Theory and Practice of Succinct Zero Knowledge Proofs

Lecture 09: SNARKs from Linear PCPs
Announcements

• Next assignment due Monday 10/09 midnight
• Next discussion-oriented class 10/10
  • If you’re presenting, reach out to me by this Friday 10/06!
• Project:
  • Project proposal **deadline is 10/10**!
  • Talk to me if you’d like to chat about project topics
Recap
PIOP + PC = SNARK

Polynomial IOPs

Compiler

zkSNARK

Polynomial Commitment
## SNARKs So Far

<table>
<thead>
<tr>
<th>PIOP</th>
<th>PC Scheme</th>
<th>Setup</th>
<th>P Time</th>
<th>V Time</th>
<th>Pf size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marlin</td>
<td>KZG</td>
<td>Trusted</td>
<td>$O(n \log n)$</td>
<td>$O(\log n)$</td>
<td>$\sim 1\text{ kB}$</td>
</tr>
<tr>
<td>Spartan</td>
<td>DL-based</td>
<td>Transparent</td>
<td>$O(n)$</td>
<td>$O(\sqrt{n})$</td>
<td>$10 - 100\text{ kB}$</td>
</tr>
</tbody>
</table>

How small can verifier time and proof size be?
New Recipe: LIPs + Linear Commitments
New Compiler

- Linear IPs
- Linear Commitment

Compiler

zkSNARK
# SNARK Comparison

<table>
<thead>
<tr>
<th>PIOP</th>
<th>PC Scheme</th>
<th>Setup</th>
<th>P Time</th>
<th>V Time</th>
<th>Pf size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marlin</td>
<td>KZG</td>
<td>Trusted</td>
<td>$O(n \log n)$</td>
<td>$O(\log n)$</td>
<td>$\sim 1$ kB</td>
</tr>
<tr>
<td>Spartan</td>
<td>DL-based</td>
<td>Transparent</td>
<td>$O(n)$</td>
<td>$O(\sqrt{n})$</td>
<td>10 - 100 kB</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LIP</th>
<th>LC Scheme</th>
<th>Setup</th>
<th>P Time</th>
<th>V Time</th>
<th>Pf size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Groth16</td>
<td>GGM</td>
<td>Circuit-specific trusted</td>
<td>$O(n \log n)$</td>
<td>$O(1)$</td>
<td>$&lt; 200B$</td>
</tr>
</tbody>
</table>
Definition: Linear IP
Recall: PIOPs \[^{[GWC19, CHMMVW20, BFS20]}\]

- **Completeness**: Whenever \((F, x, w) \in \mathcal{R}\), there is a strategy for \(P\) that outputs **only polynomials**, and which causes \(V\) to accept.

- **Knowledge Soundness**: Whenever \(V\) accepts against a \(P\) that outputs **only polynomials**, then \(P\) “knows” \(w\) such that \((F, x, w) \in \mathcal{R}\).

- **Bounded-query ZK**: Whenever \((F, x, w) \in \mathcal{R}\), a \(V\) that makes up to \(b\) queries to polynomials learns nothing about \(w\).
New: Linear IOPs \[\text{[GGPR13, BCIOP13, SBVBPW13]}\]

- **Completeness**: Whenever \((F, x, w) \in \mathcal{R}\), there is a strategy for \(P\) that outputs **only linear functions**, and which causes \(V\) to accept.

- **Knowledge Soundness**: Whenever \(V\) accepts against a \(P\) that outputs **only linear functions**, then \(P\) “knows” \(w\) such that \((F, x, w) \in \mathcal{R}\).

- **Bounded-query ZK**: Whenever \((F, x, w) \in \mathcal{R}\), a \(V\) that makes up to \(b\) queries to polys learns nothing about \(w\).
Construction:
Linear IP for R1CS
An rank-1 constraint system (R1CS) is a generalization of arithmetic circuits

\[
(F := (\mathbb{F}, n \in \mathbb{N}, A, B, C), x, w)
\]

\[
z := \begin{bmatrix} x \\ w \end{bmatrix} \quad n \begin{bmatrix} A \\ z \end{bmatrix} \cdot \begin{bmatrix} B \\ z \end{bmatrix} = \begin{bmatrix} C \end{bmatrix}
\]
Attempt #1

\[ z = (x, w) \]

- Idea 1: Just a random vector: \( \vec{r} := (1, r, r^2, \ldots, r^{n-1}) \)
  - \( \langle z, \vec{r} \rangle \) doesn’t seem that useful…
Attempt #2

**Prover**

\[ (F, x, w) \]

\[ z = (x, w) \]

**Verifier**

\[ (F, x) \]

\[ Q \]

\[ b \]

\[ | \vec{r}_A, \vec{r}_B, \vec{r}_C | \]

- **Hint:** Think of the lincheck PIOP!
- **Idea 2:** \( \vec{r} \cdot M \) for each \( M \in \{A, B, C\} \)
  - What can we do with \( \langle \vec{r}_A, z \rangle, \langle \vec{r}_B, z \rangle, \langle \vec{r}_C, z \rangle \)?

What can we put here?
Attempt #2

\[ z = (x, w) \]

• Hint: Think of the lincheck PIOP!

• Idea 2: \( \vec{r} \cdot M \) for each \( M \in \{A, B, C\} \)
  • What can we do with \( \langle \vec{r}A, z \rangle, \langle \vec{r}B, z \rangle, \langle \vec{r}C, z \rangle \)?
Attempt #2

- **Prover**
  - $(F, x, w)$
  - $z = (x, w)$

- **Verifier**
  - $(F, x)$
  - $Q$
  - $b$
  - $\langle \vec{r}A, z \rangle \cdot \langle \vec{r}B, z \rangle = \langle \vec{r}C, z \rangle$

- **Hint:** Think of the lincheck PIOP!
- **Idea 2:** $\vec{r} \cdot M$ for each $M \in \{A, B, C\}$
  - How about checking the product?
Let's analyze this

\[
\langle \vec{r}M, z \rangle = \langle \vec{r}, Mz \rangle = \sum_i r^i \langle m_i, z \rangle
\]

Then we have that

\[
\langle \vec{r}A, z \rangle \cdot \langle \vec{r}B, z \rangle
\]

\[
= \left( \sum_i r^i \langle a_i, z \rangle \right) \cdot \left( \sum_j r^j \langle b_j, z \rangle \right)
\]

\[
= \sum_{i,j} r^{i+j} \cdot \langle a_i, z \rangle \cdot \langle b_j, z \rangle
\]
Let's analyze this

\[
= \sum_{i,j} r^{i+j} \cdot \langle a_i, z \rangle \cdot \langle b_j, z \rangle
\]

\[
= \sum_i r^{2i} \cdot \langle a_i, z \rangle \cdot \langle b_i, z \rangle + \sum_{i \neq j} r^{i+j} \cdot \langle a_i, z \rangle \cdot \langle b_j, z \rangle
\]

\[
= \langle r^2, Cz \rangle + \text{junk}
\]

Almost there!
We just have to get rid of … \(O(n^2)\) junk terms 😞
Attempt #3: A Different Basis

We saw that, for each $M \in \{A, B, C\}$,

$$\langle \vec{r}M, z \rangle = \sum_i r^i \langle m_i, z \rangle$$

This looks like a polynomial!

$$p_M(r) = \sum_i r^i \langle m_i, z \rangle$$

This is a polynomial in the monomial basis. Using this basis didn’t work. What should we try next?
Attempt #3: Lagrange Basis!

- New idea: query for $\overline{L(r)} \cdot M := (L_1(r), L_2(r), \ldots, L_n(r)) \cdot M$
  - $L_i(X)$ is $i$-th Lagrange basis poly for $n$-sized domain $H$
Let's analyze this

\[ \langle \overrightarrow{L(X)}M, z \rangle = \sum_i L_i(X)\langle m_i, z \rangle \]

Then we have that

\[ \langle \overrightarrow{L(X)}A, z \rangle \cdot \langle \overrightarrow{L(X)}B, z \rangle \]

\[ = \left( \sum_i L_i(X)\langle a_i, z \rangle \right) \cdot \left( \sum_j L_j(X)\langle b_j, z \rangle \right) \]

\[ = \sum_{i,j} L_i(X)L_j(X) \cdot \langle a_i, z \rangle \cdot \langle b_j, z \rangle \]
Let’s analyze this

\[
\begin{align*}
&= \sum_{i,j} L_i(X) L_j(X) \cdot \langle a_i, z \rangle \cdot \langle b_j, z \rangle \\
&= \sum_i L_i(X)^2 \cdot \langle a_i, z \rangle \cdot \langle b_i, z \rangle + \sum_{i\neq j} L_i(X) L_j(X) \cdot \langle a_i, z \rangle \cdot \langle b_j, z \rangle \\
&= \sum_i L_i(X)^2 \cdot \langle a_i, z \rangle \cdot \langle b_i, z \rangle + \text{junk}
\end{align*}
\]

Still stuck?!?! What are we doing wrong?
Idea: Remember Hadamard PIOP

What does this remind you of?
\[ \langle \vec{L}(X)M, z \rangle = \sum_i L_i(X) \langle m_i, z \rangle \]

This is the interpolation of \( Mz \) over \( H \)!

So after queries we have \( \hat{z}_A(r), \hat{z}_B(r), \hat{z}_C(r) \)!

Q: What did we do with these in Hadamard PIOP?

A: Check \( \hat{z}_A(r) \cdot \hat{z}_B(r) - \hat{z}_C(r) = h(r) \cdot v_H(r) \)
Final Construction

\[ L(r) \cdot M \quad \forall M \]

\[ z = (x, w) \]

\[ h = \frac{\hat{z}_A \cdot \hat{z}_B - \hat{z}_C}{v_H} \]

\[ v_H(r) \cdot (1, r, \ldots, r^{n-1}) \]

Prover

\( (F, x, w) \)

Verifier

\( (F, x) \)

\[ \langle L(r)A, z \rangle \cdot \langle L(r)B, z \rangle - \langle L(r)C, z \rangle \]

\[ \overset{?}{=} \]

\[ \langle h, v_H(r) \cdot \vec{r} \rangle \]
Let’s analyze this: Completeness

\[(\sum_i L_i(X)\langle a_i, z\rangle) \cdot (\sum_j L_j(X)\langle b_j, z\rangle) - \sum_j L_j(X) \cdot \langle c_j, z\rangle) =  \hat{z}_A(X) \cdot \hat{z}_B(X) - \hat{z}_C(X) = h(X) \cdot v_H(X)\]
Let’s analyze this: Soundness

\[ \hat{z}_A(X) \cdot \hat{z}_B(X) - \hat{z}_C(X) \neq h(X) \cdot v_H(X), \]
then \( \hat{z}_A(r) \cdot \hat{z}_B(r) - \hat{z}_C(r) = h(r) \cdot v_H(r) \)
with negligible probability
Let’s analyze this: Efficiency

- Number of oracles: 2
- Number of queries: 4
- Prover work: $O(n \log n)$ (due to poly mul)
- Number of rounds: 1
- Verifier checks: $\sim 1$ multiplication
Compiling LIPs to SNARKs
Q: What is verifier computation?

Verifier computation involves a Prover and a Verifier:

- **Prover**
  - Input: $(F, x, w)$
  - Computation:
    \[ h := \frac{\hat{z}_A \cdot \hat{z}_B - \hat{z}_C}{v_H} \]
    
- **Verifier**
  - Input: $(F, x)$
  - Computation:
    \[ z = (x, w) \]
    
- **Query**
  - Computation:
    \[ h \]
    
- **Decision**
  - Computation:
    \[ b \]
    
- **Verification**
  - Computation:
    \[ \langle L(r)A, z \rangle \cdot \langle L(r)B, z \rangle - \langle L(r)C, z \rangle \]
    \[ \overset{?}{=} \]
    \[ \langle h, v_H(r) \cdot \bar{r} \rangle \]
A: $O(n)!$

\[
\overline{L(r)} \cdot M \quad \forall M
\]

**Prover**

$(F, x, w)$

\[
h := \frac{\hat{z}_A \cdot \hat{z}_B - \hat{z}_C}{v_H}
\]

$v_H(r) \cdot (1, r, \ldots, r^{n-1})$

**Verifier**

$(F, x)$

\[
\langle \overline{L(r)}A, z \rangle \cdot \langle \overline{L(r)}B, z \rangle - \langle \overline{L(r)}C, z \rangle \\
\overset{?}{=} \langle h, v_H(r) \cdot \vec{r} \rangle
\]

$b$
Can we do better?

Yes, via preprocessing!
Insight: all queries are independent of prover message

\[ h := \frac{\hat{z}_A \cdot \hat{z}_B - \hat{z}_C}{v_H} \]

• Problem: No soundness!
Idea: Encode in Exponent

Let $m_i$ be the $i$-th column of $M$.
Then $\hat{m}_i(X) := \langle L(X), m_i \rangle$ is its interpolation, and $L(X) \cdot M := (\hat{m}_1(X), \ldots, \hat{m}_n(X))$

- For each $M$, define $\text{pk}_M := (g^{\hat{m}_1(r)}, \ldots, g^{\hat{m}_n(r)})$
- Define $\text{pk}_h := (g^{v_H(r)}, g^{r \cdot v_H(r)}, \ldots, g^{r^{n-1} v_H(r)})$
Construction with encoded queries

\[ h := \frac{\hat{z}_A \cdot \hat{z}_B - \hat{z}_C}{v_H} \]

**Q:** How to perform check in exponent?
Q: How to perform check in exponent?

We have

\[ g_A := g^{\langle \overrightarrow{L(r)}A, z \rangle} \]
\[ g_B := g^{\langle \overrightarrow{L(r)}B, z \rangle} \]
\[ g_C := g^{\langle \overrightarrow{L(r)}C, z \rangle} \]
\[ g_h := g^{h(r)\nu_H(r)} \]

We need to check

\[ \langle \overrightarrow{L(r)}A, z \rangle \cdot \langle \overrightarrow{L(r)}B, z \rangle - \langle \overrightarrow{L(r)}C, z \rangle \overset{?}{=} h(r)\nu_H(r) \]
Decision via pairing

\[ h := \frac{\hat{z}_A \cdot \hat{z}_B - \hat{z}_C}{v_H} \]

Prover \((F, x, w)\)

\[ g_A, g_B, g_C, g_h \]

Verifier \((x)\)

Setup \((F)\)

\[ Q \]

QUERY

\[ b \]

DECISION

\[ e(g_A, g_B) - e(g_C, g) \]

\[ \equiv \]

\[ e(g_h, g) \]
Let’s analyze this: Soundness

Assuming GGM, malicious prover can only compute linear combinations of pk

So it must provide a linear response to encoded queries

Additionally, DL ensures that prover learns nothing about query.
Summary

• Proof size: 4 group elements
• Setup work: \( O(n \log n) \mathbb{F} + O(n) \mathbb{G} \)
• Prover work: \( O(n \log n) \mathbb{F} + O(n) \mathbb{G} \)
• Verifier work: 3 pairings

Unresolved questions:
• What about public input?
• What if prover uses different oracles for \( A, B, C \)?
• What if prover’s response is affine?