Rise of the SNARKs

an introduction to zero-knowledge proofs, zkSNARKs, and `libsnark`

Howard Wu
Blockchain at Berkeley Developers Decal
Spring 2018

Cryptography is a powerful tool for building secure systems.

Much of the cryptography used today offers security properties for **data**.

Confidentiality

Authenticity

What about security properties for **computation**?

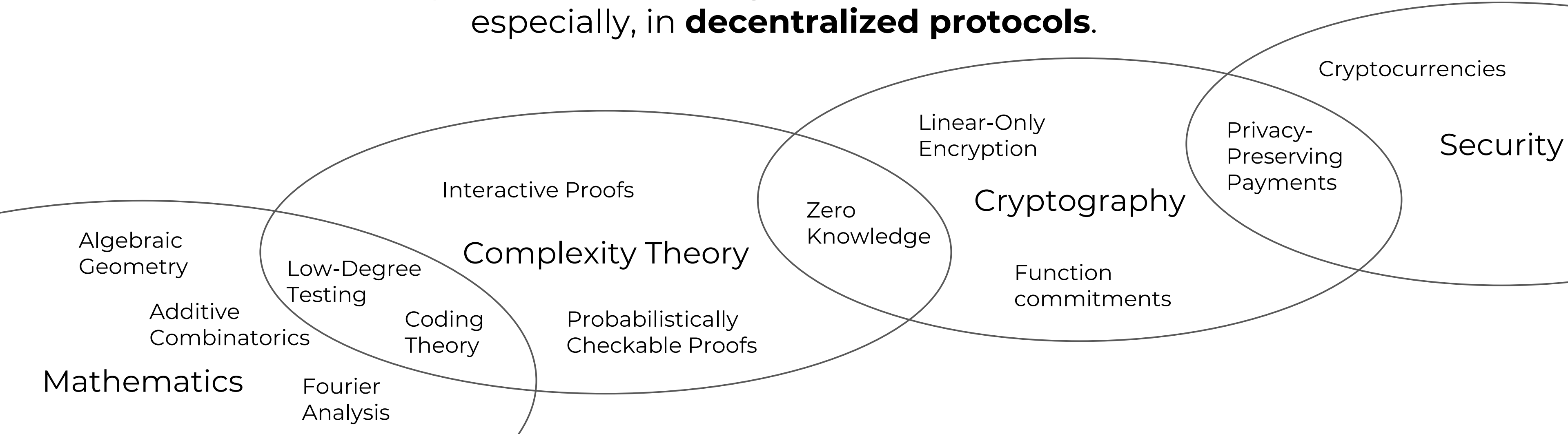
Cryptographic proofs offer
privacy-preserving integrity for computation.

Cryptographic Proofs

A powerful defense against malicious behavior, especially, in **decentralized protocols**.

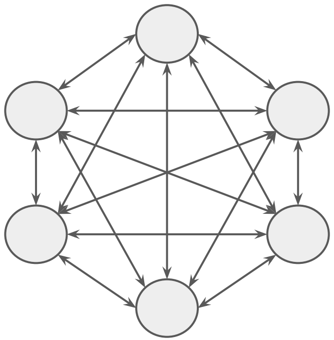
Cryptographic Proofs

A powerful defense against malicious behavior, especially, in **decentralized protocols**.



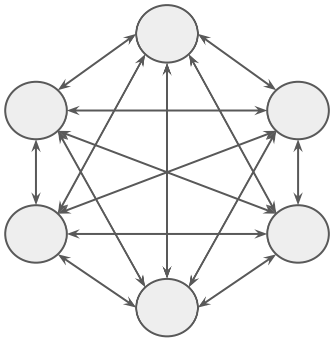
Cryptographic Proofs

1980s Securely compute $y = \boldsymbol{F}(x_1, \ldots, x_n)$ via a multi-party protocol



Cryptographic Proofs

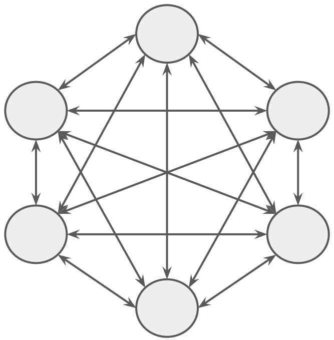
1980s Securely compute $y = \boldsymbol{F}(x_1, \ldots, x_n)$ via a multi-party protocol



Assumptions
Closed network
All nodes are online
Small number of nodes

Cryptographic Proofs

1980s Securely compute $y = \boldsymbol{F}(x_1, \dots, x_n)$ via a multi-party protocol



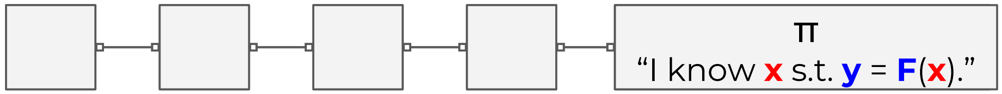
Assumptions

Closed network
All nodes are online
Small number of nodes

Zero-knowledge proof that
every message is sent according to the protocol
(consistent with the input and random tape).

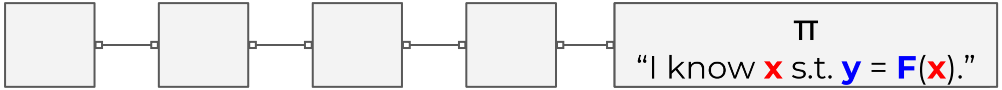
Cryptographic Proofs

2010s Blockchain Technology



Cryptographic Proofs

2010s Blockchain Technology

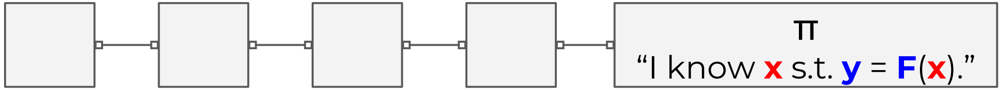


Assumptions

Open network
Nodes can be online or offline
Planetary scale

Cryptographic Proofs

2010s Blockchain Technology



Assumptions

Open network
Nodes can be online or offline
Planetary scale

Desirable Proof Properties

Succinct
Non-Interactive
Publicly Verifiable
Zero Knowledge

Zero-Knowledge Proofs

Zero-Knowledge Proofs

Privacy-preserving cryptographic proofs of computational integrity.

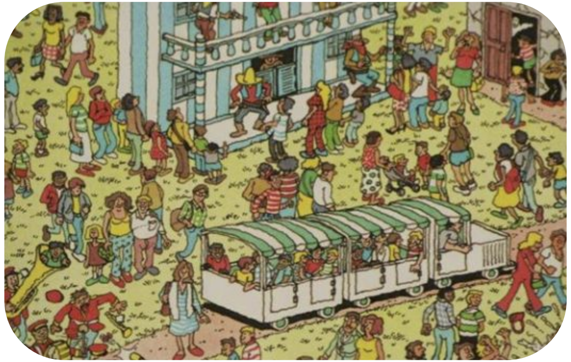
Zero-Knowledge Proofs

Privacy-preserving cryptographic proofs of computational integrity.

I can't tell you the secret,
but I can prove to you that I know the secret.

Zero-Knowledge Proofs

Privacy-preserving cryptographic proofs of computational integrity.

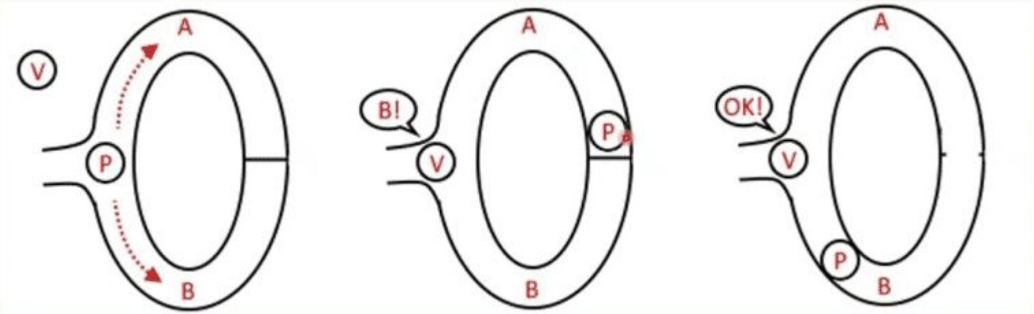


Can you prove to me where Waldo is,
without saying anything about where he is?

Zero-Knowledge Proofs

Privacy-preserving cryptographic proofs of computational integrity.

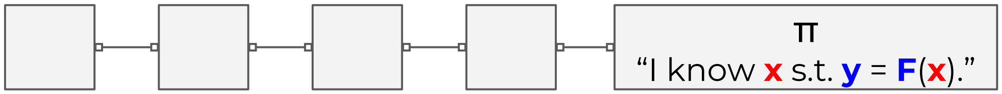
The Ali Baba Cave



from *“How to Explain Zero-Knowledge Protocols to Your Children”*

Zero-Knowledge Proofs

2010s Blockchain Technology



Assumptions

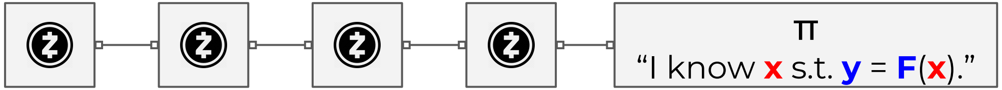
Open network
Nodes can be online or offline
Planetary scale

Proof Properties

Succinct
Non-Interactive
Publicly Verifiable
Zero Knowledge

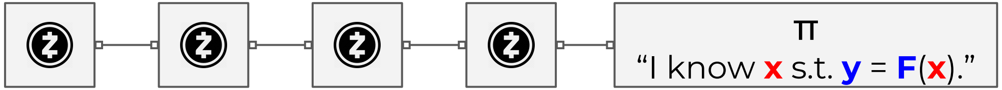
Zero-Knowledge Proofs

2010s Zerocash Protocol (Zcash)



Zero-Knowledge Proofs

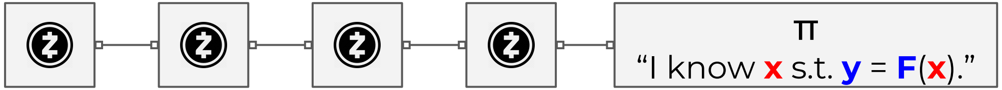
2010s Zerocash Protocol (Zcash)



A cryptographic protocol achieving a digital currency that is:

Zero-Knowledge Proofs

2010s Zerocash Protocol (Zcash)

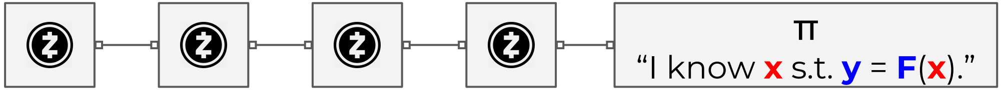


A cryptographic protocol achieving a digital currency that is:

- **Decentralized** - *works when given any (ideal) ledger*

Zero-Knowledge Proofs

2010s Zerocash Protocol (Zcash)

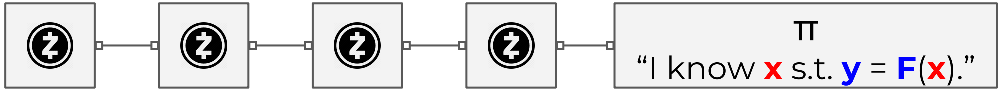


A cryptographic protocol achieving a digital currency that is:

- **Decentralized** - *works when given any (ideal) ledger*
- **Privacy-preserving** - *anyone can post a payment transaction to anyone else, while provably hiding the sender, receiver, and amount*

Zero-Knowledge Proofs

2010s Zerocash Protocol (Zcash)



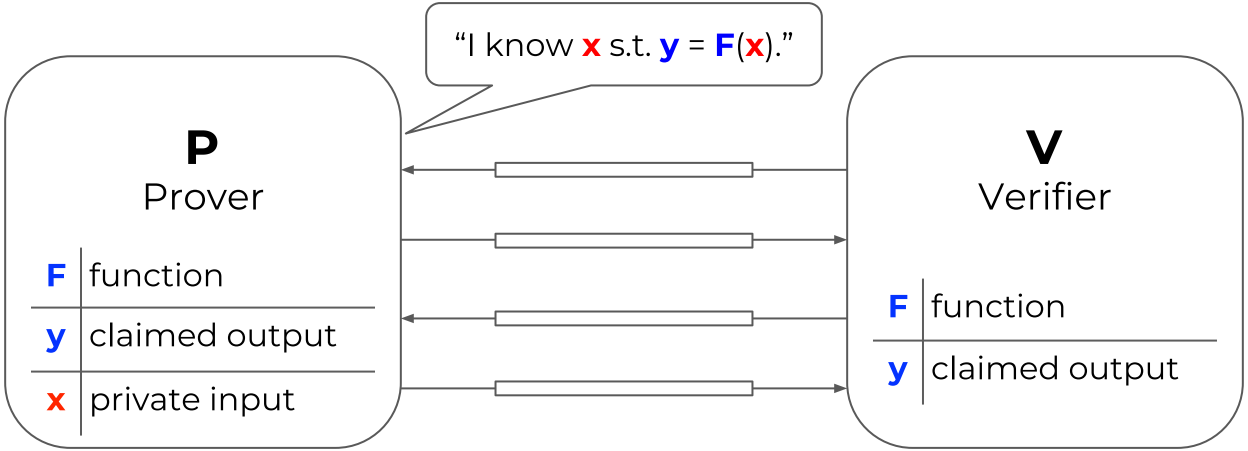
A cryptographic protocol achieving a digital currency that is:

- **Decentralized** - *works when given any (ideal) ledger*
- **Privacy-preserving** - *anyone can post a payment transaction to anyone else, while provably hiding the sender, receiver, and amount*
- **Efficient** - *payment transactions take less than 1 minute to produce, are less than 1 kB in size, and take a few milliseconds to verify*

[GMR89]

Zero-Knowledge Proofs

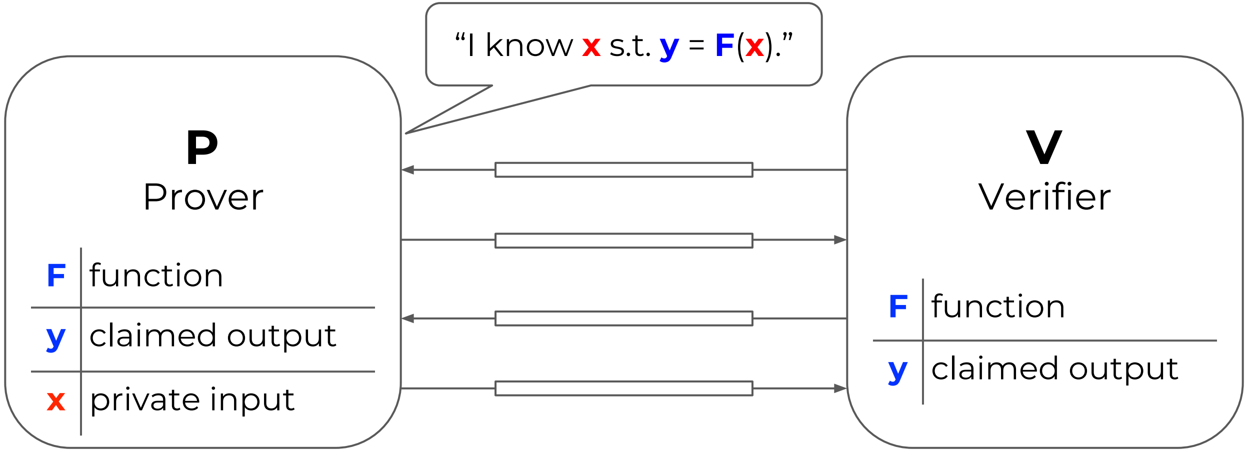
Privacy-preserving cryptographic proofs of computation integrity.



[GMR89]

Zero-Knowledge Proofs

Privacy-preserving cryptographic proofs of computation integrity.

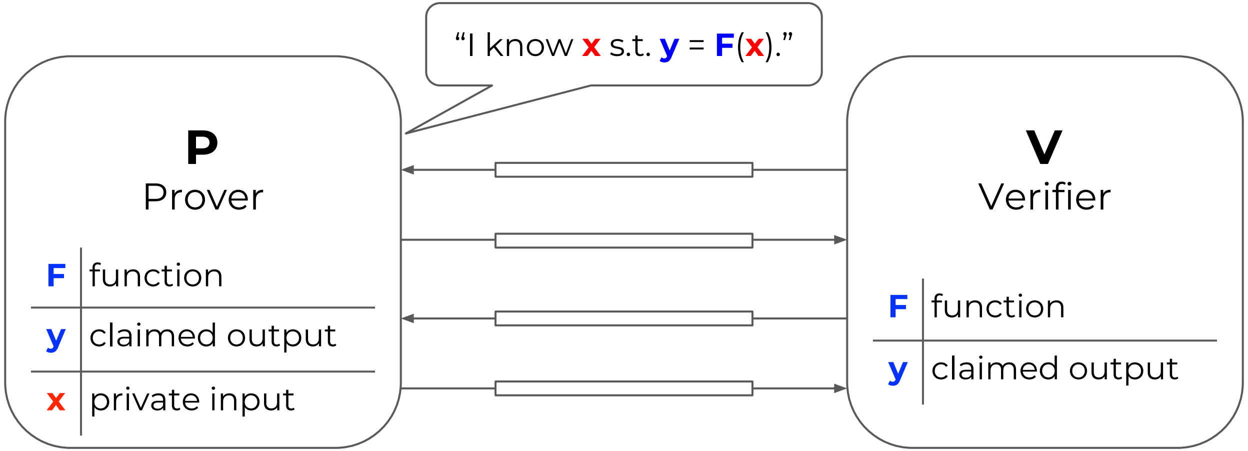


Completeness | $\exists\ \boldsymbol{x}: \boldsymbol{y} = \boldsymbol{F}(\boldsymbol{x}) \rightarrow \Pr[\mathbf{P}(\boldsymbol{F},\boldsymbol{y},\boldsymbol{x})$ convinces $\mathbf{V}(\boldsymbol{F},\boldsymbol{y})]=1$

[GMR89]

Zero-Knowledge Proofs

Privacy-preserving cryptographic proofs of computation integrity.

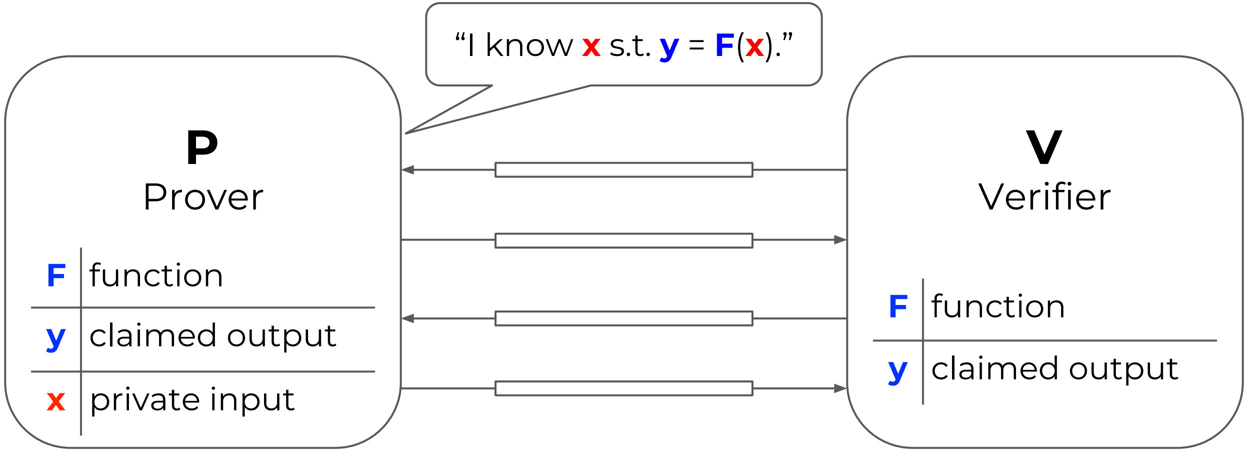


Completeness	$\exists\ x\!: y = F(x) \rightarrow \Pr[\mathbf{P}(F,y,x) \text{ convinces } \mathbf{V}(F,y)]=1$
Soundness	$\nexists\ x\!: y = F(x) \rightarrow \forall\ \mathbf{P'}\ \Pr[\mathbf{P'} \text{ convinces } \mathbf{V}(F,y)]\approx 0$

[GMR89]

Zero-Knowledge Proofs

Privacy-preserving cryptographic proofs of computation integrity.

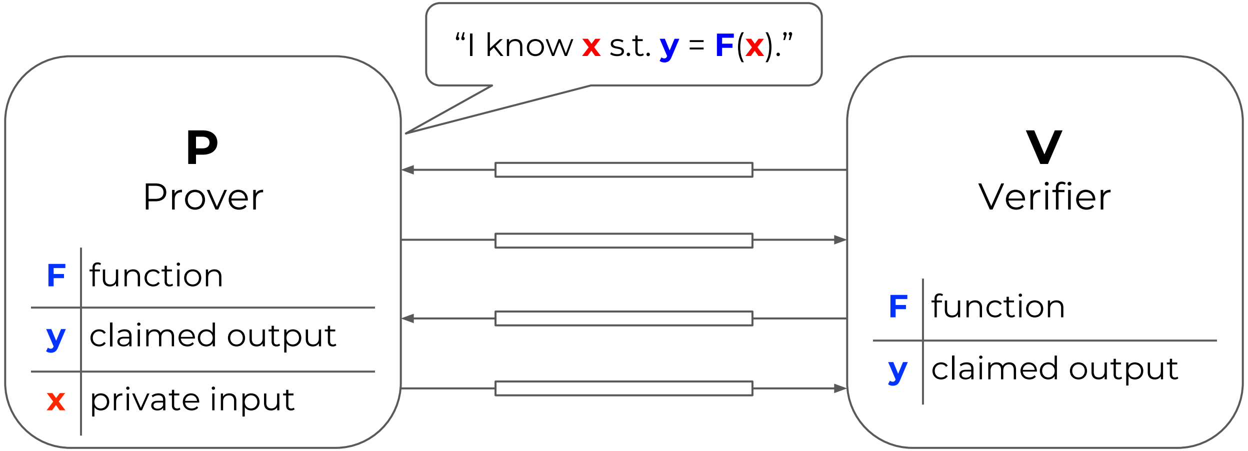


Completeness	$\exists\ x: y = F(x) \rightarrow \Pr[\mathbf{P}(F,y,x) \text{ convinces } \mathbf{V}(F,y)]=1$
Soundness	$\nexists\ x: y = F(x) \rightarrow \forall\ \mathbf{P'}\ \Pr[\mathbf{P'} \text{ convinces } \mathbf{V}(F,y)]\approx 0$
Zero-Knowledge	$\exists\ x: y = F(x) \rightarrow \underbrace{\mathbf{S}}_{\text{Simulator}}(F,y) \approx \text{view of } \mathbf{V}(F,y) \text{ after talking to } \mathbf{P}(F,y,x)$

[GMR89]

Zero-Knowledge Proofs

Privacy-preserving cryptographic proofs of computation integrity.



Powerful cryptographic primitive.

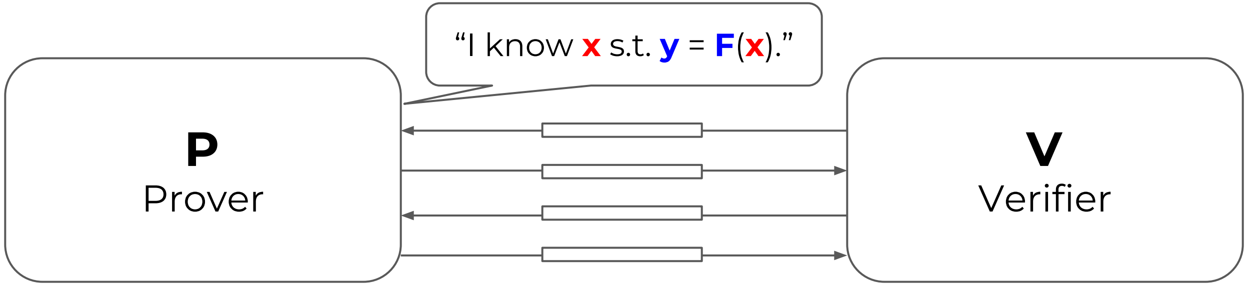
However, it is
interactive, not succinct, and has bad concrete efficiency.

Communication complexity & verification complexity are proportional to time(F)

Relies on PCPs (probabilistically checkable proofs)

Zero-Knowledge Succinct Proofs

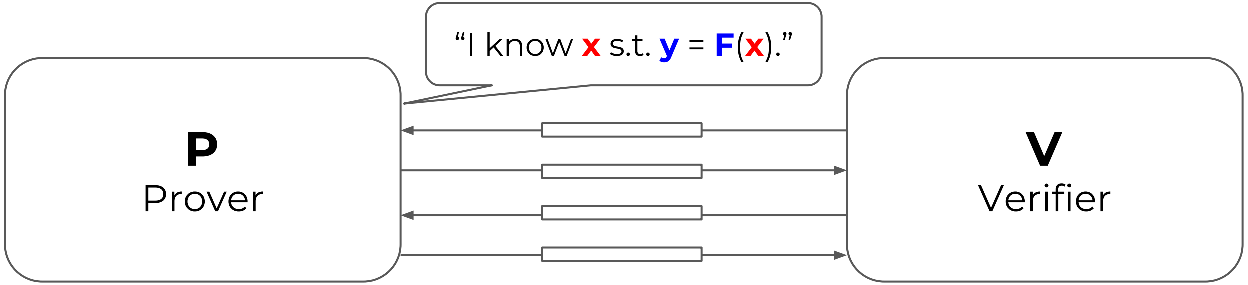
Privacy-preserving cryptographic **succinct** proofs of computation integrity.



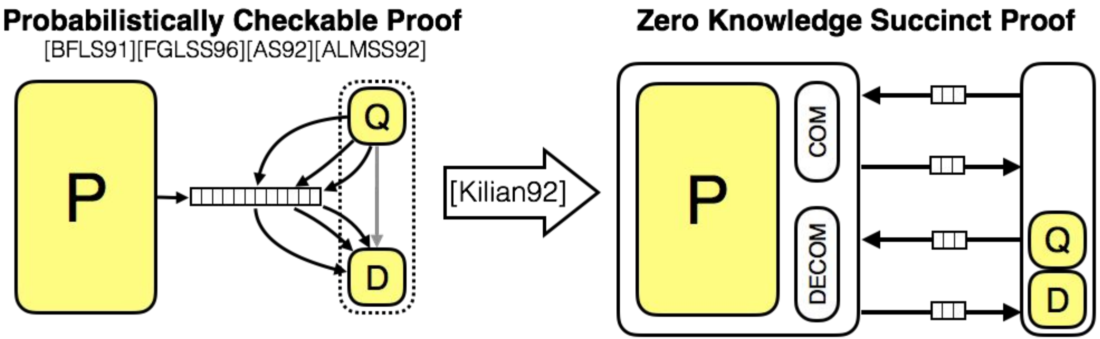
[Killian92]

Zero-Knowledge Succinct Proofs

Privacy-preserving cryptographic **succinct** proofs of computation integrity.



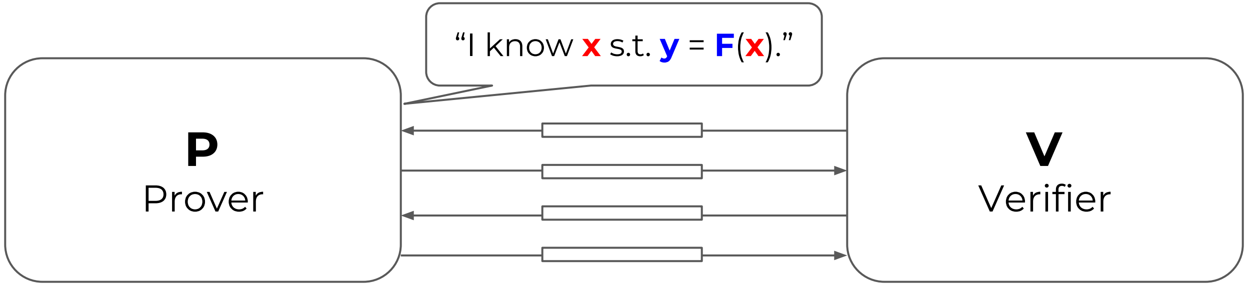
Achieving Succinctness



[Killian92]

Zero-Knowledge Succinct Proofs

Privacy-preserving cryptographic **succinct** proofs of computation integrity.

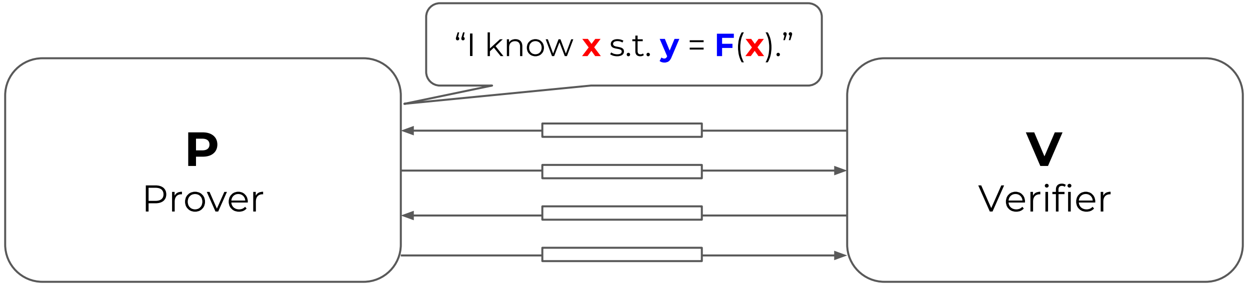


Completeness	$\exists\ x: y = F(x) \rightarrow \Pr[\mathbf{P}(F,y,x)$ convinces $\mathbf{V}(F,y)]=1$
Soundness	$\nexists\ x: y = F(x) \rightarrow \forall\ \mathbf{P'}\ \Pr[\mathbf{P'}$ convinces $\mathbf{V}(F,y)]\approx 0$
Zero-knowledge	$\exists\ x: y = F(x) \rightarrow \mathbf{S}(F,y) \approx$ view of $\mathbf{V}(F,y)$ after talking to $\mathbf{P}(F,y,x)$
Succinctness	$\mathbf{V}(F,y)$ runs in time proportional to $\lvert F\rvert+\lvert y\rvert$ (not F's runtime)

[Killian92]

Zero-Knowledge Succinct Proofs

Privacy-preserving cryptographic **succinct** proofs of computation integrity.



More powerful cryptographic primitive.

However, it is still

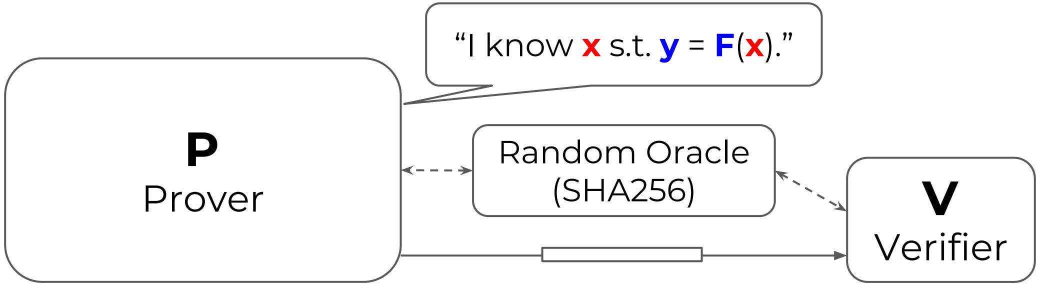
interactive, ~~not succinct~~, and has bad concrete efficiency.

Relies on PCPs (probabilistically checkable proofs)

[Micali94]

Zero-Knowledge Succinct Non-Interactive Proofs

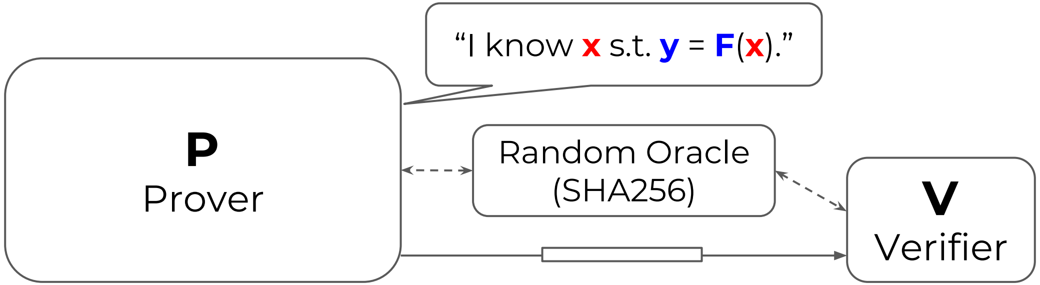
Privacy-preserving cryptographic succinct,
non-interactive proofs of computation integrity.



[Micali94]

Zero-Knowledge Succinct Non-Interactive Proofs

Privacy-preserving cryptographic succinct,
non-interactive proofs of computation integrity.

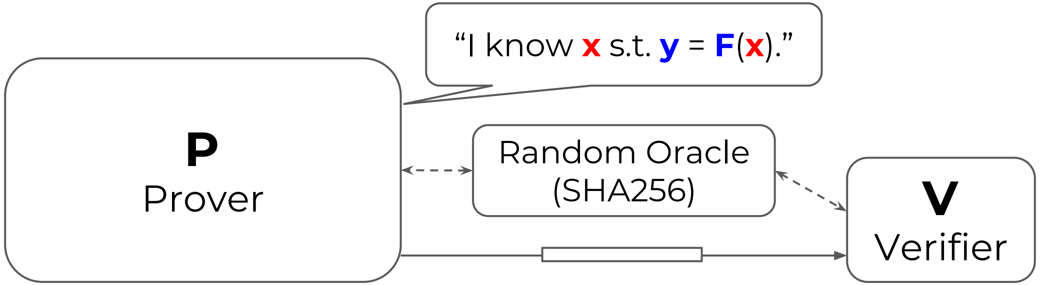


Zero-**K**nowledge **S**uccinct **N**on-interactive **Ar**gument of **K**nowledge

[Micali94]

Zero-Knowledge Succinct Non-Interactive Proofs

Privacy-preserving cryptographic succinct,
non-interactive proofs of computation integrity.



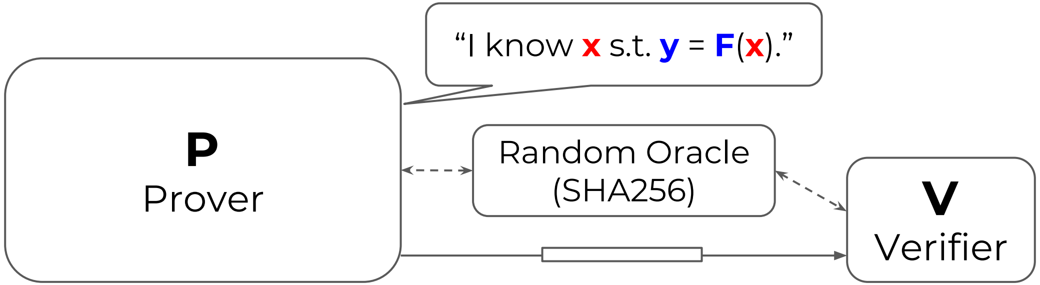
Zero-**K**nowledge **S**uccinct **N**on-interactive **Ar**gument of **K**nowledge

Zero-knowledge - *Proofs do not reveal the witness*

[Micali94]

Zero-Knowledge Succinct Non-Interactive Proofs

Privacy-preserving cryptographic succinct,
non-interactive proofs of computation integrity.



Zero-**K**nowledge **S**uccinct **N**on-interactive **Ar**gument of **K**nowledge

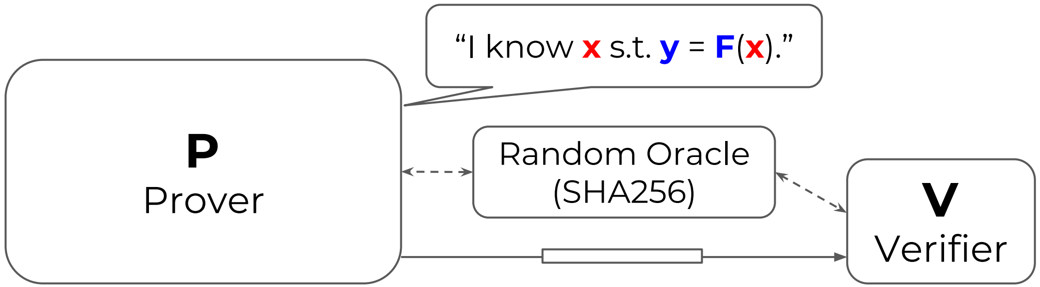
Zero-knowledge - *Proofs do not reveal the witness*

Succinct - *Proofs are short and verification is fast*

[Micali94]

Zero-Knowledge Succinct Non-Interactive Proofs

Privacy-preserving cryptographic succinct,
non-interactive proofs of computation integrity.



Zero-**K**nowledge **S**uccinct **N**on-interactive **Ar**gument of **K**nowledge

Zero-knowledge - *Proofs do not reveal the witness*

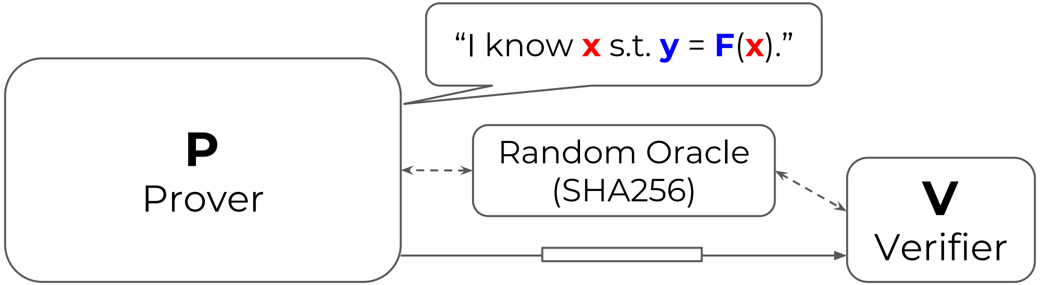
Succinct - *Proofs are short and verification is fast*

Non-interactive - *No interaction is necessary between the prover and the verifier*

[Micali94]

Zero-Knowledge Succinct Non-Interactive Proofs

Privacy-preserving cryptographic succinct,
non-interactive proofs of computation integrity.



Zero-**K**nowledge **S**uccinct **N**on-interactive **Ar**gument of **K**nowledge

Zero-knowledge - *Proofs do not reveal the witness*

Succinct - *Proofs are short and verification is fast*

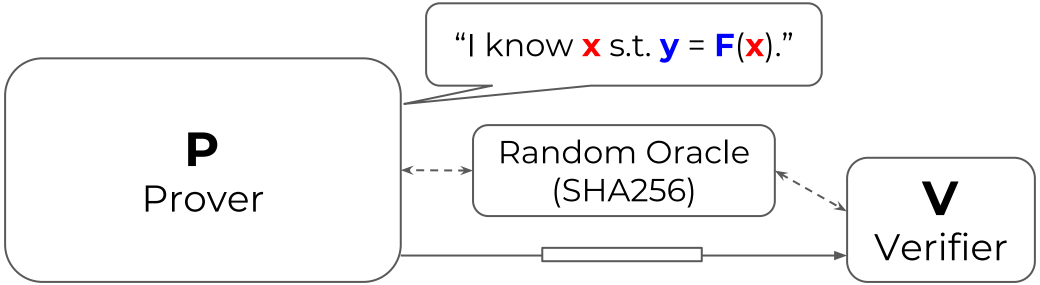
Non-interactive - *No interaction is necessary between the prover and the verifier*

Argument - *Soundness holds against a polynomially-bounded verifier*

[Micali94]

Zero-Knowledge Succinct Non-Interactive Proofs

Privacy-preserving cryptographic succinct,
non-interactive proofs of computation integrity.

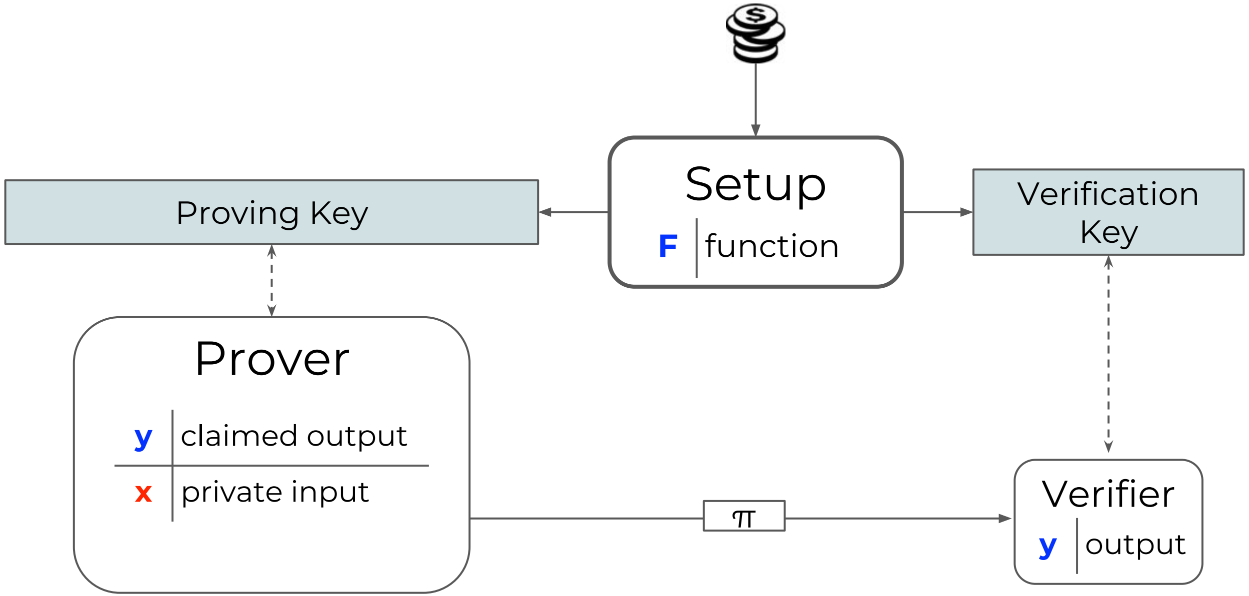


Even more powerful cryptographic primitive.

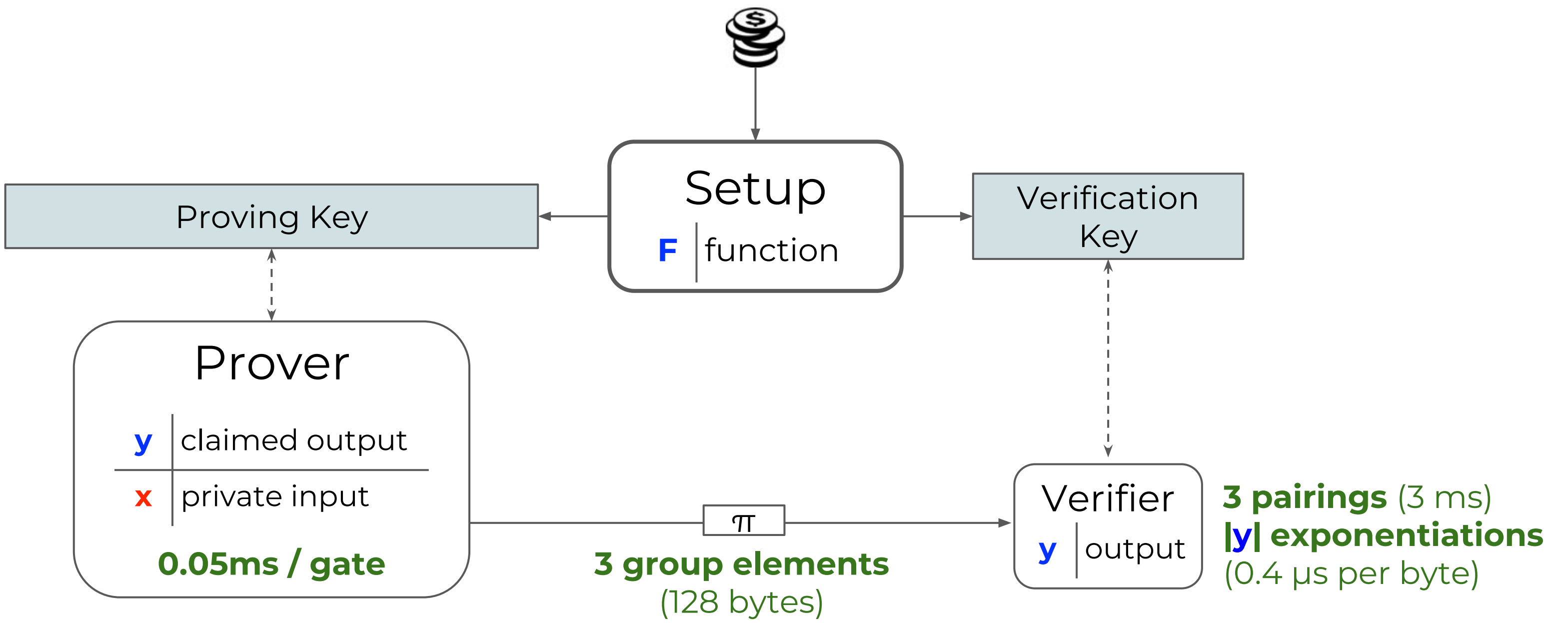
However, it is still
~~interactive~~, ~~not succinct~~, and has bad concrete efficiency.

Relies on PCPs (probabilistically checkable proofs)

Preprocessing zkSNARKs

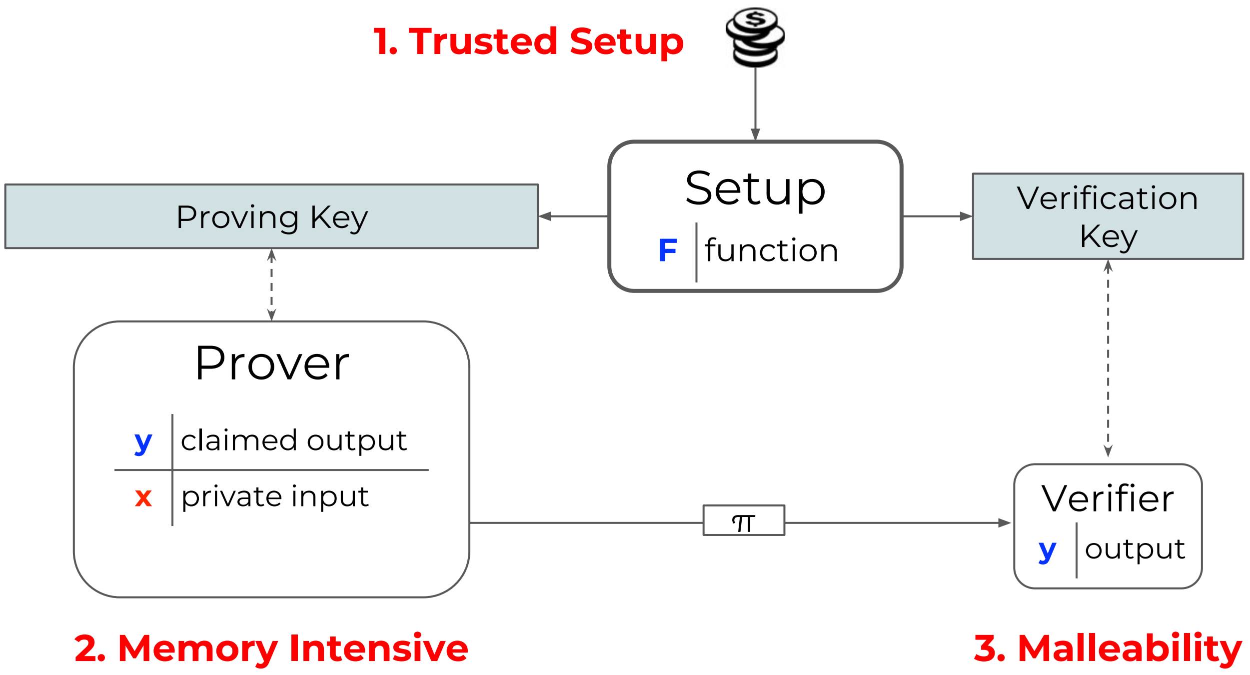


Preprocessing zkSNARKs

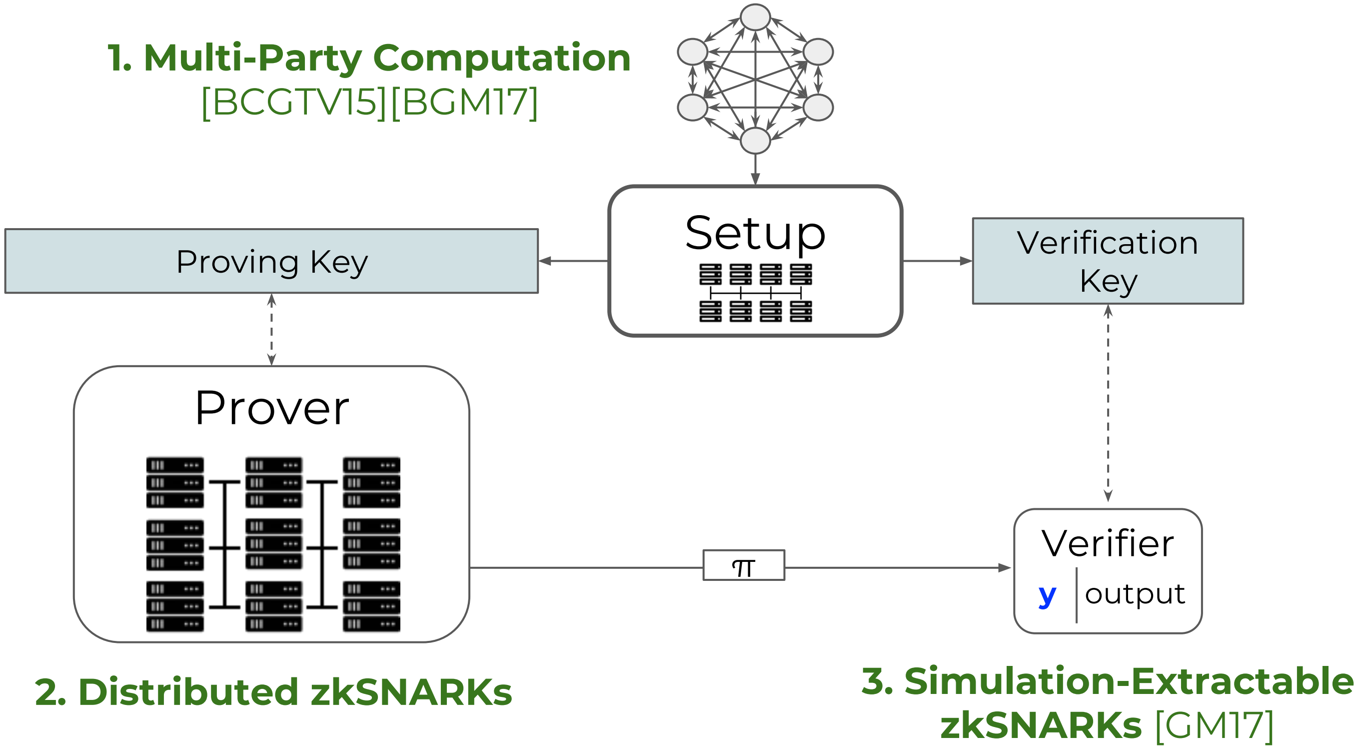


https://github.com/scipr-lab/libsnark/tree/master/src/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark

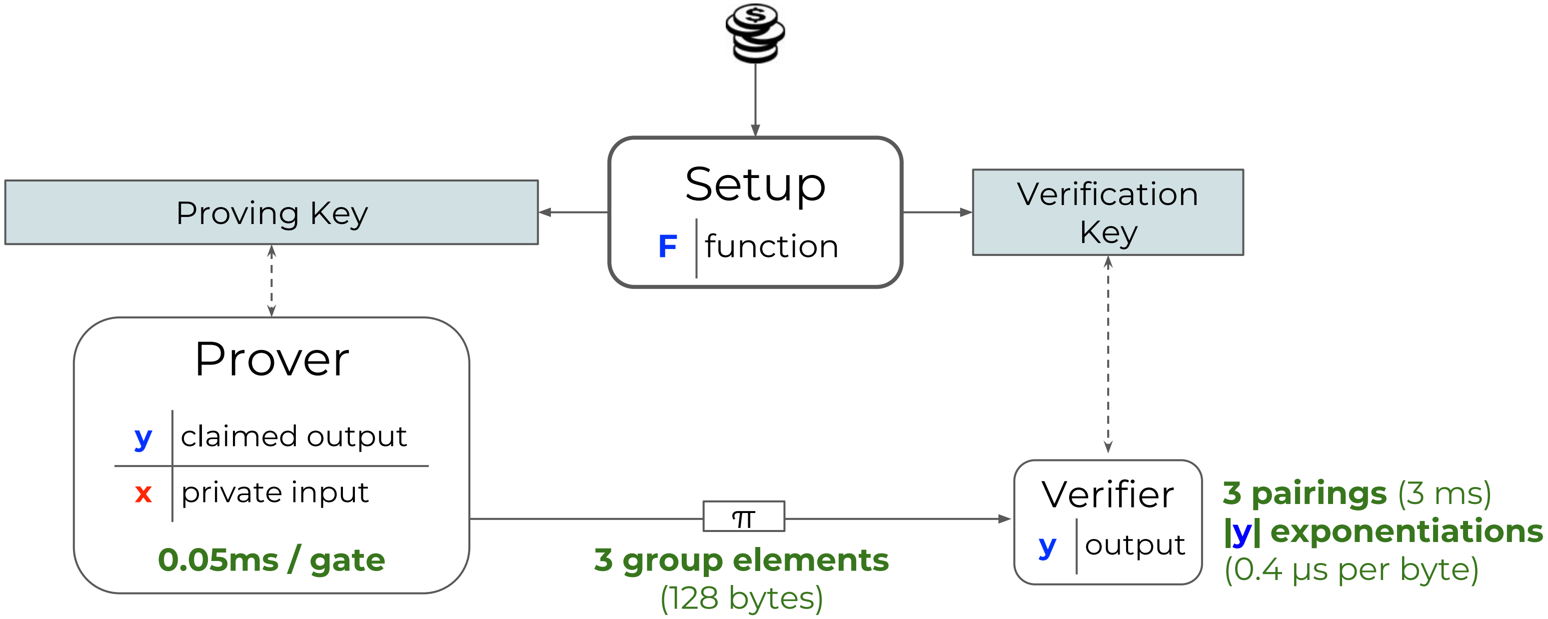
Preprocessing zkSNARKs



Preprocessing zkSNARKs

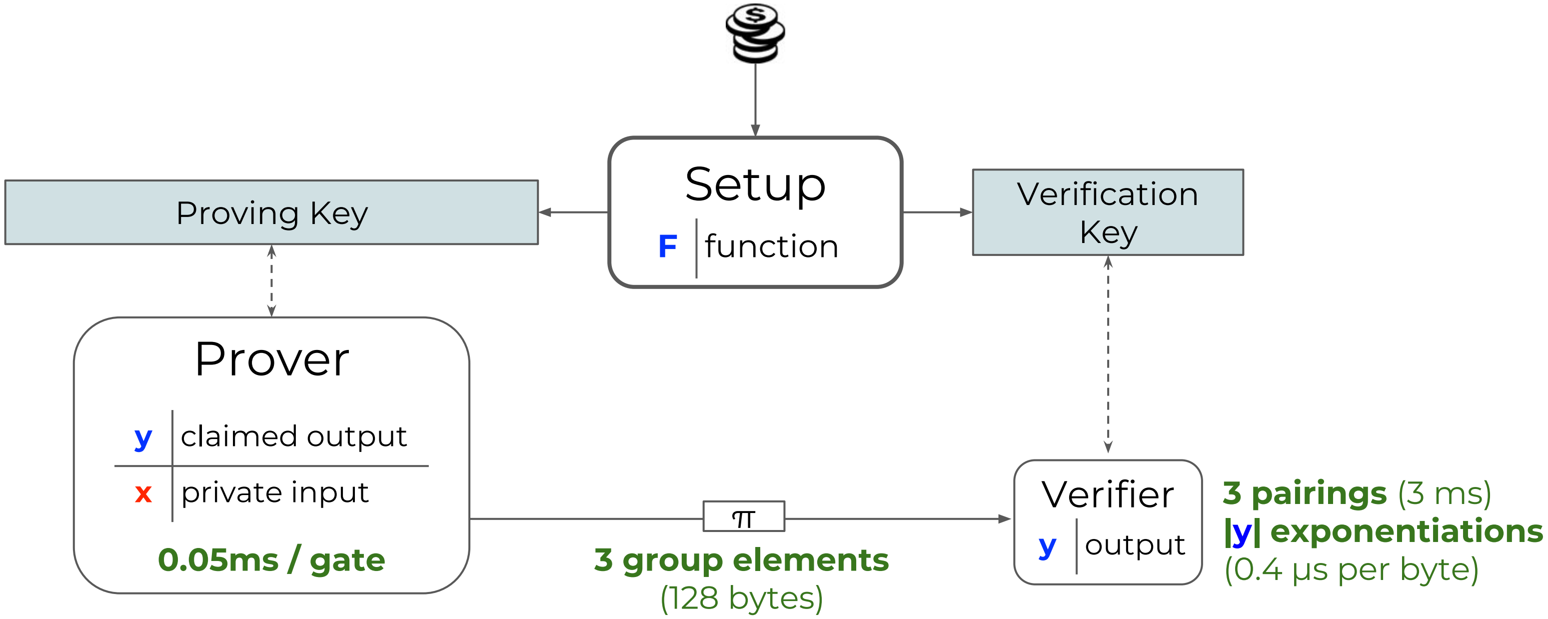


Preprocessing zkSNARKs



https://github.com/scipr-lab/libsnark/tree/master/src/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark

Pairing-based Preprocessing zkSNARKs



https://github.com/scipr-lab/libsnark/tree/master/src/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark

Prime-order bilinear map

G_1, G_2, G_T finite cyclic groups of prime order p

Prime-order bilinear map

G_1, G_2, G_T finite cyclic groups of prime order p

A pairing is a map,
$e: G_1 \times G_2 \rightarrow G_T$

Prime-order bilinear map

G_1, G_2, G_T finite cyclic groups of prime order p

A pairing is a map,
$e: G_1 \times G_2 \rightarrow G_T$

Given g in G_1 and h in G_2, a bilinear map ensures
$e(g^a, h^b) = e(g, h)^{ab}$.

Prime-order bilinear map

G_1, G_2, G_T finite cyclic groups of prime order p

A pairing is a map,
$e: G_1 \times G_2 \rightarrow G_T$

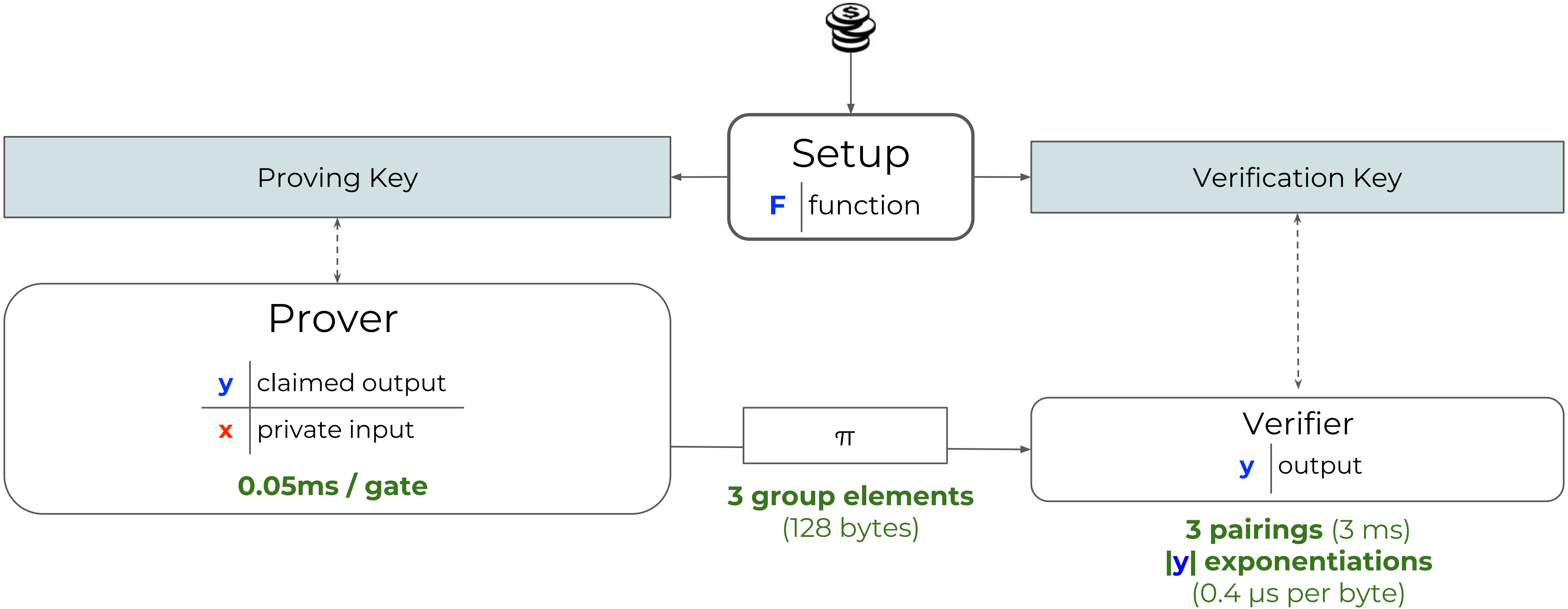
Given g in G_1 and h in G_2, a bilinear map ensures
$e(g^a, h^b) = e(g, h)^{ab}$.

Notation:

$[a]_1 = g^a$ $[b]_2 = h^b$ $[c]_T = e(g, h)^c$

[Groth16]

Pairing-based Preprocessing zkSNARKs



https://github.com/scipr-lab/libsnark/tree/master/src/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark

[Groth16]

Pairing-based Preprocessing zkSNARKs

$$\vec{\sigma}_1 = \left(\alpha, \beta, \delta, \{x^i\}, \left\{\frac{x^i t(x)}{\delta}\right\}, \left\{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\gamma}\right\}_{i\leq \ell}, \left\{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}\right\}_{i>\ell}\right)$$

Setup

$$\sigma = (\vec{\sigma}_1, \vec{\sigma}_2)$$

$$\vec{\sigma}_2 = \left(\alpha, \beta, \gamma, \delta, \{x^i\}, \left\{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\gamma}\right\}_{i\leq \ell}\right)$$

Prover

y	claimed output
x	private input

0.05ms / gate

π

3 group elements
(128 bytes)

Verifier

y	output

3 pairings (3 ms)
|y| exponentiations
(0.4 µs per byte)

https://github.com/scipr-lab/libsnark/tree/master/src/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark

[Groth16]

Pairing-based Preprocessing zkSNARKs

$$\vec{\sigma}_1 = \left(\alpha, \beta, \delta, \{x^i\}, \left\{\frac{x^i t(x)}{\delta}\right\}, \left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right\}_{i \le \ell}, \left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right\}_{i > \ell}\right)$$

Setup

$$\sigma = (\vec{\sigma}_1, \vec{\sigma}_2)$$

$$\vec{\sigma}_2 = \left(\alpha, \beta, \gamma, \delta, \{x^i\}, \left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right\}_{i \le \ell}\right)$$

Prover

$$A = \alpha + \sum a_i u_i(x) + r\delta \qquad B = \beta + \sum a_i v_i(x) + s\delta$$

$$C = \sum_{i > \ell} a_i \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta} + \frac{h(x)t(x)}{\delta} + As + rB - rs\delta$$

0.05ms / gate

π

3 group elements
(128 bytes)

Verifier

y | output

3 pairings (3 ms)
|y| exponentiations
(0.4 µs per byte)

https://github.com/scipr-lab/libsnark/tree/master/src/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark

[Groth16]

Pairing-based Preprocessing zkSNARKs

Setup

$\sigma = (\vec{\sigma}_1, \vec{\sigma}_2)$

$$\vec{\sigma}_1 = \left(\alpha, \beta, \delta, \{x^i\}, \left\{\frac{x^i t(x)}{\delta}\right\}, \left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right\}_{i \le \ell}, \left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right\}_{i > \ell}\right)$$

$$\vec{\sigma}_2 = \left(\alpha, \beta, \gamma, \delta, \{x^i\}, \left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right\}_{i \le \ell}\right)$$

Prover

$$A = \alpha + \sum a_i u_i(x) + r\delta \qquad B = \beta + \sum a_i v_i(x) + s\delta$$

$$C = \sum_{i > \ell} a_i \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta} + \frac{h(x)t(x)}{\delta} + As + rB - rs\delta$$

0.05ms / gate

$\pi = ([A]_1, [C]_1, [B]_2)$

3 group elements
(128 bytes)

Verifier

y | output

3 pairings (3 ms)
|y| exponentiations
(0.4 μs per byte)

https://github.com/scipr-lab/libsnark/tree/master/src/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark

Pairing-based Preprocessing zkSNARKs

Setup

$$\sigma = (\vec{\sigma}_1, \vec{\sigma}_2)$$

$$\vec{\sigma}_1 = \left(\alpha, \beta, \delta, \{x^i\}, \left\{\frac{x^i t(x)}{\delta}\right\}, \left\{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\gamma}\right\}_{i\le\ell}, \left\{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\delta}\right\}_{i>\ell}\right)$$

$$\vec{\sigma}_2 = \left(\alpha, \beta, \gamma, \delta, \{x^i\}, \left\{\frac{\beta u_i(x)+\alpha v_i(x)+w_i(x)}{\gamma}\right\}_{i\le\ell}\right)$$

Prover

$$A = \alpha + \sum a_i u_i(x) + r\delta \qquad B = \beta + \sum a_i v_i(x) + s\delta$$

$$C = \sum_{i>\ell} a_i \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta} + \frac{h(x)t(x)}{\delta} + As + rB - rs\delta$$

0.05ms / gate

$$\pi = ([A]_1, [C]_1, [B]_2)$$

3 group elements
(128 bytes)

Verifier

$$[A]_1 \cdot [B]_2 = [\alpha]_1 \cdot [\beta]_2 + \sum_{i=0}^{\ell} a_i \left[\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right]_1 \cdot [\gamma]_2 + [C]_1 \cdot [\delta]_2$$

3 pairings (3 ms)
|y| exponentiations
(0.4 µs per byte)

https://github.com/scipr-lab/libsnark/tree/master/src/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark

[Groth16]

Pairing-based Preprocessing zkSNARKs

Efficiency

Proof size: $2G_1$, $1G_2$

Prover: $m + 3n\ E_1, n\ E_2$

Verifier: $\ell\ E_1 + 3P$

Setup

$$\sigma = (\vec{\sigma}_1, \vec{\sigma}_2)$$

$$\vec{\sigma}_1 = \left(\alpha, \beta, \delta, \{x^i\}, \left\{\frac{x^i t(x)}{\delta}\right\}, \left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right\}_{i \le \ell}, \left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta}\right\}_{i > \ell}\right)$$

$$\vec{\sigma}_2 = \left(\alpha, \beta, \gamma, \delta, \{x^i\}, \left\{\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right\}_{i \le \ell}\right)$$

Prover

$$A = \alpha + \sum a_i u_i(x) + r\delta \qquad B = \beta + \sum a_i v_i(x) + s\delta$$

$$C = \sum_{i > \ell} a_i \frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\delta} + \frac{h(x)t(x)}{\delta} + As + rB - rs\delta$$

0.05ms / gate

$$\pi = ([A]_1, [C]_1, [B]_2)$$

3 group elements
(128 bytes)

Verifier

$$[A]_1 \cdot [B]_2 = [\alpha]_1 \cdot [\beta]_2 + \sum_{i=0}^{\ell} a_i \left[\frac{\beta u_i(x) + \alpha v_i(x) + w_i(x)}{\gamma}\right]_1 \cdot [\gamma]_2 + [C]_1 \cdot [\delta]_2$$

3 pairings (3 ms)
|y| exponentiations
(0.4 μs per byte)

https://github.com/scipr-lab/libsnark/tree/master/src/zk_proof_systems/ppzksnark/r1cs_gg_ppzksnark

`libsnark`: a C++ library for zkSNARKs

(libsnark.org)

`libsnark`: a C++ library for zkSNARKs

(libsnark.org)

Algebraic Core			
	Polynomial Interpolation / Evaluation	Fast Fourier Transforms in Finite Fields	Lagrange-Coefficient Computations
	Finite Field Arithmetic	Bilinear Group Arithmetic	Fixed & Variable Base Multi-Exponentiation

libsnark: a C++ library for zkSNARKs

(libsnark.org)

- **libff**: a C++ library for Finite Fields and Elliptic Curves

(github.com/scipr-lab/libff)

`libsnark`: a C++ library for zkSNARKs

(libsnark.org)

- **`libfqfft`**: a C++ library for FFTs in Finite Fields

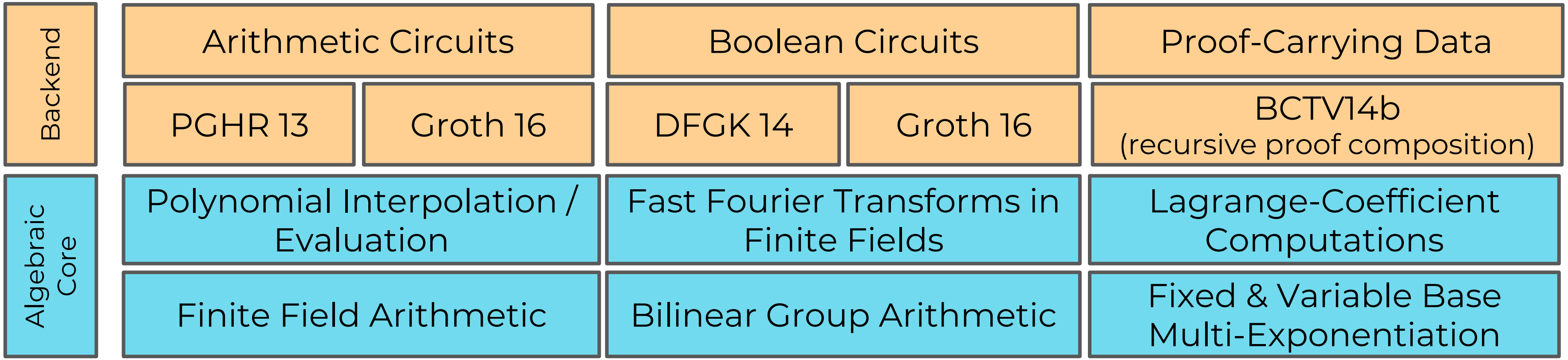
  (github.com/scipr-lab/libfqfft)

- **`libff`**: a C++ library for Finite Fields and Elliptic Curves

  (github.com/scipr-lab/libff)

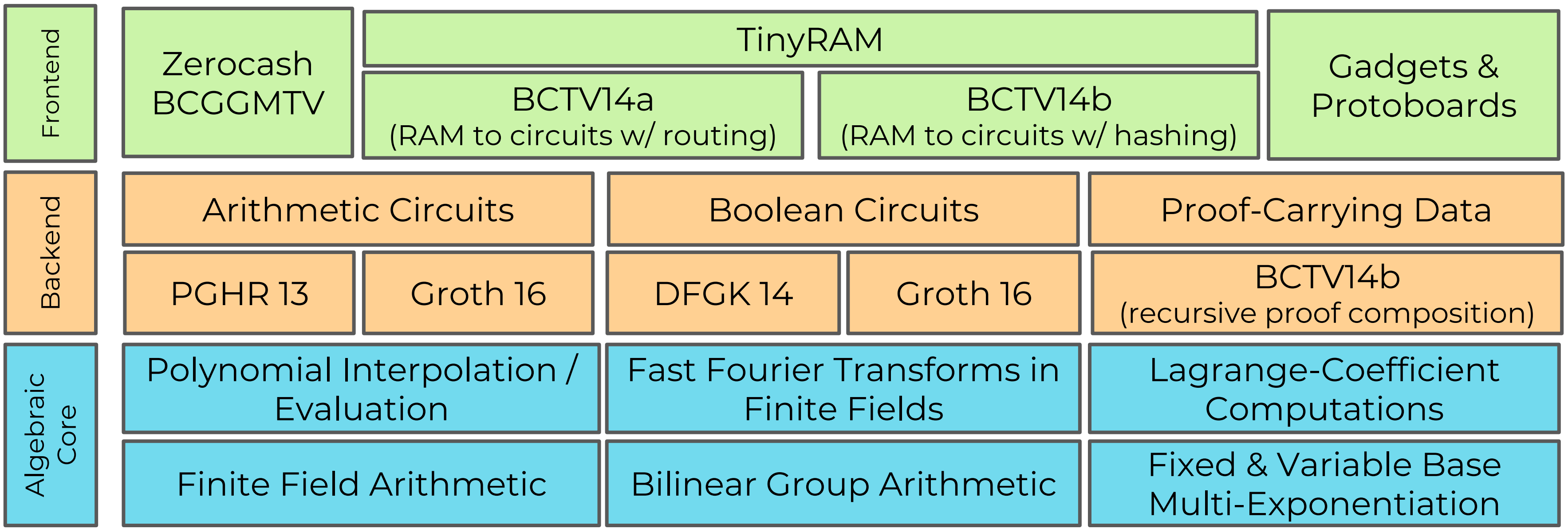
libsnark: a C++ library for zkSNARKs

(libsnark.org)



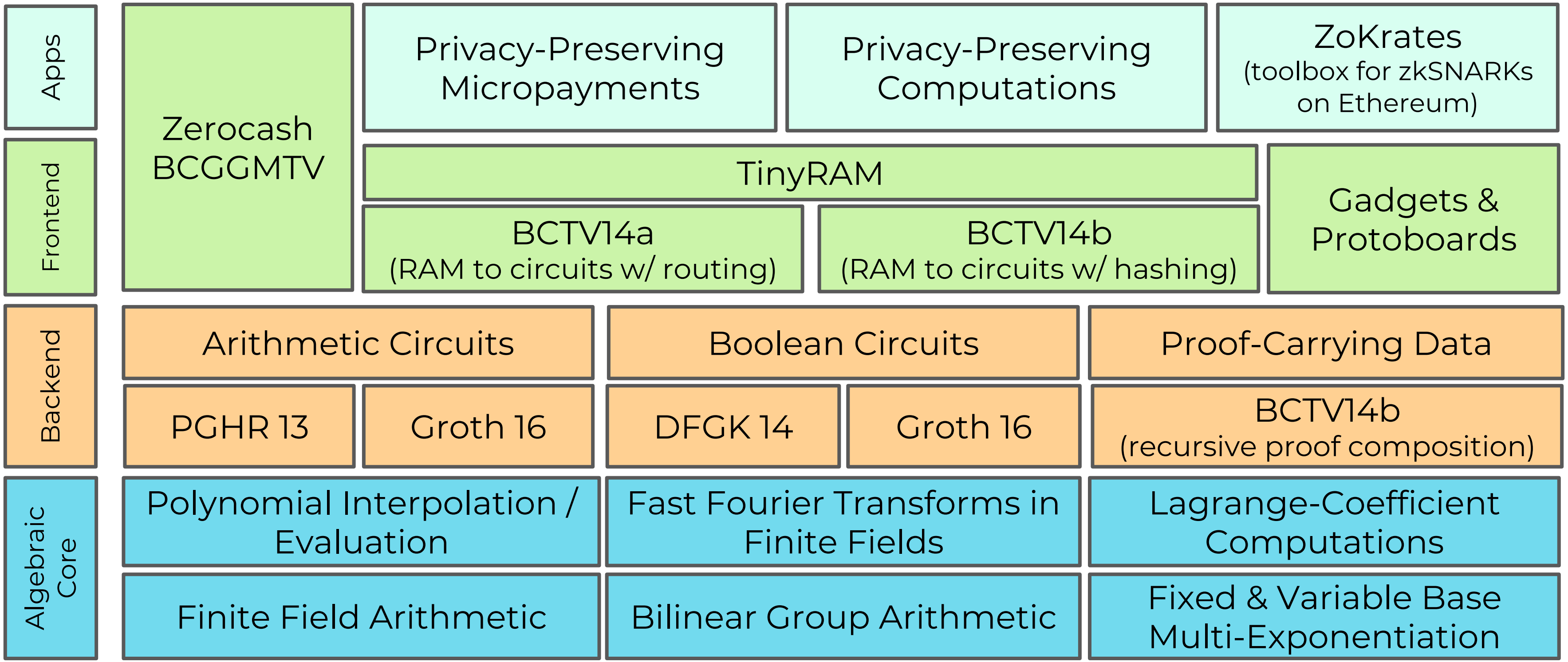
libsnark: a C++ library for zkSNARKs

(libsnark.org)



`libsnark`: a C++ library for zkSNARKs

(libsnark.org)



Developing on `libsnark`

`libff`

Choice of elliptic curve implementations

Barreto-Naehrig curve
(~128 bits of security)

Edwards curve
(~80 bits of security)

MNT4 curve
(~80 bits of security)

MNT6 curve
(~80 bits of security)

Developing on **`libsnark`**

`libff`

Choice of elliptic curve implementations

Barreto-Naehrig curve
(~128 bits of security)

Edwards curve
(~80 bits of security)

MNT4 curve
(~80 bits of security)

MNT6 curve
(~80 bits of security)

Bilinear Group Arithmetic

Developing on `libsnark`

`libff`

Choice of elliptic curve implementations

Barreto-Naehrig curve
(~128 bits of security)

Edwards curve
(~80 bits of security)

MNT4 curve
(~80 bits of security)

MNT6 curve
(~80 bits of security)

Bilinear Group Arithmetic

`libfqfft`

Choice of evaluation domains

Standard radix-2
(size $m = 2^k$ & m-th roots of unity)

Extended radix-2
(size $m = 2^{k+1}$ & m-th roots of unity, union a coset of these roots)

Step radix-2
(size $m = 2^k + 2^r$ & 2^k-th roots of unity, union a coset of 2^r-th roots of unity)

Geometric Sequence
(size m & sampled points of $a_n = r^{(n-1)}$)

Arithmetic Sequence
(size m & sampled points of $a_i = a_1 + (i - 1)*d$)

Developing on `libsnark`

`libff`

Choice of elliptic curve implementations

Barreto-Naehrig curve
(~128 bits of security)

Edwards curve
(~80 bits of security)

MNT4 curve
(~80 bits of security)

MNT6 curve
(~80 bits of security)

Bilinear Group Arithmetic

`libfqfft`

Choice of evaluation domains

Standard radix-2
(size $m = 2^k$ & m-th roots of unity)

Extended radix-2
(size $m = 2^{k+1}$ & m-th roots of unity, union a coset of these roots)

Step radix-2
(size $m = 2^k + 2^r$ & 2^k-th roots of unity, union a coset of 2^r-th roots of unity)

Geometric Sequence
(size m & sampled points of $a_n = r^{(n-1)}$)

Arithmetic Sequence
(size m & sampled points of $a_i = a_1 + (i - 1)*d$)

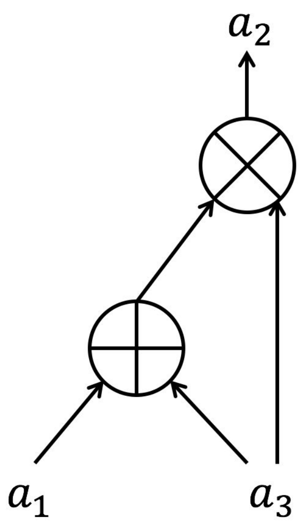
Polynomial Interpolation / Evaluation

Developing on **`libsnark`**

`libsnark` arithmetic circuits

Developing on **`libsnark`**

`libsnark` arithmetic circuits



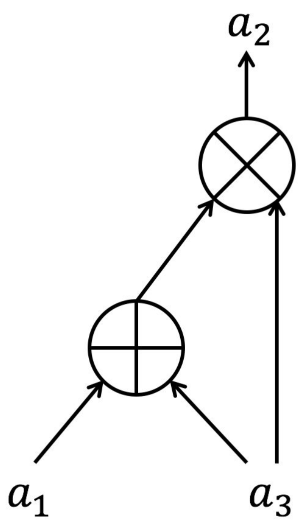
Write as quadratic equation over $\boldsymbol{F}_p$

$$(a_1 + a_3) \cdot a_3 = a_2$$

Arithmetic Circuits

Developing on **`libsnark`**

`libsnark` arithmetic circuits



Write as quadratic equation over $\boldsymbol{F}_p$

$$(a_1 + a_3) \cdot a_3 = a_2$$

Arithmetic circuits can be expressed in the form,

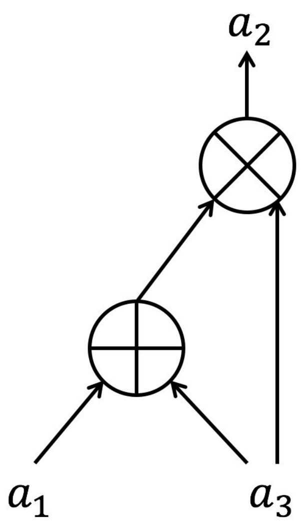
$$\sum a_i u_i \cdot \sum a_i v_i = \sum a_i w_i$$

over variables $a_1, \ldots, a_m$, where by convention $a_0 = 1$.

Arithmetic Circuits

Developing on **`libsnark`**

`libsnark` arithmetic circuits



Write as quadratic equation over $\boldsymbol{F}_p$

$$(a_1 + a_3) \cdot a_3 = a_2$$

Arithmetic circuits can be expressed in the form,

$$\sum a_i u_i \cdot \sum a_i v_i = \sum a_i w_i$$

over variables $a_1, \dots, a_m$, where by convention $a_0 = 1$.

An arithmetic circuit defines an *NP-language* with statements $(a_1, \dots, a_\ell)$ and witnesses $(a_{\ell+1}, \dots, a_m)$.

Arithmetic Circuits

Developing on `libsnark`

`libsnark` gadgetlib1 & gadgetlib2

Gadgets &
Protoboards

Developing on **`libsnark`**

`libsnark` gadgetlib1 & gadgetlib2

Low-level libraries which expose all features of the preprocessing zkSNARK

Gadgets & Protoboards

Developing on **`libsnark`**

`libsnark` gadgetlib1 & gadgetlib2

Low-level libraries which expose all features of the preprocessing zkSNARK

Finite Fields

Elliptic Curves

Pairings

Multi-Exponentiation

Hash Functions *(SHA256)*

Merkle Trees *(Authentication Paths)*

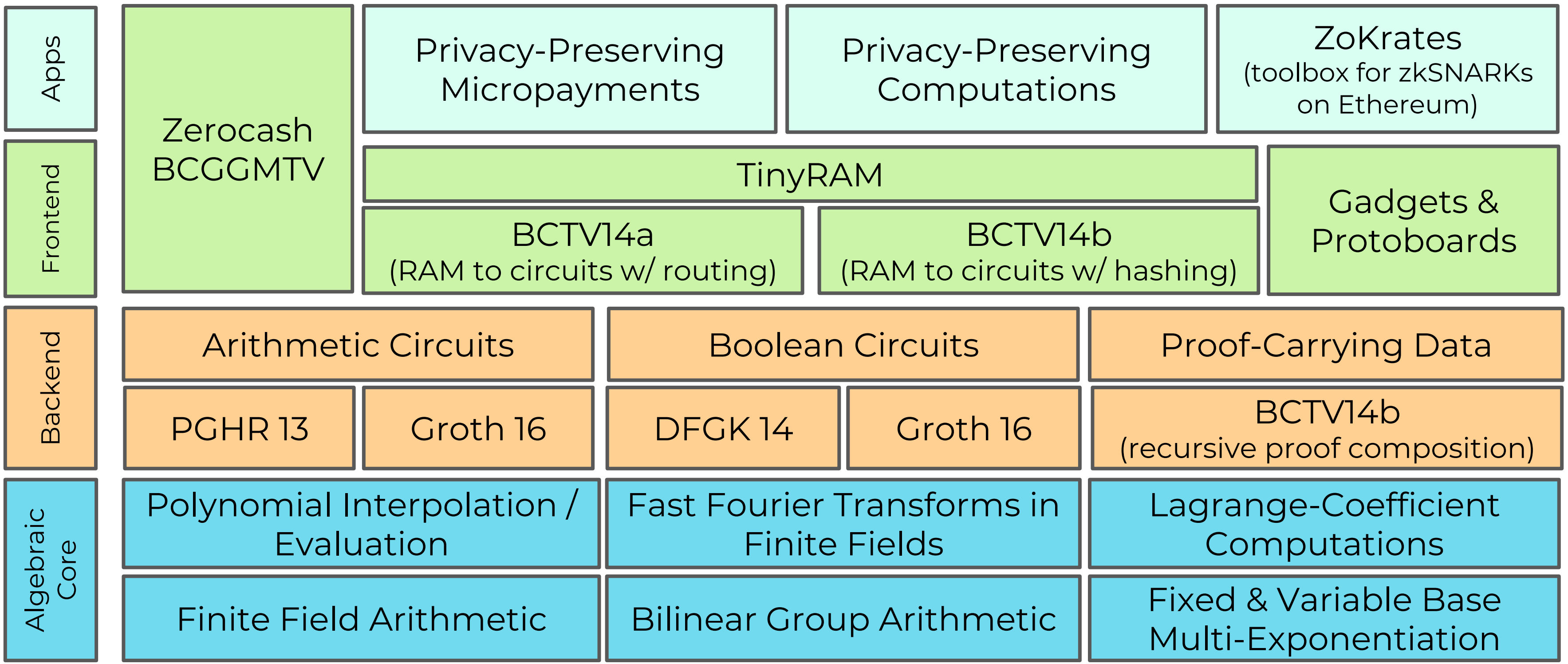
CPU Checkers *(TinyRAM)*

zkSNARK verifier

Gadgets & Protoboards

libsnark: a C++ library for zkSNARKs

(libsnark.org)

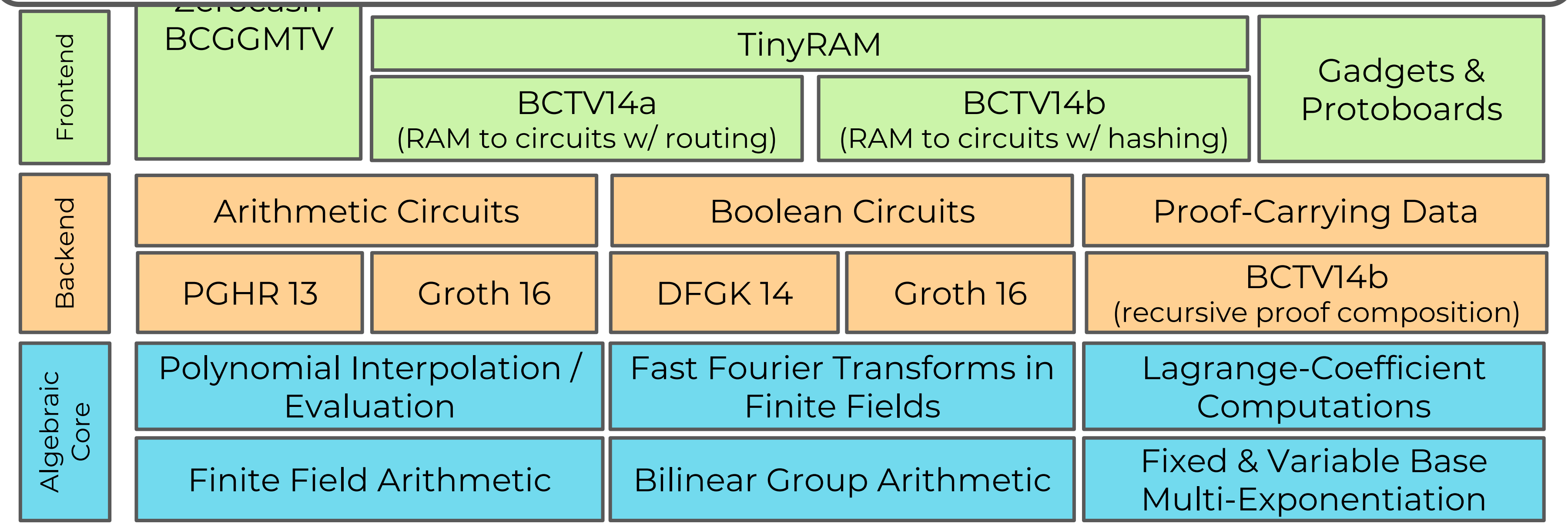


`libsnark`: a C++ library for zkSNARKs

(libsnark.org)

To get started:

`github.com/howardwu/libsnark-tutorial`



`libsnark`: a C++ library for zkSNARKs

(libsnark.org)

To get started:

`github.com/howardwu/libsnark-tutorial`

To get involved:

`libsnark.org/get-involved`

Backend	Arithmetic Circuits	Boolean Circuits	Proof-Carrying Data
Backend	PGHR 13 / Groth 16	DFGK 14 / Groth 16	BCTV14b (recursive proof composition)
Algebraic Core	Polynomial Interpolation / Evaluation	Fast Fourier Transforms in Finite Fields	Lagrange-Coefficient Computations
Algebraic Core	Finite Field Arithmetic	Bilinear Group Arithmetic	Fixed & Variable Base Multi-Exponentiation

libsnark: a C++ library for zkSNARKs

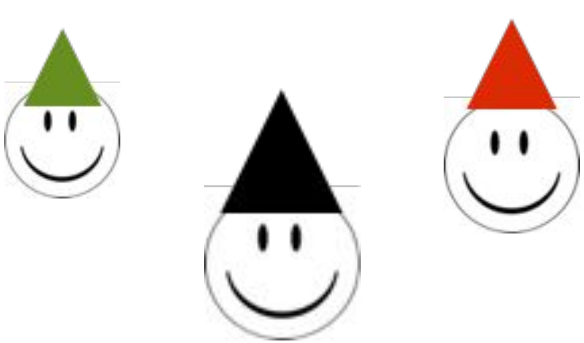
(libsnark.org)

To get started:

github.com/howardwu/libsnark-tutorial

To get involved:

libsnark.org/get-involved



Thanks!