Introduction to (zk)SNARKs

Le Trong Anh Tu

Nanyang Technological University, Singapore

July 05, 2025

Outline

Outline

Let's prove a statement

Untrusted prover P convinces V to know w satisfying some property.

- A preimage w for an output y of a cryptographic hash function h, i.e. $h(w) = y$.

$$\langle P(h, y; w), V(h, y) \rangle$$

- A private key w corresponding to a public key y for discrete log, i.e. $y = g^w \pmod{p}$ where g is a generator of a group.

$$\langle P(y, g, p; w), V(y, g, p) \rangle$$

Let's prove a statement

Untrusted prover P convinces V to know w satisfying some property.

- A preimage w for an output y of a cryptographic hash function h, i.e. $h(w) = y$.
$$\langle P(h, y; w), V(h, y)\rangle$$
- A private key w corresponding to a public key y for discrete log, i.e. $y = g^w \pmod p$ where g is a generator of a group.
$$\langle P(y, g, p; w), V(y, g, p)\rangle$$

Trivial proof: P sends w to V, and V checks if it satisfies the property.

- We don't pay attention to the privacy $\rightarrow$ Do not have the term "zk".
- V only "check", not "recompute" from witness w.

$\Rightarrow$ A SNARK proof achieves the same, but better cost to V.

SNARKs: What are they? (Informal)

SNARK = **<u>S</u>**uccinct **<u>N</u>**on-interactive **<u>AR</u>**gument of **<u>K</u>**nowledge.

- The proof π must be shorter than the witness w.
- Checking π is faster than checking $w \rightarrow$ “work-saving” for V.

SNARKs: What are they? (Informal)

SNARK = <u>**S**</u>uccinct <u>**N**</u>on-interactive <u>**AR**</u>gument of <u>**K**</u>nowledge.

- The proof π must be shorter than the witness w.
- Checking π is faster than checking w $\rightarrow$ “work-saving” for V.

In some papers, “succinct” might mean constant, logarithm, or sublinear with the length of the witness w (and, also the statement).

SNARKs: What are they? (Informal)

SNARK = **S**uccinct **N**on-interactive **AR**gument of **K**nowledge.

- The proof π must be shorter than the witness w.
- Checking π is faster than checking $w \rightarrow$ "work-saving" for V.

In some papers, "succinct" might mean constant, logarithm, or sublinear with the length of the witness w (and, also the statement).

In practice, proof size is hundred bytes or several KBs, and checking SNARK proof π is around 1 second.

SNARKs: What are they? (Informal)

SNARK = **S**uccinct **N**on-interactive **AR**gument of **K**nowledge.

- The proof π must be shorter than the witness w.
- Checking π is faster than checking $w \rightarrow$ "work-saving" for V.

In some papers, "succinct" might mean constant, logarithm, or sublinear with the length of the witness w (and, also the statement).

In practice, proof size is hundred bytes or several KBs, and checking SNARK proof π is around 1 second.

zk-SNARK: the proof π reveals nothing about m (privacy for m).

- In this talk, we only mention about SNARKs, not zk-SNARKs.

Trivial proof is not a SNARK proof

Trivial proof: P sends w to V, and V checks if it satisfies the property.

- w might be long, and not shorter than w (straightforward).
- Checking w is just "recompute", not "verify".
- (Optional) w might be secret, P might not want to reveal w to V.

Applications of SNARKs (Mainly in blockchain)

1. Outsourcing computation (no need for zero knowledge).
   - **zkRollup:** An off-chain service processes a batch of transactions by generating a succinct proof that the transactions were processed correctly. This proof is then sent to the L1 chain for verification.
   - **zkBridge:** Provide a SNARK proof that enables transfer of assets from one chain to another.

Applications of SNARKs (Mainly in blockchain)

1. Outsourcing computation (no need for zero knowledge).
   - **zkRollup:** An off-chain service processes a batch of transactions by generating a succinct proof that the transactions were processed correctly. This proof is then sent to the L1 chain for verification.
   - **zkBridge:** Provide a SNARK proof that enables transfer of assets from one chain to another.
2. Verifying private information (need zero knowledge)
   - **Tornado, Zcash, ...:** Prove a private transaction is valid on a public blockchain.
   - **Espresso:** Generate proof that a private transaction is compliant with banking regulations.

And non-blockchain applications

Blockchain drives the development of SNARKs, but their advancement has led to many non-blockchain applications (e.g. zkML, ...).

And non-blockchain applications

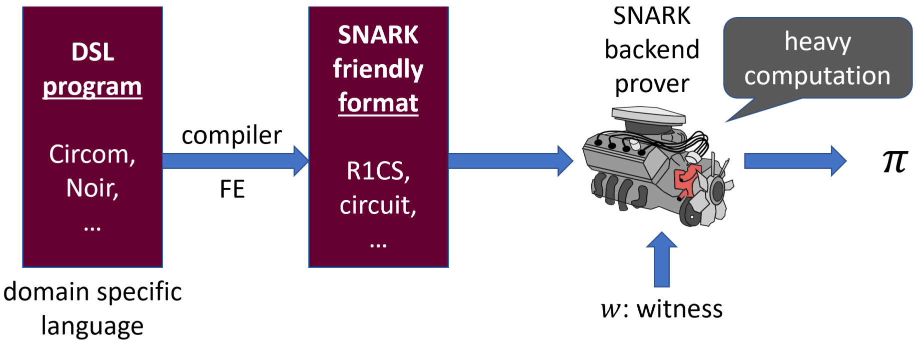
Blockchain drives the development of SNARKs, but their advancement has led to many non-blockchain applications (e.g. zkML, ...).

Zero-knowledge machine learning (zkML):

- Enable the verification of ML model computations while preserving model privacy.
- The inference process is treated as a function F, where either the model input or the model weights serve as the secret input.

Outline

General-Purpose SNARKs: Standard Paradigm



In practice, a computer program written in a high-level programming language (Java, Python, ...) is preferred.

General-Purpose SNARKs: Standard Paradigm

Computer program:

- Hash pre-image example: Program takes w and y as inputs, applies a hash function h to w, and checks that output equal y.

Step 1: Transform the program into an equivalent model (R1CS, circuit-satisfiability, ...) amenable to probabilistic checking.

- We often call this step as the Front End (FE).

Step 2: Run a SNARK for R1CS or circuit-satisfiability. We often call this step as the Back End (BE).

- E.g.: Groth16, PlonK (2017), Ligero (2022), Polymath (2024), ...

FE ensures that P claims a witness w for R1CS or circuit-satisfiability is equivalent to claiming a witness w for the computer program.

Remarks on Standard Paradigm of SNARKs

- A witness w can be extended with auxiliary elements for applying general-purpose SNARKs.
- Not all programs are “good” for generating “efficient” circuit. Without careful design, the circuit may contain million of gates, degrading protocol performance.

Back-End Goals

We want a SNARK for R1CS or circuit-SAT for which:

- P: linear time.
- Proof π: as small as possible (few hundred bytes).
  - $\text{len}(\pi) = \text{sublinear}(|w|)$.
- V: as fast as possible (milliseconds to a few seconds).
  - $\text{time}(\text{V}) = \mathcal{O}_\lambda(|x|, \text{sublinear}(|C|))$.
- Strongly succinct: logarithm instead of sublinear.

Back-End Goals

We want a SNARK for R1CS or circuit-SAT for which:

- P: linear time.
- Proof π: as small as possible (few hundred bytes).
    - $\text{len}(\pi) = \text{sublinear}(|w|)$.
- V: as fast as possible (milliseconds to a few seconds).
    - $\text{time(V)} = \mathcal{O}_\lambda(|x|, \text{sublinear}(|C|))$.
- Strongly succinct: logarithm instead of sublinear.

Also consider (with specific application):

- Transparent: no "harmful" parameters produced in a trusted setup.
- Plausibly post-quantum secure: if P* has a quantum computer, he can find w for false statement (since we only consider "argument").

Key Paradigm for BE Design

1. Give a **polynomial IOP** for R1CS or circuit-satisfiability.
2. Combine with **polynomial commitment scheme** $\rightarrow$ succinct interactive argument.
3. Fiat-Shamir transformation $\rightarrow$ non-interactive scheme.

Key Paradigm for BE Design

1. Give a **polynomial IOP** for R1CS or circuit-satisfiability.

2. Combine with **polynomial commitment scheme** $\rightarrow$ succinct interactive argument.

3. Fiat-Shamir transformation $\rightarrow$ non-interactive scheme.

Another paradigm: Linear-PCP based (Groth16, Pinocchio, ...).

Outline

Arithmetic Circuits

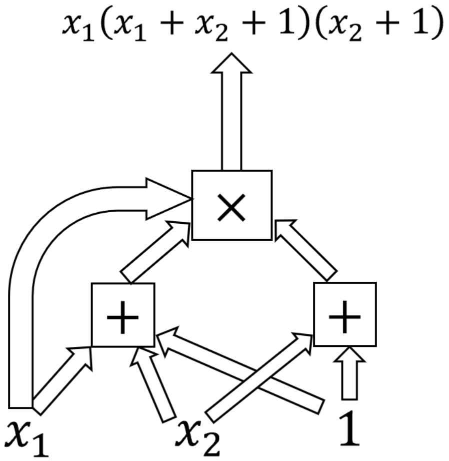
Arithmetic circuit $\mathcal{C}$: $\mathbb{F}^n \rightarrow \mathbb{F}$.

- Directed acyclic graph (DAG) where internal nodes are labeled as operation $(+, \times)$, and inputs are labeled as $(2, x_1, ..., x_n)$.
- Wires carrying values and connect to addition and multiplication of gates.
- $|\mathcal{C}| = \#$gates in $\mathcal{C}$.

Some of the input/output wires specify a statement x and the rest of the wires define a witness w.
$\rightarrow$ A binary relation $\mathcal{R} = \{(x; w) \mid \mathcal{C}(x, w) = 0\}$ consists of statement wires and witness wires that satisfy the arithmetic circuit.

Arithmetic Circuits (cont.)

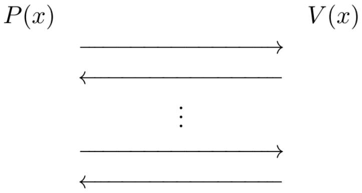
- We do not care about the fan-in of the gates.
- $|\mathcal{C}| = 3$.



Interesting Arithmetic Circuits

- $\mathcal{C}_{\mathsf{SHA}}(y; w)$ outputs 0 if $\mathsf{SHA256}(w) = y$, and $\neq 0$ otherwise.
  $\rightarrow \mathcal{C}_{\mathsf{hSHA}}(y; w) = (y - \mathsf{SHA256}(w))$, $|\mathcal{C}_{\mathsf{hSHA}}| \approx$ 20K gates.
- $\mathcal{C}_{\mathsf{sig}}(pk, m, \sigma; sk) = 0$ if σ is a valid ECDSA signature on m with respect to a key-pair (pk, sk).

Interactive Proofs (IPs)



Transcript of the protocol includes proof and messages during interaction.

(P, V) is a **multi-round** interactive proof system if the following two properties are satisfied:

- Completeness: $Pr[\langle P, V\rangle(x) = 1] \geq \frac{2}{3}$.
- Soundness: $Pr[\langle P^*, V\rangle(x) = 1] \leq \frac{1}{3}$.

Can be transformed into non-interactive via **Fiat-Shamir transformation**.

Argument Systems

Definition (Argument Systems)

An argument system is an interactive proof in which the soundness condition is only required to hold against prover strategies that run in polynomial time.

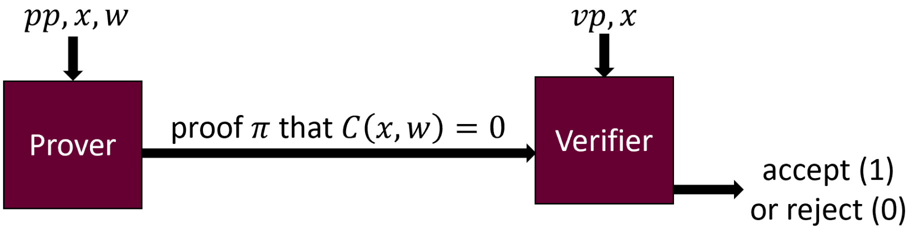
- Often called as **computational soundness**.
- The soundness for interactive proofs against computationally unbounded provers P^*, and is referred to as **statistical soundness** or **information-theoretic soundness**.

Proofs do not exist for false statements, arguments for false statements are only infeasible to find.

Non-interactive ARgument of Knowledge (NARK)

Public arithmetic circuit $\mathcal{C}(x, w) \to \mathbb{F}$, where x is a public statement, and w is a secret witness.

- Setup $\mathcal{S}(C)$ generates public parameters (pp, vp).
  - Trusted setup per circuit.
  - Trusted but universal (updatable) setup.
  - Transparent setup: do not use secret data (no trusted setup).



Non-interactive ARgument of Knowledge (NARK) (cont.)

Requirements:

- **Completeness:** If P is honest, then V can accept with high prob.
- **Knowledge soundness:** If V accepts, then P knows witness w s.t. $\mathcal{C}(x, w) = 0$, then exist an extractor $\mathcal{E}$ that can extract w.
- (Optional) **Zero knowledge:** (pp, vp, x, π) reveals nothing about w.

SNARK: Succinct NARK

SNARK: a NARK (completeness and knowledge soundness) that is succinct.

zk-SNARK: a SNARK that is also zero-knowledge.

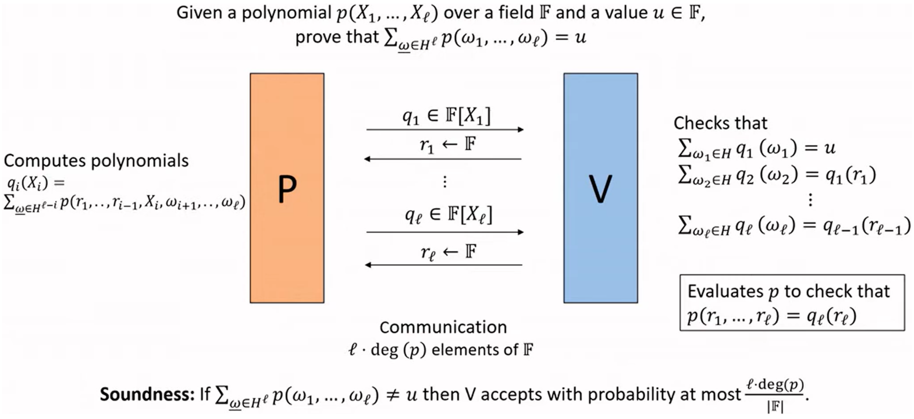
Recall: SNARK = PIOP + PCS.

Polynonial-IOP (PIOP)

The “pre”definition:

- P’s message is a polynomial $h(x)$.
- V is permitted to evaluate $h(x)$ at one point x_0, i.e. $h(x_0)$, and check some constraints.
- Goal: Prove an initial polynomial satisfies some constraints.
- Example: Sum-check protocol, ...

PIOP Example: Sum-check protocol


Given a polynomial $p(X_1, \dots, X_\ell)$ over a field $\mathbb{F}$ and a value $u \in \mathbb{F}$,
prove that $\sum_{\underline{\omega} \in H^\ell} p(\omega_1, \dots, \omega_\ell) = u$
Computes polynomials
$q_i(X_i) = \sum_{\underline{\omega} \in H^{\ell-i}} p(r_1, \dots, r_{i-1}, X_i, \omega_{i+1}, \dots, \omega_\ell)$
P
$q_1 \in \mathbb{F}[X_1]$
$r_1 \leftarrow \mathbb{F}$
$\vdots$
$q_\ell \in \mathbb{F}[X_\ell]$
$r_\ell \leftarrow \mathbb{F}$
V
Checks that
$\sum_{\omega_1 \in H} q_1(\omega_1) = u$
$\sum_{\omega_2 \in H} q_2(\omega_2) = q_1(r_1)$
$\vdots$
$\sum_{\omega_\ell \in H} q_\ell(\omega_\ell) = q_{\ell-1}(r_{\ell-1})$
Evaluates p to check that $p(r_1, \dots, r_\ell) = q_\ell(r_\ell)$
Communication
$\ell \cdot \deg(p)$ elements of $\mathbb{F}$
Soundness: If $\sum_{\underline{\omega} \in H^\ell} p(\omega_1, \dots, \omega_\ell) \neq u$ then V accepts with probability at most $\frac{\ell \cdot \deg(p)}{|\mathbb{F}|}$.

- H is often considered as a binary set $\{0, 1\}$.
- V needs oracle access to p for the final check (if p is not available).
- Do not require a trusted setup.

Polynomial-IOP (PIOP) (cont.)

So, what does a letter “O” (oracle) in PIOP mean?

- Challenge: P should not send the full polynomial (as large as the circuit), but V needs to know in a clear for evaluation.
  $\rightarrow$ How to achieve succinct (a proof smaller than a polynomial) while work-saving for V (allow to evaluate without reading full polynomial).
- An **oracle** is a “mysterious” function: it takes input, returns output, but reveals nothing about how the output is computed.
- Instead of sending full polynomial, we send **polynomial oracle**.

  E.g.: In sum-check protocol, we send an oracle of polynomial q_1, not q_1 itself $\rightarrow$ The challenge in practice is (i) how we define an oracle, and (ii) how V can compute $q_1(\omega_1)$ without knowing q_1.
- To implement such an oracle $\rightarrow$ **Polynomial Commitment Scheme.**

Polynomial Commitment Scheme (PCS)

Why is PCS suitable for securing a PIOP? Consider a “cheating” PIOP:

- P encodes a polynomial f as an oracle and sends to V.
- V samples x and querys an evaluation $f(x)$.
- However, after seeing x, P changes a polynomial f to f', and sends $f'(x)$ instead of $f(x)$.
- And, how V can verify the correctness of $f(x)$ associated with an oracle without knowledge of f.

Polynomial Commitment Scheme (PCS) (cont.)

High-level idea:

- P **binds** itself to a polynomial f by sending a short string $Com(f)$.
- V chooses x and asks P to evaluate $f(x)$.
- P sends y and a proof π s.t. y is consistent with $Com(f)$ and x.

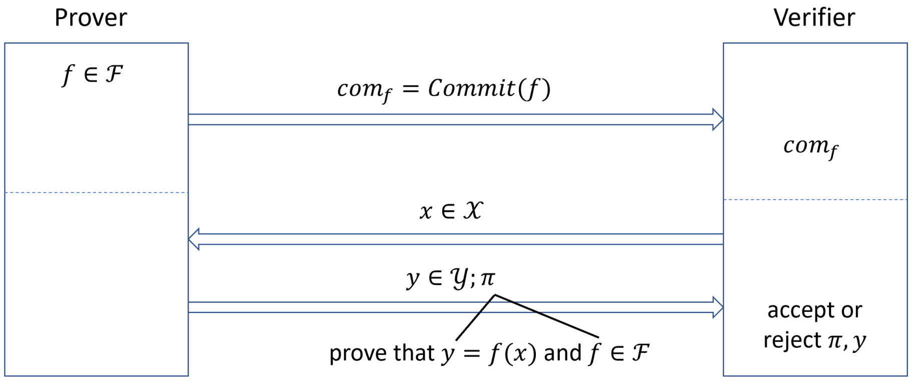
Goals:

- P cannot produce a valid proof for an incorrect evaluation.
- $Com(h)$ and π are short and easy to generate, π is easy to check.

Polynomial Commitment Scheme (PCS) (cont.)

Choose a family of function $\mathcal{F} = \{f : \mathcal{X} \to \mathcal{Y}\}$.

- A family of polynomial f of bounded degree d over a field $\mathbb{F}$, i.e. $f \in \mathbb{F}^{\leq d}[X]$.



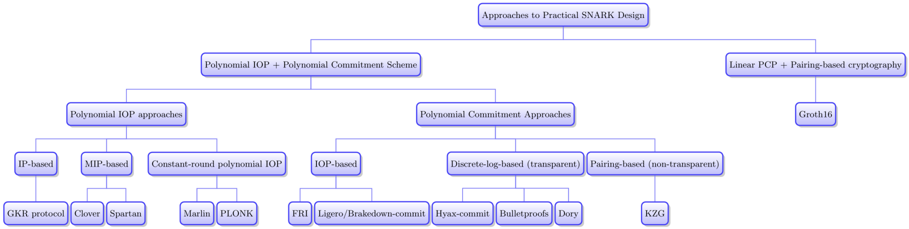
Outline

Taxonomy of SNARKs

Recall (again) that SNARK $=$ PIOP $+$ PCS.

- There are several different PIOP and PCS in literature.
- Can mix-and-match to get different tradeoffs (P time, proof size, etc.) for specific application.
- The properties of SNARK (transparent, plausible post-quantum) is determined by PCS.

Taxonomy of SNARKs (cont.)



Approaches to Practical SNARK Design
Polynomial IOP + Polynomial Commitment Scheme
Linear PCP + Pairing-based cryptography
Polynomial IOP approaches
Polynomial Commitment Approaches
Groth16
IP-based
MIP-based
Constant-round polynomial IOP
IOP-based
Discrete-log-based (transparent)
Pairing-based (non-transparent)
GKR protocol
Clover
Spartan
Marlin
PLONK
FRI
Ligero/Brakedown-commit
Hyax-commit
Bulletproofs
Dory
KZG

Figure: A taxonomy of SNARK design taken from a book "Proofs, Arguments, and Zero-Knowledge" by Justin Thaler.

Taxonomy of PIOP (3 classes)

- Based on interactive proofs (IPs).
    - Hyrax (2017), vSQL (2017), Libra (2019), Virgo (2019), ...
- Based on multi-prover interactive proofs (MIPs).
    - Spartan (2019), Brakedown (2021), Xiphos (2020), ...
- Based on constant-round polynomial IOPs (PIOPs).
    - Marlin (2019), PlonK (2019), ...

List in increasing P cost, and decreasing proof length and V cost.

Taxonomy of PCS (4 classes)

	Transparent	PQ	E.g.
Pairing			KZG10, PST13, ZGKPP18
Discrete Log	✓		BCCGP16, Bulletproofs, Hyrax
IOPs + Hash	✓	✓	Ligero, FRI, Brakedown
Unknown order group	✓		DARK, Dew

(PQ: Post-quantum)

- Pairing schemes have constant size evaluation proof for $y = f(x)$.
- The red-name schemes are popular.

Remarks

- So many SNARKs (at least $4 \times 3 = 12$).
- Choosing the suitable PIOP and PCS can depend on more than two properties listed above, i.e. the choice of field to work, ...
    - ECDSA works over elliptic curve groups, which are not FFT-friendly. Many SNARKs require P to perform FFTs.
    - DL-based and pairing-based PCS **must** use a field of size equal to the order of the cryptographic group. In contrast, IOP-based SNARKs avoid this limitation by using hash function working over arbitrary data.
- Again, we only mention SNARKs, but not zk-SNARKs.

Outline

SNARKs based on Linear-PCPs

E.g.: Groth16 (application in ZCash).

- Pros: Shortest proof (3 group elements), fastest V.
- Cons: Circuit-specific trusted setup, slow and space-intensive P, not post-quantum.

Technique: Arithmetic circuit $\rightarrow$ Rank 1 Constraint System (R1CS) $\rightarrow$ Quadratic Arithmetic Program (QAP) for prove system.

Introduction to R1CS

Goal: Generate a system of equations of the form

$$La \circ Ra = Oa$$

where L, R, O are public matrices and a is a witness vector.

- A witness vector contains input variables, output variable(s), and the intermediate values.
- L, R, O represent an arithmetic circuit.
- Every circuit can be represented as a R1CS.

Step-by-step from circuit to R1CS

Step 1: Define all equations (fan-in 2) for an arithemtic circuit $\rightarrow$ The number of rows in L, R, O.

- Addition is free, but not for multiplication.

Step 2: Define a witness vector contains input, output variables and some of the intermediate values.

- Witness is often in the form $[1, \text{out}[:], \text{in}[:], \dots]$.
- The size of witness is the number of columns in L, R, O.

Step 3: Define matrix L, R, O by looking at the equation in step 1.

Example 1: $z = xy$ into R1CS

Equation (fan-in 2) in a circuit:

$$z = xy.$$

- 1 equation $\rightarrow n = 1$.
- A witness vector $a = [1, z, x, y] \rightarrow m = 4$.

$$\begin{bmatrix} \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix} = \begin{bmatrix} \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix} \circ \begin{bmatrix} \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix}$$

Example 1: $z = xy$ into R1CS

Equation (fan-in 2) in a circuit:

$$z = xy.$$

- 1 equation $\rightarrow n = 1$.
- A witness vector $a = [1, z, x, y] \rightarrow m = 4$.

$$\begin{bmatrix} 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix}$$

Example 2: $z = xyuv$ into R1CS

Equations (fan-in 2) in a circuit:

$$\begin{aligned} t_1 &= xy \\ t_2 &= uv \\ z &= t_1 t_2 \end{aligned}$$

- 3 equations $\rightarrow n = 3$.
- A witness vector $a = [1, z, x, y, u, v, t_1, t_2] \rightarrow m = 8$.

Example 2: $z = xyuv$ into R1CS

Equations (fan-in 2) in a circuit:

$$\begin{aligned} t_1 &= xy \\ t_2 &= uv \\ z &= t_1 t_2 \end{aligned}$$

$$\begin{bmatrix} - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \\ u \\ v \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \\ u \\ v \\ t_1 \\ t_2 \end{bmatrix} \circ \begin{bmatrix} - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \\ u \\ v \\ t_1 \\ t_2 \end{bmatrix}$$

Example 2: $z = xyuv$ into R1CS

Equations (fan-in 2) in a circuit:

$$\begin{aligned} t_1 &= xy \\ t_2 &= uv \\ z &= t_1 t_2 \end{aligned}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \\ u \\ v \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \\ u \\ v \\ t_1 \\ t_2 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ & \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \\ u \\ v \\ t_1 \\ t_2 \end{bmatrix}$$

Example 2: $z = xyuv$ into R1CS

Equations (fan-in 2) in a circuit:

$$\begin{aligned} t_1 &= xy \\ t_2 &= uv \\ z &= t_1 t_2 \end{aligned}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ - & - & - & - & - & - & - & - \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \\ u \\ v \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ - & - & - & - & - & - & - & - \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \\ u \\ v \\ t_1 \\ t_2 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ - & - & - & - & - & - & - & - \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \\ u \\ v \\ t_1 \\ t_2 \end{bmatrix}$$

Example 2: $z = xyuv$ into R1CS

Equations (fan-in 2) in a circuit:

$$\begin{aligned} t_1 &= xy \\ t_2 &= uv \\ z &= t_1 t_2 \end{aligned}$$

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \\ u \\ v \\ t_1 \\ t_2 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \\ u \\ v \\ t_1 \\ t_2 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \\ u \\ v \\ t_1 \\ t_2 \end{bmatrix}$$

Example 3: $z = xy + 2$ into R1CS

Equations (fan-in 2) in a circuit:

$$\begin{aligned} t_1 &= xy \\ z &= t_1 + 2 \end{aligned}$$

- 2 equations $\rightarrow n = 2$.
- A witness vector $a = [1, z, x, y, t_1] \rightarrow m = 5$.

Example 3: $z = xy + 2$ into R1CS

Equation (fan-in 2) in a circuit:

$$-2 + z = xy$$

- 1 equation $\rightarrow n = 1$.
- A witness vector $a = [1, z, x, y] \rightarrow m = 4$.

$$\begin{bmatrix} \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix} = \begin{bmatrix} \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix} \circ \begin{bmatrix} \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix}$$

Example 3: $z = xy + 2$ into R1CS

Equation (fan-in 2) in a circuit:

$$-2 + z = xy$$

- 1 equation $\rightarrow n = 1$.
- A witness vector $a = [1, z, x, y] \rightarrow m = 4$.

$$\begin{bmatrix} \_ & \_ & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix}$$

Example 3: $z = xy + 2$ into R1CS

Equation (fan-in 2) in a circuit:

$$-2 + z = xy$$

- 1 equation $\rightarrow n = 1$.
- A witness vector $a = [1, z, x, y] \rightarrow m = 4$.

$$\begin{bmatrix} -2 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ y \end{bmatrix}$$

Example 4: $z = x^3 + x + 5$ into R1CS

Equations (fan-in 2) in a circuit:

$$\begin{aligned} t_1 &= xx \\ -5 - z - x &= t_1 x \end{aligned}$$

- 2 equations $\rightarrow n = 2$.
- A witness vector $a = [1, z, x, t_1] \rightarrow m = 4$.

$$\begin{bmatrix} \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ t_1 \end{bmatrix} = \begin{bmatrix} \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ t_1 \end{bmatrix} \circ \begin{bmatrix} \_ & \_ & \_ & \_ \\ \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ t_1 \end{bmatrix}$$

Example 4: $z = x^3 + x + 5$ into R1CS

Equations (fan-in 2) in a circuit:

$$\begin{aligned} t_1 &= xx \\ -5 - z - x &= t_1 x \end{aligned}$$

- 2 equations $\rightarrow n = 2$.
- A witness vector $a = [1, z, x, t_1] \rightarrow m = 4$.

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ t_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ t_1 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 1 & 0 \\ \_ & \_ & \_ & \_ \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ t_1 \end{bmatrix}$$

Example 4: $z = x^3 + x + 5$ into R1CS

Equations (fan-in 2) in a circuit:

$$\begin{aligned} t_1 &= xx \\ -5 - z - x &= t_1 x \end{aligned}$$

- 2 equations $\rightarrow n = 2$.
- A witness vector $a = [1, z, x, t_1] \rightarrow m = 4$.

$$\begin{bmatrix} 0 & 0 & 0 & 1 \\ -5 & -1 & -1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ t_1 \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ t_1 \end{bmatrix} \circ \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 \\ z \\ x \\ t_1 \end{bmatrix}$$

Quadratic Arithmetic Circuit (QAP)

Rewrite the formula of R1CS for specific row q as

$$(\sum_{i=1}^{m} L_{q,i}a_i) \cdot (\sum_{i=1}^{m} R_{q,i}a_i) = \sum_{i=1}^{m} O_{q,i}a_i.$$

Sample n element $t_1, ..., t_n \in \mathbb{F}$, and compute $\ell_i(x), r_i(x), o_i(x)$ satisfy:

$$\ell_i(t_q) = L_{q,i};\ \ r_i(t_q) = R_{q,i};\ \ o_i(t_q) = O_{q,i}\ \forall i \in [m], q \in [n].$$

$$\rightarrow (\sum_{i=1}^{m} \ell_i(t_q) \cdot a_i) \cdot (\sum_{i=1}^{m} r_i(t_q) \cdot a_i) = \sum_{i=1}^{m} o_i(t_q) \cdot a_i.$$

Define $t(X) = \prod_{i=1}^{n}(X - t_i)$, we can have the following formula:

$$(\sum_{i=1}^{m} \ell_i(t_q) \cdot a_i) \cdot (\sum_{i=1}^{m} r_i(t_q) \cdot a_i) \equiv \sum_{i=1}^{m} o_i(t_q) \cdot a_i \pmod{t(X)}.$$

Schwart-Zippel Lemma

Lemma (Schwart-Zippel Lemma)

Let $\mathbb{F}$ be any field, and let $g : \mathbb{F}^m \to \mathbb{F}$ be a nonzero m-variate polynomial of total degree at most d. Then on any finite set $S \subseteq \mathbb{F}$,

$$Pr[g(x) = 0 \mid x \leftarrow S^m] \leq \frac{d}{|S|}.$$

PCP for vanishing polynomial

Goal: Prove $f(x) = 0\ \forall x \in H$ where $H = \{h_1, ..., h_n\}$.

Trivial solution:

- Send a set H to V and check if $f(h_i) = 0\ \forall i \in [n]$.

PCP protocol:

- $Z_H(X) = \prod_{i=1}^{n}(X - h_i)$ is a vanishing polynomial.
  $\rightarrow f(X) = Z_H(X) \cdot t(X)$ where $t(X)$ is a quotient polynomial.

$$\begin{tikzcd}
\text{P}(f(X), t(X)) & & \text{V} \\
{} & & {} \arrow[ll, Leftarrow, "r \leftarrow \mathbb{F}"'] \\
{} \arrow[rr, Rightarrow, "f(r);\ t(r)"] & & \text{Check } f(r) = Z_H(r) \cdot t(r)
\end{tikzcd}$$

Constant-round PIOP + KZG polynomial commitment

E.g.: Marlin, PlonK.

- Pros: Universal trusted setup.
- Cons: Proofs are 4x-6x larger and P is slower than Groth16, not post-quantum.
- Often uses for small circuit.

Technique for PlonK (high-level):

- Prove for gate constraints and wire constraints (two components of arithmetic circuit).
- Computer program $\rightarrow$ Arithmetic circuit $\rightarrow$ **PLONK's equations** (proportion with the number of gates) $\rightarrow$ Polynomial equations.

Outline

MIPs and IPs + fast-P polynomial commitments

E.g.: Spartan, Brakedown, Xiphos, Hyrax.

- Pros: Fastest P in literature, plausibly post-quantum, transparent iff PCS.
- Cons: Bigger proof size than Groth16 and Marlin/PlonK.
- Not widely use in practice due to large proof size, which increases the verification cost (a critical factor in current SNARK applications).

Spartan's Technique

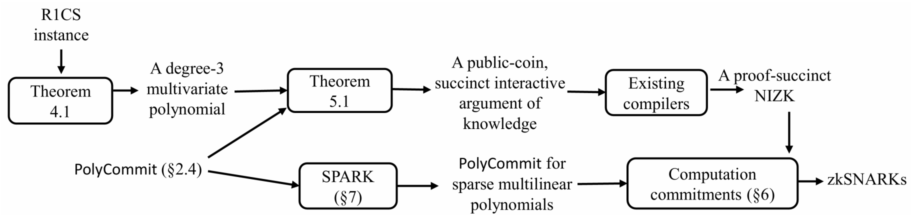

R1CS instance
Theorem 4.1
A degree-3 multivariate polynomial
Theorem 5.1
A public-coin, succinct interactive argument of knowledge
Existing compilers
A proof-succinct NIZK
PolyCommit (§2.4)
SPARK (§7)
PolyCommit for sparse multilinear polynomials
Computation commitments (§6)
zkSNARKs

FIGURE 1—Overview of our techniques for constructing zkSNARKs.

Any PIOP + FRI polynomial commitment

E.g.: Fractal, Aurora, Virgo, Ligero++.

- Pros: Shortest proofs amongst plausibly post-quantum SNARKs.
- Cons: Slow P, proofs still large $\mathcal{O}(\lambda \log^2) > \mathcal{O}(\log) > \mathcal{O}(1)$.

Outline

Promised approach: Composition proof

Take "fast P, big proof" SNARK and compose with small-proof SNARK.

- P does not send "big proof" π.
- P proves knowing π using small-proof SNARK.
  $\rightarrow$ Need to reperesent "big proof" SNARK verifier as a circuit to apply small-proof SNARK.

$\Rightarrow$ Yield the proof size and V time from small-proof SNARK, and achieve the P efficiency of the big-proof SNARK.

If any of SNARK is not transparent, then the result is not transparent. Require transparent for both SNARKs to preserve this property.