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?
Neither Bitcoin nor Ethereum are private

etherscan.io:

Address 0x1654b0c3f62902d7A86237...

Balance: 1.114479450024297906 Ether

Ether Value: $4,286.34 (@ $3,846.05/ETH)

<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>0x1463283c495069d61c...</td>
<td>Atomic Match...</td>
<td>13387703</td>
</tr>
</tbody>
</table>

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 $m$
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 \( \times \)
  - inputs are labeled \( 1, x_1, \ldots, x_n \)
  - defines an n-variate polynomial with an evaluation recipe

- \( |C| = \# \text{ gates in } C \)
Interesting arithmetic circuits

Examples:

• $C_{hash}(h, m)$: outputs 0 if $\text{SHA256}(m) = h$, and $\neq 0$ otherwise
  
  $C_{hash}(h, m) = (h - \text{SHA256}(m))$, $|C_{hash}| \approx 20K$ gates

• $C_{sig}(pk, m, \sigma)$: outputs 0 if $\sigma$ 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 \( \mathbb{F}^n \)
- secret witness in \( \mathbb{F}^m \)

Preprocessing (setup): \( S(C) \rightarrow \) public parameters \((pp, vp)\)

---

\( pp, x, w \) → **Prover** → proof \( \pi \) that \( C(x, w) = 0 \) → **Verifier** → \( vp, x \)

accept or reject
A preprocessing NARK is a triple \((S, P, V)\):

- \(S(C) \rightarrow\) public parameters \((pp, vp)\) for prover and verifier
- \(P(pp, x, w) \rightarrow\) proof \(\pi\)
- \(V(vp, x, \pi) \rightarrow\) accept or reject
NARK: requirements (informal)

Prover $P(pp, x, w)$ \hspace{2cm} Verifier $V(vp, x, \pi)$

proof $\pi$ \hspace{2cm} accept or reject

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

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

Optional: Zero knowledge: $(C, pp, vp, x, \pi)$ “reveal nothing new” about $w$
SNARK: a **Succinct ARgument of Knowledge**

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

- **\(S(C)\)** $\rightarrow$ public parameters \((pp, vp)\) for prover and verifier

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

- **\(V(vp, x, \pi)\)** **fast to verify** ; time(V) = \(O(|x|, \log(|C|))\)

short "summary" of circuit  

Why preprocess \(C\)??
A succinct preprocessing NARK is a triple \((S, P, V)\): 

- \(S(C) \rightarrow\) public parameters \((pp, vp)\) for prover and verifier
- \(P(pp, x, w) \rightarrow\) short proof \(\pi\) \(\quad |\pi| = O_\lambda(\log(|C|))\)
- \(V(vp, x, \pi)\) fast to verify \(\quad\) time\((V) = O_\lambda(|x|, \log(|C|))\)

**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 $w$ 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
Types of preprocessing Setup

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

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 $C$

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

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

<table>
<thead>
<tr>
<th></th>
<th>size of proof $\pi$</th>
<th>verifier time</th>
<th>Setup</th>
<th>post-quantum?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groth’16</td>
<td>$\approx 200$ Bytes $O_\lambda(1)$</td>
<td>$\approx 1.5$ ms $O_\lambda(1)$</td>
<td>trusted per circuit</td>
<td>no</td>
</tr>
<tr>
<td>Plonk / Marlin</td>
<td>$\approx 400$ Bytes $O_\lambda(1)$</td>
<td>$\approx 3$ ms $O_\lambda(1)$</td>
<td>universal trusted setup</td>
<td>no</td>
</tr>
</tbody>
</table>

(for a circuit with $2^{20}$ gates)
Significant progress in recent years  

<table>
<thead>
<tr>
<th></th>
<th>size of proof $\pi$</th>
<th>verifier time</th>
<th>setup</th>
<th>post-quantum?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groth’16</td>
<td>$\approx 200$ Bytes $O_\lambda(1)$</td>
<td>$\approx 1.5$ ms $O_\lambda(1)$</td>
<td>trusted per circuit</td>
<td>no</td>
</tr>
<tr>
<td>Plonk / Marlin</td>
<td>$\approx 400$ Bytes $O_\lambda(1)$</td>
<td>$\approx 3$ ms $O_\lambda(1)$</td>
<td>universal trusted setup</td>
<td>no</td>
</tr>
<tr>
<td>Bulletproofs</td>
<td>$\approx 1.5$ KB $O_\lambda(\log</td>
<td>C</td>
<td>)$</td>
<td>$\approx 3$ sec $O_\lambda(</td>
</tr>
<tr>
<td>STARK</td>
<td>$\approx 100$ KB $O_\lambda(\log^2</td>
<td>C</td>
<td>)$</td>
<td>$\approx 10$ ms $O_\lambda(\log</td>
</tr>
</tbody>
</table>

(for a circuit with $2^{20}$ gates)
Significant progress in recent years (partial list)

<table>
<thead>
<tr>
<th></th>
<th>size of proof $\pi$</th>
<th>verifier time</th>
<th>setup</th>
<th>post-quantum?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groth’16</td>
<td>$O(\lambda \log^2</td>
<td>C</td>
<td>)$</td>
<td>$O(\lambda \log</td>
</tr>
<tr>
<td>Plonk / Marlin</td>
<td>$O(\lambda \log^2</td>
<td>C</td>
<td>)$</td>
<td>$O(\lambda \log</td>
</tr>
<tr>
<td>Bulletproofs</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>STARK</td>
<td>$O(\lambda \log^2</td>
<td>C</td>
<td>)$</td>
<td>$O(\lambda \log</td>
</tr>
</tbody>
</table>

Prover time is almost linear in $|C|$

(for a circuit with $2^{20}$ gates)
How to define “knowledge soundness” and “zero 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) 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

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

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

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

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

\[
\Pr[C(x, w) = 0] > \frac{1}{10^6} - \epsilon \quad \text{(for a negligible $\epsilon$)}
\]
(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 $\implies$ it learned nothing new from $\pi$

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

Main point: $\text{Sim}(C, x)$ simulates $\pi$ 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:

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

is indistinguishable from the distribution:

\[ (C, pp, vp, x, \pi) : \text{ where } (pp, vp, \pi) \leftarrow Sim(C, x) \]

Main point: \(Sim(C, x)\) simulates \(\pi\) without knowledge of \(w\)
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.
**100 DAI pool:** each coin = 100 DAI

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

**Treasury:** 300 DAI

**nullifiers**

- $nf_1$
- $nf_2$

Contract state:
- next = 4

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

**Coins Merkle root**

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

**public list of coins**

**H_1, H_2:** R $\rightarrow \{0,1\}^{256}$ CRHF
Tornado cash: deposit

100 DAI pool:
each coin = 100 DAI

Alice deposits 100 DAI:

100 DAI

\[ C_4, \text{MerkleProof}(4) \]

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
nullifiers

Merkle root

(32 bytes)

next = 4

nullifiers

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

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

public list of coins

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

![Tornado cash: deposit diagram]

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

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

- $100$ DAI
- $C_4$, MerkleProof(4)
- Coins Merkle root (32 bytes)
- Tree of height 20 ($2^{20}$ leaves)
- Public list of coins $C_1 C_2 C_3 0 0 \ldots 0$
Tornado cash: deposit (simplified)

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:**

100 DAI $\rightarrow$ 100 DAI $C_4$, MerkleProof(4)

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

**Treasury:** 400 DAI

updated Merkle root

$nf_1, nf_2$

(32 bytes)

updated contract state

nullifiers

next = 5

Every deposit: new Coin added sequentially to tree

updated Merkle root

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

public list of coins

an observer sees who owns which leaves
Box 1: *100 DAI pool:* each coin = 100 DAI

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

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

**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

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’).
Tornado cash: withdrawal (simplified)

Withdrawing coin #3 to address A:

- 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 \( x = (\text{root}, \text{nf}, A) \)

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

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')

\[ H_1, H_2: R \rightarrow \{0,1\}^{256} \]
Tornado cash: withdrawal

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

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

Bob’s ID and coin \( C_3 \) are not revealed

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

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

nullifiers

nf1

nf2

nf

next = 5

H1, H2: R → \{0,1\}^{256}

Merkle root

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

C1 C2 C3 C4 0 ... 0

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

$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
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
The Ronin-bridge hack (2022):

- In late March: $\approx600M$ USD stolen ... $\approx80M$ 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
“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:

```solidity
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: \[ C_4 = [ H_1(k, r), \text{msg.sender} ] \]

• During withdrawal Bob proves in ZK that `msg.sender` in his leaf is not currently on sanctions list.
(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) $\pi$ is ZK proof that $ct$ is computed correctly

$\Rightarrow$ 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.
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