Using zk-SNARKs for Privacy on the Blockchain

Dan Boneh
Can we have private transactions on a public blockchain?

Naïve reasoning:

universal verifiability \Rightarrow \text{transaction data must be public. otherwise, how we can the public verify Tx ??}

Goal for this lecture:

crypto magic \Rightarrow \text{private Tx on a publicly verifiable blockchain}

Crypto tools: commitments and zero knowledge proofs
The need for privacy in the financial system

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

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

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

Neither Bitcoin nor Ethereum are private

etherscan.io:

Address 0x1654b0c3f62902d7A86237...

<table>
<thead>
<tr>
<th>Balance:</th>
<th>1.114479450024297906 Ether</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ether Value:</td>
<td>$4,286.34 (@ $3,846.05/ETH)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Txn Hash</th>
<th>Method</th>
<th>Block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0269eff8b4196558c07...</td>
<td>Set Approval For...</td>
<td>13426561</td>
</tr>
<tr>
<td>0xa3dacb0e7c579a99cd...</td>
<td>Cancel Order...</td>
<td>13397993</td>
</tr>
<tr>
<td>0x73785abcc7ccf030d6a...</td>
<td>Set Approval For...</td>
<td>13387834</td>
</tr>
<tr>
<td>0x1463293c495069d61c...</td>
<td>Atomic Match...</td>
<td>13387703</td>
</tr>
</tbody>
</table>
Simple blockchain anonymity via mixing

- 1 ETH to M from Alice
- 1 ETH to M from Bob
- 1 ETH to M from Carol

fresh addr X from Alice
fresh addr Y from Bob
fresh addr Z from Carol

mixer address: M
has 3 ETH

Observer knows Y belongs to one of {Alice, Bob, Carol} but does not know which one
⇒ anonymity set of size 3.
⇒ Bob can mix again with different parties to increase anonymity set.

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

Mixing without a mixer? on Bitcoin: CoinJoin (e.g., Wasabi), on Ethereum: Tornado cash
What is a zk-SNARK?

An central tool for privacy on the blockchain
zk-SNARK: Blockchain Applications

Private Tx on a public blockchain:
• Confidential transactions
• Tornado cash, Zcash, IronFish
• Private Dapps: Aleo

Compliance:
• Proving solvency in zero-knowledge
• Zero-knowledge taxes

Scalability: privacy in zk-SNARK Rollup (next week)
• Fix a finite field $\mathbb{F} = \{0, \ldots, p - 1\}$ for some prime $p > 2$.

• **Arithmetic circuit:** $C: \mathbb{F}^n \to \mathbb{F}$
  - directed acyclic graph (DAG) where internal nodes are labeled $+$, $-$, or $\times$
  - 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

Examples:

• $C_{\text{hash}}(h, m)$: outputs 0 if $\text{SHA}256(m) = h$, and $\neq 0$ otherwise

  $$C_{\text{hash}}(h, m) = (h - \text{SHA}256(m))$$

  $|C_{\text{hash}}| \approx 20$K gates

• $C_{\text{sig}}(pk, m, \sigma)$: outputs 0 if $\sigma$ is a valid ECDSA signature on $m$ with respect to $pk$
Public arithmetic circuit: \[ C(x, w) \rightarrow \mathbb{F} \]

P’s goal: “convince” V that \( \exists w \) s.t. \( C(x, w) = 0 \)

Prover

Verifier

accept or reject
Preprocessing argument systems

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

- Public statement in $\mathbb{F}^n$
- Secret witness in $\mathbb{F}^m$

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

Diagram:

- $S_p, x, w$ to Prover
- $S_v, x$ to Verifier
- Proof $\pi$ from Prover to Verifier
- Verifier accepts or rejects proof
A preprocessing argument system is a triple \((S, P, V)\):

- \(S(C) \rightarrow\) public parameters \((S_p, S_v)\) for prover and verifier
- \(P(S_p, x, w) \rightarrow\) proof \(\pi\)
- \(V(S_v, x, \pi) \rightarrow\) accept or reject
**Argument system: requirements (informal)**

**Prover** $P(S_p, x, w)$

**Verifier** $V(S_v, x, \pi)$

Proof $\pi$

Accept or reject

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

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

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

**Optional:** Zero knowledge: $(S_v, x, \pi)$ “reveals nothing” about $w$
SNARK: a **Succinct ARgument of Knowledge**

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

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

Why preprocess $C$??

short “summary” of circuit
SNARK: a **Succinct ARgument of Knowledge**

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

- **\(S(C) \rightarrow\)** public parameters \((S_p, S_v)\) for prover and verifier

- **\(P(S_p, x, w) \rightarrow\)** short proof \(\pi\) ; \(|\pi| = O(\log(|C|), \lambda)\)

- **\(V(S_v, x, \pi)\)** fast to verify ; \(\text{time}(V) = O(|x|, \log(|C|), \lambda)\)

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 \((x_1, \ldots, x_n) \in X\) such that \(H(x_1, \ldots, x_n) = y\)

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

How is this possible???

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

Prover

Verifier

statement: \(y\)

witness: \(x_1, \ldots, x_n\)

Proof \(\pi\)

accept or reject
Types of preprocessing Setup

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

Types of setup:

- **trusted setup per circuit:** $S(C; r)$ random bits $r$ must be kept secret
  compromised trusted setup $\Rightarrow$ can prove false statements

- **trusted but universal (updatable) setup:** secrets in $S(C)$ are independent of $C$
  
  \[ S = (S_{\text{init}}, S_{\text{pre}}): \quad S_{\text{init}}(\lambda; r) \rightarrow U, \quad S_{\text{pre}}(U, C) \rightarrow (S_p, S_v) \]
  
  one-time \hspace{2cm} no secret data

- **transparent setup:** $S(C)$ does not use secret data (no trusted setup)
Significant progress in recent years

- Kilian’92, Micali’94: succinct transparent arguments from PCP
  - impractical prover time

- GGPR’13, Groth’16, …: linear prover time, constant size proof \( O_1(1) \)
  - 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)

<table>
<thead>
<tr>
<th></th>
<th>size of proof $\pi$</th>
<th>size of $S_p$</th>
<th>verifier time</th>
<th>trusted setup?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groth’16</td>
<td>$O(1)$</td>
<td>$O(</td>
<td>C</td>
<td>)$</td>
</tr>
<tr>
<td>Plonk/Marlin</td>
<td>$O(1)$</td>
<td>$O(</td>
<td>C</td>
<td>)$</td>
</tr>
<tr>
<td>Bulletproofs</td>
<td>$O(\log</td>
<td>C</td>
<td>)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>STARK</td>
<td>$O(\log</td>
<td>C</td>
<td>)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>DARK</td>
<td>$O(\log</td>
<td>C</td>
<td>)$</td>
<td>$O(1)$</td>
</tr>
</tbody>
</table>
A SNARK software system

- DSL programs: Circom, ZoKrates, Leo, Zinc, Cairo, Snarky, ...
- SNARK compiler: friendly format (R1CS, AIR, Plonk-CG)
- SNARK backend
- Proof $\pi$
- CPU heavy
- Setup
- (setup) $S_p, S_v$
- Witness $x$
- Verifier
- Accept/reject
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

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

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

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

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

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

If holds for all $A$, then $(S, P, V)$ is a proof of knowledge.
(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 $w$
(but also outputs $S_p, S_v$)
Definitions: (2) Zero knowledge (against an honest verifier)

Formally: \((S, P, V)\) is (honest verifier) **zero knowledge** for a circuit \(C\) if there is an efficient simulator \(Sim\) such that for all \(x \in \mathbb{F}^n\) s.t. \(\exists w: C(x, w) = 0\) the distribution:

\[
(S_p, S_v, x, \pi): \text{ where } (S_p, S_v) \leftarrow S(C), \quad \pi \leftarrow P(S_p, x, w)
\]

is indistinguishable from the distribution:

\[
(S_p, S_v, x, \pi): \text{ where } (S_p, S_v, \pi) \leftarrow Sim(C, x)
\]
Recall: A zero knowledge preprocessing argument system.

Prover generates a short proof that is fast to verify

How to build a zk-SNARK ??

Next lecture
Tornado cash: a zk-based mixer

Launched on the Ethereum blockchain on May 2020 (v2)
Tornado Cash: a ZK-mixer

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

100 DAI pool:
each coin = 100 DAI

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

Treasury: 300 DAI

nullifiers

nf₁

nf₂

coins

Merkle root

(32 bytes)

next = 4

explicit list:
one entry per spent coin

H₁, H₂: R → {0,1}^{256}

Coins
Merkle root
tree of height 20
(2^{20} leaves)

public list of coins

C₁ C₂ C₃ 0 0 ... 0
**Tornado cash: deposit** (simplified)

**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

**nullifiers**

$nf_1$, $nf_2$

(32 bytes)

Next = 4

**Merkle root**

**Coins**

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

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

**contract state**

next = 4

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

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

**nullifiers**

$nf_1$, $nf_2$

(32 bytes)

Next = 4

**Merkle root**

**Coins**

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

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

**contract state**

next = 4

**explicit list:** one entry per spent coin
Tornado cash: deposit

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

100 DAI $C_4$, MerkleProof(4)

coins Merkle root
(32 bytes)

next = 4

H₁, H₂: $R \rightarrow \{0,1\}^{256}$

Coins Merkle root

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

public list of coins

$C_1 C_2 C_3 0 0 \ldots 0$
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

100 DAI pool:
each coin = 100 DAI

Alice deposits 100 DAI:

Alice deposits 100 DAI: 100 DAI

updated contract state

updated Merkle root

nf1

nullifiers

nf2

(32 bytes)

next = 5

Every deposit: new Coin added sequentially to tree

an observer sees who owns which coins

updated Merkle root

tree of height 20
(2^20 leaves)

public list of coins

note: (k, r)

Alice keeps secret
(one note per coin)
**Tornado cash: withdrawal**

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 \( \text{nf} \)” (without revealing which coin)
Withdraw coin #3 to addr A:

has note= (k’, r’) set nf = H₂(k’)

Bob builds zk-SNARK proof π for public statement x = (root, nf, A)

secret witness w = (k’, r’, C₃, MerkleProof(C₃))

where Circuit(x,w)=0 iff:

(i) C₃ = (leaf #3 of root), i.e. MerkleProof(C₃) is valid,
(ii) C₃ = H₁(k’, r’), and
(iii) nf = H₂(k’).

(address A not used in Circuit)
Tornado cash: withdrawal (simplified)

Withdraw coin #3 to addr A:

\[
C_1 \quad C_2 \quad C_3 \quad C_4 \quad 0 \ldots 0
\]

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

Has note = (k', r') set \( nf = H_2(k') \)

Bob builds zk-SNARK proof \( \pi \) for public statement \( x = (\text{root}, \ nf, \ A) \)

Secret witness \( w = (k', \ r', \ C_3, \ \text{MerkleProof}(C_3)) \)

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: \( R \rightarrow \{0,1\}^{256} \)

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:
Tornado cash: withdrawal

**100 DAI pool:**
each coin = 100 DAI

**Withdraw coin #3 to addr A:**

nf, proof \(\pi\), A
(over Tor)
Bob’s ID and coin \(C_3\) are not revealed

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

Treasury: 400 DAI

coins

Merkle root

\(nf_1\)

\(nf_2\)

nullifiers

next = 5

contract state

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

Merkle root

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

\(C_1, C_2, C_3, C_4, 0 \ldots 0\)

public list of coins

Withdraw coin #3 to addr A:

nf, proof \(\pi\), A
(over Tor)
Bob’s ID and coin \(C_3\) are not revealed

Contract checks (i) proof \(\pi\) is valid for (root, nf, A), and (ii) nf is not in the list of nullifiers
Tornado cash: withdrawal

100 DAI pool:
each coin = 100 DAI

Withdraw coin #3
to addr A:

\[ \text{nf, proof } \pi, A \]
(over Tor)

100 DAI
to address A

nf and \( \pi \) reveal nothing about which coin was spent.

But, coin #3 cannot be spent again, because \( \text{nf} = H_2(k') \) is now nullified.
Problem: how does Bob pay for gas for the withdrawal Tx?

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

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

Note: relay and Tornado also charge a fee
Tornado Cash: the UI

After deposit: get a note
Later, use note to withdraw
(wait before withdrawing)
Anonymity set

88,036
Total deposits

$3,798,916,834
Total USD deposited

# leaves occupied over all pools

1 ETH pool

30141
Equal user deposits

Latest deposits

30141. 4 minutes ago
30140. 9 minutes ago
30139. 2 hours ago
30138. 3 hours ago
30137. 3 hours ago

30136. 3 hours ago
30135. 4 hours ago
30134. 5 hours ago
30133. 5 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.
Compliance tool

Reveals source address and destination address of funds
ZCASH / IRONFISH

Two L1 blockchains that extend Bitcoin.
Sapling (Zcash v2) launched in Aug. 2018.

Similar use of Nullifiers, support for any value Tx.
Quick review

A zk-SNARK for a circuit $C$:
• For a public statement $x$, prover outputs a proof that “convinces” verifier that prover knows $w$ s.t. $C(x, w) = 0$.
• Proof is short and fast to verify

What is it good for?
• Private payments and private Dapp business logic (Aleo)
• Private compliance and L2 scalability with privacy

More to think about:
• private DAO participation? private governance?
Further topics

Privately communicating with the blockchain: Nym
  • How to privately compensate proxies for relaying traffic

Next lecture: how to build a SNARK
END OF LECTURE
Two types of argument systems: interactive vs. non-interactive

Interactive: proof takes multiple $P \leftrightarrow V$ 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
- SNARK: short proof that is fast to verify