Doubly-efficient zkSNARKs without trusted setup

Riad S. Wahby*, Ioanna Tzialla◦, abhi shelat†, Justin Thaler‡, and Michael Walfish◦

*Stanford University
◦New York University
†Northeastern University
‡Georgetown University

May 23rd, 2018
zkSNARK

Argument A “proof”...
Argument A “proof”...
of knowledge ... that you know a secret, and...
zkSNARK

Argument A “proof” . . .

of knowledge . . . that you know a secret, and . . .

Zero knowledge . . . it doesn’t reveal the secret.
zkSNARK

**Argument** A “proof”...

**of knowledge** ...that you know a secret, and...

**Zero knowledge** ...it doesn’t reveal the secret.

**Succinct** It’s short...
zkSNARK

Argument A “proof”...

of knowledge ... that you know a secret, and...

Zero knowledge ... it doesn’t reveal the secret.

Succinct It’s short...

Non-interactive ... and it can be written down...
zkSNARK

Argument A “proof”...

of knowledge ... that you know a secret, and...

Zero knowledge ... it doesn’t reveal the secret.

Succinct It’s short...

Non-interactive ... and it can be written down...

(Publicly verifiable) ... so that anyone can check it.
zkSNARKs: Costs and desiderata

Proof size
zkSNARKs: Costs and desiderata

Proof size

Prover (P) time

Verifier (V) time

Cryptographic assumptions

Trusted setup?
zkSNARKs: Costs and desiderata

Proof size

Prover ($\mathcal{P}$) time

Verifier ($\mathcal{V}$) time
zkSNARKs: Costs and desiderata

Proof size

Prover ($\mathcal{P}$) time

Verifier ($\mathcal{V}$) time

Cryptographic assumptions
zkSNARKs: Costs and desiderata

Proof size

Prover ($P$) time

Verifier ($V$) time

Cryptographic assumptions

Trusted setup?
Our contributions

We design and implement Hyrax, a zkSNARK for “parallel” arithmetic circuit satisfiability:
for V’s input $x$, $\exists w : C(x, w) = 1$ (and $\mathcal{P}$ knows $w$)

Proof size is sub-linear in $|C|$ and $|w|
Prover time is linear in $|C|
Verifier time is sublinear in $|C|$ and $|w|

Good constants: concrete costs are low
Cryptographic assumptions: discrete log
No trusted setup
Hyrax is one useful point in a large tradeoff space
Our contributions

We design and implement *Hyrax*, a zkSNARK for “parallel” arithmetic circuit satisfiability:
for $V$’s input $x$, $\exists w : C(x, w) = 1$ (and $P$ knows $w$)

- Proof size is sub-linear in $|C|$ and $|w|$.
- Prover time is linear in $|C|$.
- Verifier time is sublinear in $|C|$ and $|w|$.
Our contributions

- We design and implement *Hyrax*, a zkSNARK for “parallel” arithmetic circuit satisfiability:
  for \( V \)'s input \( x \), \( \exists w : C(x, w) = 1 \) (and \( P \) knows \( w \))

Proof size is sub-linear in \(|C|\) and \(|w|\)

Prover time is linear in \(|C|\)

Verifier time is sublinear in \(|C|\) and \(|w|\)

Good constants: concrete costs are low
Our contributions

→ We design and implement Hyrax, a zkSNARK for “parallel” arithmetic circuit satisfiability:

for V’s input $x$, $\exists w : C(x, w) = 1$ (and $P$ knows $w$)

Proof size is sub-linear in $|C|$ and $|w|$

Prover time is linear in $|C|$

Verifier time is sublinear in $|C|$ and $|w|$

Good constants: concrete costs are low

Cryptographic assumptions: discrete log

No trusted setup
Our contributions

- We design and implement *Hyrax*, a zkSNARK for “parallel” arithmetic circuit satisfiability:
  
  for $\forall$’s input $x$, $\exists w : C(x, w) = 1$ (and $P$ knows $w$)

- We evaluate *Hyrax* and five other ZK systems.

We find that:

- Hyrax’s proofs are small: to get smaller, you have to pay more computation.
- Hyrax is fast: to get faster, you have to accept bigger proofs.
- Hyrax is one useful point in a large tradeoff space.
Our contributions

→ We design and implement Hyrax, a zkSNARK for “parallel” arithmetic circuit satisfiability:
for V’s input $x$, $\exists w : C(x, w) = 1$ (and $P$ knows $w$)

→ We evaluate Hyrax and five other ZK systems.
We find that:

Hyrax’s proofs are small:
to get smaller, you have to pay more computation.
Our contributions

→ We design and implement *Hyrax*, a zkSNARK for “parallel” arithmetic circuit satisfiability:
  for \( \forall \)’s input \( x \), \( \exists w : C(x, w) = 1 \) (and \( P \) knows \( w \))

→ We evaluate Hyrax and five other ZK systems.

We find that:

Hyrax’s proofs are **small**:
  to get smaller, you have to pay more computation.

Hyrax is **fast**:
  to get faster, you have to accept bigger proofs.
Our contributions

→ We design and implement *Hyrax*, a zkSNARK for “parallel” arithmetic circuit satisfiability:
   for \( V \)'s input \( x \), \( \exists w : C(x, w) = 1 \) (and \( P \) knows \( w \))

→ We evaluate Hyrax and five other ZK systems.

We find that:

- Hyrax’s proofs are **small**: to get smaller, you have to pay more computation.
- Hyrax is **fast**: to get faster, you have to accept bigger proofs.

Hyrax is one useful point in a large tradeoff space
Roadmap

1. General-purpose ZK proof systems

2. Hyrax at a high level

3. Evaluation
General-purpose ZK proof systems for NP

On input $x$, $\mathcal{P}$ convinces $\mathcal{V}$ that $\Phi(x, w) = 1$ (for a witness $w$ that $\mathcal{P}$ knows)
General-purpose ZK proof systems for NP

On input $x$, $P$ convinces $V$ that $\Phi(x, w) = 1$ (for a witness $w$ that $P$ knows)
General-purpose ZK proof systems for NP

On input $x$, $P$ convinces $V$ that $\Phi(x, w) = 1$  
(for a witness $w$ that $P$ knows)
General-purpose ZK proof systems for NP

On input $x$, $\mathcal{P}$ convinces $\mathcal{V}$ that $\Phi(x, w) = 1$ (for a witness $w$ that $\mathcal{P}$ knows)

- $\Phi$: witness checking computation
- arithmetic circuit $\mathcal{C}$
- ZK proof machinery
- $\mathcal{V}$ computation
- $\mathcal{P}$ computation

generalized boolean circuit over $\mathbb{F}_p$

$\land \rightarrow \times \quad \lor \rightarrow +$
General-purpose ZK proof systems for NP

On input $x$, $P$ convinces $V$ that $\Phi(x, w) = 1$ (for a witness $w$ that $P$ knows)

Diagram:

Front-end:
- $\Phi$: witness checking computation
- arithmetic circuit $C$

Back-end:
- ZK proof machinery
- $V$ computation
- $P$ computation
General-purpose ZK proof systems for NP

On input $x$, $P$ convinces $V$ that $\Phi(x, w) = 1$ (for a witness $w$ that $P$ knows)

```
Φ: witness checking computation
⇒ arithmetic circuit $C$
ZK proof machinery
⇒ valid proof
V computation
P computation
```

front-end
arithmetic circuit $C$ satisfies $w$ is correct

back-end
valid proof satisfies arithmetic circuit $C$ is satisfied
Existing systems use a wide range of proof machinery

**Linear PCPs** [IKO07, Gro09, Gro10, BG12, Lip12, BCIOP13, GGPR13, …]
- Pinocchio [PGHR13], libsnark [BCTV14]

<table>
<thead>
<tr>
<th>Short Proofs</th>
<th>Fast $\mathcal{P}$</th>
<th>Fast $\mathcal{V}$</th>
<th>Trusted setup?</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>libsnark</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
Existing systems use a wide range of proof machinery

**Linear PCPs** [IKO07, Gro09, Gro10, BG12, Lip12, BCIOP13, GGPR13, ...]
- Pinocchio [PGHR13], libsnark [BCTV14]
- [BCCGP16], Bulletproofs [BBBPWM18]

<table>
<thead>
<tr>
<th></th>
<th>Short Proofs</th>
<th>Fast $\mathcal{P}$</th>
<th>Fast $\mathcal{V}$</th>
<th>Trusted setup?</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>libsnark</td>
<td>✔</td>
<td>×</td>
<td>✔</td>
<td>×</td>
<td>Knowledge of exponent</td>
</tr>
<tr>
<td>Bulletproofs</td>
<td>✔</td>
<td>×</td>
<td>×</td>
<td>✔</td>
<td>discrete log</td>
</tr>
</tbody>
</table>
Existing systems use a wide range of proof machinery

**Linear PCPs** [IKO07, Gro09, Gro10, BG12, Lip12, BCIOP13, GGPR13, ...]
- Pinocchio [PGHR13], libsnark [BCTV14]
- [BCCGP16], Bulletproofs [BBBPWM18]

**Multiparty computation–in-the-head** [IKOS07]
- ZKBoo [GMO16], ZKB++ [CDGORRSZ17]

<table>
<thead>
<tr>
<th></th>
<th>Short Proofs</th>
<th>Fast (\mathcal{P})</th>
<th>Fast (\mathcal{V})</th>
<th>Trusted setup?</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>libsnark</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>Knowledge of exponent</td>
</tr>
<tr>
<td>Bulletproofs</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>discrete log</td>
</tr>
<tr>
<td>ZKB++</td>
<td>✗</td>
<td>✓</td>
<td>✗ (ish)</td>
<td>✓</td>
<td>collision-resistant hashes</td>
</tr>
</tbody>
</table>
Existing systems use a wide range of proof machinery

**Linear PCPs** [IKO07, Gro09, Gro10, BG12, Lip12, BCIOP13, GGPR13, ...]
- Pinocchio [PGHR13], libsnark [BCTV14]
- [BCCGP16], Bulletproofs [BBBPWM18]

**Multiparty computation—in-the-head** [IKOS07]
- ZKBoo [GMO16], ZKB++ [CDGORRSZ17]
- Ligero [AHIV17]

<table>
<thead>
<tr>
<th></th>
<th>Short Proofs</th>
<th>Fast $P$</th>
<th>Fast $V$</th>
<th>Trusted setup?</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>libsnark</td>
<td>✓</td>
<td>$\times$</td>
<td>✓</td>
<td>$\times$</td>
<td>Knowledge of exponent</td>
</tr>
<tr>
<td>Bulletproofs</td>
<td>✓</td>
<td>$\times$</td>
<td>$\times$</td>
<td>✓</td>
<td>discrete log</td>
</tr>
<tr>
<td>ZKB++</td>
<td>$\times$</td>
<td>✓</td>
<td>$\times$ (ish)</td>
<td>✓</td>
<td>collision-resistant hashes</td>
</tr>
<tr>
<td>Ligero</td>
<td>✓ (ish)</td>
<td>✓</td>
<td>✓ (ish)</td>
<td>✓</td>
<td>collision-resistant hashes</td>
</tr>
</tbody>
</table>
Existing systems use a wide range of proof machinery

**Linear PCPs** [IKO07, Gro09, Gro10, BG12, Lip12, BCIOP13, GGPR13, ... ]
- Pinocchio [PGHR13], libsnark [BCTV14]
- [BCCGP16], Bulletproofs [BBBPWM18]

**Multiparty computation–in-the-head** [IKOS07]
- ZKBoo [GMO16], ZKB++ [CDGORRSZ17]
- Ligero [AHIV17]

**Short PCPs** [Kil94, Mic00, BS08, BCN16, RRR16, BBC+17, BBHR17, ... ]
- libSTARK [BBHR18]

<table>
<thead>
<tr>
<th>Pr</th>
<th>Pr</th>
<th>Pr</th>
<th>Assumption</th>
</tr>
</thead>
<tbody>
<tr>
<td>libsnark</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
<tr>
<td>Bulletproofs</td>
<td>✓</td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>ZKB++</td>
<td>X</td>
<td>✓</td>
<td>X(ish)</td>
</tr>
<tr>
<td>Ligero</td>
<td>✓ (ish)</td>
<td>✓</td>
<td>✓ (ish)</td>
</tr>
<tr>
<td>libSTARK</td>
<td>✓</td>
<td>X</td>
<td>✓</td>
</tr>
</tbody>
</table>
Roadmap

1. General-purpose ZK proof systems
2. Hyrax at a high level
3. Evaluation
Hyrax: a ZK argument from Interactive Proofs (IPs)

Hyrax builds on the interactive proofs of GKR/CMT

[Bab85, GMR89, GKR08, CMT12, Tha13, WJBsTWW17, ZGKPP17, ...]
Hyrax: a ZK argument from Interactive Proofs (IPs)

Hyrax builds on the interactive proofs of GKR/CMT [Bab85, GMR89, GKR08, CMT12, Tha13, WJBsTWW17, ZGKPP17, ...]

We compile Hyrax’s IP to a ZK argument using the techniques of [BGGHKMR88] and [CD98]...
Hyrax: a ZK argument from Interactive Proofs (IPs)

Hyrax builds on the interactive proofs of GKR/CMT [Bab85,GMR89,GKR08,CMT12,Tha13,WJBsTWW17,ZGKPP17,…]

We compile Hyrax’s IP to a ZK argument using the techniques of [BGGHKMR88] and [CD98]…

…plus refinements that result in multiple orders of magnitude savings in V time and proof size.
Hyrax: a ZK argument from Interactive Proofs (IPs)

Hyrax builds on the interactive proofs of GKR/CMT [Bab85,GMR89,GKR08,CMT12,Tha13,WJBsTWW17,ZGKPP17,…]

We compile Hyrax’s IP to a ZK argument using the techniques of [BGGHKMR88] and [CD98]…

… plus refinements that result in multiple orders of magnitude savings in \( \mathcal{V} \) time and proof size.

High-level idea: Replace each of \( \mathcal{P} \)’s messages in the IP with a commitment to the message; \( \mathcal{V} \) runs checks “under the commitments.”
Cryptographic commitments

 Sender computes $C \leftarrow \text{Com}(m)$, sends to receiver. Later, sender can open $C$, convincing the receiver that $m$ was the committed message.
Cryptographic commitments

*Sender* computes $C \leftarrow \text{Com}(m)$, sends to *receiver*. Later, sender can *open* $C$, convincing the receiver that $m$ was the committed message.

In general, $\text{Com}(m)$ has two important properties:

**Hiding:** $C$ reveals nothing about $m$.

**Binding:** Cannot produce $m' \neq m$ s.t. $C = \text{Com}(m')$
Cryptographic commitments (with a linear homomorphism)

*Sender* computes $C \leftarrow \text{Com}(m)$, sends to *receiver*. Later, sender can open $C$, convincing the receiver that $m$ was the committed message.

In general, $\text{Com}(m)$ has two important properties:

**Hiding:** $C$ reveals nothing about $m$.

**Binding:** Cannot produce $m' \neq m$ s.t. $C = \text{Com}(m')$

We also require a *linear homomorphism*, $\odot$:

given $C_0 \leftarrow \text{Com}(m_0)$, $C_1 \leftarrow \text{Com}(m_1)$, we have

$$C_0 \odot C_1 \triangleq \text{Com}(m_0 + m_1)$$

$$C_1^k \triangleq C_1 \odot \cdots \odot C_1 = \text{Com}(k \cdot m_1)$$

The Pedersen commitment has this property.
Witness checker must be expressed as a \textit{layered} AC.
GKR08: IP for arithmetic circuit evaluation (non-ZK)

1. $V$ sends inputs

2. $P$ evaluates, returns output $y$

3. $V$ constructs a polynomial relating $y$ to the last layer's input wires

4. $V$ engages $P$ in a sum-check, gets a claim about the second-last layer

5. $V$ iterates, gets a claim about the inputs, which it can check

$V$, $P$: thinking...

$y$: thinking...

... sum-check

[LFKN90]

more sum-checks
GKR08: IP for arithmetic circuit evaluation (non-ZK)

1. $V$ sends inputs
2. $P$ evaluates
3. $V$ constructs polynomial relating $y$ to last layer's input wires
4. $V$ engages $P$ in a sum-check, gets claim about second-last layer
5. $V$ iterates, gets claim about inputs, which it can check

$V$ thinking...
$y$ thinking...
... sum-check

[LFKN90]
more sum-checks
1. $V$ sends inputs
2. $P$ evaluates

GKR08: IP for arithmetic circuit evaluation (non-ZK)
1. $V$ sends inputs
2. $P$ evaluates

$V$ constructs polynomial relating $y$ to last layer's input wires
$V$ engages $P$ in a sum-check, gets claim about second-last layer
$V$ iterates, gets claim about inputs, which it can check

thinking...

$y$ thinking...

... sum-check

$[LFKN90]$ more sum-checks
GKR08: IP for arithmetic circuit evaluation (non-ZK)

1. \( V \) sends inputs
2. \( P \) evaluates, returns output \( y \)
3. \( V \) constructs polynomial relating \( y \) to last layer's input wires
4. \( V \) engages \( P \) in a sum-check, gets claim about second-last layer
5. \( V \) iterates, gets claim about inputs, which it can check

\( V \) thinking... \( y \) thinking...

... sum-check

[LFKN90]
more sum-checks

\( y \)
GKR08: IP for arithmetic circuit evaluation (non-ZK)

1. $V$ sends inputs
2. $P$ evaluates, returns output $y$
3. $V$ constructs polynomial relating $y$ to last layer's input wires
4. $V$ engages $P$ in a sum-check, gets claim about second-last layer
5. $V$ iterates, gets claim about inputs, which it can check

$V \xrightarrow{x} P$

... sum-check

$[LFKN90]$ more sum-checks
GKR08: IP for arithmetic circuit evaluation (non-ZK)

1. \( \mathcal{V} \) sends inputs
2. \( \mathcal{P} \) evaluates, returns output \( y \)
3. \( \mathcal{V} \) constructs polynomial relating \( y \) to last layer’s input wires
4. \( \mathcal{V} \) engages \( \mathcal{P} \) in a sum-check

[LFKN90] more sum-checks
GKR08: IP for arithmetic circuit evaluation (non-ZK)

1. \( \mathcal{V} \) sends inputs
2. \( \mathcal{P} \) evaluates, returns output \( y \)
3. \( \mathcal{V} \) constructs polynomial relating \( y \) to last layer’s input wires
4. \( \mathcal{V} \) engages \( \mathcal{P} \) in a sum-check, gets claim about second-last layer

\[ \text{sum-check} \quad [\text{LFKN90}] \]
GKR08: IP for arithmetic circuit evaluation (non-ZK)

1. $V$ sends inputs
2. $P$ evaluates, returns output $y$
3. $V$ constructs polynomial relating $y$ to last layer's input wires
4. $V$ engages $P$ in a sum-check, gets claim about second-last layer
5. $V$ iterates

[Diagram of arithmetic circuit with sum-check notation]

$V$ and $P$ thinking...

$y$ thinking...

... sum-check

[Ref: LFKN90]

More sum-checks
GKR08: IP for arithmetic circuit evaluation (non-ZK)

1. \( \mathcal{V} \) sends inputs
2. \( \mathcal{P} \) evaluates, returns output \( y \)
3. \( \mathcal{V} \) constructs polynomial relating \( y \) to last layer's input wires
4. \( \mathcal{V} \) engages \( \mathcal{P} \) in a sum-check, gets claim about second-last layer
5. \( \mathcal{V} \) iterates

\[ \text{sum-check} \quad \text{[LFKN90]} \]

more sum-checks
GKR08: IP for arithmetic circuit evaluation (non-ZK)

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates, returns output $y$
3. $\mathcal{V}$ constructs polynomial relating $y$ to last layer's input wires
4. $\mathcal{V}$ engages $\mathcal{P}$ in a sum-check, gets claim about second-last layer
5. $\mathcal{V}$ iterates

$\mathcal{V} \ x \ \mathcal{P}$

sum-check [LFKN90]

more sum-checks
GKR08: IP for arithmetic circuit evaluation (non-ZK)

1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates, returns output $y$
3. $\mathcal{V}$ constructs polynomial relating $y$ to last layer's input wires
4. $\mathcal{V}$ engages $\mathcal{P}$ in a sum-check, gets claim about second-last layer
5. $\mathcal{V}$ iterates, gets claim about inputs, which it can check

[LFKN90]

more sum-checks
1. \( V \) sends inputs
2. \( P \) evaluates, returns output \( y \)
3. \( V \) constructs polynomial relating \( y \) to last layer's input wires
4. \( V \) engages \( P \) in a sum-check, gets claim about second-last layer
5. \( V \) iterates, gets claim about inputs, which it can check

To make this protocol ZK, \( P \) sends commitments to its messages [CD98].
1. \( V \) sends inputs
2. \( \mathcal{P} \) evaluates, returns output \( y \)
3. \( V \) constructs polynomial relating \( y \) to last layer’s input wires
4. \( V \) engages \( \mathcal{P} \) in a sum-check, gets claim about second-last layer
5. \( V \) iterates, gets claim about inputs, which it can check

In a ZK proof, AC inputs include \( w \), so \( V \) cannot check them directly!
Idea: use a \textit{polynomial commitment} [KZG10]

$\mathcal{V}$’s final check is to evaluate a polynomial $\tilde{m}$ that encodes input $x$ and witness $w$.  

\hspace{1cm} 1. $P$ commits to $\tilde{m}$ at the start of the protocol
2. $P$ and $\mathcal{V}$ run the interactive proof
3. $P$ evaluates $\tilde{m}(\cdot)$ at a point of $\mathcal{V}$’s choosing. . .
4. . . . and proves consistency with initial commitment.
Idea: use a *polynomial commitment* [KZG10]

\( \mathcal{V} \)'s final check is to evaluate a polynomial \( \tilde{m} \) that encodes input \( x \) and witness \( w \).

Instead of having \( \mathcal{V} \) evaluate \( \tilde{m} \) directly:

1. \( \mathcal{P} \) commits to \( \tilde{m} \) at the start of the protocol
Idea: use a *polynomial commitment* [KZG10]

V’s final check is to evaluate a polynomial $\tilde{m}$ that encodes input $x$ and witness $w$.

Instead of having $V$ evaluate $\tilde{m}$ directly:

1. $P$ commits to $\tilde{m}$ at the start of the protocol
2. $P$ and $V$ run the interactive proof
Idea: use a polynomial commitment [KZG10]

\( \mathcal{V} \)'s final check is to evaluate a polynomial \( \tilde{m} \) that encodes input \( x \) and witness \( w \).

Instead of having \( \mathcal{V} \) evaluate \( \tilde{m} \) directly:

1. \( \mathcal{P} \) commits to \( \tilde{m} \) at the start of the protocol
2. \( \mathcal{P} \) and \( \mathcal{V} \) run the interactive proof
3. \( \mathcal{P} \) evaluates \( \tilde{m}(\cdot) \) at a point of \( \mathcal{V} \)'s choosing...
Idea: use a polynomial commitment [KZG10]

\( \mathcal{V}'s \) final check is to evaluate a polynomial \( \tilde{m} \) that encodes input \( x \) and witness \( w \).

Instead of having \( \mathcal{V} \) evaluate \( \tilde{m} \) directly:

1. \( \mathcal{P} \) commits to \( \tilde{m} \) at the start of the protocol
2. \( \mathcal{P} \) and \( \mathcal{V} \) run the interactive proof
3. \( \mathcal{P} \) evaluates \( \tilde{m}(\cdot) \) at a point of \( \mathcal{V}'s \) choosing
4. ... and proves consistency with initial commitment.
Idea: use a polynomial commitment [KZG10]

V’s final check is to evaluate a polynomial \( \tilde{m} \) that encodes input \( x \) and witness \( w \).

Instead of having \( V \) evaluate \( \tilde{m} \) directly:

1. \( P \) commits to \( \tilde{m} \) at the start of the protocol
2. \( P \) and \( V \) run the interactive proof
3. \( P \) evaluates \( \tilde{m}(\cdot) \) at a point of \( V \)’s choosing...
4. ... and proves consistency with initial commitment.

Hyrax uses a new polynomial commitment scheme tailored to \textit{multilinear}\(^*\) polynomials like \( \tilde{m} \)

\(^*\)multivariate, linear in each variable
A polynomial commitment for $\tilde{m}$

$\tilde{m}(r) \triangleq L \cdot T \cdot R^T$

$\mathcal{V}$ can compute $L$ and $R$ from $r$, and

$T \triangleq \begin{bmatrix}
    w_0 & w_\ell & \cdots & w_{\ell^2-\ell} \\
    w_1 & w_{\ell+1} & \cdots & w_{\ell^2-\ell+1} \\
    \vdots & \vdots & \ddots & \vdots \\
    w_{\ell-1} & w_{2\cdot\ell-1} & \cdots & w_{\ell^2-1}
\end{bmatrix}$
A polynomial commitment for $\tilde{m}$

$$\tilde{m}(r) \triangleq L \cdot T \cdot R^T$$

$V$ can compute $L$ and $R$ from $r$, and

$$T \triangleq \begin{bmatrix}
w_0 & w_\ell & \cdots & w_{\ell^2-\ell} \\
w_1 & w_{\ell+1} & \cdots & w_{\ell^2-\ell+1} \\
& \vdots & \ddots & \vdots \\
w_{\ell-1} & w_{2\cdot\ell-1} & \cdots & w_{\ell^2-1}
\end{bmatrix}$$

Naive: $P$ sends commitments to each $w_i$
A polynomial commitment for $\tilde{m}$

$$\tilde{m}(r) \triangleq L \cdot T \cdot R^T$$

$\mathcal{V}$ can compute $L$ and $R$ from $r$, and

$$T \triangleq \begin{bmatrix} w_0 & w_\ell & \cdots & w_{\ell^2-\ell} \\ w_1 & w_{\ell+1} & \cdots & w_{\ell^2-\ell+1} \\ \vdots & \vdots & \ddots & \vdots \\ w_{\ell-1} & w_{2\cdot\ell-1} & \cdots & w_{\ell^2-1} \end{bmatrix}$$

Naive: $\mathcal{P}$ sends commitments to each $w_i$

$\times$ Proof size and $\mathcal{V}$ time are both $O(|w|)!$
A polynomial commitment for $\tilde{m}$

$$\tilde{m}(r) \triangleq L \cdot T \cdot R^T$$

$\mathcal{V}$ can compute $L$ and $R$ from $r$, and

$$T \triangleq \begin{bmatrix}
    w_0 & w_\ell & \cdots & w_{\ell^2-\ell} \\
    w_1 & w_{\ell+1} & \cdots & w_{\ell^2-\ell+1} \\
    \vdots & \vdots & \ddots & \vdots \\
    w_{\ell-1} & w_{2\ell-1} & \cdots & w_{\ell^2-1}
\end{bmatrix}$$

Better: $\mathcal{P}$ sends a *multi-commitment* to each row:

$$T_0 = \text{Com}(w_0, w_\ell, \ldots, w_{\ell^2-\ell}) \ [\text{Gro09}]$$
A polynomial commitment for $\tilde{m}$

$$\tilde{m}(r) \triangleq L \cdot T \cdot R^T$$

$\mathcal{V}$ can compute $L$ and $R$ from $r$, and

$$T \triangleq \begin{bmatrix}
  w_0 & w_\ell & \cdots & w_{\ell^2-\ell} \\
  w_1 & w_{\ell+1} & \cdots & w_{\ell^2-\ell+1} \\
  \vdots & \vdots & \ddots & \vdots \\
  w_{\ell-1} & w_{2\cdot\ell-1} & \cdots & w_{\ell^2-1}
\end{bmatrix}$$

Better: $\mathcal{P}$ sends a *multi-commitment* to each row:

$$T_0 = \text{Com}(w_0, w_\ell, \ldots, w_{\ell^2-\ell}) \quad [\text{Gro09}]$$

Pedersen commitments: vector-wise homomorphism.
A polynomial commitment for $\tilde{m}$ (cont’d)

$$\tilde{m}(r) \triangleq L \cdot T \cdot R^T$$

$$T \triangleq \begin{bmatrix}
  w_0 & w_\ell & \cdots & w_{\ell^2-\ell} \\
  w_1 & w_{\ell+1} & \cdots & w_{\ell^2-\ell+1} \\
  \vdots & \vdots & \ddots & \vdots \\
  w_{\ell-1} & w_{2\cdot\ell-1} & \cdots & w_{\ell^2-1}
\end{bmatrix}$$

1. $V$ uses homomorphism to compute $\text{Com}(L \cdot T)$. 

2. $P$ sends a commitment to an evaluation of $\tilde{m}(r)$.

3. $P$ uses a dot-product argument to convince $V$ that $\text{Com}(\tilde{m}(r))$ is consistent with $R$ and $\text{Com}(L \cdot T)$. 

A polynomial commitment for \( \tilde{m} \) (cont’d)

\[
\tilde{m}(r) \triangleq L \cdot T \cdot R^T
\]

\[
T \triangleq \begin{bmatrix}
w_0 & w_\ell & \cdots & w_{\ell^2-\ell} \\
w_1 & w_{\ell+1} & \cdots & w_{\ell^2-\ell+1} \\
\vdots & \vdots & \ddots & \vdots \\
w_{\ell-1} & w_{2\cdot\ell-1} & \cdots & w_{\ell^2-1}
\end{bmatrix}
\]

1. \( V \) uses homomorphism to compute \( \text{Com}(L \cdot T) \).
2. \( P \) sends a commitment to an evaluation of \( \tilde{m}(r) \).
A polynomial commitment for $\tilde{m}$ (cont’d)

$$\tilde{m}(r) \triangleq L \cdot T \cdot R^T$$

$$T \triangleq \begin{bmatrix}
    w_0 & w_\ell & \cdots & w_{\ell^2 - \ell} \\
    w_1 & w_{\ell+1} & \cdots & w_{\ell^2 - \ell + 1} \\
    \vdots & \vdots & \ddots & \vdots \\
    w_{\ell-1} & w_{2\ell - 1} & \cdots & w_{\ell^2 - 1}
\end{bmatrix}$$

1. $\mathcal{V}$ uses homomorphism to compute $\text{Com}(L \cdot T)$.
2. $\mathcal{P}$ sends a commitment to an evaluation of $\tilde{m}(r)$
3. $\mathcal{P}$ uses a dot-product argument to convince $\mathcal{V}$ that $\text{Com}(\tilde{m}(r))$ is consistent with $R$ and $\text{Com}(L \cdot T)$. 
A polynomial commitment for \( \tilde{m} \) (cont’d)

\[
\tilde{m}(r) \triangleq L \cdot T \cdot R^T
\]

\[
T \triangleq \begin{bmatrix}
w_0 & w_\ell & \cdots & w_{\ell^2-\ell} \\
w_1 & w_{\ell+1} & \cdots & w_{\ell^2-\ell+1} \\
\vdots & \vdots & \ddots & \vdots \\
w_{\ell-1} & w_{2\cdot\ell-1} & \cdots & w_{\ell^2-1}
\end{bmatrix}
\]

Dot-product argument has \( 2 \log |R| \) communication (adapted from Bulletproofs [BBBPWM18])
A polynomial commitment for $\tilde{m}$ (cont’d)

$$\tilde{m}(r) \triangleq L \cdot T \cdot R^T$$

$$T \triangleq \begin{bmatrix}
    w_0 & w_\ell & \cdots & w_{\ell^2 - \ell} \\
    w_1 & w_{\ell+1} & \cdots & w_{\ell^2 - \ell + 1} \\
    \vdots & \vdots & \ddots & \vdots \\
    w_{\ell-1} & w_{2\cdot\ell-1} & \cdots & w_{\ell^2 - 1}
\end{bmatrix}$$

Dot-product argument has $2 \log |R|$ communication (adapted from Bulletproofs [BBBPWM18])

$\mathcal{P}$ sends one commitment per row: $S_\mathcal{P} \in O\left(\sqrt{|w|}\right)$
A polynomial commitment for $\tilde{m}$ (cont’d)

$$\tilde{m}(r) \triangleq L \cdot T \cdot R^T$$

$$T \triangleq \begin{bmatrix}
    w_0 & w_\ell & \cdots & w_{\ell^2-\ell} \\
    w_1 & w_{\ell+1} & \cdots & w_{\ell^2-\ell+1} \\
    \vdots & \vdots & \ddots & \vdots \\
    w_{\ell-1} & w_{2\cdot \ell-1} & \cdots & w_{\ell^2-1}
\end{bmatrix}$$

Dot-product argument has $2 \log |R|$ communication (adapted from Bulletproofs [BBBPWM18])

$P$ sends one commitment per row: $S_P \in O\left(\sqrt{|w|}\right)$

$V$’s time is $O(|R| + |L|)$: $T_V \in O\left(\sqrt{|w|}\right)$
A polynomial commitment for \( \tilde{m} \) (cont’d)

\[
\tilde{m}(r) \triangleq L \cdot T \cdot R^T
\]

\[
T \triangleq \begin{bmatrix}
w_0 & w_\ell & \cdots & w_{\ell^2 - \ell} \\
w_1 & w_{\ell + 1} & \cdots & w_{\ell^2 - \ell + 1} \\
\vdots & \vdots & \ddots & \vdots \\
w_{\ell - 1} & w_{2 \cdot \ell - 1} & \cdots & w_{\ell^2 - 1}
\end{bmatrix}
\]

Dot-product argument has 2\( \log |R| \) communication (adapted from Bulletproofs [BBBPWM18])

\( \mathcal{P} \) sends one commitment per row: \( S_{\mathcal{P}} \in O\left(\sqrt{|w|}\right) \)

\( \mathcal{V} \)'s time is \( O(|R| + |L|) \): \( T_{\mathcal{V}} \in O\left(\sqrt{|w|}\right) \)

Can choose \( S_{\mathcal{P}} \cdot T_{\mathcal{V}} \in O(|w|) \) s.t. \( T_{\mathcal{V}} \in \Omega\left(\sqrt{|w|}\right) \)
Details and refinements (see paper)

Use Fiat-Shamir heuristic [FS86] to make non-interactive (in the random oracle model)
Details and refinements (see paper)

Use Fiat-Shamir heuristic [FS86] to make non-interactive (in the random oracle model)

Tailored ZK transform [CD98] using multi-commitments reduces proof size and \( \mathcal{V} \) time
Details and refinements (see paper)

Use Fiat-Shamir heuristic [FS86] to make non-interactive (in the random oracle model)

Tailored ZK transform [CD98] using multi-commitments → reduces proof size and \( V \) time

Redistribution layer
→ lets Hyrax extract parallelism from serial computations
Use Fiat-Shamir heuristic [FS86] to make non-interactive (in the random oracle model)

Tailored ZK transform [CD98] using multi-commitments → reduces proof size and \( \nu \) time

Redistribution layer
→ lets Hyrax extract parallelism from serial computations

Gir\(^{++} \) IP: Giraffe [WJBsTWW17] plus a tweak [CFS17] → reduces proof size
Roadmap

1. General-purpose ZK proof systems
2. Hyrax at a high level
3. Evaluation
Evaluation overview

Baselines:
- BCCGP-sqrt [BCCGP16]—re-implemented
- Bulletproofs [BBBPWM18]—re-implemented
- ZKB++ [CDGORRSZ17]—ran authors’ implementation
- Ligero [AHIV17]—ran authors’ implementation
- libSTARK [BBHR18]—ran authors’ implementation

- Hyrax-$^{1/3}$—$T$ has $\ell$ rows, $\ell^2$ columns
- Hyrax-naive—no refinements
Evaluation overview

Baselines:

- BCCGP-sqrt [BCCGP16]—re-implemented
- Bulletproofs [BBBPWM18]—re-implemented
- ZKB++ [CDGORRSZ17]—ran authors’ implementation
- Ligero [AHIV17]—ran authors’ implementation
- libSTARK [BBHR18]—ran authors’ implementation

- Hyrax-\(1/3\)—\(T\) has \(\ell\) rows, \(\ell^2\) columns
- Hyrax-naive—no refinements

Parameters: \(\approx\)90-bit security (M191 elliptic curve)
Evaluation overview

Baselines:
- BCCGP-sqrt [BCCGP16]—re-implemented
- Bulletproofs [BBBPWM18]—re-implemented
- ZKB++ [CDGORRSZ17]—ran authors’ implementation
- Ligero [AHIV17]—ran authors’ implementation
- libSTARK [BBHR18]—ran authors’ implementation

- Hyrax-$1/3$—$T$ has $\ell$ rows, $\ell^2$ columns
- Hyrax-naive—no refinements

Parameters: $\approx90$-bit security (M191 elliptic curve)

Benchmark: SHA-256 Merkle tree, varying number of leaves
Proof size

log₂ \( M \), number of leaves in Merkle tree

- 2
- 4
- 6
- 8

proof size, kiB (lower is better)

Hyrax-\( \frac{1}{2} \)
Hyrax-naive
BCCGP-sqrt
Bulletproofs
ZKB++
Ligero
libSTARK

\( \times \)
**P time**

![Graph showing the relationship between log₂ M, number of leaves in Merkle tree, and prover time (seconds) for different proof systems: Hyrax-1/2, Hyrax-naive, BCCGP-sqrt, Bulletproofs, ZKB++, Ligero, and libSTARK.](image)

- **Log₂ M, number of leaves in Merkle tree:**
  - 1
  - 10
  - 100
  - 10^3
  - 10^4

- **Prover time, seconds (lower is better):**
  - 2
  - 4
  - 6
  - 8

**Proof size, kiB**

- Hyrax-1/2
- Hyrax-naive
- BCCGP-sqrt
- Bulletproofs
- ZKB++
- Ligero
- libSTARK

- 100
- 2 × 100
- 3 × 100
- 4 × 100
- 6 × 100
Recap

We design, implement, and evaluate *Hyrax*, a zkSNARK for “data-parallel” AC satisfiability.
Recap

We design, implement, and evaluate *Hyrax*, a zkSNARK for “data-parallel” AC satisfiability.

✔️ Hyrax’s proofs are **small**: to get smaller, you have to pay more computation.

https://hyrax.crypto.fyi

https://github.com/hyrax:K
Recap

We design, implement, and evaluate Hyrax, a zkSNARK for “data-parallel” AC satisfiability

✓ Hyrax’s proofs are small:
   to get smaller, you have to pay more computation.

✓ Hyrax is fast:
   to get faster, you have to accept bigger proofs.
Recap

We design, implement, and evaluate Hyrax, a zkSNARK for “data-parallel” AC satisfiability

- Hyrax’s proofs are **small**: to get smaller, you have to pay more computation.

- Hyrax is **fast**: to get faster, you have to accept bigger proofs.

Hyrax is one useful point in a large tradeoff space. There is still plenty of room for improvement!

https://hyrax.crypto.fyi
https://github.com/hyrax
Recap

We design, implement, and evaluate Hyrax, a zkSNARK for “data-parallel” AC satisfiability

✓ Hyrax’s proofs are small:
  to get smaller, you have to pay more computation.

✓ Hyrax is fast:
  to get faster, you have to accept bigger proofs.

Hyrax is one useful point in a large tradeoff space. There is still plenty of room for improvement!

https://hyrax.crypto.fyi
https://github.com/hyraxZK