Using zkSNARKs

Dan Boneh
Recap: high-level goals

• Private transactions on a public blockchain
• Blockchain scaling, such as proof-based Rollup
• Privately prove compliance, such as a private proof of solvency
Recap: non-interactive proof systems (for NP)

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

public statement in \( \mathbb{F}_p^n \) \( \rightarrow \) secret witness in \( \mathbb{F}_p^m \)

Let \( \mathbf{x} \in \mathbb{F}_p^n \). Two standard goals for prover \( P \):

1. **Soundness**: convince Verifier that \( \exists \mathbf{w} \text{ s.t. } C(\mathbf{x}, \mathbf{w}) = 0 \)
   (e.g., \( \exists \mathbf{w} \text{ such that } [ H(\mathbf{w}) = \mathbf{x} \text{ and } 0 < \mathbf{w} < 2^{60} ] \) )

2. **Knowledge**: convince Verifier that \( P \) “knows” \( \mathbf{w} \text{ s.t. } C(\mathbf{x}, \mathbf{w}) = 0 \)
   (e.g., \( P \) knows a \( \mathbf{w} \) such that \( H(\mathbf{w}) = \mathbf{x} \) )
A non-interactive proof system is a triple \((S, P, V)\):

- \(S(C) \rightarrow \) public parameters \((S_p, S_v)\) for prover and verifier
  
  \((S_p, S_v)\) is called a *reference string*

- \(P(S_p, x, w) \rightarrow \) proof \(\pi\)

- \(V(S_v, x, \pi) \rightarrow \) accept or reject
proof systems: properties (informal)

Prover $P(pp, x, w)$

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

proof $\pi$

accept or reject

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

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

in some cases, **soundness** is sufficient: $\exists w$ s.t. $C(x, w) = 0$

**Zero knowledge** (optional): $(x, \pi)$ “reveals nothing” about $w$
SNARK: succinct argument of knowledge

Goal: P wants to show that it knows \( w \) s.t. \( C(x, w) = 0 \)

**Succinct:**

- Proof \( \pi \) should be **short** [i.e., \( |\pi| = O(\log(|C|), \lambda) \)]
- Verifying \( \pi \) should be **fast** [i.e., \( \text{time}(V) = O(|x|, \log(|C|), \lambda) \)]

note: if SNARK is zero-knowledge, then called a **zkSNARK**
zkSNARK applications
Blockchain Applications

Scalability:
- SNARK Rollup (zkSNARK for privacy from public)

Privacy:
- Private Tx on a public blockchain
  - Confidential transactions
  - Zcash

Compliance:
- Proving solvency in zero-knowledge
- Zero-knowledge taxes
... but first: commitments

Cryptographic commitment: emulates an envelope

Many applications: e.g., a DAPP for a sealed bid auction

• Every participant commits to its bid,
• Once all bids are in, everyone opens their commitment
Cryptographic Commitments

Syntax: a commitment scheme is two algorithms

- **commit**(msg, r) \(\rightarrow\) com
  - secret randomness in \(R\)
  - commitment string

- **verify**(msg, com, r) \(\rightarrow\) accept or reject

  anyone can verify that commitment was opened correctly
Commitments: security properties

• **binding**: Bob cannot produce two valid openings for \( \text{com} \).
  Formally: no efficient adversary can produce
  \[ \text{com}, (m_1, r_1), (m_2, r_2) \]
  such that
  \[
  \text{verify}(m_1, \text{com}, r_1) = \text{verify}(m_2, \text{com}, r_2) = \text{accept}
  \]
  and \( m_1 \neq m_2 \).

• **hiding**: \( \text{com} \) reveals nothing about committed data
  \[
  \text{commit}(m, r) \rightarrow \text{com}, \text{ and } r \text{ is uniform in } R \ (r \leftarrow R),
  \]
  then \( \text{com} \) is statistically independent of \( m \).
Example 1: hash-based commitment

Fix a hash function \( H: M \times R \rightarrow C \) (e.g., SHA256)

where \( H \) is collision resistant, and \(|R| \gg |C|\)

- commit\((m \in M, \ r \leftarrow R)\): \( \text{com} = H(m, r) \)
- verify\((m, \text{com}, r)\): accept if \( \text{com} = H(m, r) \)

binding: follows from collision resistance of \( H \)

hiding: follows from a mild assumption on \( H \)
Example 2: Pedersen commitment

\[ G = \text{finite cyclic group} = \{1, g, g^2, ..., g^{q-1}\} \]

where \( g^i \cdot g^j = g^{(i+j \mod q)} \)

\[ q = |G| \text{ is called the order of } G. \text{ Assume } q \text{ is a prime number.} \]

Fix \( h \) in \( G \) and let \( R = \{0, 1, ..., q-1\} \). For \( m, r \in R \) define

\[ H(m, r) = g^m \cdot h^r \in G \]

**Fact:** for a “cryptographic” group \( G \), this \( H \) is collision resistant.

\[ \Rightarrow \text{ commitment scheme: } \text{commit} \text{ and verify as in example 1} \]

\[ \text{commit}(m \in R, r \leftarrow R) = H(m, r) = g^m \cdot h^r \]
An interesting property

\[ \text{commit}(m \in R, \ r \leftarrow R) = H(m, r) = g^m \cdot h^r \]

Suppose:
\[ \text{commit}(m_1 \in R, \ r_1 \leftarrow R) \rightarrow com_1 \]
\[ \text{commit}(m_2 \in R, \ r_2 \leftarrow R) \rightarrow com_2 \]

Then: \[ com_1 \times com_2 = g^{m_1+m_2} \cdot h^{r_1+r_2} = \text{commit}(m_1+m_2, \ r_1+r_2) \]

\[ \Rightarrow \text{anyone can sum committed value} \]
Confidential Transactions
Confidential Tx (CT)

Goal: hide amounts in Bitcoin transactions.

⇒ businesses cannot use for supply chain payments

will not hide Tx fee
Confidential Tx: how?

Bitcoin Tx today: Google: 30 → Alice: 1, Google: 29

The plan: replace amounts by commitments to amounts

Google: $\text{com}_1 \rightarrow$ Alice: $\text{com}_2$, Google: $\text{com}_3$

where $\text{com}_1 = \text{commit}(30, r_1)$, $\text{com}_2 = \text{commit}(1, r_2)$, $\text{com}_3 = \text{commit}(29, r_3)$
Now blockchain hides amounts

<table>
<thead>
<tr>
<th>Source Address</th>
<th>Destination Address</th>
<th>Transaction ID</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>c2561b292ed4878bb28478a8cafd1f99a01faeb9c5a906715fa595cac0e8d1d8</td>
<td>ae23b452d8</td>
<td>16k4365rzdecpgwjdnnbekxj696mbchwx</td>
<td>3bd6e25fqd</td>
</tr>
<tr>
<td>1bsh4kd9zjt4djc0o755us1jvttmvtmreb7</td>
<td>187b6cf54a8</td>
<td>1jgbpwstdmrrozg9xppdqrhtnb5cspa</td>
<td>8c528ad9fa</td>
</tr>
<tr>
<td>FEE: 0.00179523 BTC</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

How much was transferred ???
The problem: how will miners verify Tx?

Solution: zkSNARK (special purpose, optimized for this problem)

- Google: (1) privately send $r_2$ to Alice
  (2) construct a zkSNARK $\pi$ where
  
  statement = $x = (\text{com}_1, \text{com}_2, \text{com}_3)$

  witness = $w = (m_1, r_1, m_2, r_2, m_3, r_3)$

  and circuit $C(x, w)$ outputs 0 if:
  
  (i) $\text{com}_i = \text{commit}(m_i, r_i)$ for $i=1,2,3$,
  (ii) $m_1 = m_2 + m_3 + \text{Tx Fees}$,
  (iii) $m_2 \geq 0$ and $m_3 \geq 0$

Google: $\text{com}_1 \rightarrow$ Alice: $\text{com}_2$, Google: $\text{com}_3$

$\text{com}_1 = \text{commit}(30, r_1)$, $\text{com}_2 = \text{commit}(1, r_2)$, $\text{com}_3 = \text{commit}(29, r_3)$
The problem: how will miners verify Tx?

- Google: (1) privately send $r_2$ to Alice
  (2) construct zkSNARK proof $\pi$ that Tx is valid
  (3) append $\pi$ to Tx (need short proof! $\Rightarrow$ zkSNARK)

Miners: accept Tx if proof $\pi$ is valid (need fast verification)
  $\Rightarrow$ learn Tx is valid, but amounts are hidden
Note: Alice needs $r_2$ to spend her UTXO otherwise: she cannot construct proof $\pi$

statement = $x = (\text{com}_1, \text{com}_2, \text{com}_3)$

witness = $w = (m_1, r_1, m_2, r_2, m_3, r_3)$

Circuit $C(x,w)$ outputs 0 if:

(i) $\text{com}_i = \text{commit}(m_i, r_i)$,

(ii) $m_1 = m_2 + m_3 + \text{Tx Fees}$,

(iii) $m_2 \geq 0$ and $m_3 \geq 0$

Easy to check with Pedersen:

set $\text{com} = \text{com}_1/\text{com}_2 \cdot \text{com}_3 \cdot g^\text{Tx Fees}$

prove that $\text{com} = \text{commit}(0, r)$

remaining proof is $\approx$400 bytes
Zcash  (simplified)
**Zcash**

**Goal**: fully private payments ... like cash, but across the Internet

challenge: will governments allow this ???

Zcash blockchain supports two types of TXOs:

- transparent TXO (as in Bitcoin)
- shielded (anonymized)

a Tx can have both types of inputs, both types of outputs
Addresses and TXOs

H₁, H₂, H₃: cryptographic hash functions.

(1) shielded address: random sk \(\leftarrow X\), \(pk = H_1(sk)\)

(2) shielded TXO (note) owned by address \(pk\):
- TXO owner has (from payer): value \(v\) and \(r \leftarrow R\)
- on blockchain: \(coin = H_2((pk, v), r)\) (commit to \(pk, v\))

pk: addr. of owner,  \(v\): value of coin,  \(r\): random chosen by payer
The blockchain

<table>
<thead>
<tr>
<th>coins</th>
<th>nullifiers</th>
<th>transparent-TXOs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{coin}_1$</td>
<td>$\text{nfa}_1$</td>
<td>similar to Bitcoin UTXO set</td>
</tr>
<tr>
<td>$\text{coin}_2$</td>
<td>$\text{nfa}_2$</td>
<td></td>
</tr>
<tr>
<td>$\text{coin}_3$</td>
<td>$\vdots$</td>
<td></td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\vdots$</td>
<td></td>
</tr>
</tbody>
</table>

just Merkle root ... append only tree (coins are never removed)

explicit list: one entry per spent coin
Transactions: an example

owner of \textbf{coin} = H_2((pk, v), r) \quad \text{(Tx input)}

wants to send \textbf{coin} funds to:

\begin{align*}
(v = v' + v'') & \quad \text{shielded } pk', v' \\
\text{transp. } pk'', v'' & \quad \text{(Tx output)}
\end{align*}

\textbf{step 1}: construct new coin: \textbf{coin'} = H_2((pk', v'), r')

by choosing random \quad r' \leftarrow R \quad \text{(and sends } v', r' \text{ to owner of } pk')

\textbf{step 2}: compute \textbf{nullifier} for spent coin \quad nf = H_3(sk, \text{index of coin in Merkle tree})

nullifier \textbf{nf} is used to “cancel” \textbf{coin} \quad \text{(no double spends)}

key point: miners learn that some coin was spent, but not which one!
Transactions: an example

**step 3:** construct a zkSNARK proof $\pi$ for

statement = $x = (\text{current Merkle root, coin', nf, v''})$

witness = $w = (\text{sk, (v, r), (pk', v', r'), Merkle proof for coin})$

$C(x, w)$ outputs 0 if: with $coin := H_2((pk=H_1(sk), v), r)$ check

1. Merkle proof for $coin$ is valid,
2. $coin' = H_2((pk', v'), r')$
3. $v = v' + v''$ and $v' \geq 0$ and $v'' \geq 0$,
4. $nf = H_3(sk, \text{index-of-coin-in-Merkle-tree})$

The Zcash circuit

from Merkle proof
step 4: send \( (\text{coin}', \text{nf}, \text{transparent-TXO}, \text{proof } \pi) \) to miners,

send \( (v', r') \) to owner of pk'

step 5: miners verify

(i) proof \( \pi \) and transparent-TXO

(ii) verify that \( \text{nf} \) is not in nullifier list (prevent double spending)

if so, add \text{coin}' to Merkle tree, add \text{nf} to nullifier list,

add transparent-TXO to UTXO set.
Summary

• Tx hides which coin was spent
  ⇒ coin is never removed from Merkle tree, but cannot be double spent thanks to nullifer

  note: prior to spending coin, only owner knows nf:
  \[ nf = H_3(Sk, \text{index of coin in Merkle tree}) \]

• Tx hides address of coin’ owner

• Miners can verify Tx is valid, but learn nothing about Tx details.
A simple PCP-based SNARK

[Kilian’92, Micali’94]
A simple construction: PCP-based SNARK

The PCP theorem: Let $C(x, w)$ be a circuit where $x \in \mathbb{F}_p^n$.
there is a proof system that for every $x$ proves $\exists w: C(x, w) = 0$
as follows:

Prover $P(S_p, x, w)$

proof $\pi$

Verifier $V(S_v, x)$

read only $O(\lambda)$ bits of $\pi$,
output accept or reject

V always accepts valid proof. If no $w$, then V rejects with high prob.

size of proof is $poly(|C|)$. (not succinct)
Converting a PCP proof to a SNARK

Prover $P(S_p, x, w)$

Verifier $V(S_v, x)$

Verifier sees $O(\lambda \log |C|)$ data $\Rightarrow$ succinct proof.

Problem: interactive
Making the proof non-interactive

The Fiat-Shamir heuristic:

• public-coin interactive protocol ⇒ non-interactive protocol
  public coin: all verifier randomness is public (no secrets)

\[
\text{Prover } P(S_p, x, w) \quad \text{Verifier } V(S_v, x)
\]

\[
\begin{align*}
\text{msg1} & \quad r \\
\text{choose random bits } r & \quad \text{msg2}
\end{align*}
\]

accept or reject
Making the proof non-interactive

**Fiat-Shamir heuristic:** \( H: M \mapsto R \) a cryptographic hash function

- idea: prover generates random bits on its own (!)

Prover \( P(S_p, x, w) \)
- generate msg1
- \( r \leftarrow H(x, \text{msg1}) \)
- generate msg2

Verifier \( V(S_v, x) \)
- \( \pi = (\text{msg1}, \text{msg2}) \)
- \( |\pi| = O(\lambda \log |C|) \)
- \( r \leftarrow H(x, \text{msg1}) \)
- accept or reject

**Thm:** this is a secure SNARK assuming \( H \) is a random oracle
Are we done?

Simple transparent SNARK from the PCP theorem

• Use Fiat-Shamir heuristic to make non-interactive
• We will apply Fiat-Shamir in many other settings

The bad news: an impractical SNARK --- Prover time too high

Better SNARKs: next lecture! Goal: Time(Prover) = O(|C|)
Next lecture: How to build an efficient SNARK