Modern Zero Knowledge Cryptography - MIT IAP, 2023.1

Lecture 9: Proof systems stack

27 January 2023

Lecturer: Ying Tong Lai

In the previous lectures, we went through the components of modern SNARKs (*Lecture 5 (Commitment Schemes); Lecture 7 (Arithmetisations)*), as well as the construction of the PlonK [3] (*Lecture 8*) zk-SNARK. This note illustrates a few more end-to-end constructions of proof systems, with the hope of beginning to construct some kind of taxonomy. We briefly explore **recursion**, which lets us check an arbitrarily long chain of computation with a constant-size proof; and **proof composition**, where multiple proof systems are embedded in a single protocol.

1 A taxonomy of proof systems

Most modern proof systems are designed in a modular way, using *algebraic holographic proofs* as the information-theoretic component. These are polynomial interactive oracle proofs (PIOPs) where the verifier does not have access to the full encoding of the relation being proven, but instead *queries* it via oracle access. Because AHPs work with any polynomial commitment scheme, the setups for these proof systems are specific to just the chosen commitment scheme, and not the relation being proven. To contrast the IOP-based approach against older schemes, we go through two proof system stacks that both start with the R1CS arithmetisation: Pinocchio [6] constructs a *linear probabilistically checkable proof* (LPCP), which is then compiled using pairing-based cryptography, while Marlin [2] and Spartan [7] construct *algebraic holographic proofs* (AHPs), compiled using a polynomial commitment scheme.

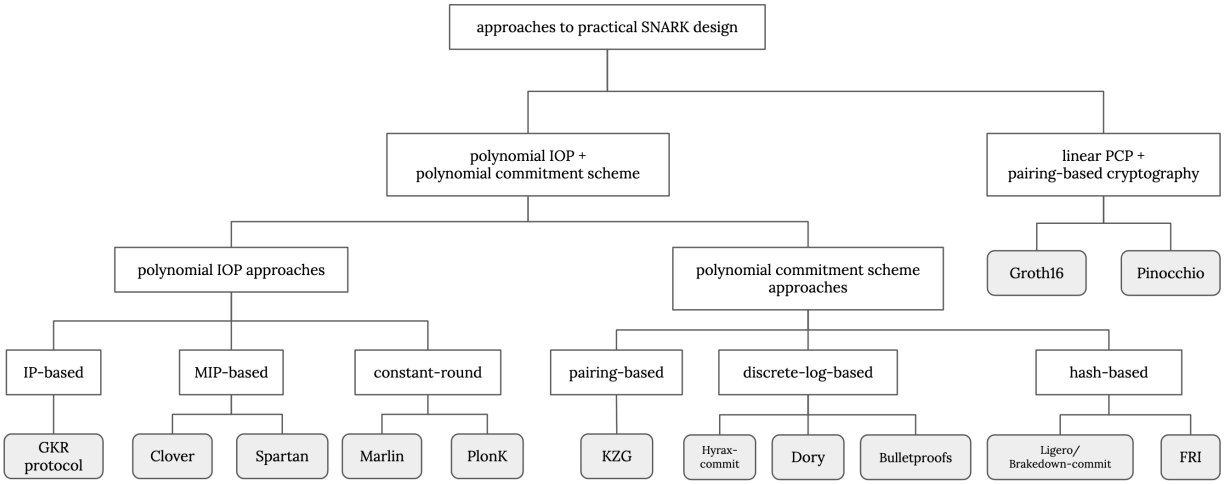

Figure 1: A taxonomy of modern SNARKs. Adapted from Fig. 19.1 of [9].

1.1 R1CS $\rightarrow$ QAP + LPCP + pairing-based-cryptography (Pinocchio)

Recall from *Lecture 7 (Arithmetisations)* that a Rank-1 Constraint System (R1CS) consists of the matrices $\mathcal{L}, \mathcal{R}, \mathcal{O}$. A satisfying assignment $\vec{a} = (a_0, \ldots, a_{m-1})$ fulfils $\mathcal{L} \cdot \vec{a} + \mathcal{R} \cdot \vec{a} - \mathcal{O} \cdot \vec{a} = \vec{0}$.

The R1CS can be expressed as a Quadratic Arithmetic Program (QAP) Q, which consists of the polynomials $\{L_j\}, \{R_j\}, \{O_j\}, j \in [m]$, and a target polynomial $T(X)$ that evaluates to zero at all $i \in [d]$. To check that an assignment $\vec{a}$ is satisfying, we check that

$$H(X) := P(X) \,/\, T(X),$$

where:

- $P(X) := L(X) \cdot R(X) - O(X)$ checks the assignment at each point in the evaluation domain;
- $L(X) := \sum a_j \cdot L_j(X), R(X) := \sum a_j \cdot R_j(X), O(X) := \sum a_j \cdot O_j(X)$.

Definition 1.1. [Linear Probabilistically Checkable Proof (LPCP)] *A Linear Probabilistically Checkable Proof (LPCP) of length m is an oracle computing a linear function $\pi : \mathbb{F}^m \rightarrow \mathbb{F}$; namely, the answer to each oracle query $\vec{q_i} \in \mathbb{F}^m$ is $a_i = \langle \pi, q_i \rangle$.*

To check the correctness of the QAP, Pinocchio [6] constructs an LPCP compiled with pairing-based cryptography. Here, we use a symmetric pairing $e : \mathbb{G}_1 \times \mathbb{G}_1 \rightarrow \mathbb{G}_T$ where $G \in \mathbb{G}_1$ is a generator. We proceed as follows to check the divisibility of the QAP:

QAP divisibility check

- $\mathsf{Setup}(\{L_j\}, \{R_j\}, \{O_j\}, T(X)) \rightarrow \mathsf{srs}$:
  - sample a random evaluation point s;
  - output the commitments to the evaluations of the QAP's polynomials at s :

$$\mathsf{srs} = (\{[L_j(s)]_1\}, \{[R_j(s)]_1\}, \{[O_j(s)]_1\}, [T(s)]_1)\,.$$

- $\mathsf{Prove}(\mathsf{srs}, \vec{a}) \rightarrow \pi$:
  - compute

$$\pi_L = \sum_{j=0}^{m-1} [a_j][L_j(s)]_1, \pi_R = \sum_{j=0}^{m-1} [a_j][R_j(s)]_1, \pi_O = \sum_{j=0}^{m-1} [a_j][O_j(s)]_1;$$

  - compute $\pi_H = [H(s)]_1$;
  - output $\pi = (\pi_L, \pi_R, \pi_O, \pi_H)$.

- $\mathsf{Verify}([T(s)]_1, \pi) \rightarrow \{0, 1\}$: check $\boxed{e(\pi_L, \pi_R) - e(\pi_O, G) \stackrel{?}{=} e([T(s)]_1, \pi_H);}$ this corresponds to checking $L(s) \cdot R(s) - O(s) = T(s) \cdot H(s)$ "in the exponent".

However, we haven't actually constrained the prover to use the provided $\{L_j\}, \{R_j\}, \{O_j\}$ polynomials from the setup. To enforce that π_L, π_R, π_O were indeed constructed as linear combinations of the srs elements, we proceed as follows:

QAP linear combination check

- $\mathsf{Setup}(\{L_j\}, \{R_j\}, \{O_j\}, T(X)) \rightarrow \mathsf{srs}$:
  - sample random shifts $\alpha_L, \alpha_R, \alpha_O$;
  - compute the commitments to the evaluations of the "$\alpha -$ *shifted*" L_j polynomials at s: $\{[L'_j(s)]_1\} = \{[\alpha_L \cdot L_j(s)]_1\}$;
  - similarly, compute

$$\{[R'_j(s)]_1\} = \{[\alpha_R \cdot R_j(s)]_1\}, \{[O'_j(s)]_1\} = \{[\alpha_O \cdot O_j(s)]_1\};$$

  - compute the rest of the srs as in the QAP divisibility check, and output

$$\mathsf{srs} = \begin{pmatrix} \{[L_j(s)]_1\}, \{[R_j(s)]_1\}, \{[O_j(s)]_1\}, \\ \{[L'_j(s)]_1\}, \{[R'_j(s)]_1\}, \{[O'_j(s)]_1\}, \\ [\alpha_L]_1, [\alpha_R]_1, [\alpha_O]_1 \end{pmatrix}.$$

- $\mathsf{Prove}(\mathsf{srs}, \vec{a}) \rightarrow \pi$:
  - compute the "α-shifted"

$$\pi'_L = \sum_{j=0}^{m-1} [a_j][L'_j(s)]_1, \pi'_R = \sum_{j=0}^{m-1} [a_j][R'_j(s)]_1, \pi'_O = \sum_{j=0}^{m-1} [a_j][O'_j(s)]_1;$$

  - compute π_L, π_R, π_O as in the QAP divisibility check;
  - output $\pi = (\pi_L, \pi_R, \pi_O, \pi'_L, \pi'_R, \pi'_O)$.
- $\mathsf{Verify}([\alpha_L]_1, [\alpha_R]_1, [\alpha_O]_1, \pi) \rightarrow \{0, 1\}$: check that

$$e(\pi_L, [\alpha_L]_1) \stackrel{?}{=} e(\pi'_L, G_1), e(\pi_R, [\alpha_R]_1) \stackrel{?}{=} e(\pi'_R, G_1),$$
$$e(\pi_O, [\alpha_O]_1) \stackrel{?}{=} e(\pi'_O, G_1).$$

Since the prover does not know the shift values $\alpha_L, \alpha_R, \alpha_O$, the only way they could have constructed valid α-shifted pairs is to use the actual commitments to the QAP polynomials and their α-shifted versions, provided in the srs. In other words, since the prover does not know the value s and cannot recover it from the encrypted values in the srs, they are only able to compute linear combinations of the encrypted QAP polynomials to compute the proof.

A final problem remains: how do we know constrain the prover to use the same coefficients $\vec{a}$ in computing each of the π_L, π_R, π_O commitments?

QAP coefficients consistency check

- $\mathsf{Setup}(\{L_j\}, \{R_j\}, \{O_j\}, T(X)) \to \mathsf{srs}$:
  - sample random β, γ;
  - compute the commitments to the evaluations of the "β-*shifted*" L_j polynomials at s: $\{[L''_j(s)]_1\} = \{[\beta \cdot L_j(s)]_1\}$;
  - similarly, compute

$$\{[R''_j(s)]_1\} = \{[\beta \cdot R_j(s)]_1\}, \{[O''_j(s)]_1\} = \{[\beta \cdot O_j(s)]_1\};$$

  - compute the rest of the srs as in the QAP divisibility check, and output

$$\mathsf{srs} = \begin{pmatrix} [\gamma]_1, [\beta\gamma]_1, \\ \{[L'_j(s)]_1\}, \{[R'_j(s)]_1\}, \{[O'_j(s)]_1\}, \\ \{[L_j(s)]_1\}, \{[R_j(s)]_1\}, \{[O_j(s)]_1\} \end{pmatrix}.$$

- $\mathsf{Prove}(\mathsf{srs}, \vec{a}) \to \pi$:
  - compute the polynomial $F(X) = \sum a_j \cdot (L_j(X) + R_j(X) + O_j(X))$;
  - using the shifted commitments from the srs, compute a commitment to the "β-shifted" $S(X)$ evaluated at s:

$$\pi'_F := \sum [a_j]([L''_j(s)]_1 + [R''_j(s)]_1 + [O''_j(s)]_1)$$

  - compute π_L, π_R, π_O as in the QAP divisibility check;
  - output $\mathsf{srs} = (\pi'_F, \pi_L, \pi_R, \pi_O)$.
- $\mathsf{Verify}([\gamma]_1, [\beta\gamma]_1, \pi) \to \{0, 1\}$: check

$$e(\pi_L + \pi_R + \pi_O, [\beta\gamma]_1) \stackrel{?}{=} e(\pi'_F, [\gamma]_1);$$

  this corresponds to checking $(L(s) + R(s) + O(s)) \cdot \beta \cdot \gamma \stackrel{?}{=} \beta \cdot F(s) \cdot \gamma$ "in the exponent".

The QAP $\to$ LPCP approach was introduced in [4], and followed by many improvements and refinements such as Pinocchio [6] (explained above) and Zaatar [8]. Groth16 [5] improved the proof size to only 3 group elements (possibly the optimal bound). The drawback of this approach is that queries to the linear PCP are "hard-coded": they are encrypted and stored in the srs of the scheme. This means the srs is *not universal*, and must be generated anew for each QAP.

1.2 R1CS $\rightarrow$ AHP + polynomial commitment scheme

For the same R1CS arithmetisation, both Marlin [2] and Spartan [7] introduce *algebraic holographic proofs* (AHPs) that allow a *universal setup* to be reused for arbitrary relations. The intuition here is that the circuit's wiring predicates are not hard-coded in the setup; rather, their polynomial encoding is sent during the offline phase. Most of the work involves massaging the matrices into polynomial encodings, and reducing the verifier cost of accessing these encodings. To achieve this, both Marlin and Spartan introduce additional rounds of interaction where the verifier queries the matrix values using the sumcheck protocol.

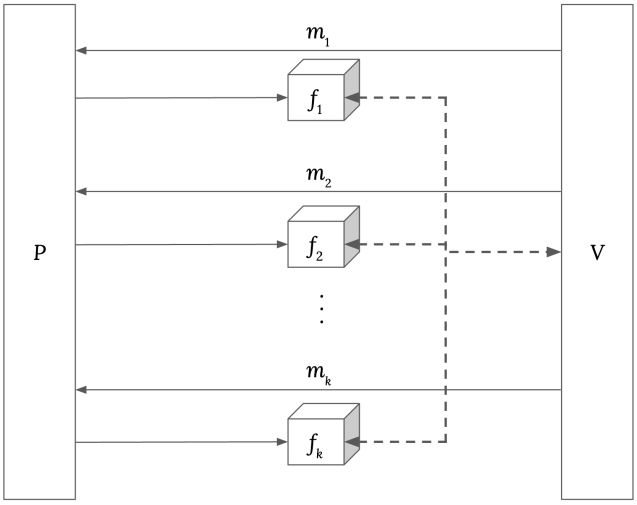

Figure 2: Recall the interactive oracle proof (IOP) from *Lecture 5 (Commitment Schemes)*. An algebraic holographic proof (AHP) is closely related to a polynomial IOP.

Definition 1.2. [Algebraic Holographic Proof (AHP)] *An algebraic holographic proof (AHP) for an indexed relation $\mathcal{R}$ over a field $\mathbb{F}$ is a tuple $\mathsf{AHP} = (\mathsf{k}, \mathsf{s}, \mathsf{d}, \mathsf{I}, \mathsf{P}, \mathsf{V})$, where k specifies the number of interaction rounds; $\mathsf{s} : \{0,1\}^* \rightarrow \mathbb{N}$ specifies the number of polynomials in each round; $\mathsf{d} : \{0,1\}^* \rightarrow \mathbb{N}$ specifies degree bounds on these polynomials; and $\mathsf{I}, \mathsf{P}, \mathsf{V}$ are the **indexer**, **prover**, and **verifier**. The rounds of interaction proceed as follows:*

- *Round 0 (**offline** phase): the indexer receives an index $\mathsf{I}(\mathbb{i}) \rightarrow p_{0,1}, \ldots, p_{0,\mathsf{s}(0)} \in \mathbb{F}^{\mathsf{d}(0)}[X]$ and outputs $\mathsf{s}(0)$ polynomials encoding the given index.*
- *Round $i \in [\mathsf{k}]$ (**online** phase): the verifier V sends a message $\rho_i \in \mathbb{F}^*$ to the prover P; P replies with $\mathsf{s}(i)$ oracle polynomials $p_{i,1}, \ldots, p_{i,\mathsf{s}(i)} \in \mathbb{F}^{\mathsf{d}(i)}[X]$. The verifier may query any of the polynomials it has received any number of times. A query consists of an evaluation point z, and its corresponding answer is $p_{i,j}(z) \in \mathbb{F}$.*

After the interaction, the verifier either accepts or rejects.

Math building block: Indexed relations. *An indexed relation $\mathcal{R}$ is a set of triples $(\mathbb{i}, \mathbb{x}, \mathbb{w})$ where $\mathbb{i}$ is the index, $\mathbb{x}$ is the instance, and $\mathbb{w}$ is the witness; the corresponding indexed language $\mathcal{L}(\mathcal{R})$ is the set of pairs $(\mathbb{i}, \mathbb{x})$ for which there exists a witness $\mathbb{w}$ such that $(\mathbb{i}, \mathbb{x}, \mathbb{w}) \in \mathcal{R}$.*

1.3 PAIR + polynomial IOP + KZG (PlonK)

The PlonK proof system [3] is very similar to Marlin [2]: they both encode their values in Lagrange interpolation polynomials over a multiplicative subgroup, and check relations on them using polynomial identities; then, they both construct polynomial IOPs compiled with the KZG polynomial commitment scheme.

Their key difference is in *arithmetisation*: since the verifier's checks are now performed "in the clear", as opposed to "in the exponent", PlonK no longer limits itself to degree-2 gates (formerly imposed by the bilinear pairing). This lets us define higher-degree custom gates than in R1CS.

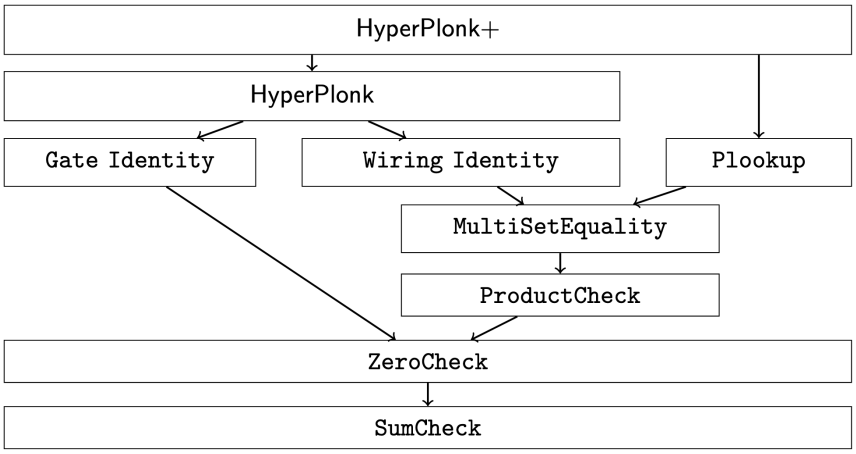

Figure 3: Recall from *Lecture 8 (PlonK)* the multiple subprotocols making up PlonK's IOP, including gate constraints, the permutation argument, and the lookup argument. These are randomly combined into a "zero check" over the vanishing polynomial. (Figure taken from [1]).

The modular construction of PlonK has allowed implementers to experiment with alternative arithmetisations and polynomial commitment schemes. This opens up possibilities for composition across different proof systems.

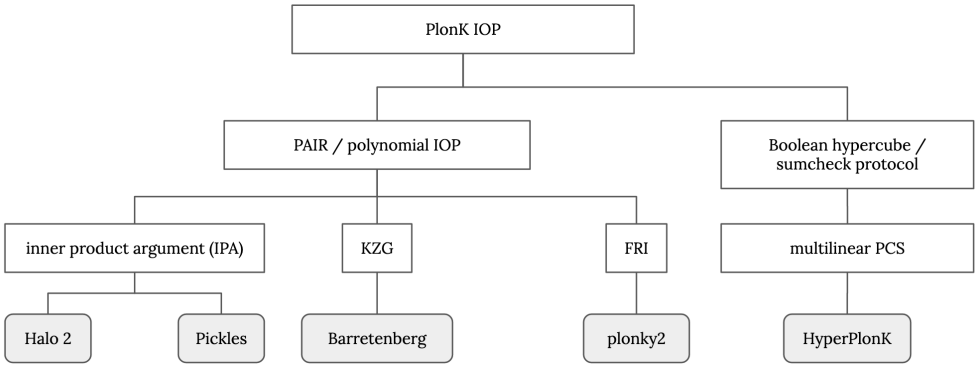

Figure 4: "PlonKish" arithmetisation and IOP is the core of many production proof system stacks.

References

[1] B. Chen, B. Bünz, D. Boneh, and Z. Zhang. Hyperplonk: Plonk with linear-time prover and high-degree custom gates. *Cryptology ePrint Archive*, 2022.

[2] A. Chiesa, Y. Hu, M. Maller, P. Mishra, N. Vesely, and N. Ward. Marlin: Preprocessing zksnarks with universal and updatable srs. In *Advances in Cryptology–EUROCRYPT 2020: 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10–14, 2020, Proceedings, Part I 39*, pages 738–768. Springer, 2020.

[3] A. Gabizon, Z. J. Williamson, and O. Ciobotaru. Plonk: Permutations over lagrange-bases for oecumenical noninteractive arguments of knowledge. *Cryptology ePrint Archive*, 2019.

[4] R. Gennaro, C. Gentry, B. Parno, and M. Raykova. Quadratic span programs and succinct nizks without pcps. In *Advances in Cryptology–EUROCRYPT 2013: 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques, Athens, Greece, May 26-30, 2013. Proceedings 32*, pages 626–645. Springer, 2013.

[5] J. Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology–EUROCRYPT 2016: 35th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Vienna, Austria, May 8-12, 2016, Proceedings, Part II 35*, pages 305–326. Springer, 2016.

[6] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. *Communications of the ACM*, 59(2):103–112, 2016.

[7] S. Setty. Spartan: Efficient and general-purpose zksnarks without trusted setup. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III*, pages 704–737. Springer, 2020.

[8] S. Setty, B. Braun, V. Vu, A. J. Blumberg, B. Parno, and M. Walfish. Resolving the conflict between generality and plausibility in verified computation. In *Proceedings of the 8th ACM European Conference on Computer Systems*, pages 71–84, 2013.

[9] J. Thaler et al. Proofs, arguments, and zero-knowledge. *Foundations and Trends® in Privacy and Security*, 4(2–4):117–660, 2022.