Theory and Practice of Succinct Zero Knowledge Proofs

Lecture 03:
SNARKs from Polynomial Interactive Oracle Proofs
Succinct Non-Interactive Arguments (SNARGs)

\[ O(\log(F)) \]

Prover
- \( \text{pk} \): proving key
- \( x \): public input
- \( w \): private witness

Verifier
- \( \text{vk} \): verifying key
- \( x \): public input

\[ I \text{ know } w \text{ s.t. } F(x, w) = 1 \]

Setup
- \( F \): function
- \( \text{pk} \): proving key
- \( \text{vk} \): verifying key

\[ \tilde{O}(F) \]

References:
- Mic94, Groth10, GGPR13, Groth16, ...
- GWC19, CHMMVW20, ...
SNARKs

- **Completeness**: \( \forall (F, x, w) \in \mathcal{R}, \Pr \left[ V(pk, x, \pi) = 1 : \pi \leftarrow P(pk, x, w) \right] = 1. \)

- **Knowledge Soundness**: \( \forall \) efficient \( \tilde{P} \), \( \exists \) extractor \( E \) s.t.

  \[
  \Pr \left[ V(pk, x, \pi) = 1 : \pi \leftarrow \tilde{P}(pk, x) \right] \approx 0
  \]

- **Zero Knowledge**: \( \exists \) simulator \( \text{Sim} \) s.t. \( \forall (F, x, w) \in \mathcal{R} \), and all \( \tilde{V} \),

  \[
  \Pr \left[ V(pk, x, \pi) : \pi \leftarrow \text{Sim}(pk, x) \right] = \Pr \left[ V(pk, x, \pi) : \pi \leftarrow P(pk, x, w) \right]
  \]

- **Succinctness**: \(| \pi | = O(\log |F|) \) and \( \text{Time}(\mathcal{V}) = O(\log |F|, |x|) \)
Constructing zkSNARKs
Blueprint

Functional Proof System

"Functional" Commitment

Compiler

zkSNARK
Which function to pick?

**Polynomial commitments:**
- $F_z(m)$: Interpret $m$ as univariate poly $f(X)$ in $\mathbb{F}[X]$ and evaluate at $z$.

**Multilinear commitments:**
- $F_{\vec{z}}(m)$: Interpret $m$ as multilinear poly $f(X)$ in $\mathbb{F}[\vec{X}]$ and evaluate at $\vec{z}$.

**Vector commitments:**
- $F_i(m)$: Interpret $m$ as vector $\vec{v}$ in $\mathbb{F}^n$ and return $\vec{v}_i$.

**Inner-product commitments:**
- $F_{\vec{q}}(m)$: Interpret $m$ as vector $\vec{v}$ in $\mathbb{F}^n$ and return $\langle \vec{v}, \vec{q} \rangle$.

Which to pick?
A: Polynomials!
Let’s pick polynomials

- ???
- Polynomial Commitment
- Compiler
- zkSNARK
Polynomial
Interactive
Oracle
Proofs
Polynomial IOPs \textsuperscript{[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 polys learns nothing about \(w\).
Majority of innovation is in PIOPs

Sonic: Zero-Knowledge SNARKs from Linear-Size Universal and Updatable Structured Reference Strings

Mary Mailler
mary.mailer.is@rud.ac.uk
University College London

Markulf Kohlweiss
mkelhweis@ed.ac.uk
University of Edinburgh

Sean Bowe
sean@tcash
Electric Coin Company

Sarah Meiklejohn
s.meiklejohn@ed.ac.uk
University College London

Sean Bowe
sean@tcash
Electric Coin Company

Sarah Meiklejohn
s.meiklejohn@ed.ac.uk
University College London

Sean Bowe
sean@tcash
Electric Coin Company

Sarah Meiklejohn
s.meiklejohn@ed.ac.uk
University College London


PlonK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge

Ariel Gabizon
Ariel Gabizon
Aztec
Aztec

Zachary J. Williamson
Oana Ciotobotaru

Spartan: Efficient and general-purpose zkSNARKs without trusted setup

Srinath Setty
Srinath Setty
Microsoft Research

HyperPlonk: Plonk with Linear-Time Prover and High-Degree Custom Gates

Binyi Chen
Benedikt Bünz
Benedikt Bünz

Dan Boneh
Zhenfei Zhang

Espresso Systems
Espresso Systems

Caulk: Lookup Arguments in Sublinear Time

Arantxa Zapico1, Vitalik Buterin2, Dmitry Khovratovich2, Mary Maller2, Anca Nitulescu2, and Mark Simkin2

1 Universitat Pompeu Fabra1
2 Ethereum Foundation2
3 Protocol Labs3

plonkup: A simplified polynomial protocol for lookup tables

Liam Fagen
Liam Fagen
Blockstream

Dario Fiore
Dario Fiore
IMEDEA software institute

Baloo: Nearly Optimal Lookup Arguments

Ariel Gabizon
Ariel Gabizon
Zeta Function Technologies

Ariel Gabizon
Ariel Gabizon
Zeta Function Technologies
Polynomial Commitments
Polynomial Commitments

Maximum degree $D$ → **SETUP** → Committer key $ck$ Verifier key $vk$

**SENDER**
1. $cm \leftarrow \text{COMMIT}(ck, p)$
2. $v \leftarrow p(z)$
3. $\pi \leftarrow \text{OPEN}(ck, cm, p, z)$

**RECEIVER**

- **Completeness**: Whenever $p(z) = v$, $R$ accepts.
- **Extractability**: Whenever $R$ accepts, $S$’s commitment $cm$ “contains” a polynomial $p$ of degree at most $D$.
- **Hiding**: $cm$ and $\pi$ reveal no information about $p$ other than $v$
Polynomial Commitments

Maximum degree $D$ → SETUP → Committer key $ck$
Verifier key $vk$

**S E N D E R**
1. $[cm] \leftarrow \text{COMMIT}(pk, [p], [d])$
2. $[v] \leftarrow [p](Q)$
3. $\pi \leftarrow \text{OPEN}(pk, [p], [d], Q)$

**R E C E I V E R**
$[cm] \rightarrow Q \rightarrow ([v], \pi) \rightarrow \text{CHECK}(vk, [cm], Q, [v], [d], \pi)$

For efficiency improvements, you need

- Batch commitment
- Batch opening
A selection of constructions

In the last 10 years, several constructions with different

- Cryptographic assumptions
- Prover and verifier efficiency and proof sizes
- Homomorphism and batching properties

<table>
<thead>
<tr>
<th></th>
<th>KZG10</th>
<th>PST13</th>
<th>IPA</th>
<th>Hyrax</th>
<th>Dory</th>
<th>BFS20</th>
</tr>
</thead>
<tbody>
<tr>
<td>crypto</td>
<td>Pairings</td>
<td>Pairings</td>
<td>DLog + RO</td>
<td>DLog + RO</td>
<td>Pairing + RO</td>
<td>GUO + RO</td>
</tr>
<tr>
<td># variables</td>
<td>1</td>
<td>$m$</td>
<td>1</td>
<td>$m$</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>setup type</td>
<td>Private</td>
<td>Private</td>
<td>Public</td>
<td>Public</td>
<td>Public</td>
<td>Public</td>
</tr>
<tr>
<td>commitment size</td>
<td>$O(1) G$</td>
<td>$O(1) G$</td>
<td>$O(1) G$</td>
<td>$O(2^{m/2}) G$</td>
<td>$O(1) G$</td>
<td>$O(1) G$</td>
</tr>
<tr>
<td>proof size</td>
<td>$O(1) G$</td>
<td>$O(m) G$</td>
<td>$O(\log d) G$</td>
<td>$O(2^{m/2}) G$</td>
<td>$O(\log d) G$</td>
<td>$O(\log d) G$</td>
</tr>
<tr>
<td>verifier time</td>
<td>$O(1) G$</td>
<td>$O(m) G$</td>
<td>$O(d) G$</td>
<td>$O(2^{m/2}) G$</td>
<td>$O(\log d) G$</td>
<td>$O(\log d) G$</td>
</tr>
</tbody>
</table>
PIOP + PC = SNARK
PIOPs + PC Schemes → SNARK

\[ \text{PIOPs + PC Schemes → SNARK} \]

\[ \text{SETUP}(1^\lambda, N) \]

- max degree \( D \)
- output \( (ck, vk) \)
- prover key \( pk = ck \)
- verifier key \( vk \)

- \( \text{PIOP}(N) \)
- \( \text{PC.SETUP}(D) \)

\[ \text{P}(pk, F, x, w) \]

- \( p_1 \) \( \xrightarrow{\text{PC.COMMIT}} \) cm_1
- \( p_t \) \( \xrightarrow{\text{PC.COMMIT}} \) cm_t
- \( Q \) \( \xrightarrow{\text{PC.OPEN}} (\pi, [v]) \)

\[ \text{V}(vk, F, x) \]

- \( r_1 \)
- \( r_t \)

\[ \text{DECISION} \]

\[ \text{PC.CHECK} \]

+ Fiat—Shamir to get non-interactivity
Properties

- **Completeness**: Follows from completeness of PC and AHP.

- **Proof of Knowledge**: Whenever V accepts but $C(x, w) = 0$, we can construct either an adversarial prover against PIOP, or an adversary that breaks extractability of PC.

- **Zero Knowledge**: Follows from hiding of PC and bounded-query ZK of AHP.

- **Verifier efficiency**: 
  $$T(\text{ARG.VERIFY}) = T(\text{PIOP.VERIFY}) + T(\text{PC.CHECK})$$
Verifier Complexity of PIOP-based SNARKs

\[ T(SNARK.V) = T(\text{CHECK}) + T(\text{PIOP.V}) \]

Can make this sublinear (eg: KZG)

**What about this?**

Prover

\[ (F, x, w) \]

\[ \mathbb{P} \]

Verifier

\[ (F, x) \]

\[ \mathbb{V} \]

PIOP Verifier has to **at least** read \((F, x)\)

- When size of \(F \ll\) size of computation (eg machine computations), \(\text{TIME}(\mathbb{V})\) **is sublinear**.
- When size of \(F = \) size of computation (eg circuit computations), \(\text{TIME}(\mathbb{V})\) **is linear!**
A simple PIOP
Background on polynomials

**Polynomial over** $\mathbb{F}$:
\[ p(X) = a_0 + a_1X + \ldots + a_dX^d \] where $a_i \in \mathbb{F}$ and $X$ takes values in $\mathbb{F}$.

**Polynomial Interpolation:**
Given a list $A = (a_0, \ldots, a_d)$, and a set $H \subseteq \mathbb{F}$, we can interpolate $A$ over $H$ to obtain $p(X)$ such that $p(h_i) = a_i$ where $h_i$ is the $i$-th element of $H$.

**Vanishing polynomial:**
The vanishing polynomial for $H \subseteq \mathbb{F}$ is $v_H(X)$ such that $v_H(h) = 0 \quad \forall \ h \in H$
Warmup: PIOP for Equality

• **Completeness:** If \( p_1 = p_2 \), then definitely \( p_1(r) = p_2(r) \).

• **Soundness:** If \( p_1 \neq p_2 \), then \( p_1(r) = p_2(r) \) \( \implies \) \( r \) is a root of 

\[
q := p_1 - p_2.
\]

But since \( r \) is random, this happens with probability \(
\frac{\text{deg}(q)}{|F|}
\)
Warmup: PIOP for Equality over Domain

**Completeness:** If $p_1 = p_2$, then definitely $p_1(r) = p_2(r)$.

**Soundness:** Define $q := p_1 - p_2$. Then $\forall h \in H, p_1(h) = p_2(h)$ if and only if $q = s \cdot v_H$. But we can check this via the previous PIOP.