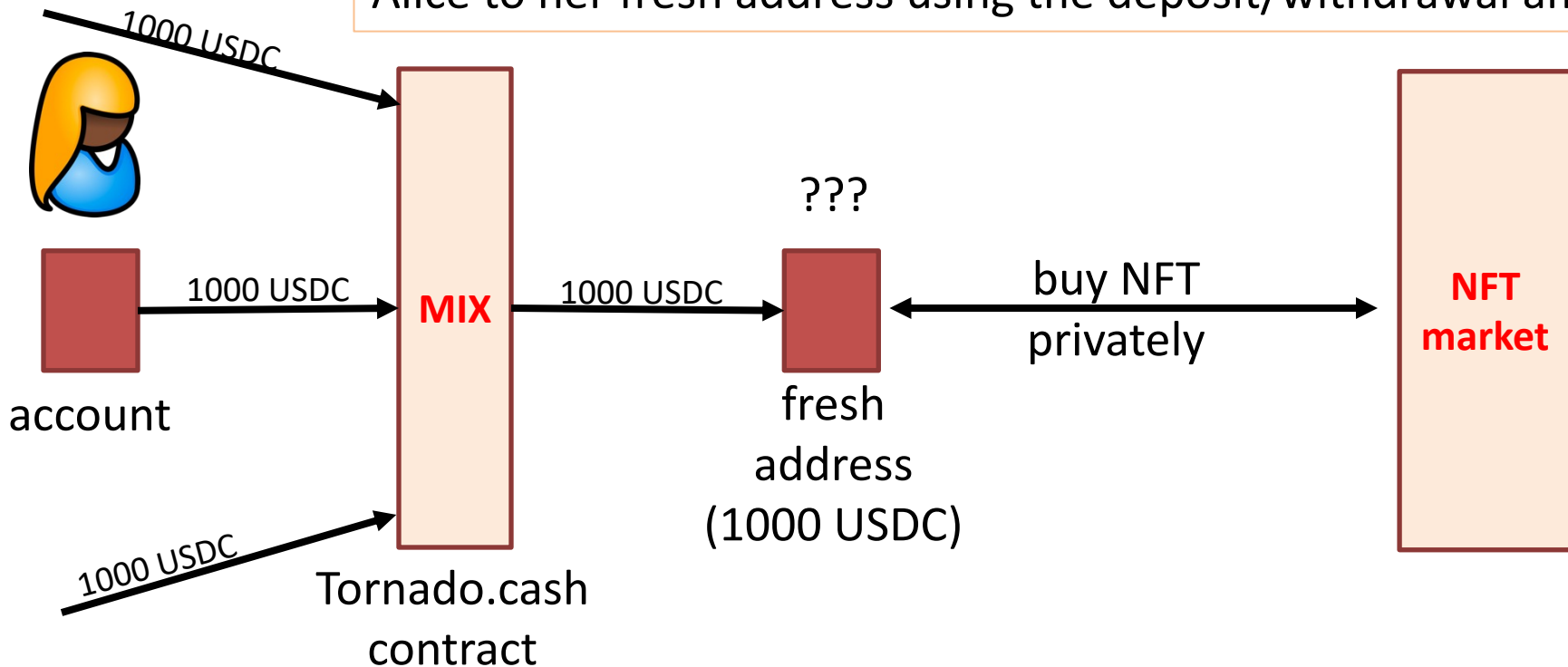
Applications of zk-SNARKs:

(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 USDC) is needed to prevent linking Alice to her fresh address using the deposit/withdrawal amount



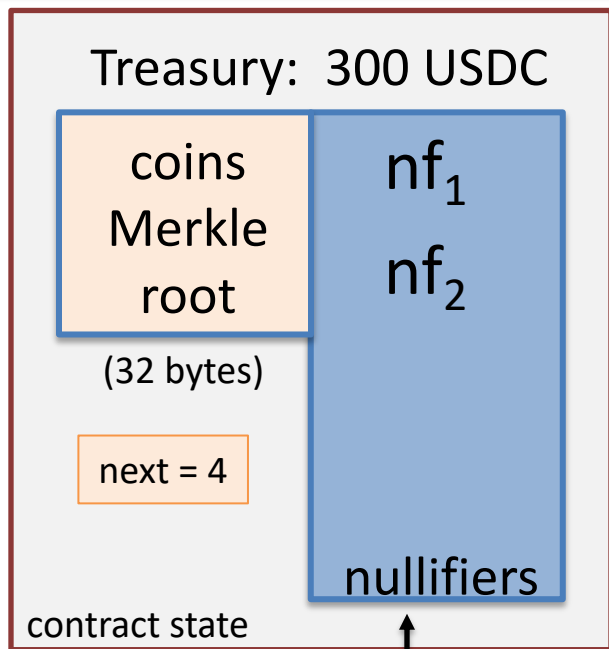
The tornado cash contract (simplified)

100 USDC pool:

each coin = 100 USDC

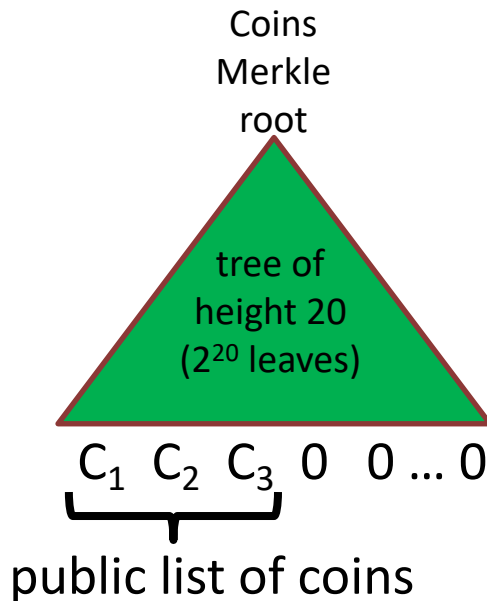
Currently:

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



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

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



Tornado cash: deposit

(simplified)

100 USDC pool:

each coin = 100 USDC

Alice deposits 100 USDC:



100 USDC

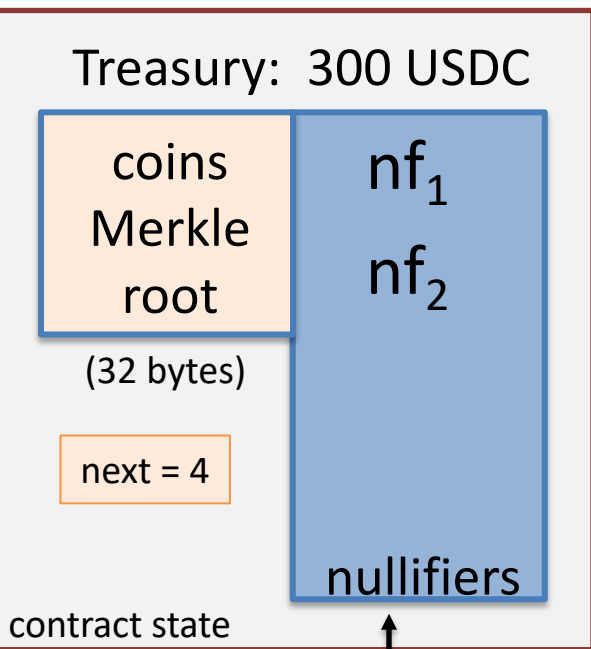
C_4 , MerkleProof(4)

Build Merkle proof for leaf #4:

MerkleProof(4) (leaf=0)

choose random k, r in R

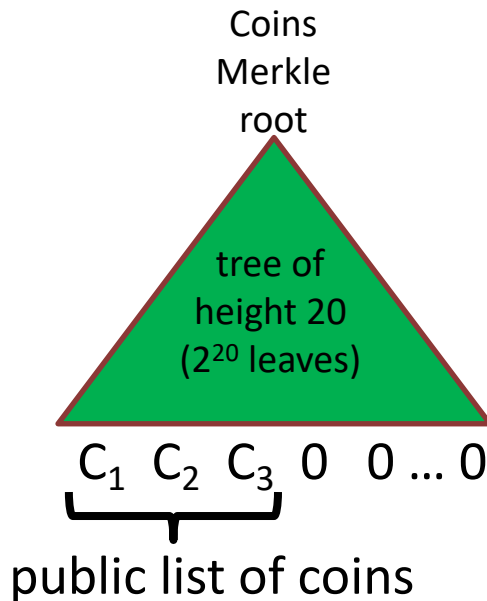
set $C_4 = H_1(k, r)$



explicit list:

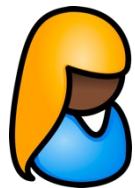
one entry per **spent coin**

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



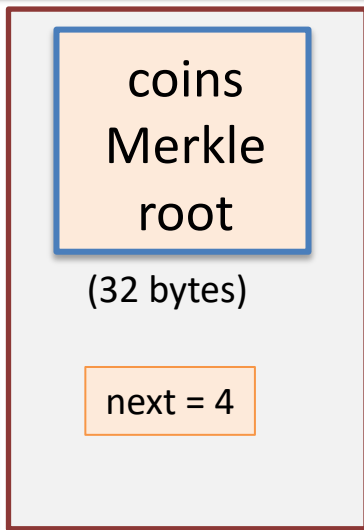
Tornado cash: deposit

(simplified)



100 USDC

C_4 , MerkleProof(4)



Tornado contract

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

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

Coins
Merkle
root

tree of
height 20
(2^{20} leaves)

C_1 C_2 C_3 0 0 ... 0

public list of coins

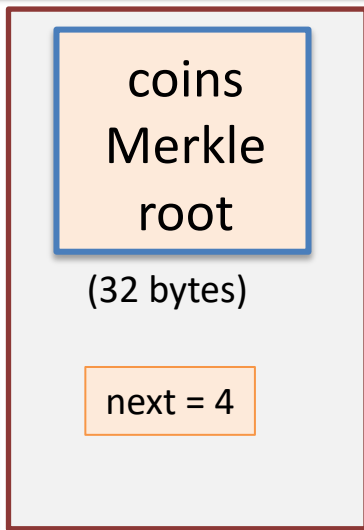
Tornado cash: deposit

(simplified)



100 USDC

C_4 , MerkleProof(4)



Tornado contract

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

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

updated
Merkle
root

tree of
height 20
(2^{20} leaves)

C_1 C_2 C_3 C_4 0 ... 0

public list of coins

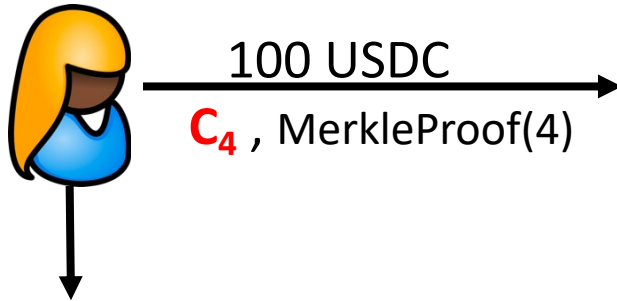
Tornado cash: deposit

(simplified)

100 USDC pool:

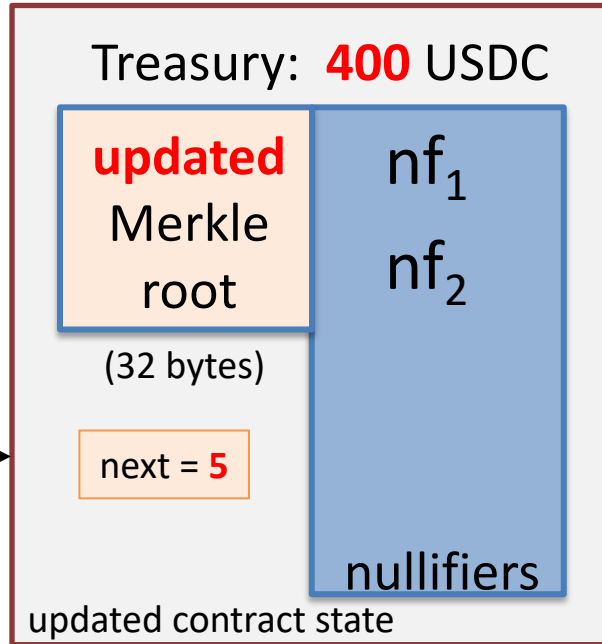
each coin = 100 USDC

Alice deposits 100 USDC:

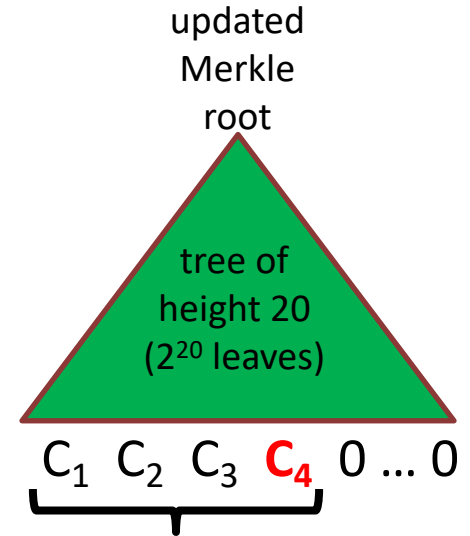


note: (k, r)

Alice keeps secret
(one note per coin)



Every deposit: new Coin
added sequentially to tree



public list of coins

an observer sees who
owns which leaves

Tornado cash: withdrawal

(simplified)

100 USDC pool:

each coin = 100 USDC

Withdraw coin #3
to addr A:



has note = (k', r')

set $nf = H_2(k')$

Treasury: **400** USDC

coins
Merkle
root

(32 bytes)

next = 5

nf_1
 nf_2

nullifiers

contract state

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

Merkle
root

tree of
height 20
(2^{20} leaves)

C_1 C_2 **C_3** C_4 0 ... 0

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)

Withdraw coin #3 to addr A:



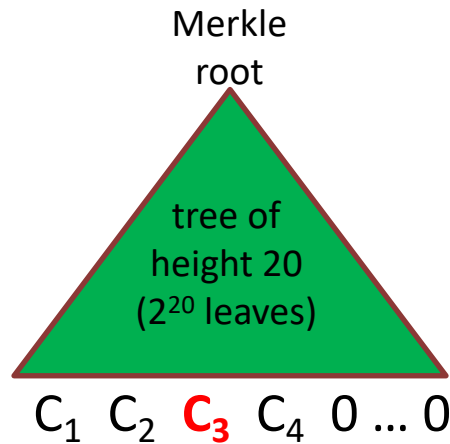
has note = (k', r') set **nf** = $H_2(k')$

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

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

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

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



(address A not used in Circuit)

Tornado cash: withdrawal (simplified)

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

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 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')

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

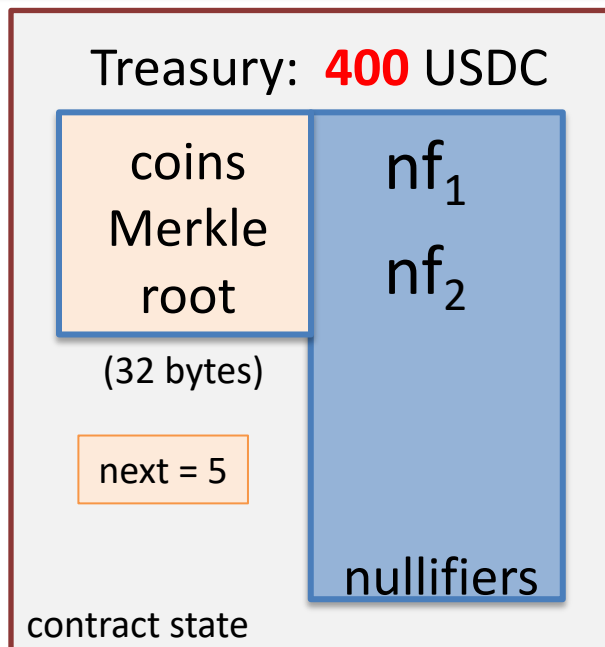
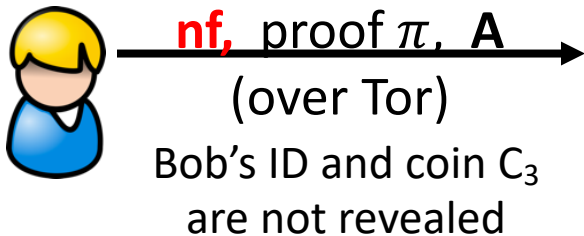
Bob builds zk-SNARK proof π for
public statement $x = (\text{root}, \text{nf}, A)$
secret witness $w = (k', r', C_3, \text{MerkleProof}(C_3))$

Tornado cash: withdrawal (simplified)

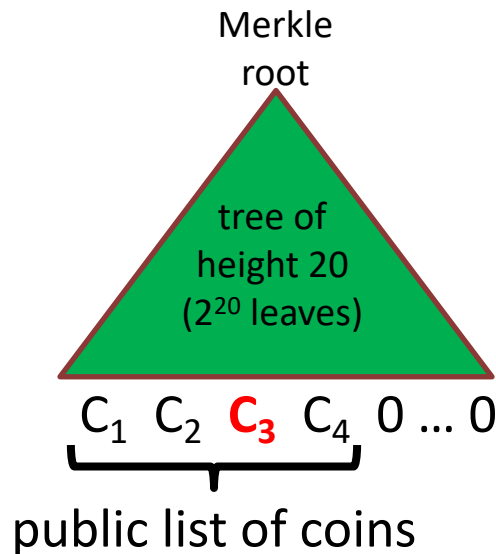
100 USDC pool:

each coin = 100 USDC

Withdraw coin #3
to addr A:



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

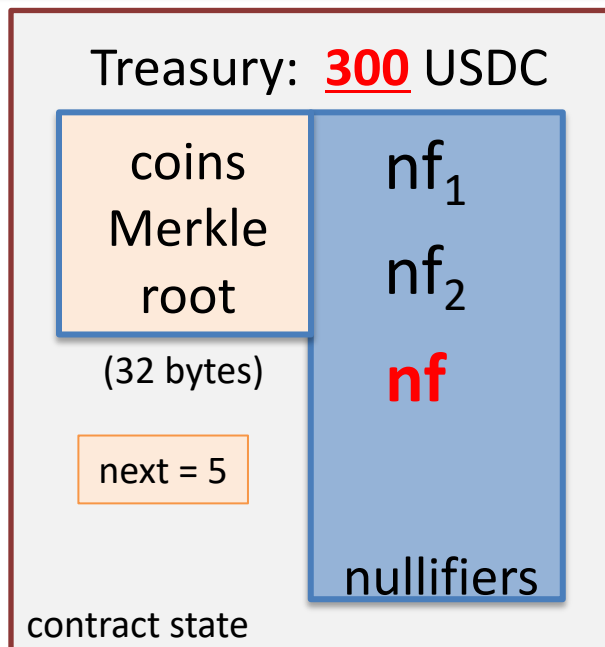
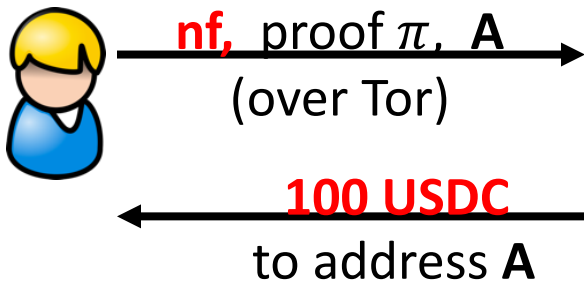


Contract checks (i) proof π is valid for (root, **nf**, **A**), and
(ii) **nf** is not in the list of nullifiers

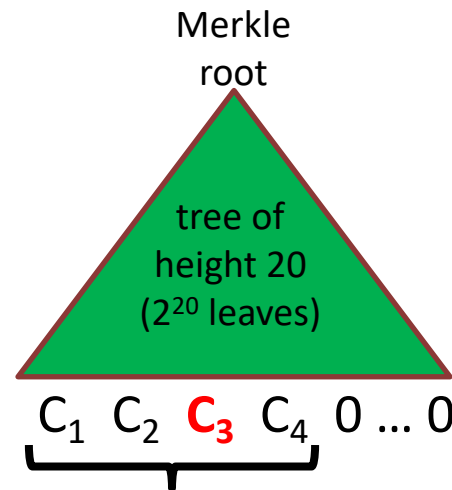
Tornado cash: withdrawal (simplified)

100 USDC pool:
each coin = 100 USDC

Withdraw coin #3
to addr A:



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



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

nf and π reveal nothing about which coin was spent.

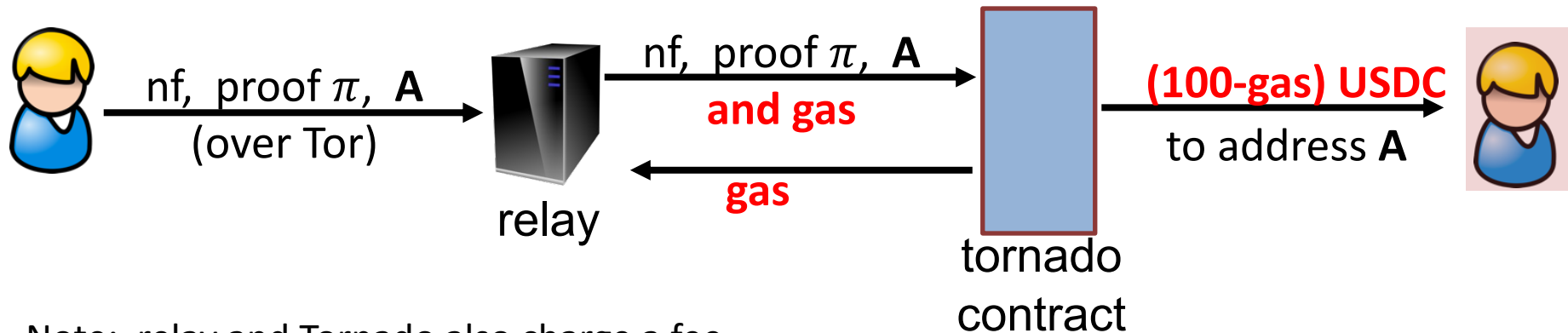
But, coin #3 cannot be spent again, because **nf** = $H_2(k')$ is now nullified.

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**



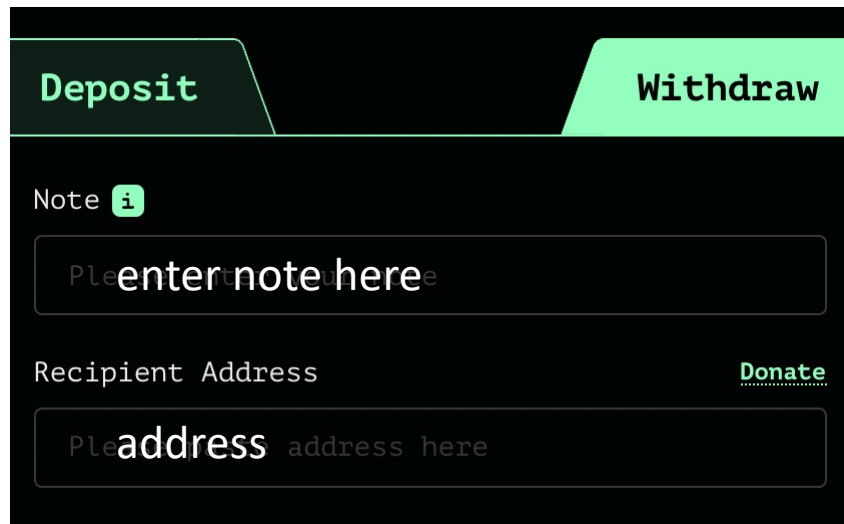
Note: relay and Tornado also charge a fee

Tornado Cash: the UI



The Deposit UI features a dark theme with two tabs: 'Deposit' (active, highlighted in red) and 'Withdraw' (greyed out). Below the tabs, there is a 'Token' dropdown menu currently set to 'DAI' with a green checkmark. Underneath is an 'Amount' section with an information icon and a horizontal slider. The slider has four circular markers corresponding to the values '100 DAI', '1K DAI' (which is selected and highlighted in red), '10K DAI', and '100K DAI'.

After deposit: get a note



The Withdraw UI also has a dark theme with two tabs: 'Deposit' (greyed out) and 'Withdraw' (active, highlighted in red). It contains a 'Note' field with an information icon and a placeholder text 'Please enter note here'. Below this is a 'Recipient Address' field with a placeholder 'Please enter address here' and a red 'Donate' button to its right.

Later, use note to withdraw

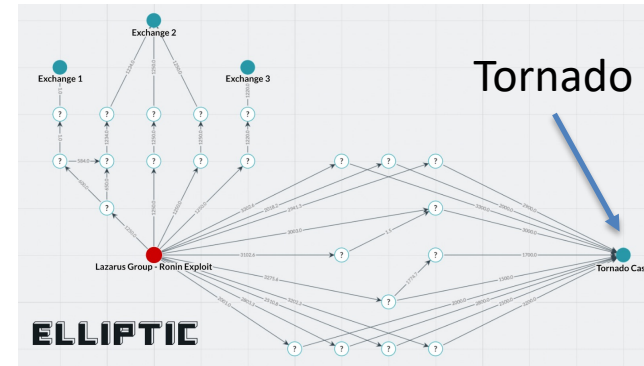
(wait before withdrawing)

Tornado trouble ... U.S. sanctions

The Ronin-bridge hack (2022):

- In late March: $\approx 600\text{M}$ USD stolen ... $\$80\text{M}$ 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 removed: March 2025

March 21, 2025

WASHINGTON — Based on the Administration’s review of the novel legal and policy issues raised by use of financial sanctions against financial and commercial activity occurring within evolving technology and legal environments, we have exercised our discretion to remove the economic sanctions against Tornado Cash as reflected in Treasury’s Monday filing in Van Loon v. Department of the Treasury.

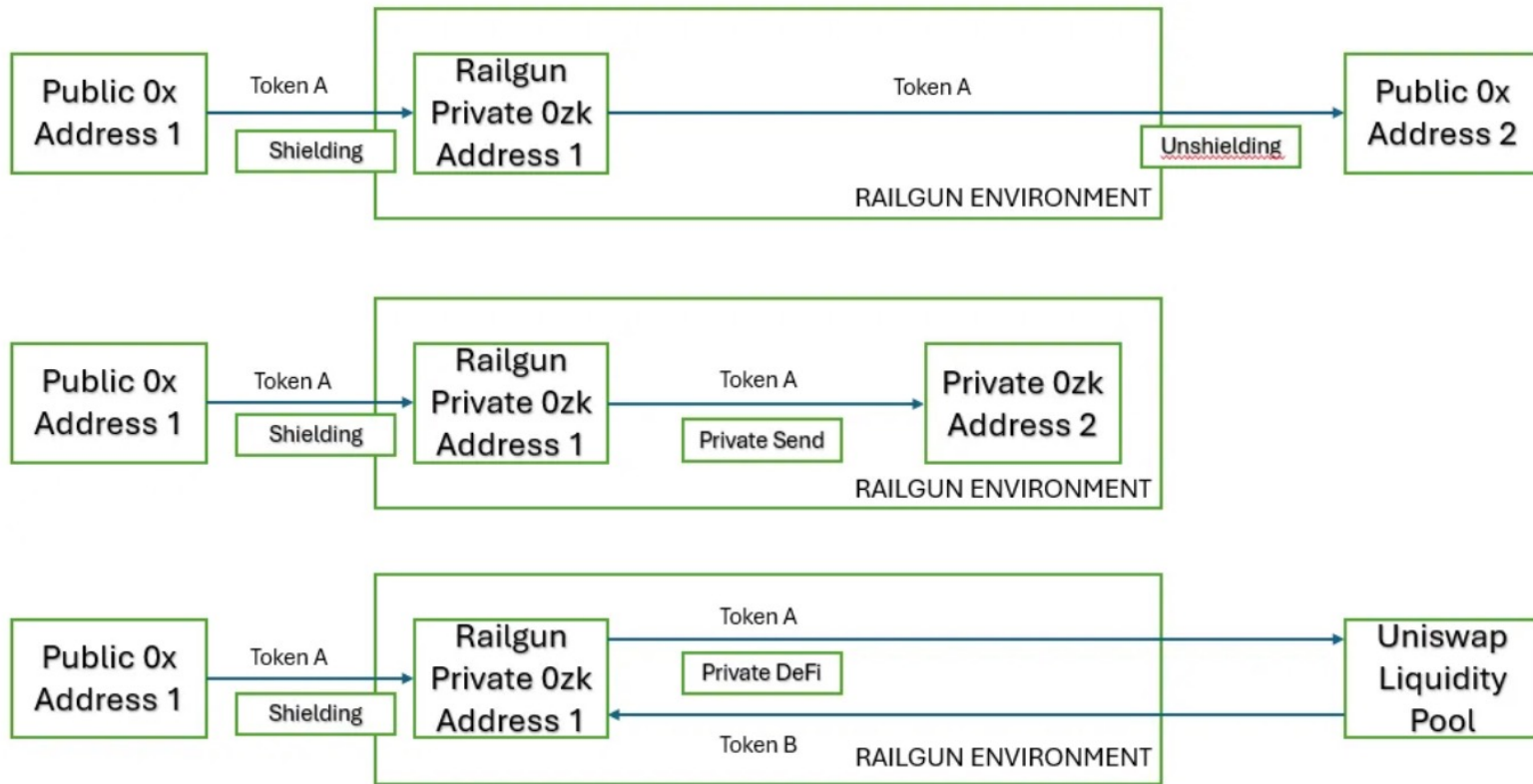
...

“Digital assets present enormous opportunities for innovation and value creation for the American people,” said Secretary of the Treasury Scott Bessent. “Securing the digital asset industry from abuse by North Korea and other illicit actors is essential to establishing U.S. leadership and ensuring that the American people can benefit from financial innovation and inclusion.”

Building a Compliant Mixer?

Examples: Railgun, PrivacyPools
(2021) (2025)

Railgun Functionality



A compliant mixer

Defense #1: deposit filtering. Ensure incoming funds are not sanctioned

Chainalysis **SanctionsList** contract:

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

When funds are moved into Railgun **they are locked for an hour**

- Victim of a theft has an hour to report the theft
... funds rejected if source address is reported to be a bad actor

A compliant mixer

Defense #2: withdrawal filtering. At withdrawal, require a ZK proof that the source of funds is not currently on a sanctioned list

How?

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

in our example Alice sets: $\mathbf{C}_4 = [H_1(k, r), \mathbf{msg.sender}]$

- During withdrawal Bob proves in ZK that **msg.sender** in his leaf is not currently on a sanctioned list (Proof-of-Innocence, PoI)

Proof of innocence providers

Several Proof of Innocence (Pol) list providers:
Elliptic, ScamSniffer, PureFi, SlowMist, Chainalysis

RAILGUN Flagged Shields Explorer

View the transaction details of shields flagged as high risk.

Network: Ethereum ▾

Total Flagged Shields for Ethereum: 611

May not display in same order of on-chain execution.

⚠ Flagged Shield #1

Reason: Address identified as high-risk by independent list providers

Commitment: 0x113a71037dbefeaa43729c20e91f1961a57454974a6d48533ec07d58dec2ae9

Show Details

Railgun:
Pol is provided
on deposit (!)

A compliant mixer

Defense #3: viewing keys. At withdrawal, require nullifier to include an encryption of deposit msg.sender under government public key.

How? Merkle leaf C_4 is computed as on previous slide.

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

⇒ If needed, law enforcement can trace funds through Railgun

So only good actors?

Jan. 2023: a [wallet](#) associated with the Harmony Bridge hack sent **897 ETH** through Railgun.

- A single deposit Tx using 31 distinct smart contract events, many of which escaped detection by investigative tools

Other approaches to compliant privacy?

Example: **ERC-7984** (ConfidentialFungibleToken)

- A confidential ERC-20.
 - All account balances are encrypted under a global public key
 - The secret decryption key is held by a trusted party (or parties)
- Transfers are done using homomorphic encryption
- Authorized parties can request to decrypt current balance
 - account owner, relevant law-enforcement

Importantly: hides amount, but not who is paying who

END OF LECTURE

Next lecture: how to build a SNARK