recursive proofs:
applications and affordances

Ying Tong
Core Engineer, Electric Coin Company

Nalin
Person, 0xPARC
what are recursive proofs?

Verify SNARK proof $N$ and Step $N+1$ (instead of verifying all $N$ steps again)
unlocks from recursion

ZK-SNARKs

Succinctness

Zero-knowledge

Recursion?
unlocks from recursion
Section 1

compression
(supercharging succinctness)
compression: supercharging succinctness

for a low verifier cost, a prover can show:

"i know \( \forall \) pieces of knowledge"
compression: supercharging succinctness

for a low verifier cost, a prover can show

"i know \( N \) pieces of knowledge"

"i know one piece of knowledge"

"i know a proof of \( N-1 \) pieces of knowledge"
compression: supercharging succinctness

for a low verifier cost, a prover can show

"i know $N$ pieces of knowledge"

→ "i know one piece of knowledge"

→ "i know a proof of $N-1$ pieces of knowledge"

→ "i know one piece of knowledge"

→ "i know a proof of $N-2$ pieces of knowledge"
compression: supercharging succinctness

for a low verifier cost, a prover can show

"i know $N$ pieces of knowledge"

\[ \rightarrow \text{"i know one piece of knowledge"} \]

\[ \rightarrow \text{"i know a proof of } N-1 \text{ pieces of knowledge"} \]

\[ \rightarrow \text{"i know one piece of knowledge"} \]

\[ \rightarrow \text{"i know a proof of } N-2 \text{ pieces of knowledge"} \]

\[ \rightarrow \text{"i know one piece of knowledge"} \]

\[ \rightarrow \text{"i know a proof of } N-3 \text{ pieces of knowledge"} \]

\[ \rightarrow \text{"i know one piece of knowledge"} \]

\[ \rightarrow \text{"i know a proof of } N-4 \text{ pieces of knowledge"} \]

\[ \rightarrow \text{"i know one piece of knowledge"} \]

\[ \rightarrow \text{"i know a proof of } N-5 \text{ pieces of knowledge"} \]

\[ \rightarrow \ldots \]
prover / verifier complexity tradeoffs 🍰

<table>
<thead>
<tr>
<th></th>
<th>fast prover</th>
<th>small proof / fast verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;wide&quot; proof</td>
<td>✔</td>
<td>❌</td>
</tr>
</tbody>
</table>

- wide circuit
  - fast prover
  - large proof
  - $\pi_{\text{wide}}$
prover / verifier complexity tradeoffs 🍰

<table>
<thead>
<tr>
<th></th>
<th>fast prover</th>
<th>small proof / fast verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;wide&quot; proof</td>
<td>✔</td>
<td>❌</td>
</tr>
<tr>
<td>&quot;narrow&quot; proof</td>
<td>❌</td>
<td>✔</td>
</tr>
</tbody>
</table>

Wide circuit $\pi_{\text{wide}}$ to fast prover $\pi_{\text{wide}}$

Narrow circuit $\pi_{\text{narrow}}$ to slow prover tiny proof $\pi_{\text{narrow}}$
# Prover / Verifier Complexity Tradeoffs

<table>
<thead>
<tr>
<th></th>
<th>Fast Prover</th>
<th>Small Proof / Fast Verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;Wide&quot; Proof</td>
<td>✔</td>
<td>❌</td>
</tr>
<tr>
<td>&quot;Narrow&quot; Proof</td>
<td>❌</td>
<td>✔</td>
</tr>
</tbody>
</table>

---

![Diagram showing prover and verifier complexity tradeoffs](image-url)
### Interoperability between Proof Systems

<table>
<thead>
<tr>
<th></th>
<th>Fast Prover</th>
<th>Small Proof / Fast Verifier</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>STARK</strong></td>
<td>✔</td>
<td>❌</td>
</tr>
<tr>
<td><strong>Groth16</strong></td>
<td>❌</td>
<td>✔</td>
</tr>
</tbody>
</table>

#### Diagram

![Diagram showing interoperability between proof systems](image)

- **STARK Prover Circuit**
- **Fast Prover**
- **π_{STARK}**
- **Large Proof**
- **STARK Verifier in Groth16 Prover Circuit**
- **π_{Groth16}**
- **Slow Prover**
- **Tiny Proof**
compression: examples

a rollup of...

Signatures

light client proofs

txes
Section 2

composability
(taking zk a step further)
composability

typically, zk proofs are thought of in the context:

"a prover shows knowledge to a verifier, without revealing the underlying fact."

recursive zk proofs, in fact, unlock a stronger property:

"a prover shows knowledge to a verifier, without fully knowing the underlying facts themselves"
ETHdos: Erdős numbers on social graphs

vitalik.eth

?????.eth

Friendships authenticated by signature

?????.eth

adhyyan.eth

nibnalin.eth

I am 4 degrees away from vitalik.eth, but I do not know my path to him
composability: incomplete information games

- telephone/chinese whispers
- mafia
- private state channels
implementations
schemes for recursive proof composition

- Groth16
- FRI
- Inner Product Argument (IPA)
- Nova
schemes for recursive proof composition

needs:
- full recursion
- succinct verifier
- Groth16
- FRI

relax requirements on proof system
IVC from SNARKs with sublinear verification

\[ z_{i+1} = F(z_i; w_i) \]
IVC from SNARKs with sublinear verification

\[ z_{i+1} = F(z_i; w_i) \]

\[ V(\pi_i) = 1 \]
IVC from SNARKs with sublinear verification

\[ z_{i+1} = F(z_i; w_i) \]
\[ V(\pi_i) = 1 \]

\[ z_{i+2} = F(z_{i+1}; w_{i+1}) \]
\[ V(\pi_{i+1}) = 1 \]
IVC from SNARKs with sublinear verification

\[ z_i = F(z_{i-1}; w_i) \]
\[ V(\pi_i) = 1 \]

\[ z_{i+1} = F(z_i; w_i) \]
\[ V(\pi_{i+1}) = 1 \]

\[ z_{i+2} = F(z_{i+1}; w_{i+1}) \]
\[ V(\pi_{i+2}) = 1 \]
schemes for recursive proof composition

- full recursion
- atomic accumulation

needs:
- succinct verifier
- succinct accumulator

- Groth16
- FRI
- Inner Product Argument (IPA)
IVC from atomic accumulation
IVC from atomic accumulation
IVC from atomic accumulation
IVC from atomic accumulation
IVC from atomic accumulation
IVC from atomic accumulation
schemes for recursive proof composition

- full recursion
- atomic accumulation
- split accumulation

needs:
- succinct verifier
- succinct accumulator
- succinct public accumulator

relax requirements on proof system

- Groth16
- FRI
- Inner Product Argument (IPA)
- Nova
IVC from split accumulation [BCLMS20]
future of proving systems

- modular design $\rightarrow$ customisable proving stack
- recursion $\rightarrow$ composition across proof systems
modular proof systems: **Halo 2**

- **arithmetisation**
  - encode values in the Lagrange basis; constraints as polynomial identities

- **information-theoretic PIOP**
  - vanishing argument; multipoint opening argument

- **cryptographic compiler**
  - inner product argument; Fiat-Shamir transform

- "post-processing"
  - atomic accumulation
modular proof systems

arithmetisation

information-theoretic proofs

cryptographic compiler

"post-processing"

turns a relation into a constraint system involving native operations over a finite field

provides soundness and zk guarantees even when prover and verifier are computationally unbounded

transforms proof system into concrete protocol involving direct interaction between prover and verifier

compositional schemes for the protocol (e.g. distributed proving, streaming prover, aggregation, accumulation, recursion)
types of proof composition

1. information-theoretic compilers
e.g. "MPC-in-the-head" [IKOS07]
types of proof composition

2. composing cryptographic compilers
   e.g. STARK verifier in Groth16 prover

- FRI (Fast Reed-Solomon IOPP)
- hash commitment
- R1CS → QAP
- bi-linear pairing
types of proof composition

3. tailor-made cryptographic compilers
e.g. GKR verifier in Groth16 prover [BSB22]
types of proof composition

3. tailor-made cryptographic compilers
e.g. GKR verifier in Groth16 prover [BSB22]
future of proving systems

- modular design → customisable proving stack
- recursion → composition across proof systems

can we systematise the composition of proof protocols?
future of proving systems

- modular design → customisable proving stack
- recursion → composition across proof systems

can we systematise the composition of proof protocols?

- better benchmarks for primitives (e.g. hashes, bigint, signatures, range proofs, …)
- standardised criteria for comparing different compositions
- auditing / formal verification when composing proof systems
recursion, aggregation, composition

task force
Thank you!

Ying Tong
Core Engineer, Electric Coin Company
yingtong@z.cash
@therealyingtong

Nalin
Person, 0xPARC
devcon@nibnalin.me
@nibnalin