CS251 Fall 2022

(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 <u>succinct</u> proof that a certain statement is true

Example statement: "I know an m such that 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

zk-SNARK: Blockchain Applications

Private Tx on a public blockchain:

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

Compliance:

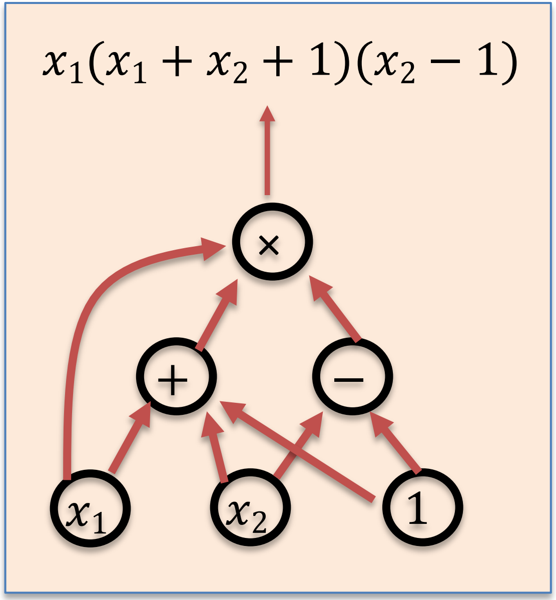
- Proving that a private Tx are in compliance with banking laws
- Proving solvency in zero-knowledge

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

Bridging between blockchains: zkBridge

Review: arithmetic circuits

- Fix a finite field $\mathbb{F} = \{0, \ldots, p-1\}$ for some prime p>2.
- **Arithmetic circuit:** $C: \mathbb{F}^n \rightarrow \mathbb{F}$
  - directed acyclic graph (DAG) where
    internal nodes are labeled +, −, or ×
    inputs are labeled $1, x_1, \ldots, x_n$
  - 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 $\neq 0$ otherwise

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

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

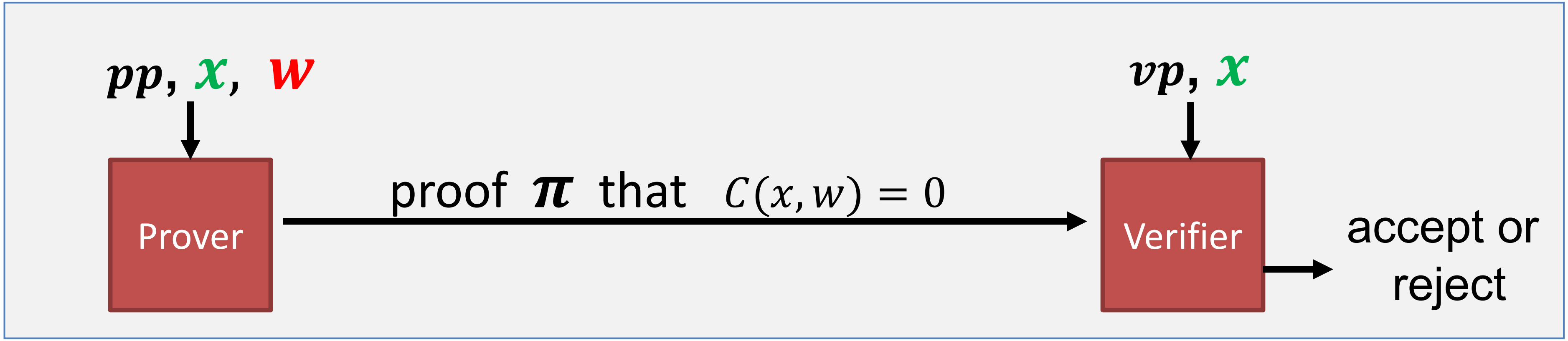
(preprocessing) NARK: Non-interactive ARgument of Knowledge

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

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

secret witness in $\mathbb{F}^m$ → w

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

NARK: requirements (informal)

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

proof π →

accept or reject

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

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 ; $|\pi| = O_\lambda(\log(|C|))$
- **V**$(vp, \boldsymbol{x}, \boldsymbol{\pi})$ **fast to verify** ; time(V) $= O_\lambda(|x|, \log(|C|))$

short "summary" of circuit

Why preprocess C??

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 ; $|\pi| = O_\lambda(\, \mathbf{log}(|C|))$
- **V**(vp, $\boldsymbol{x}$, $\boldsymbol{\pi}$) **fast to verify** ; time(V) = $O_\lambda(|x|,\ \mathbf{log}(|C|))$

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

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

The trivial SNARK is not a SNARK

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

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

<u>**Problems with this**</u>:

(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

Types of preprocessing Setup

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

random bits

Types of setup:

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

$\boldsymbol{S} = (S_{init}, S_{index})$: $S_{init}(\lambda; r) \to gp$, $S_{index}(gp, C) \to (pp, vp)$

one-time — no secret data from prover

transparent setup: **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 $O_\lambda(\log\|C\|)$	$\approx$ 3 sec $O_\lambda(\|C\|)$	transparent	no
STARK	$\approx$ 100 KB $O_\lambda(\log^2\|C\|)$	$\approx$ 10 ms $O_\lambda(\log\|C\|)$	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				
Plonk / Marlin				
Bulletproofs				
STARK	$O_\lambda(\log^2\|C\|)$	$O_\lambda(\log\|C\|)$		
⋮				⋮

Prover time is almost linear in $|C|$

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

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

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

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

$gp \leftarrow S_{init}(\,)$, $(C, x, st) \leftarrow A_0(gp)$, $(pp, vp) \leftarrow S_{index}(C)$, $\pi \leftarrow A_1(pp, x, st)$:

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

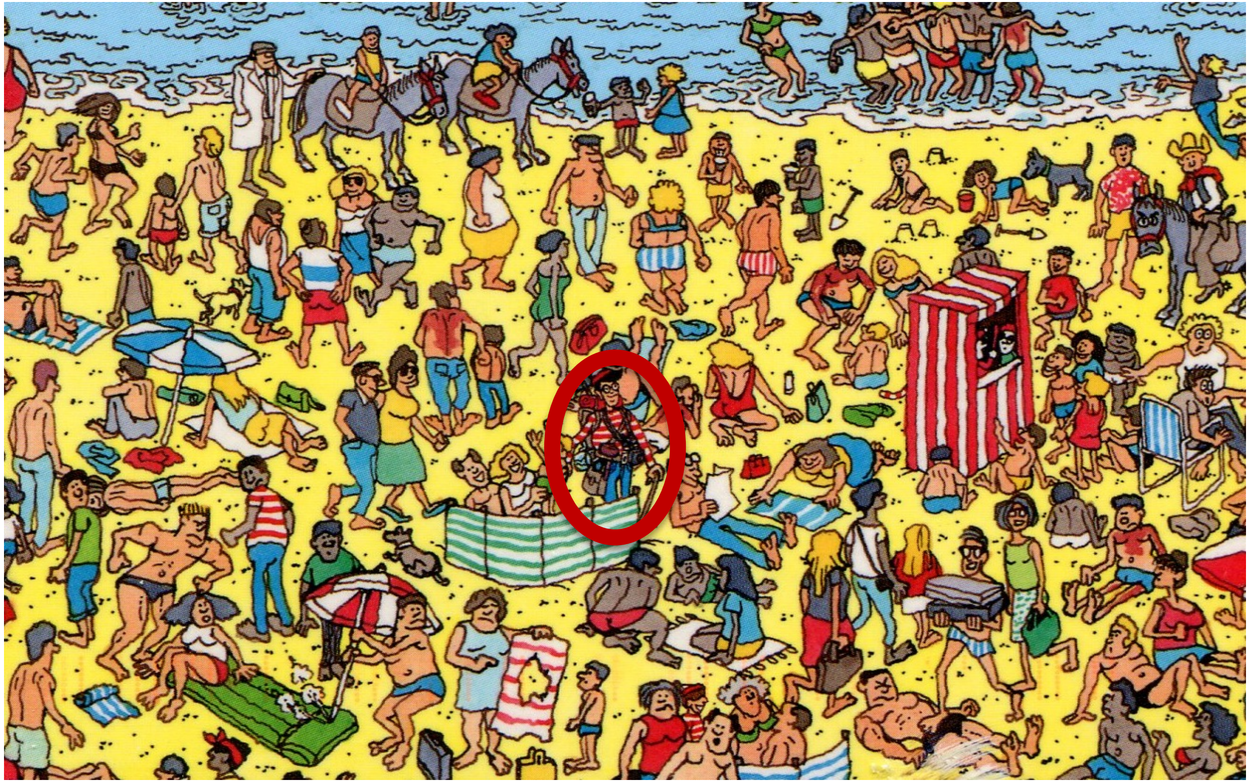
there is an efficient **extractor** E (that uses A_1 as a black box) s.t.

$gp \leftarrow S_{init}(\,)$, $(C, x, st) \leftarrow A_0(gp)$, $w \leftarrow E^{A_0,\, A_1(pp,x,st)}(gp, C, x)$:

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

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** $\Longrightarrow$ 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 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($pp, x, \boldsymbol{w}$)

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 ??

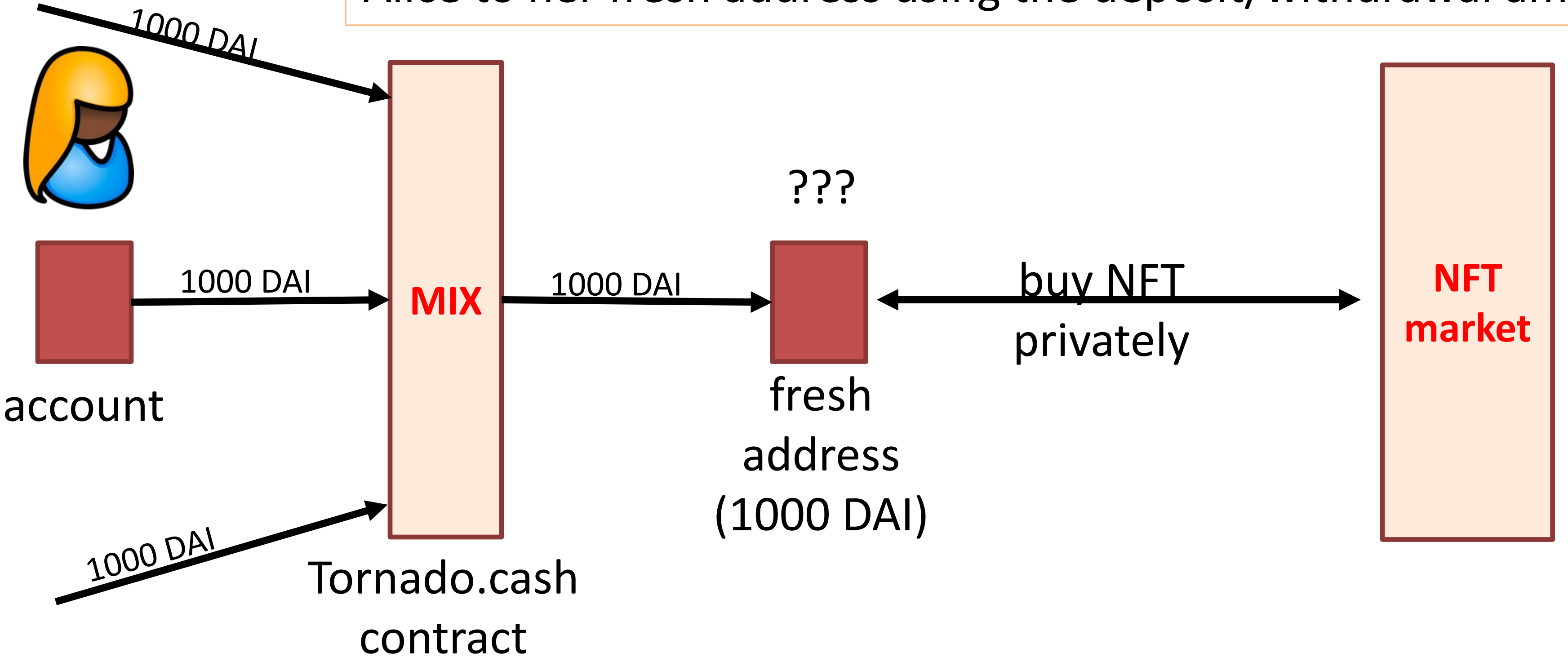
Next lecture

Tornado cash: a zk-based mixer

Launched on the Ethereum blockchain on May 2020 (v2)

Tornado Cash: a ZK-mixer

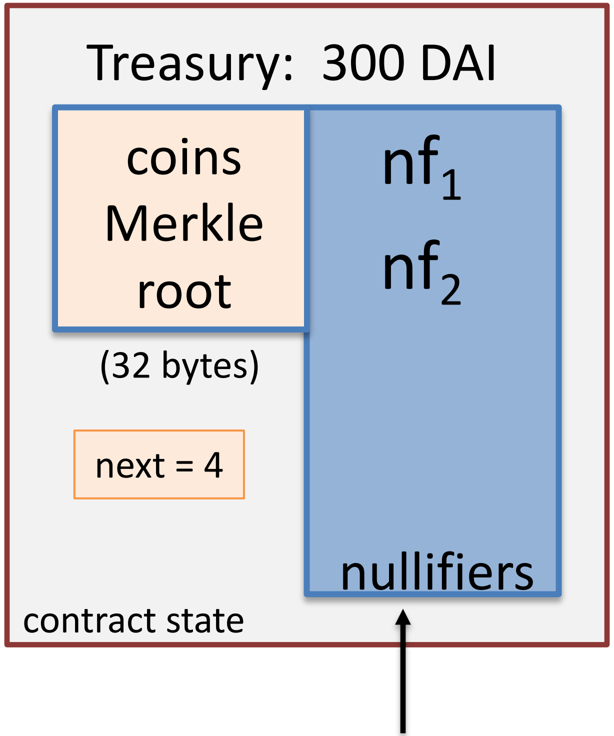
A common denomination (1000 DAI) is needed to prevent linking Alice to her fresh address using the deposit/withdrawal amount



The tornado cash contract (simplified)

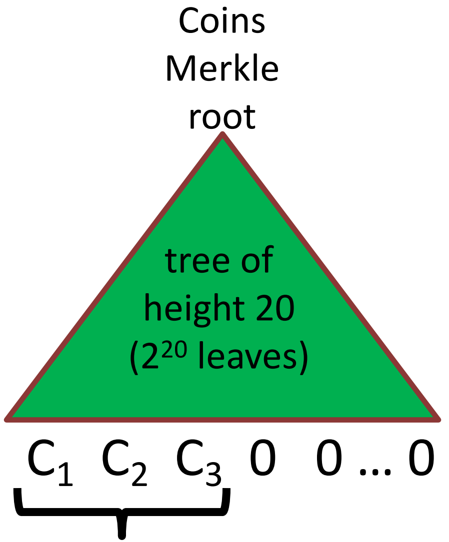
100 DAI pool:
each coin = 100 DAI

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



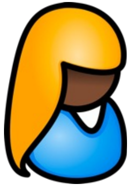
public list of coins

Tornado cash: deposit (simplified)

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

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)

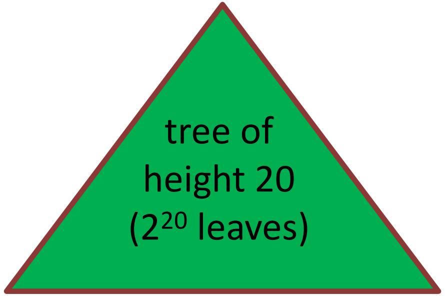
next = 4

nf_1
nf_2
nullifiers

contract state

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

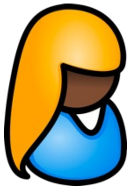
Coins Merkle root



C_1 C_2 C_3 0 0 … 0

public list of coins

Tornado cash: deposit (simplified)



100 DAI

C_4 , MerkleProof(4)

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

coins Merkle root

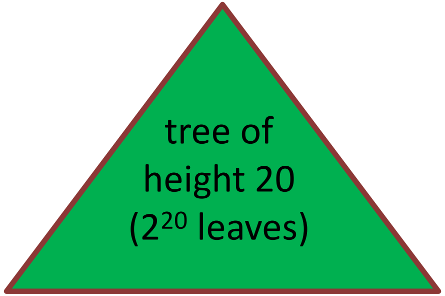
(32 bytes)

next = 4

Tornado contract

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

Coins Merkle root

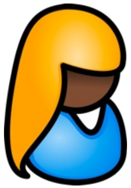


C_1 C_2 C_3 0 0 … 0

public list of coins

Tornado cash: deposit (simplified)

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



100 DAI

C_4 , MerkleProof(4)

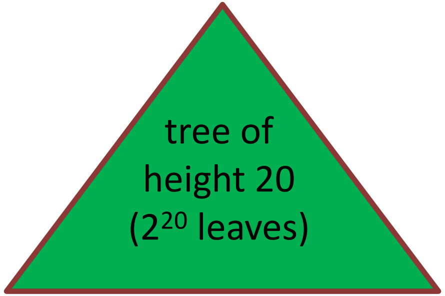
coins Merkle root

(32 bytes)

next = 4

Tornado contract

updated Merkle root



C_1 C_2 C_3 C_4 0 … 0

public list of coins

Tornado contract does:

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

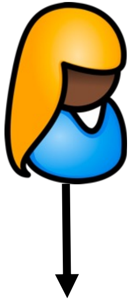
(2) use C_4 and MerkleProof(4) to compute updated Merkle root

(3) update state

Tornado cash: deposit (simplified)

100 DAI pool:
each coin = 100 DAI

Alice deposits 100 DAI:



100 DAI →
C_4 , MerkleProof(4)

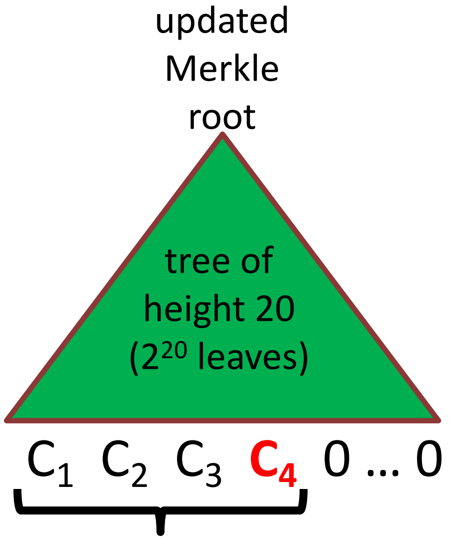
note: (k, r)
Alice keeps secret
(one note per coin)

Treasury: **400** DAI

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

updated contract state

Every deposit: new Coin added sequentially to tree



an observer sees who owns which leaves

Tornado cash: withdrawal (simplified)

100 DAI pool:
each coin = 100 DAI

Withdraw coin #3 to addr A:

has note= (k', r')

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

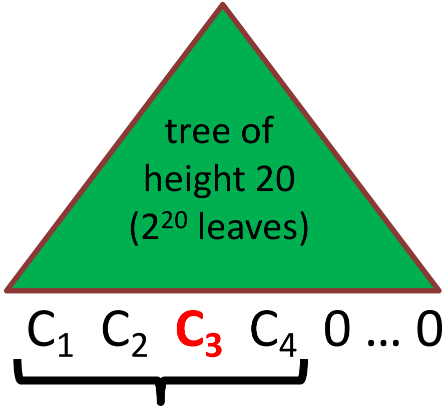
Treasury: **400** DAI

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

contract state

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

Merkle root



public list of coins

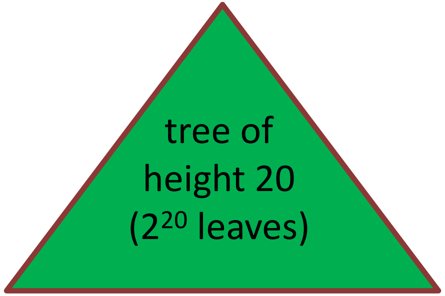
Bob proves "I have a note for some leaf in the coins tree, and its nullifier is **nf**"
(without revealing which coin)

Tornado cash: withdrawal (simplified)

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

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

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)

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

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

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

Treasury: **400** DAI

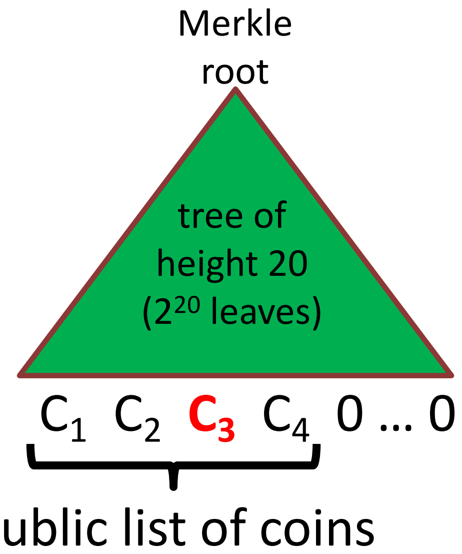
coins Merkle root
(32 bytes)

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**

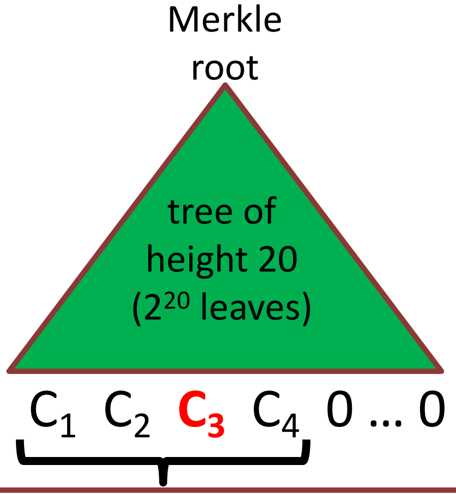
Treasury: **300** DAI

coins Merkle root (32 bytes)	nullifiers
	nf_1
	nf_2
	nf

next = 5

contract state

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



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

nf and π reveal nothing about which coin was spent.

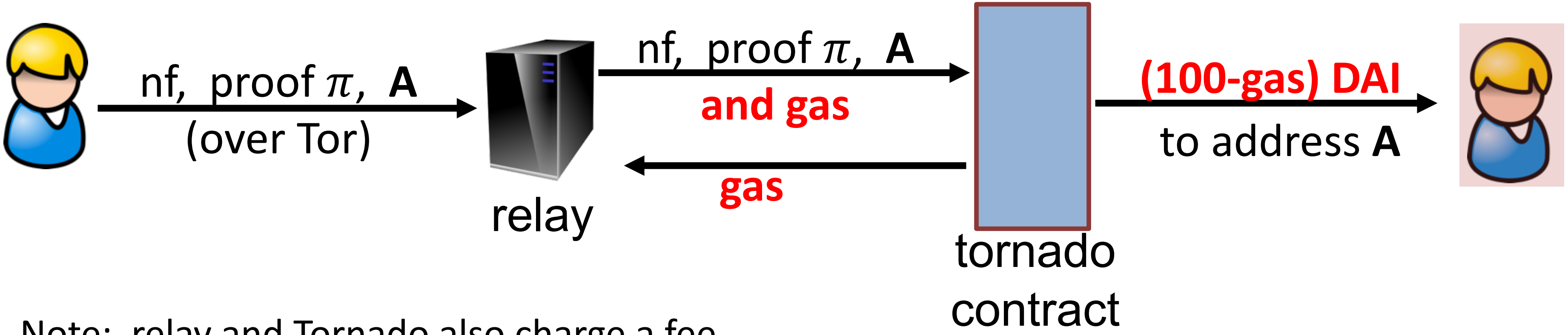
But, coin #3 cannot be spent again, because **nf = H_2(k')** is now nullified.

Who pays the withdrawal gas fee?

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

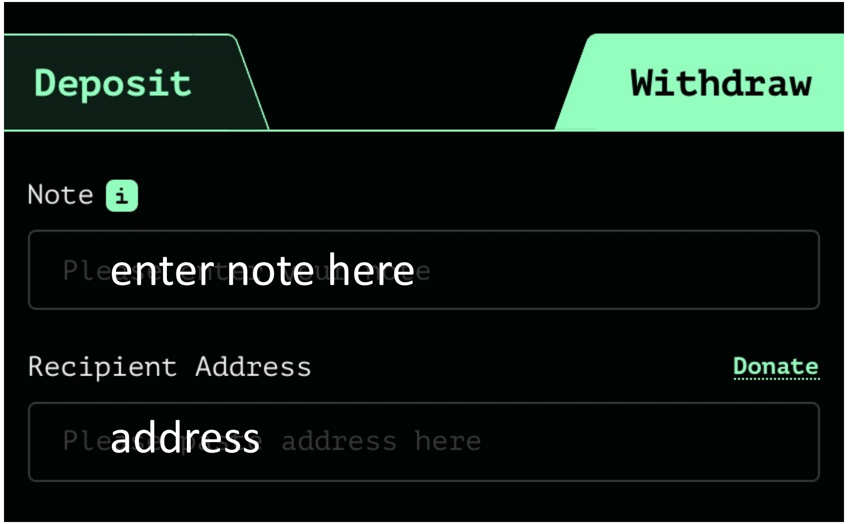
- If paid from Bob's address, then fresh address is linked to Bob

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)

Anonymity set

88,036
Total deposits

leaves occupied over all pools

$3,798,916,834
Total USD deposited

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

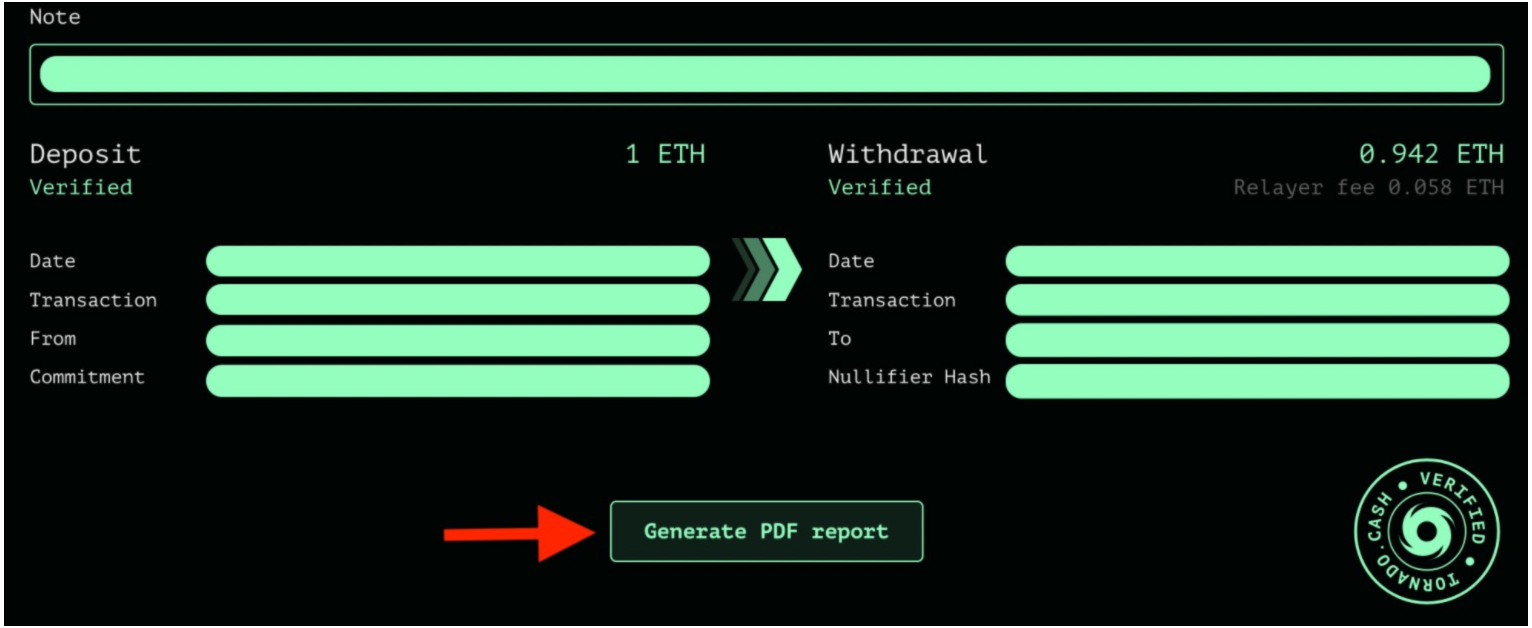
Tornado.cash compliance tool

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

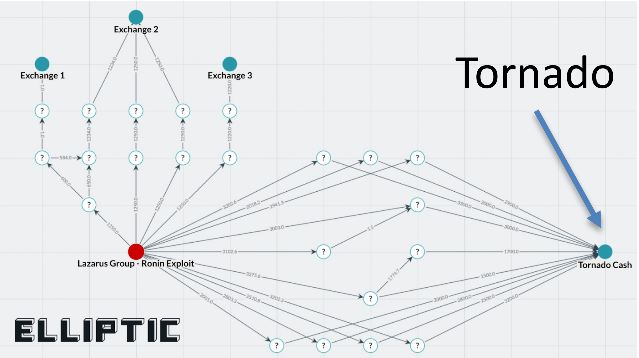


Reveals source address and destination address of funds

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: C_4 = [H_1(k, r), **msg.sender**]

- During withdrawal Bob proves in ZK that **msg.sender** in his leaf is not currently 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 **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 …

ZCASH / IRONFISH

Two L1 blockchains that extend Bitcoin.

Sapling (Zcash v2) launched in Aug. 2018.

Similar use of Nullifiers, support for any value Tx, and in-system transfers

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