Proof Systems and SNARKs

Benedikt Bünz, Stanford
Managing assets on a blockchain: key principles

• **Universal verifiability** of blockchain rules
  ⇒ all data written to the blockchain is public; everyone can verify
  ⇒ added benefit: interoperability between chains

• Assets are **controlled by signature keys**
  ⇒ assets **cannot** be transferred without a valid signature
    (of course, users can choose to custody their keys)
Naïve reasoning:

universal verifiability ⇒ blockchain data is public
⇒ all transactions data is public
otherwise, how we can verify Tx?

not quite ...

crypto magic ⇒ private Tx on a publicly verifiable blockchain
Public blockchain & universal verifiability

<table>
<thead>
<tr>
<th>public blockchain</th>
<th>(abstractly)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>current state</strong></td>
<td><strong>Tx</strong>  <strong>π</strong></td>
</tr>
<tr>
<td>encrypted (or committed)</td>
<td>encrypted (or committed)</td>
</tr>
</tbody>
</table>

- **Tx data**: encrypted (or committed)
- **Proof π**: zero-knowledge proof that (reveals nothing about Tx data)
  1. plaintext Tx data is consistent with plaintext current state
  2. plaintext new state is correct
Public blockchain & universal verifiability

- **Tx data**: encrypted (or committed)
- **Proof π**: zero-knowledge proof that (reveals nothing about Tx data)
  1. Plaintext Tx data is consistent with plaintext current state
  2. Plaintext new state is correct
Zero Knowledge Proof Systems
(1) 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 $+, -, \text{ or } \times$
    • inputs are labeled $1, x_1, \ldots, x_n$
  • defines an $n$-variate polynomial with an evaluation recipe

• $|C| = \# \text{ multiplication gates in } C$
Boolean circuits as arithmetic circuits

Boolean circuits: circuits with AND, OR, NOT gates

Encoding a boolean circuit as an arithmetic circuit over \( \mathbb{F}_p \):

- **AND** \((x, y)\) encoded as \(x \cdot y\)
- **OR** \((x, y)\) encoded as \(x + y - x \cdot y\)
- **NOT** \((x)\) encoded as \(1 - x\)

<table>
<thead>
<tr>
<th>(x)</th>
<th>(y)</th>
<th>OR ((x, y))</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Interesting arithmetic circuits

• \( C_{\text{hash}}(h, m) \): outputs 0 if \( \text{SHA256}(m) = h \), and \( \neq 0 \) otherwise

\[
C_{\text{hash}}(h, m) = (h - \text{SHA256}(m)), \quad |C_{\text{hash}}| \approx 20K \text{ gates}
\]

• \( C_{\text{sig}}((pk, m), \sigma) \): output 0 if \( \sigma \) is a valid ECDSA signature of \( m \) under \( pk \)
(2) non-interactive proof systems

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

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

2. **Knowledge**: convince Verifier that $P$ “knows” $w$ s.t. $C(x, w) = 0$

   (e.g., $P$ knows a $w$ such that $H(w) = x$)
Why can’t prover simply send \( w \) to verifier?

- Verifier checks if \( C(x, w) = 0 \) and accepts if so.

**Problems with this:**

1. \( w \) might be secret: prover cannot reveal \( w \) to verifier
2. \( w \) might be long: we want a “short” proof
3. computing \( C(x, w) \) may be hard: want to minimize Verifier’s work
Non-interactive Proof Systems (for NP)

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

Public input in \( \mathbb{F}_p^n \) \hspace{1cm} secret witness in \( \mathbb{F}_p^m \)

setup: \( S(C) \rightarrow \text{public parameters } (S_p, S_v) \)

Prover \( P(S_p, x, w) \) \hspace{2cm} Verifier \( V(S_v, x, \pi) \)

proof \( \pi \) \hspace{2cm} output accept or reject
A non-interactive proof 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
Proof systems: properties (informal)

Prover $P(pp, x, w)$

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

proof $\pi$

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

Proof of knowledge: $V$ accepts $\implies P \text{ "knows" } w \text{ s.t. } C(x, w) = 0$

Zero knowledge (optional): $(x, \pi) \text{ "reveals nothing" about } w$
(b) Zero knowledge

(S, P, V) is zero knowledge if proof \(\pi\) “reveals nothing” about \(w\)

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

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

is indistinguishable from the distribution:

\[
(S_p, S_v, x, \pi) \quad \text{where} \quad (S_p, S_v, \pi) \leftarrow Sim(x)
\]

key point: \(Sim(x)\) simulates proof \(\pi\) without knowledge of \(w\)
(3) Succinct arguments: SNARKs

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**
(3) Succinct arguments: SNARKs

Goal: P wants to show that it knows \( w \) s.t. \( C(x, w) = 1 \)

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

note: if SNARK is zero-knowledge, then called a zkSNARK
An example

Prover says: I know \((x_1, \ldots, x_n) \in X\) such that \(H(x_1, \ldots, x_n) = y\)

**SNARK:** size\((\pi)\) and VerifyTime\((\pi)\) should be \(O(\log n)\)!!
An example

How is this possible ???

**SNARK:** size(\(\pi\)) and VerifyTime(\(\pi\)) should be \(O(\log n)\) !!

Prover

statement: \(y\)

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

Proof \(\pi\)

Verifier

statement: \(y\)

accept or reject
Types of pre-processing Setup

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

Types of setup:

**trusted setup per circuit:** $S(C)$ uses data that must be kept secret

compromised trusted setup $\Rightarrow$ can prove false statements

**updatable universal trusted setup:** $(S_p, S_v)$ can be updated by anyone

**transparent:** $S()$ 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 $O_\lambda(1)$
  • 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)

|                  | size of $|\pi|$ | size of $|S_p|$ | verifier time | trusted setup?     |
|------------------|------------|---------------|----------------|-------------------|
| Groth’16         | $O(1)$     | $O(|C|)$      | $O(1)$         | yes/per circuit   |
| PLONK/MARLIN     | $O(1)$     | $O(|C|)$      | $O(1)$         | yes/updatable     |
| Bulletproofs     | $O(\log|C|)$ | $O(1)$      | $O(|C|)$       | no                |
| STARK            | $O(\log|C|)$ | $O(1)$      | $O(\log|C|)$   | no                |
| DARK             | $O(\log|C|)$ | $O(1)$      | $O(\log|C|)$   | no                |

\[ \vdots \]
A typical SNARK software system

- **DSL program**: Circom, ZoKrates, ...
- **SNARK friendly format**: R1CS, AIR, TurboPlonk
- **SNARK backend**: Proof $\pi$
- **Proof $\pi$**: $x$, witness
- **Verifier**: accept/reject
- **CPU heavy**
- **Compiler**
- **Setup**: $(S_p, S_v)$
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
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
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
  \( \text{commit}(m, r) \rightarrow \text{com}, \) and \( r \) is uniform in \( R \) \( (r \leftarrow R) \), then \( \text{com} \) is statistically independent of \( m \).
Confidential Transactions
Confidential Tx (CT)

Goal: hide amounts in Bitcoin transactions.

<table>
<thead>
<tr>
<th>BTC Address 1</th>
<th>BTC Amount 1</th>
<th>BTC Address 2</th>
<th>BTC Amount 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>16k4365RzdeCPKGwJDNNBEkXj696MbChwx</td>
<td>0.53333328 BTC</td>
<td>1JgVBpwSTDMTRoZXg9XpPDQRRHtNb5CsPA</td>
<td>0.01031593 BTC (U)</td>
</tr>
<tr>
<td>1Bsh4KD9ZJT4dJcoo755suS1jvtmVmREb7</td>
<td>1.4787788 BTC</td>
<td>1AFLeD4EtG2uZmFxfmfdXCyGUNqCqD5887u</td>
<td>2 BTC (S)</td>
</tr>
</tbody>
</table>

FEE: 0.00179523 BTC

⇒ businesses cannot use for supply chain payments
Confidential Tx: how?

Bitcoin Tx today:

Google: 30 $\rightarrow$ 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

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:

CT arithmetic circuit

(i) $\text{com}_i = \text{commit}(m_i, r_i)$ for $i=1,2,3$,
(ii) $m_1 = m_2 + m_3 + \text{TxFees}$,
(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! ⇒ zkSNARK)

Miners: accept Tx if proof $\pi$ is valid (need fast verification) ⇒ learn Tx is valid, but amounts are hidden
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

$H_1$, $H_2$, $H_3$: 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: $\text{coin} = H_2((pk, v), r)$ (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>coin₁</td>
<td>nf₁</td>
<td>similar to Bitcoin UTXO set</td>
</tr>
<tr>
<td>coin₂</td>
<td>nf₂</td>
<td></td>
</tr>
<tr>
<td>coin₃</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td>...</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*}
&\text{shielded} \quad pk', v' \\
&(v = v' + v'') \\
&\text{transp.} \quad pk'', v'' \quad \text{(Tx output)}
\end{align*}

\textbf{step 1:} construct new \textbf{coin}: \begin{align*}
\textbf{coin'} &= H_2((pk', v'), r') \\
&\text{by choosing random } r' \leftarrow R \quad \text{(and sends } v', r' \text{ to owner of } pk')
\end{align*}

\textbf{step 2:} compute \textbf{nullifier} for spent \textbf{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!
step 3: construct a zkSNARK proof $\pi$ for

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

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

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

1. Merkle proof for coin is valid,
2. $\text{coin'} = H_2((pk', v'), r')$
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
  \[ \Rightarrow \text{coin is never removed from Merkle tree, but cannot be double spent thanks to nullifier} \]

  note: prior to spending \textbf{coin}, only owner knows \textbf{nf}:

  \[ nf = H_3(Sk, \text{index of coin in Merkle tree}) \]

• Tx hides address of \textbf{coin}' owner

• Miners can verify Tx is valid, but learn nothing about Tx details.