History of Succinct Arguments

Nicholas Ward
zkSNARKs

zero knowledge
succinct
non-interactive
argument
of knowledge
Why?

Delegation of Computation
History of Zero Knowledge

[GMR85]: introduced ZKP, with its simulator-based definition, and gave an example

[GKR89]: gave ZKP for an NP-complete problem (3-colorability), and thus for all of NP
History of Non-Interactive Arguments

[FS87]: generically turn public-coin IP into non-interactive proof by generating verifier’s next queries from input and conversation so far
History of Non-Interactive Arguments

(secure assuming *random oracle*, but heuristically valid)
History of Succinct Arguments
PCPs
The PCP Theorem

$\text{NP} \subseteq \text{PCP}[O(\log n), O(1)]$

& easy to add ZK!
From PCP to Succinct Argument

Send entire PCP? Not succinct!

Verifier sends query locations? Easy for prover to cheat!
From PCP to Succinct Argument

[Kil93, Mic94]: Encode PCP over Merkle tree
From PCP to Succinct Argument
From PCP to Succinct Argument

[Kil93, Mic94]: Encode PCP over Merkle tree

Gives good asymptotics, but _bad_ practical efficiency
## Linear PCPs

<table>
<thead>
<tr>
<th>PCP</th>
<th>query</th>
<th>query</th>
<th>query</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Achieved by *moving to the exponent* of a group with hard discrete logs

Requires shared *structured reference string*, involving *trusted setup*

Introduced in [IKO07], made efficient in [GGPR13] using *pairings*
Linear PCPs

- Computation
- Algebraic Circuit
- R1CS
- QAP
- Linear PCP
- zkSNARK
IOPs

IP

PCP

P  V

P  V
IOPs
Linear IOPs

IOP where each PCP is linear

[GKR08] & protocols based off it
Polynomial IOPs

Special case of linear IOP:
PCP is coefficients
Query is of the form \[ 1, z, z^2, z^3, z^4, \ldots, z^n \]

STARK
DARK
PLONK
Marlin!
Polynomial IOPs

- Computation
- Algebraic Circuit
- R1CS
- Polynomial IOP
- *(using polynomial commitments)*
- zkSNARK
MARLIN:
Preprocessing zkSNARKs with Universal and Updatable Setup

Alessandro Chiesa  UC Berkeley
Yuncong Hu       UC Berkeley
Mary Maller      University College London
Pratyush Mishra UC Berkeley
Noah Vesely     University College London
Nicholas Ward   UC Berkeley

https://erint.iacr.org/2019/1047
untapped potential
Preprocessing zkSNARKs with circuit-specific setup

\[ C(x, w) = 1 \]

Setup for CSAT

\[ \text{SETUP}(\lambda, C) \rightarrow (pk_C, vk_C) \]

Prove

\[ \text{PROVE}(pk_C, x, w) \rightarrow \pi \]

Verify

\[ \text{VERIFY}(vk_C, x, \pi) \rightarrow b \]

Problem: new setup for every circuit; to be trustworthy, this requires a *global* MPC
Goal: universal setup

Universal Trusted Setup:
\[ \text{US}_{\text{ETUP}}(1^\lambda, \mathcal{N}) \rightarrow (\text{upk}, \text{uvk}) \]

Circuit-specific deterministic preprocessing:

\[ \text{CP}_{\text{ROCESS}}(\text{upk}, C_1) \downarrow (\text{cpk}_1, \text{cvk}_1) \]
\[ \text{CP}_{\text{ROCESS}}(\text{upk}, C_2) \downarrow (\text{cpk}_2, \text{cvk}_2) \]
\[ \text{CP}_{\text{ROCESS}}(\text{upk}, C_3) \downarrow (\text{cpk}_3, \text{cvk}_3) \]

\[ \text{P}_{\text{ROVE}}(\text{cpk}_2, x, w) \rightarrow \pi \]
\[ \text{V}_{\text{ERIFY}}(\text{cvk}_2, x, \pi) \rightarrow b \]
Goal: *updatable* setup

Initial Setup:
\[
\text{SETUP}(1^\lambda) \rightarrow (\text{srs}, \rho)
\]

Each update:
\[
\text{UPDATE}(1^\lambda, \text{srs}, (\rho_i)_{i=1,\ldots,n}) \rightarrow (\text{srs}', \rho')
\]

Verification:
\[
\text{VERIFY}(1^\lambda, \text{srs}, (\rho_i)_{i=1,\ldots,n}) \rightarrow b
\]
Contributions

1. Methodology
   - Preprocessing zkSNARK for R1CS

2. Efficient ingredients for an efficient SNARK
   - Universal SRS
   - Preprocessing zkSNARK
   - AHP for R1CS
   - Extended KZG10 Scheme

3. Rust implementation
   - https://github.com/scipr-lab/marlin
Evaluation over BLS12–381

Proof size:
- Marlin: 1296B
- Groth16: 192B

Prover time:
- ~10x

Verifier time:
- ~4x

Concurrent Work:
- Marlin: good for R1CS
- PLONK: good for CSAT
This Talk

1. Methodology

A. Provides a clean and straightforward way to construct preprocessing SNARKs
B. Shows that the key to achieving preprocessing is holography
Compiler

Algebraic Holographic Proof

Extractable Polynomial Commitment

Preprocessing zkSNARK with Universal SRS
A set of triples \((i, x, w)\) satisfying a prescribed condition

Example: Arithmetic Circuit Satisfaction (CSAT):
\[
\{(C, x, w) \mid C(x, w) = 1\}
\]

Example: Rank-1 Constraint System (R1CS):
\[
\{((A, B, C), x, w) \mid AZ \cdot BZ = CZ\}
\]

where \(z = \begin{bmatrix} x \\ w \end{bmatrix}\)
Algebraic Holographic Proofs

- **Completeness**: Whenever \((i, x, w) \in R\), \(V\) accepts.

- **Proof of Knowledge**: Whenever \(V\) accepts, \(P\) “knows” \(w\) such that \((i, x, w) \in R\).

- **Bounded-query ZK**: Whenever \((i, x, w) \in R\), a verifier that makes up to \(b\) queries to polys learns nothing about \(w\).

What about verifier efficiency?
**Problem:** Verifier is linear in circuit!

- When size of circuit $<<$ size of computation (like in machine computations), this is OK.
- When size of circuit $=$ size of computation (like in CSAT/R1CS), this is bad!

\[ (C, x, w) \]
Algebraic Holographic Proofs

Verifier efficiency: $|x| + T(\text{Interaction}) + T(\text{Query}) + T(\text{Decision})$
Compiler

- Algebraic Holographic Proof
- Extractable Polynomial Commitment

Preprocessing zkSNARK with Universal SRS
Polynomial Commitments

**Maximum degree** $D$

**Send**
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**: If $R$ makes up to $b$ queries, it learns nothing about $p$. 

**Setup**

**Committer key** $ck$

**Verifier key** $vk$

$\text{CHECK}(vk, cm, z, v, \pi)$
Polynomial Commitments

Maximum degree $D$

Our compiler needs more
- Batch commitment
- Batch opening
- Multiple rounds
- Per-poly degree bounds

**Sender**
1. $[cm] \leftarrow \text{Commit}(pk, [p], [d])$
2. $[v] \leftarrow [p](Q)$
3. $\pi \leftarrow \text{Open}(pk, [p], [d], Q)$

**Receiver**

**Setup**

**Committer key** $ck$

**Verifier key** $vk$

**Check**($vk$, $[cm]$, $Q$, $[v]$, $[d]$, $\pi$)
Idea underlying compiler:

Holography $\Rightarrow$ Preprocessing
Preprocessing zkSNARKs

\[ \text{ARG.\textsc{Setup}}(1^\lambda, N) \rightarrow (\text{upk}, \text{uvk}) \]
\[ \text{ARG.\textsc{Index}}(\text{upk}, i) \rightarrow (\text{ipk}, \text{ivk}) \]
\[ \text{ARG.\textsc{Prove}}(\text{ipk}, x, w) \rightarrow \pi \]
\[ \text{ARG.\textsc{Verify}}(\text{ivk}, x, \pi) \rightarrow b \in \{0, 1\} \]

- **Completeness**: Whenever \((i, x, w) \in R\), \(V\) accepts.
- **Proof of Knowledge**: Whenever \(V\) accepts, \(P\) “knows” \(w\) such that \((i, x, w) \in R\).
- **Zero Knowledge**: Whenever \((i, x, w) \in R\), \(V\) learns nothing about \(w\).
- **Verifier efficiency**: \(T(V) = O(\log(|i|) + |x|)\)
Universal Setup

ARG.S SETUP(1^λ, N)

1. Maximum degree $D$ ← AHP(ℕ)
2. Committer key $ck$
   Verifier key $vk$ ← PC.S SETUP(D)
3. Output
   Universal prover key $upk = (ck, vk)$
   Universal verifier key $uvk = vk$
Index-specific Setup

ARG.INDEXER(upk, i)

1. Index polys $I$ ← AHP.INDEXER(i)

2. Index comms. [cm] ← PC.COMMIT(upk, $I$)

3. Output

   Index verifier key $ivk = (upk.uvk, [cm])$
   Index prover key $ipk = (ivk, upk, I)$
Prove and Verify

ARG.P

AHP.PROVER

$[v] \leftarrow [p](Q)$

PC.OPEN(upk, [cm], [p], [d], Q)

ARG.V

AHP.VERIFIER

+ Fiat-Shamir to get non-interactivity
Properties

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

- **Proof of Knowledge**: Whenever ARG.VERIFY accepts but 
  \((i, x, w) \notin R\), we can construct either an adversarial prover against 
  AHP, or an adversary that breaks extractability of PC.

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

- **Verifier efficiency**: 
  \(T(\text{ARG.VERIFY}) = T(\text{AHP.VERIFY}) + T(\text{PC.CHECK})\)
Conclusion

In the talk:

*algebraic holographic proof* +

*extractable polynomial commitment scheme*

into a

*universal preprocessing zkSNARK*

In the paper:

**Efficient AHP for R1CS:**

- Protocol to evaluate low-degree extension for arbitrary R1CS matrices

**Extending KZG10 to achieve:**

- Extractability across multiple rounds
- Batch commitment and opening
- Individual degree bounds
Paper:  https://eprint.iacr.org/2019/1047
Code:   https://github.com/scipr-lab/marlin
[KZG10] Polynomial Commitments
Polynomial Commitments: Definition

PC.Setup(\(1^\lambda\), degree bound \(D\)) \rightarrow (committer key \(ck\), receiver key \(rk\))

PC.Commit(\(ck\), polynomial \(p\)) \rightarrow commitment \(c\)

PC.Open(\(ck\), \(p\), eval point \(z\)) \rightarrow proof \(\pi\)

PC.Check(\(rk\), \(c\), \(z\), claimed value \(v\), \(\pi\)) \rightarrow bit \(b\)
Polynomial Commitments: Security

Completeness: if \( v = p(z) \), then PC.Check outputs 1

Extractability: anyone who produces a commitment \( c \) that cause PC.Check to accept “knows” a corresponding poly \( p \)

Succinctness: \( c \) and \( \pi \) sizes, PC.Check time independent of \( D \)

Hiding: commitment reveals no information about polynomial
What Are Polynomial Commitments?

A polynomial commitment is a mechanism that allows a computation to be committed to at the outset, with the possibility of proving later which value was committed to. The prover encodes a polynomial $f$ and commits to it. The verifier queries an oracle to obtain an evaluation point $z$ and later proves that $f(z)$ is the evaluation of the polynomial at that point.
# A Wider View: Oracles & Primitives

<table>
<thead>
<tr>
<th>Oracle Type</th>
<th>Cryptographic Primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Set</td>
<td>Accumulator</td>
</tr>
<tr>
<td>Vector</td>
<td>Vector commitment</td>
</tr>
<tr>
<td>Low-degree</td>
<td>Low-degree test</td>
</tr>
<tr>
<td>Polynomial</td>
<td>Polynomial commitment</td>
</tr>
<tr>
<td>Inner product</td>
<td>Inner product argument</td>
</tr>
</tbody>
</table>

More expressive → More expensive
Kate et al. Polynomial Commitments

Setup(\lambda, D):

- computes groups \( G, G_T \) of prime order \( p \) with pairing \( e \)
- chooses generator \( g \in G \), random \( \alpha \) from \( \{1, ..., p-1\} \)
- outputs \( pk = rk = (g, \alpha g, \alpha^2 g, ..., \alpha^t g) \)
Kate et al. Polynomial Commitments

Commit(ck, p):

- outputs $c = p(\alpha)g$, pulling monomials from ck
Kate et al. Polynomial Commitments

Open(ck, p, z):

- computes witness poly $\phi(x) := (p(x) - p(z))/(x - z)$
- outputs proof $\pi = \phi(\alpha)g$

_witness poly because it shows the value for $p(z)$ is correct_
Kate et al. Polynomial Commitments

Check($rk, c, z, v, \pi$):

- checks whether

$$e(c, g) = e(\pi, (\alpha - z)g) \cdot e(g, g)^v$$

$$e(c, g) = e(p(\alpha)g, g) = e(g, g)^{p(\alpha)} = e(g, g)\cdot \phi(\alpha)(\alpha-z)+p(z)$$

$$= e(\phi(\alpha)g, (\alpha-z)g) \cdot e(g, g)^{p(z)}$$

$$= e(\pi, (\alpha-z)g) \cdot e(g, g)^v \quad \text{if} \quad c = p(\alpha)g, \quad v = p(z), \quad \pi = \phi(\alpha)g$$