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

**Previous lecture**

etherscan.io:

- **Address**: 0x1654b0c3f62902d7A8623f...
- **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>0x1463293c495069d61...</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 $m$
Commercial interest in SNARKs

Many more building applications that use SNARKs
**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.
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 $\mathbb{F} = \{0, \ldots, p - 1\}$ 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

$|C| = \# \text{ gates in } C$
**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 \((pp, vp)\)

Prover

\( pp, x, w \)

- 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

all algs. and adversary have access to a random oracle
NARK: requirements (informal)

Prover $P(pp, x, w)$

Verifier $V(vp, x, \pi)$

proof $\pi$ → 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 $w$

(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**\((pp, x, w)\) → **short** proof \(\pi\);

  \[
  \text{len}(\pi) = O_\lambda(\text{polylog}(|C|))
  \]

- **V**\((vp, x, \pi)\) **fast to verify**;

  \[
  \text{time}(V) = O_\lambda(|x|, \text{polylog}(|C|))
  \]

  short “summary” of circuit

  \[\text{V has no time to read } C !!\]

  [ 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 $(pp, vp)$

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 $C$

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

- **transparent setup**: $S(C)$ does not use secret data (no trusted setup)
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>$\approx 200$ 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)
<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 $O(1)$</td>
<td>$\approx 1.5$ ms $O(1)$</td>
<td>trusted per circuit</td>
<td>no</td>
</tr>
<tr>
<td>Plonk / Marlin</td>
<td>$\approx 400$ Bytes $O(1)$</td>
<td>$\approx 3$ ms $O(1)$</td>
<td>universal trusted setup</td>
<td>no</td>
</tr>
<tr>
<td>Bulletproofs</td>
<td>$\approx 1.5$ KB $O(\log</td>
<td>C</td>
<td>)$</td>
<td>$\approx 3$ sec $O(\log</td>
</tr>
<tr>
<td>STARK</td>
<td>$\approx 100$ KB $O(\log^2</td>
<td>C</td>
<td>)$</td>
<td>$\approx 10$ ms $O(\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 $O$</th>
<th>setup $O$</th>
<th>post-quantum?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groth’16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Plonk / Marlin</td>
<td></td>
<td></td>
<td></td>
<td></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”?
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.

$$
\text{if } \begin{align*}
gp & \leftarrow S_{\text{init}}(), \\
(C, x, \text{state}) & \leftarrow A_0(gp), \\
(pp, vp) & \leftarrow S_{\text{index}}(gp, C), \\
\pi & \leftarrow A_1(pp, x, \text{state}), \\
w & \leftarrow Ext(gp, C, x)
\end{align*}
$$

Then

$$\Pr \left[ V(vp, x, \pi) = \text{accept} \Rightarrow C(x, w) = 0 \right] \geq 1 - \epsilon \quad \text{(for a negl. } \epsilon \text{)}$$
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 \( \pi \) “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. \( (pp, vp, \pi) \leftarrow \text{Sim}(C, x) \) “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 \(\text{Sim}\) such that for all \(x \in \mathbb{F}^n\) s.t. \(\exists w: C(x, w) = 0\) the distribution:

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

is indistinguishable from the distribution:

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

**Main point:** \(\text{Sim}(C, x)\) simulates \(\pi\) without knowledge of \(w\)
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

Treasury: 300 DAI

nullifiers

nf_1

nf_2

coins Merkle root

(32 bytes)

next = 4

Contract state

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

CRHF

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

C_1 C_2 C_3 0 0 ... 0

public list of coins

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

100 DAI pool:
each coin = 100 DAI

Alice deposits 100 DAI:

Build Merkle proof for leaf #4:
MerkleProof(4) (leaf=0)
choose random $k, r$ in $R$
set $C_4 = H_1(k, r)$

100 DAI

$C_4, \text{MerkleProof}(4)$

Treasury: 300 DAI

nullifiers

coins

Merkle root

(32 bytes)

next = 4

$nf_1$

$nf_2$

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

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

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

Merkle root (32 bytes)
next = 4

H_1, H_2: R → \{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 ... 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

100 DAI $\xrightarrow{}$ $C_4$, MerkleProof(4)

H$_1$, H$_2$: $R \rightarrow \{0,1\}^{256}$

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

public list of coins
Tornado cash: deposit

100 DAI pool:
each coin = 100 DAI

Alice deposits 100 DAI:

100 DAI

C₄, MerkleProof(4)

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

Treasury: 400 DAI

nullifiers

updated Merkle root
(32 bytes)

updated contract state

Every deposit: new Coin
added sequentially to tree

updated Merkle root

tree of height 20
(2²⁰ leaves)

public list of coins

an observer sees who owns which leaves
**Tornado cash: withdrawal**

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

Withdraw coin #3 to addr A:

- has note = (k', r')
- set nf = H₂(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 \( \text{nf} = H_2(k’) \)

Bob builds zk-SNARK proof \( \pi \) for

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

- secret witness \( w = (k’, r’, C_3, \text{MerkleProof}(C_3)) \)

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

1. \( C_3 = \) (leaf #3 of \text{root}), i.e. \( \text{MerkleProof}(C_3) \) is valid,
2. \( C_3 = H_1(k’, r’) \), and
3. \( \text{nf} = H_2(k’) \).

**Diagram:**

- Merkle root
- tree of height 20 (\( 2^{20} \) leaves)
- \( C_1, C_2, C_3, C_4, 0 \ldots 0 \)

**Notes:**

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

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

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 $\pi$ for $x$ to build a proof $\pi'$ for some “related” $x'$ (e.g., where in $x'$ the address A is replaced by some $A'$)

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

secret witness $w = (k', r', C_3, \text{MerkleProof}(C_3))$
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
Tornado cash: withdrawal

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

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

nf, proof π, A → 100 DAI to address A (over Tor)

**Treasury:** 300 DAI

coins

Merkle root

nf₁, nf₂, nf

nullifiers

next = 5

contract state

**Merkle root**

tree of height 20

(2¹⁰ leaves)

H₁, H₂: R → {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 \( nf = H₂(k') \) 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)
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
“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 ...
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