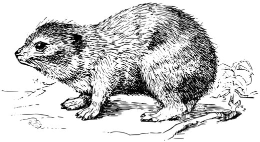
Doubly-efficient zkSNARKs without trusted setup

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"...

zkSNARK

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 ($\mathcal{P}$) time

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 ($\mathcal{P}$) time

Verifier ($\mathcal{V}$) time

Cryptographic assumptions

Trusted setup?

Our contributions

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

Our contributions

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

Proof size is sub-linear in $|\mathcal{C}|$ and $|w|$

Prover time is linear in $|\mathcal{C}|$

Verifier time is sublinear in $|\mathcal{C}|$ and $|w|$

Our contributions

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

Proof size is sub-linear in $|\mathcal{C}|$ and $|w|$

Prover time is linear in $|\mathcal{C}|$

Verifier time is sublinear in $|\mathcal{C}|$ and $|w|$

Good constants: concrete costs are low

Our contributions

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

Proof size is sub-linear in $|\mathcal{C}|$ and $|w|$

Prover time is linear in $|\mathcal{C}|$

Verifier time is sublinear in $|\mathcal{C}|$ 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 $\mathcal{V}$'s input x, $\exists w : \mathcal{C}(x, w) = 1$ (and $\mathcal{P}$ knows w)

➔ We evaluate Hyrax and five other ZK systems.

We find that:

Our contributions

➔ We design and implement *Hyrax*, a zkSNARK for "parallel" arithmetic circuit satisfiability: for $\mathcal{V}$'s input x, $\exists w : \mathcal{C}(x, w) = 1$ (and $\mathcal{P}$ 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 $\mathcal{V}$'s input x, $\exists w : \mathcal{C}(x, w) = 1$ (and $\mathcal{P}$ 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 $\mathcal{V}$'s input x, $\exists w : \mathcal{C}(x, w) = 1$ (and $\mathcal{P}$ 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.

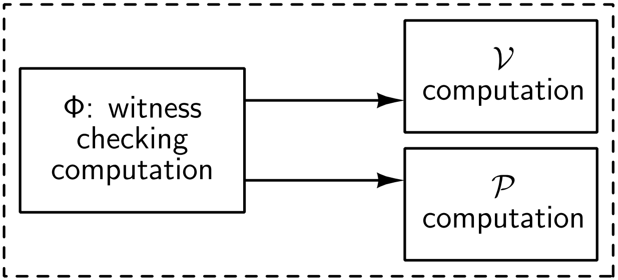
Hyrax is one useful point in a large tradeoff space

Roadmap

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

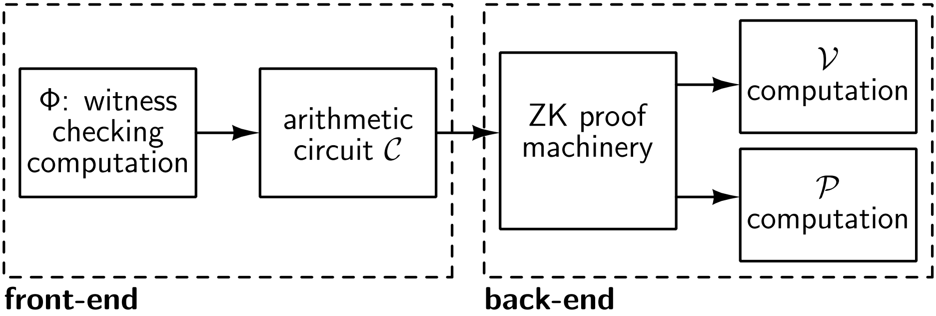
General-purpose ZK proof systems for NP

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



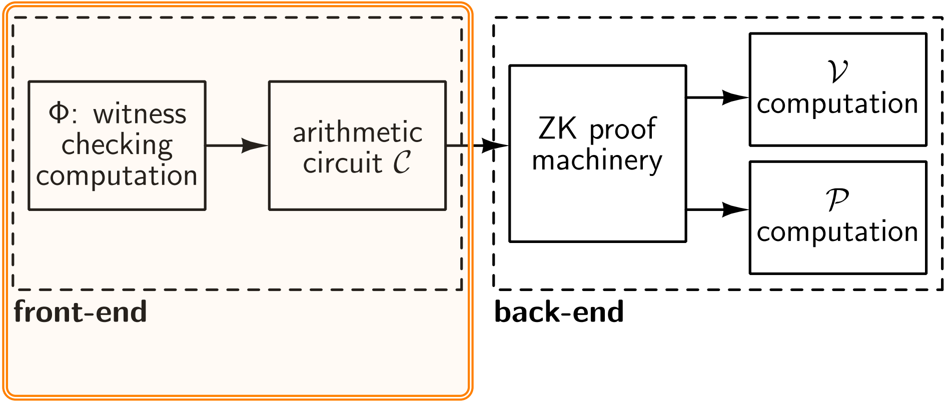
General-purpose ZK proof systems for NP

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



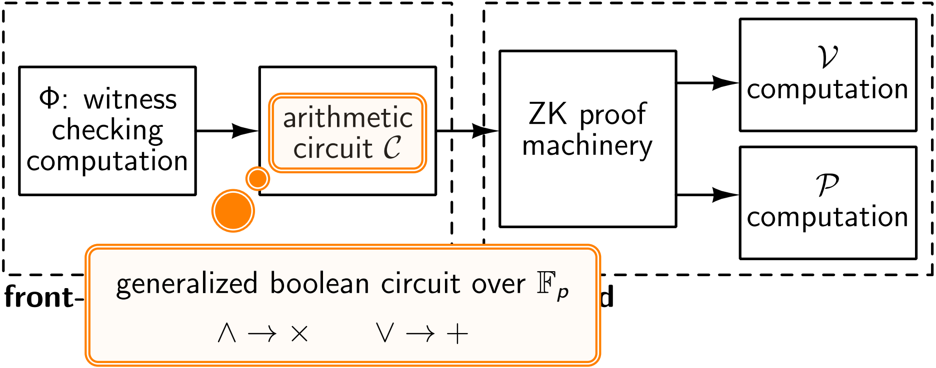
General-purpose ZK proof systems for NP

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



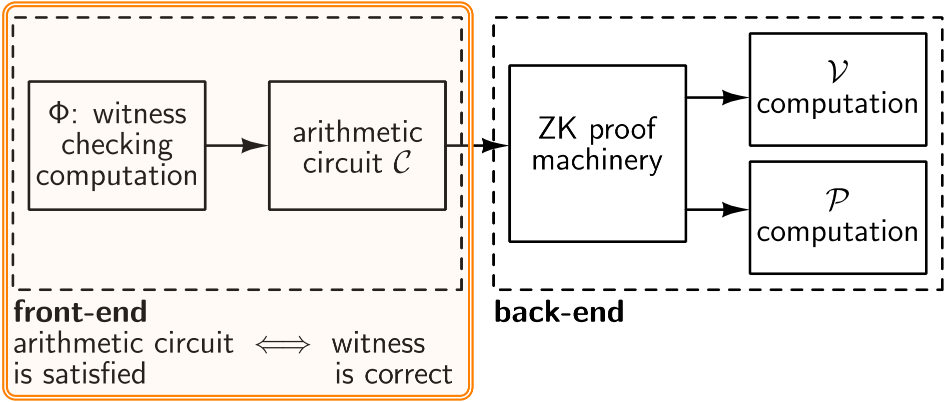
General-purpose ZK proof systems for NP

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



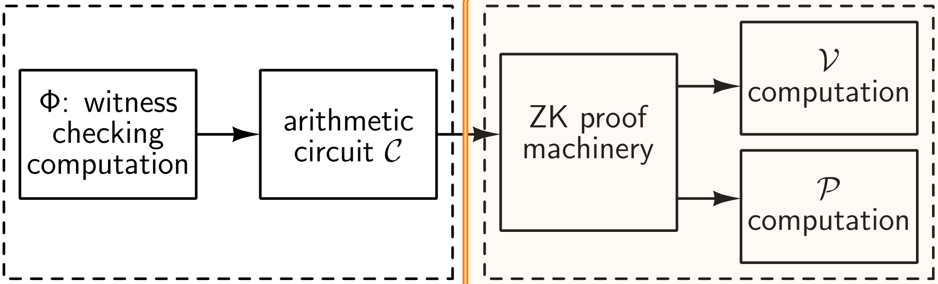
General-purpose ZK proof systems for NP

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



front-end
arithmetic circuit is satisfied $\iff$ witness is correct

back-end
valid proof $\iff$ arithmetic circuit is satisfied

Existing systems use a wide range of proof machinery

Linear PCPs [IKO07,Gro09,Gro10,BG12,Lip12,BCIOP13,GGPR13,...]

- Pinocchio [PGHR13], libsnark [BCTV14]

	Short Proofs	Fast $\mathcal{P}$	Fast $\mathcal{V}$	Trusted setup?	Assumption
libsnark	✓	✗	✓	✗	Knowledge of exponent

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]

	Short Proofs	Fast $\mathcal{P}$	Fast $\mathcal{V}$	Trusted setup?	Assumption
libsnark	✓	✗	✓	✗	Knowledge of exponent
Bulletproofs	✓	✗	✗	✓	discrete log

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]

	Short Proofs	Fast $\mathcal{P}$	Fast $\mathcal{V}$	Trusted setup?	Assumption
libsnark	✓	✗	✓	✗	Knowledge of exponent
Bulletproofs	✓	✗	✗	✓	discrete log
ZKB++	✗	✓	✗(ish)	✓	collision-resistant hashes

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 Proofs	Fast $\mathcal{P}$	Fast $\mathcal{V}$	Trusted setup?	Assumption
libsnark	✓	✗	✓	✗	Knowledge of exponent
Bulletproofs	✓	✗	✗	✓	discrete log
ZKB++	✗	✓	✗(ish)	✓	collision-resistant hashes
Ligero	✓(ish)	✓	✓(ish)	✓	collision-resistant hashes

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]

	Short Proofs	Fast $\mathcal{P}$	Fast $\mathcal{V}$	Trusted setup?	Assumption
libsnark	✓	✗	✓	✗	Knowledge of exponent
Bulletproofs	✓	✗	✗	✓	discrete log
ZKB++	✗	✓	✗(ish)	✓	collision-resistant hashes
Ligero	✓(ish)	✓	✓(ish)	✓	collision-resistant hashes
libSTARK	✓	✗	✓	✓	Reed-Solomon conjecture

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 $\mathcal{V}$ 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 $C \leftarrow \mathsf{Com}(m)$, 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

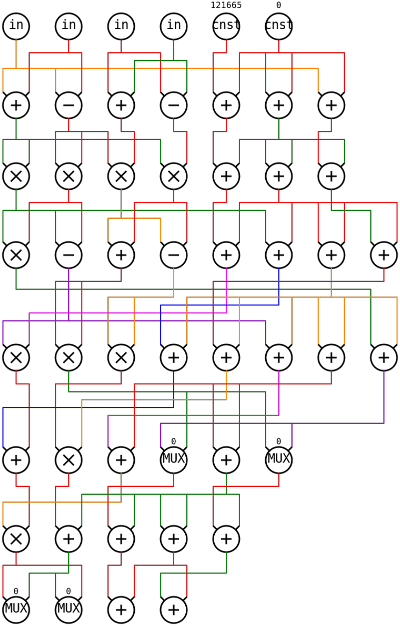
$$C_0 \odot C_1 \triangleq \text{Com}(m_0 + m_1)$$

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

The Pedersen commitment has this property.

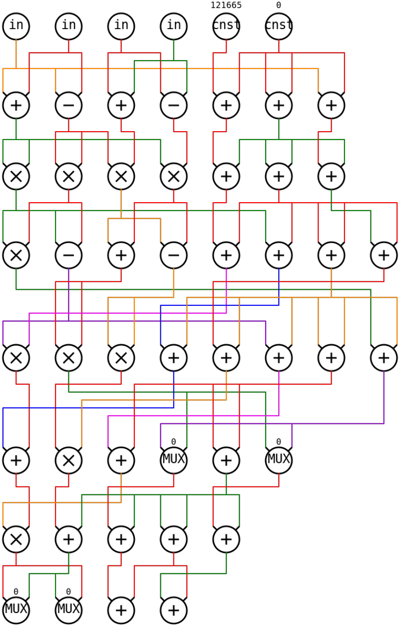
GKR08: IP for arithmetic circuit evaluation (non-ZK)

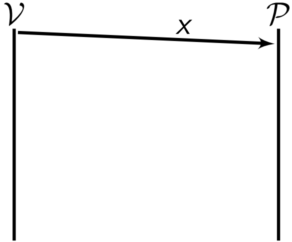
Witness checker must be expressed as a *layered* AC.



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

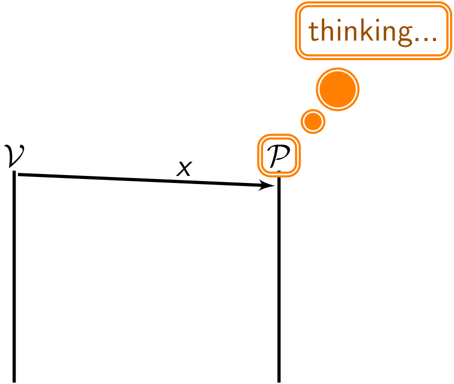
1. $\mathcal{V}$ sends inputs

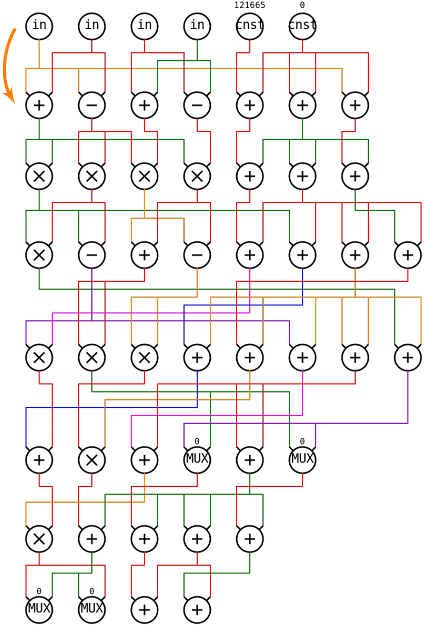




GKR08: IP for arithmetic circuit evaluation (non-ZK)

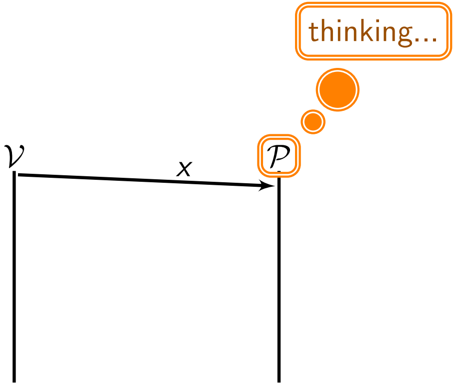
1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates

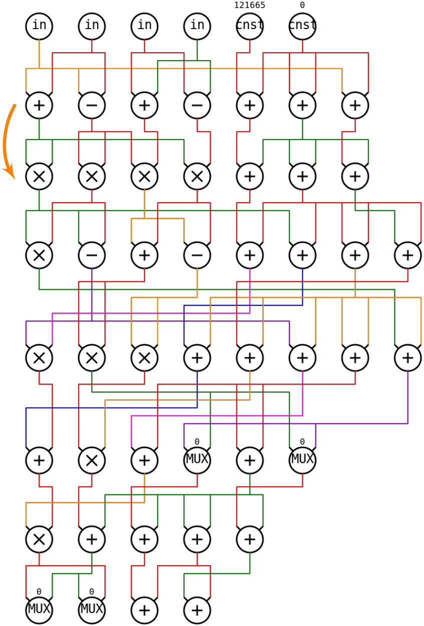




GKR08: IP for arithmetic circuit evaluation (non-ZK)

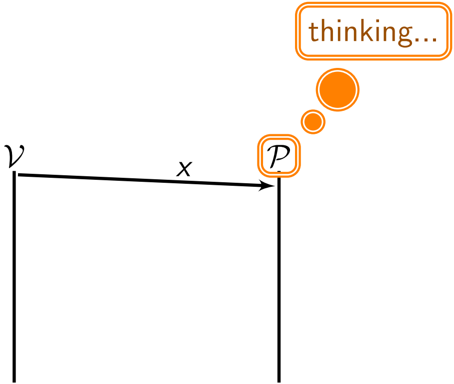
1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates

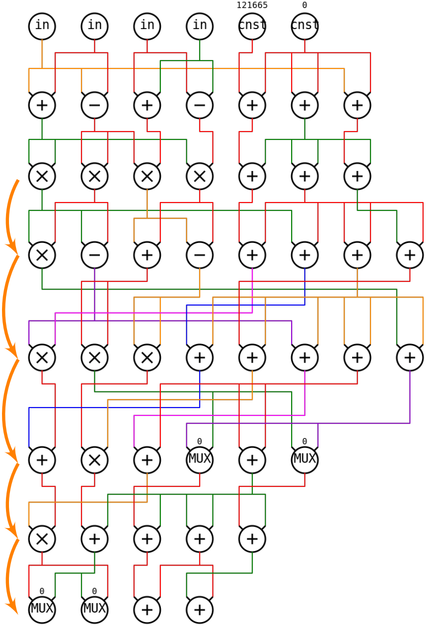




GKR08: IP for arithmetic circuit evaluation (non-ZK)

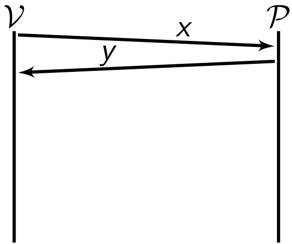
1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates

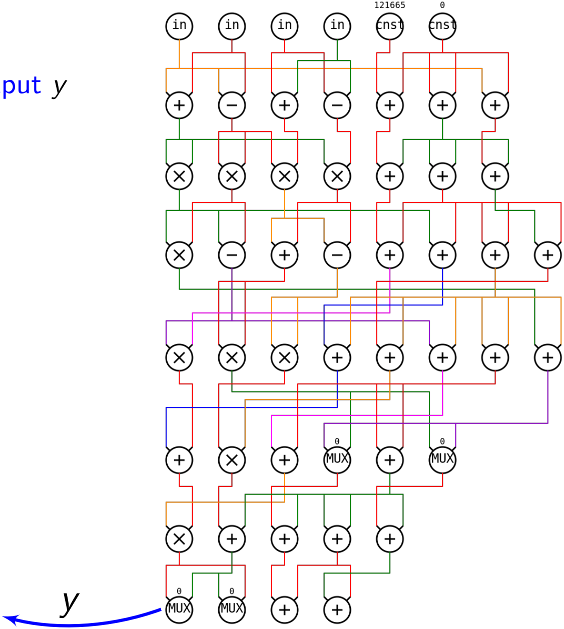




GKR08: IP for arithmetic circuit evaluation (non-ZK)

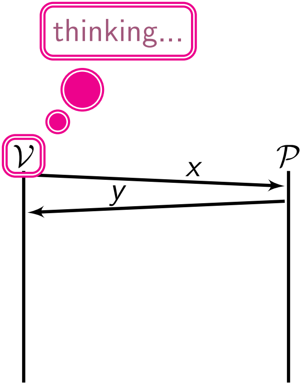
1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates, returns output y

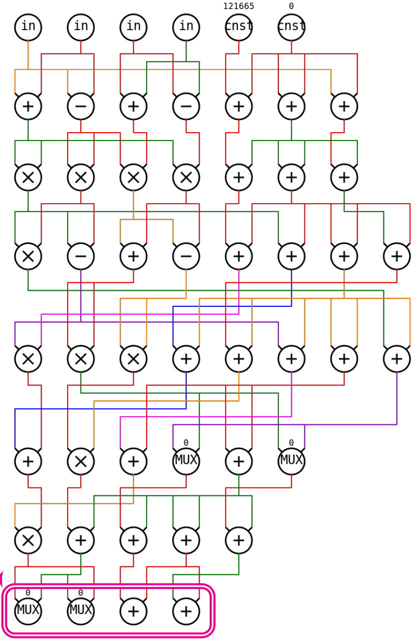




GKR08: IP for arithmetic circuit evaluation (non-ZK)

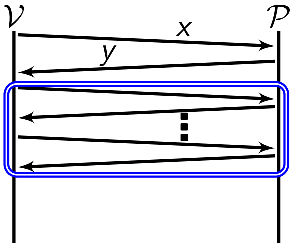
1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates, returns output y
3. $\mathcal{V}$ constructs polynomial relating y to last layer's input wires



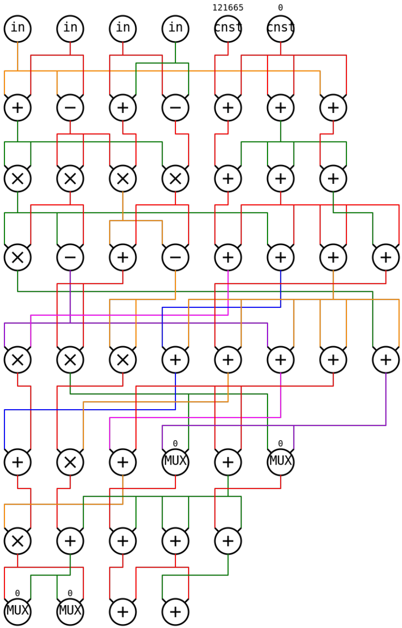


GKR08: IP for arithmetic circuit evaluation (non-ZK)

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

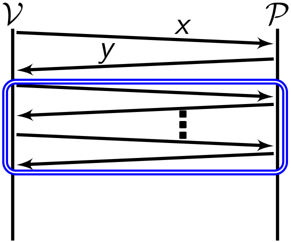


sum-check [LFKN90]

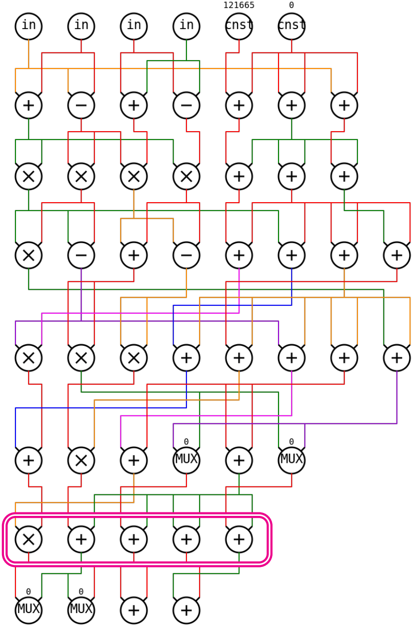


GKR08: IP for arithmetic circuit evaluation (non-ZK)

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

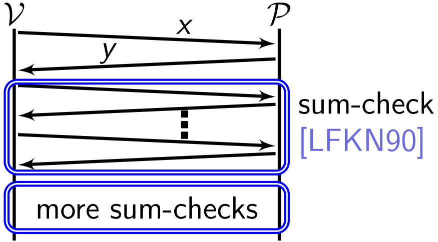


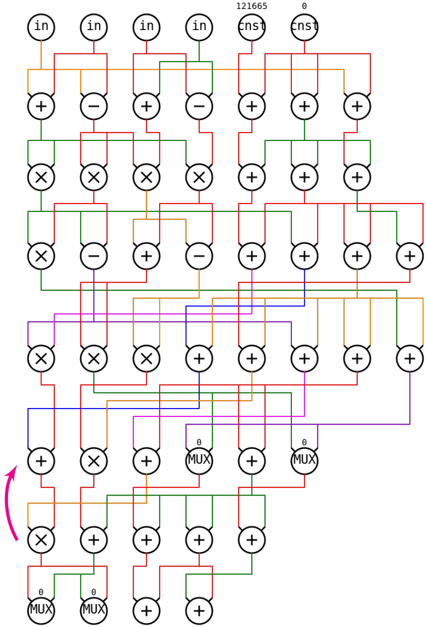
sum-check
[LFKN90]



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

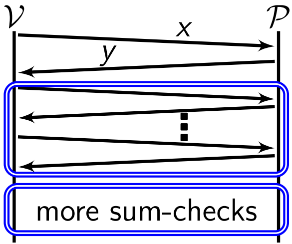
1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates, returns output y
3. $\mathcal{V}$ constructs polynomial relating y to last layer's input wires
4. $\mathcal{V}$ engages $\mathcal{P}$ in a sum-check, gets claim about second-last layer
5. $\mathcal{V}$ iterates



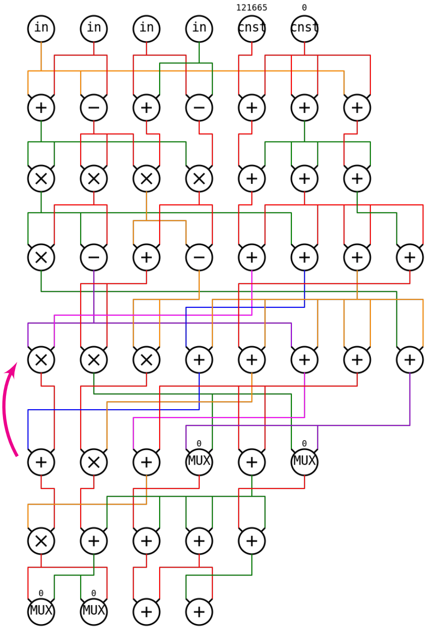


GKR08: IP for arithmetic circuit evaluation (non-ZK)

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

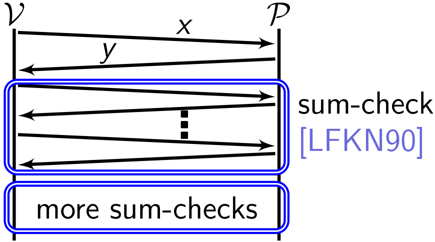


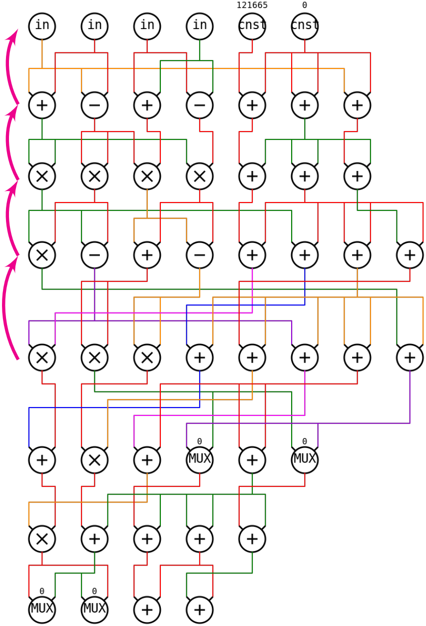
sum-check [LFKN90]



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

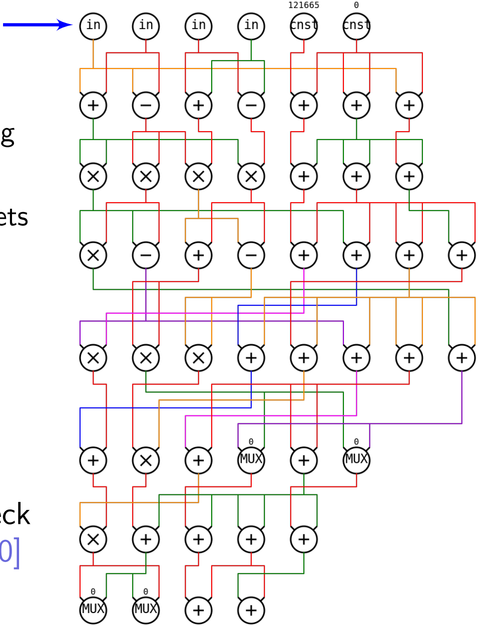
1. $\mathcal{V}$ sends inputs
2. $\mathcal{P}$ evaluates, returns output y
3. $\mathcal{V}$ constructs polynomial relating y to last layer's input wires
4. $\mathcal{V}$ engages $\mathcal{P}$ in a sum-check, gets claim about second-last layer
5. $\mathcal{V}$ iterates

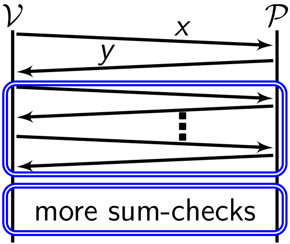




GKR08: IP for arithmetic circuit evaluation (non-ZK)

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

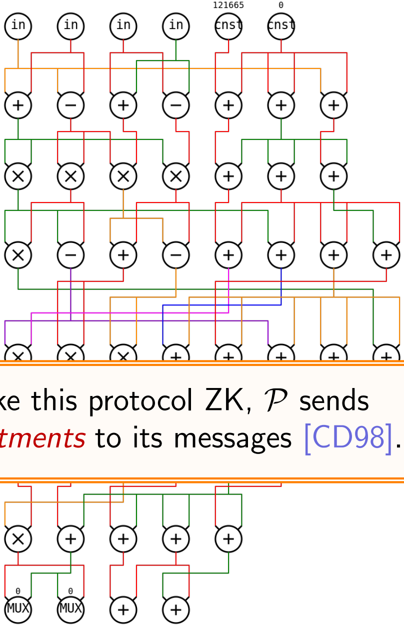




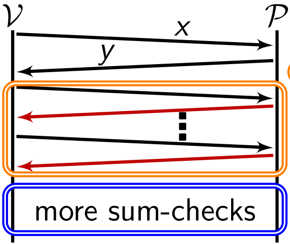
sum-check [LFKN90]

GKR08: IP for arithmetic circuit evaluation (with 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



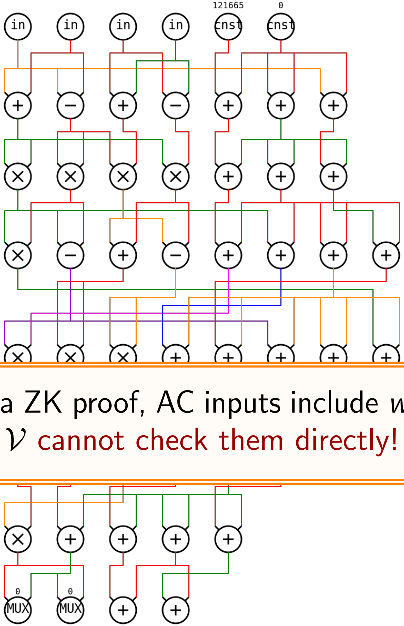
To make this protocol ZK, $\mathcal{P}$ sends *commitments* to its messages [CD98].



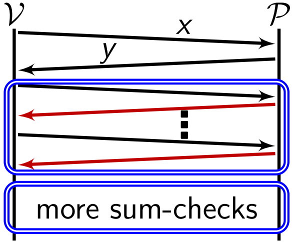
sum-check [LFKN90]

GKR08: IP for arithmetic circuit evaluation (with 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



In a ZK proof, AC inputs include w, so $\mathcal{V}$ cannot check them directly!



sum-check [LFKN90]

Idea: use a *polynomial commitment* [KZG10]

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

Idea: use a *polynomial commitment* [KZG10]

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

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

1. $\mathcal{P}$ commits to $\widetilde{m}$ at the start of the protocol

Idea: use a *polynomial commitment* [KZG10]

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

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

1. $\mathcal{P}$ commits to $\widetilde{m}$ at the start of the protocol
2. $\mathcal{P}$ and $\mathcal{V}$ run the interactive proof

Idea: use a *polynomial commitment* [KZG10]

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

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

1. $\mathcal{P}$ commits to $\widetilde{m}$ at the start of the protocol
2. $\mathcal{P}$ and $\mathcal{V}$ run the interactive proof
3. $\mathcal{P}$ evaluates $\widetilde{m}(\cdot)$ at a point of $\mathcal{V}$'s choosing...

Idea: use a *polynomial commitment* [KZG10]

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

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

1. $\mathcal{P}$ commits to $\widetilde{m}$ at the start of the protocol
2. $\mathcal{P}$ and $\mathcal{V}$ run the interactive proof
3. $\mathcal{P}$ evaluates $\widetilde{m}(\cdot)$ 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 $\widetilde{m}$ that encodes input x and witness w.

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

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

Hyrax uses a new polynomial commitment scheme tailored to *multilinear** polynomials like $\widetilde{m}$

*multivariate, linear in each variable

A polynomial commitment for $\widetilde{m}$

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

$\mathcal{V}$ 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 $\widetilde{m}$

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

$\mathcal{V}$ 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

A polynomial commitment for $\widetilde{m}$

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

$\mathcal{V}$ 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

✗ Proof size and $\mathcal{V}$ time are both O($|w|$)!

A polynomial commitment for $\widetilde{m}$

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

$\mathcal{V}$ 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:

$T_0 = \mathrm{Com}(w_0, w_\ell, \ldots, w_{\ell^2-\ell})$ [*Gro*09]

A polynomial commitment for $\widetilde{m}$

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

$\mathcal{V}$ 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:

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

Pedersen commitments: vector-wise homomorphism.

A polynomial commitment for $\widetilde{m}$ (cont'd)

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

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

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

A polynomial commitment for $\widetilde{m}$ (cont'd)

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

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

1. $\mathcal{V}$ uses homomorphism to compute $\mathrm{Com}(L \cdot T)$.
2. $\mathcal{P}$ sends a commitment to an evaluation of $\widetilde{m}(r)$

A polynomial commitment for $\widetilde{m}$ (cont'd)

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

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

1. $\mathcal{V}$ uses homomorphism to compute $\mathrm{Com}(L \cdot T)$.
2. $\mathcal{P}$ sends a commitment to an evaluation of $\widetilde{m}(r)$
3. $\mathcal{P}$ uses a *dot-product argument* to convince $\mathcal{V}$ that $\mathrm{Com}(\widetilde{m}(r))$ is consistent with R and $\mathrm{Com}(L \cdot T)$.

A polynomial commitment for $\widetilde{m}$ (cont'd)

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

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

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

A polynomial commitment for $\widetilde{m}$ (cont'd)

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

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

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

$\mathcal{P}$ sends one commitment per row: $\mathsf{S}_{\mathcal{P}} \in \mathrm{O}\left(\sqrt{|w|}\right)$

A polynomial commitment for $\widetilde{m}$ (cont'd)

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

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

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

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

$\mathcal{V}$'s time is $\mathsf{O}(|R| + |L|)$: $\mathsf{T}_{\mathcal{V}} \in \mathsf{O}\left(\sqrt{|w|}\right)$

A polynomial commitment for $\widetilde{m}$ (cont'd)

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

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

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

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

$\mathcal{V}$'s time is $\mathsf{O}(|R| + |L|)$: $\mathsf{T}_{\mathcal{V}} \in \mathsf{O}\left(\sqrt{|w|}\right)$

Can choose $\mathsf{S}_{\mathcal{P}} \cdot \mathsf{T}_{\mathcal{V}} \in \mathsf{O}(|w|)$ s.t. $\mathsf{T}_{\mathcal{V}} \in \Omega\left(\sqrt{|w|}\right)$

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 $\mathcal{V}$ time

Redistribution layer
➔ lets Hyrax extract parallelism from serial computations

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

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 ℓ rows, ℓ^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 ℓ rows, ℓ^2 columns
- ★ Hyrax-naive—no refinements

Parameters: ≈90-bit security (M191 elliptic curve)

Evaluation overview

Baselines:

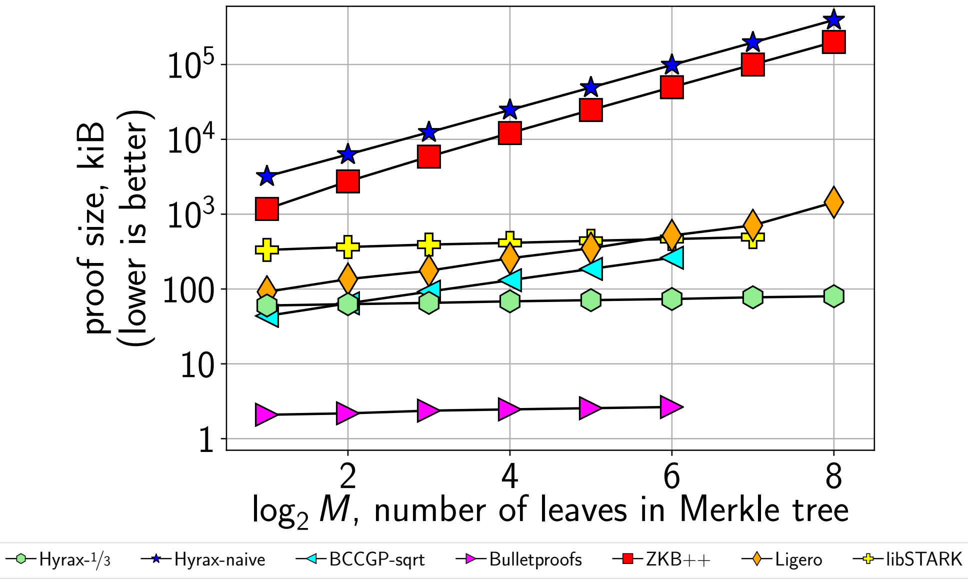
- BCCGP-sqrt [BCCGP16]—re-implemented
- Bulletproofs [BBBPWM18]—re-implemented
- ZKB++ [CDGORRSZ17]—ran authors' implementation
- Ligero [AHIV17]—ran authors' implementation
- libSTARK [BBHR18]—ran authors' implementation

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

Parameters: ≈90-bit security (M191 elliptic curve)

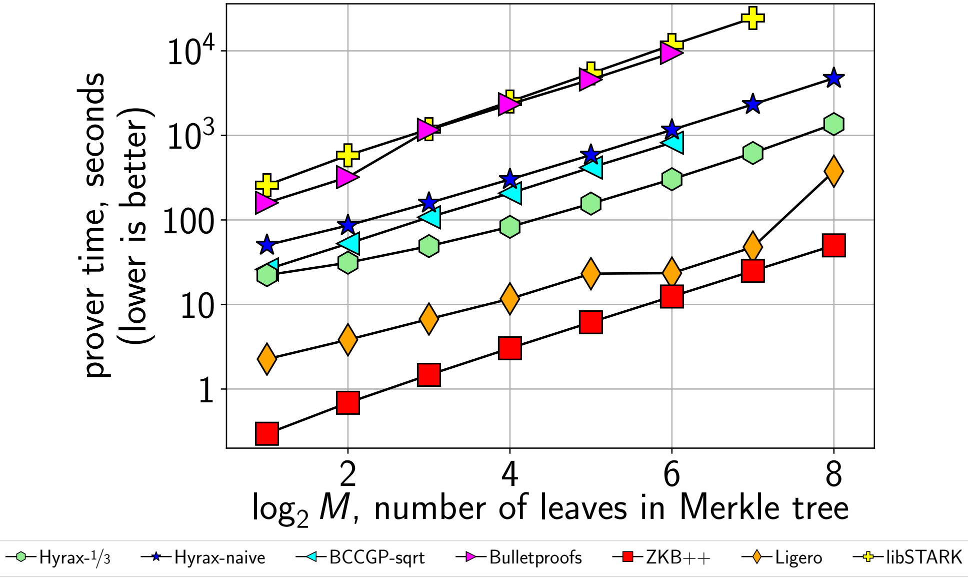
Benchmark: SHA-256 Merkle tree,
varying number of leaves

Proof size

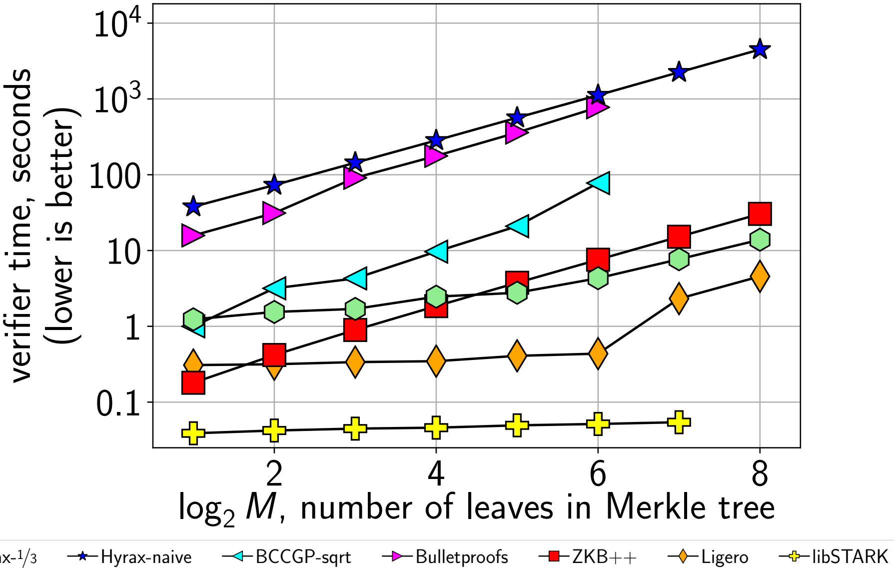


proof size, kiB
(lower is better)
10^5
10^4
10^3
100
10
1
2
4
6
8
$\log_2 M$, number of leaves in Merkle tree
Hyrax-1/3
Hyrax-naive
BCCGP-sqrt
Bulletproofs
ZKB++
Ligero
libSTARK

$\mathcal{P}$ time



$\mathcal{V}$ time



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.

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!

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`