Verifiable Computing

Compiled by Sabrina Hirani

Summer 2025

These are personal notes on verifiable computing referencing notes by Justin Thaler, Alessandro Chiesa, etc.

Contents

1 Information-Theoretic Models

1.1 Interactive Proofs (IP)

In this model, we have a prescribed "honest" deterministic prover algorithm $\mathcal{P}$ and a probabilistic polynomial-time verifier algorithm $\mathcal{V}$ (with internal randomness r). Given a common input x, the prover and verifier exchange messages and the verifier will output either *accept* or *reject* at the end of the protocol:

$$\text{out}(\mathcal{V}, x, r, \mathcal{P}) \in \{\text{accept}, \text{reject}\}$$

Definition 1.1 (Interactive Proof System)**.** Let $(\mathcal{P}, \mathcal{V})$ be a k-message interactive proof system for a language $\mathcal{L} \subseteq \{0,1\}^n$. Then:

Completeness: For every $x \in \mathcal{L}$,

$$\Pr_r[\text{out}(\mathcal{V}, x, r, \mathcal{P}) = 1] \geq 1 - \delta_c$$

Soundness: For every $x \notin \mathcal{L}$ and any (possibly cheating) prover strategy $\mathcal{P}'$,

$$\Pr_r[\text{out}(\mathcal{V}, x, r, \mathcal{P}') = 1] \leq \delta_s$$

Here, δ_c denotes the completeness error and δ_s denotes the soundness error.

Remark. The system is considered valid if δ_c, δ_s are sufficiently small.

Definition 1.2 (Argument System)**.** 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. This is known as *computational soundness*.

This is in contrast to the soundness from the previous definition which can be referred to as *statistical soundness* (or information-theoretic soundness). Through the use of cryptographic primitives, e.g., *polynomial commitment schemes*, we can introduce such form of handicaps to the prover. This use of cryptography often allows argument systems to achieve additional desirable properties, i.e. succinctness, non-interactivity, etc., that are unattainable for interactive proofs.

Remark. Additionally, the system may satisfy some knowledge properties described in Section 1.5.

Example: IP for SAT

Problem Definition

The satisfiability problem (SAT) asks whether a given boolean formula ϕ in conjunctive normal form (CNF) is satisfiable. Formally, let ϕ be a CNF formula over variables $x_1, \ldots, x_n$ with disjunctive clauses $C_1, \ldots, C_m$:

$$\phi(x_1, \ldots, x_n) = \bigwedge_{j=1}^{m} C_j$$

Then language of satisfiable formulas is given by:

$$\text{SAT} = \{\ \phi \mid \exists\, a \in \{0,1\}^n \ \text{ s.t. } \phi(a) = 1\ \}$$

In this setting, the common input is ϕ and the prover wishes to convince the verifier that $\phi \in \text{SAT}$.

Description of Protocol

1. **Prover $\mathcal{P}$:** Sends a candidate assignment $a \in \{0,1\}^n$ to the verifier.
2. **Verifier $\mathcal{V}$:** Samples $j \xleftarrow{\$} [m]$ uniformly at random and sends j to $\mathcal{P}$.
3. **Verifier $\mathcal{V}$:** Checks $C_j(a) = 1$. If so, output *accept*; otherwise, *reject*.

Completeness. If ϕ is satisfiable, then there exists $a^\star$ with $\phi(a^\star) = 1$ so $C_j(a^\star) = 1$ for all $j \in [m]$. The honest prover sends $a^\star$ and the verifier accepts with probability 1.

Soundness. Suppose ϕ is unsatisfiable. Then, for every assignment $a \in \{0,1\}^n$, there is at least one clause C_j that is not satisfied, i.e., $C_j(a) = 0$.

Let:

$$s(a) \;=\; \underbrace{|\{j \in [m] : C_j(a) = 1\}|}_{\text{number of clauses satisfied by } a} \;\leq\; m - 1$$

Since j is uniform in $[m]$, the probability that the verifier accepts a single round is:

$$\Pr[\text{accept}] \;=\; \frac{s(a)}{m} \;\leq\; 1 - \frac{1}{m}$$

By repeating the protocol independently for k rounds, the soundness error decreases exponentially:

$$\Pr[\text{verifier accepts all } k \text{ rounds}] \;\leq\; \left(1 - \frac{1}{m}\right)^k.$$

Thus, by choosing k appropriately, the soundness error can be made arbitrarily small.

Note: This is an illustrative example. We will discuss performance considerations in future sections.

Remark. This protocol does not yield negligible soundness error unless repeated many times but it illustrates how randomness allows the verifier to check global structure using only local queries.

Example: The Sum-Check Protocol

Problem Definition

Suppose we are given a v-variate polynomial g defined over a finite field $\mathbb{F}$.
The sum-check protocol allows a verifier to efficiently check claims of the following sum:

$$H := \sum_{x_1 \in \{0,1\}} \cdots \sum_{x_n \in \{0,1\}} g(x_1, \ldots, x_v),$$

Description of Protocol

1. **Round 1:** $\mathcal{P}$ sends the univariate polynomial:

   $$g_1(X_1) = \sum_{x_2, \ldots, x_v \in \{0,1\}^{v-1}} g(X_1, x_2, \ldots, x_v)$$

   $\mathcal{V}$ checks that g_1 is a univariate polynomial of degree at most $\deg_1(g)$, and that $H = g(0) + g(1)$. $\mathcal{V}$ chooses a random element $r_1 \in \mathbb{F}$ and sends r_1 to $\mathcal{P}$.

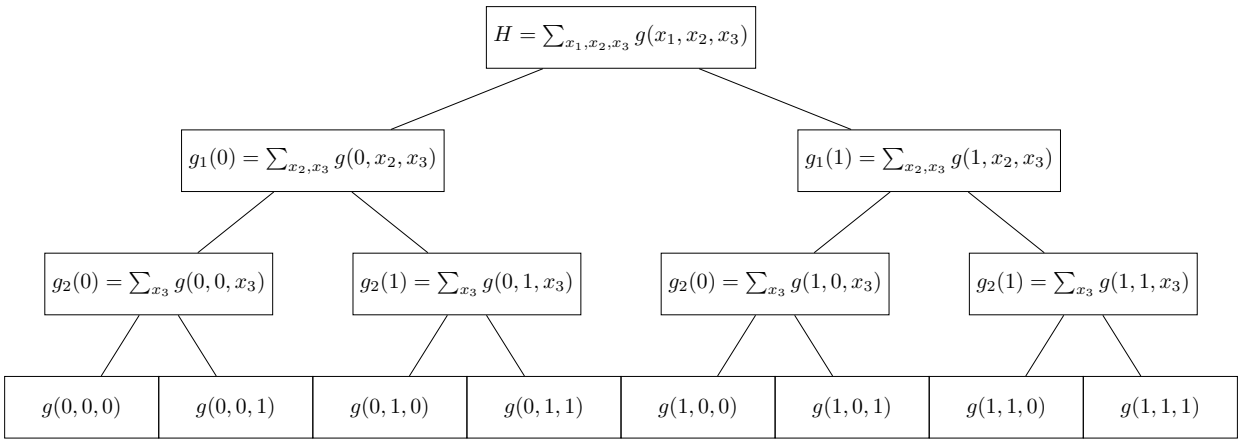
2. **Round i $(2 \leq i \leq v)$:** $\mathcal{P}$ sends the univariate polynomial:

   $$g_i(X_i) = \sum_{x_{i+1}, \ldots, x_v \in \{0,1\}^{v-i}} g(r_1, \ldots, r_{i-1}, X_i, x_{i+1}, \ldots, x_v)$$

   $\mathcal{V}$ checks that g_i is a univariate polynomial of degree at most $\deg_i(g)$, and that $g_{i-1}(r_{i-1}) = g_i(0) + g_i(1)$. $\mathcal{V}$ chooses a random element $r_i \in \mathbb{F}$ and sends r_i to $\mathcal{P}$.

3. **Final Check:** After round v, $\mathcal{V}$ evaluates $g(r_1, \ldots, r_v)$ directly and verifies consistency with $g_v(r_v)$. $\mathcal{V}$ outputs *accept* if all checks pass.

Example: Full Tree Illustrating The Sum-Check Protocol (where $v = 3$)



Completeness. Similar argument to IP for SAT.

Theorem 1.1 (Schwartz–Zippel Lemma)**.** *Let $\mathbb{F}$ be a finite field. Let $p \in \mathbb{F}[x_1, \ldots, x_n]$ be a non-zero polynomial of total degree $d \geq 0$. Choose $r = (r_1, \ldots, r_n)$ uniformly at random from $\mathbb{F}^n$. Then*

$$\Pr_{r \in \mathbb{F}^n}[\, p(r) = 0 \,] \;\leq\; \frac{d}{|\mathbb{F}|}.$$

Soundness. Suppose $\mathcal{P}'$ sends a wrong polynomial g_i at some round i. Let g_i^* denote the correct polynomial at this round. Observe that $g_i - g_i^*$ is a non-zero polynomial of degree at most $\deg_i(g)$. The verifier chooses $r_i \in \mathbb{F}$ uniformly at random and continues only if $g_i(r_i) = g_i^*(r_i)$.

By Theorem 1.1, we know the following:

$$\Pr[\, g_i(r_i) - g_i^*(r_i) = 0 \,] \;\leq\; \frac{\deg_i(g)}{|\mathbb{F}|}.$$

If this equality check does not hold, then the verifier rejects immediately. Thus, the protocol continues with a wrong polynomial in round i with probability at most $\deg_i(g)/|\mathbb{F}|$.

Applying a union bound over all v rounds, the overall soundness error is bounded by the following:

$$\delta_s \;\leq\; \sum_{i=1}^{v} \frac{\deg_i(g)}{|\mathbb{F}|}.$$

In particular, if $d = \max_i \deg_i(g)$, then:

$$\delta_s \;\leq\; \frac{v \cdot d}{|\mathbb{F}|}.$$

Thus, by choosing $\mathbb{F}$ to be sufficiently large, the soundness error can be made arbitrarily small.

1.2 Multi-prover Interactive Proofs (MIP)

We can generalize interactive proofs by introducing multiple provers. These provers may coordinate on a joint strategy beforehand but are not able to communicate during the protocol.

Let $k \geq 2$. A *k-prover interactive proof system* for a language $\mathcal{L} \subseteq \{0,1\}^n$ involves $k+1$ parties: a probabilistic polynomial-time verifier $\mathcal{V}$ and k provers $\mathcal{P}_1, \ldots, \mathcal{P}_k$. Given a common input x, the provers and verifier exchange messages and the verifier will output either *accept* or *reject* at the end of the protocol:

$$\mathrm{out}(\mathcal{V}, x, r, \mathcal{P}_1, \ldots, \mathcal{P}_k) \in \{\mathrm{accept}, \mathrm{reject}\}$$

Definition 1.3 (Multi-Prover Interactive Proof System)**.** Let $(\mathcal{P}_1, \ldots, \mathcal{P}_k, \mathcal{V})$ be a k-party interactive proof system for a language $\mathcal{L} \subseteq \{0,1\}^n$. Then:

- **Completeness:** There exists a tuple of prover strategies $(\mathcal{P}_1, \ldots, \mathcal{P}_k)$ such that for every $x \in \mathcal{L}$,

$$\Pr_r[\mathrm{out}(\mathcal{V}, x, r, \mathcal{P}_1, \ldots, \mathcal{P}_k) = \mathrm{accept}] \geq 1 - \delta_c.$$

- **Soundness:** For every $x \notin \mathcal{L}$ and for every tuple of prover strategies $(\mathcal{P}'_1, \ldots, \mathcal{P}'_k)$,

$$\Pr_r[\text{out}(\mathcal{V}, x, r, \mathcal{P}'_1, \ldots, \mathcal{P}'_k) = \text{accept}] \leq \delta_s.$$

Here, δ_c denotes the completeness error and δ_s denotes the soundness error.

Remark. The system is considered valid if δ_c, δ_s are sufficiently small.

Remark. The primary advantage of multiple provers is non-adaptivity, i.e., preventing adaptive cheating strategies.

Example: MIP for SAT

Problem Definition

Let ϕ be a 3-CNF (each clause contains exactly 3 literals) formula over variables $x_1, \ldots, x_n$ with clauses $C_1, \ldots, C_m$. Again, the common input is ϕ, and the prover wishes to convince the verifier that $\phi \in \text{SAT}$.

Description of Protocol

1. **Verifier $\mathcal{V}$:** Sample a clause index $j \xleftarrow{\$} [m]$ uniformly at random and a position $t \xleftarrow{\$} \{1, 2, 3\}$ uniformly at random. Let the t-th literal in C_j be $\ell_{j,t}$ over variable $x_{i_{j,t}}$ (possibly negated).
2. **Queries:**
   - Send j to $\mathcal{P}_1$ and request the three bits $(b_{j,1}, b_{j,2}, b_{j,3}) \in \{0,1\}^3$ giving purported values of the underlying variables of C_j (in clause order).
   - Send (j, t) (or equivalently the variable index $i_{j,t}$) to $\mathcal{P}_2$ and request a single bit $c_{j,t} \in \{0, 1\}$ giving the purported value of $x_{i_{j,t}}$.
3. **Checks:**
   - *Clause check:* Using $(b_{j,1}, b_{j,2}, b_{j,3})$, evaluate C_j; require $C_j(b_{j,1}, b_{j,2}, b_{j,3}) = 1$.
   - *Consistency check:* Require that $c_{j,t}$ equals the t-th component $b_{j,t}$.
4. **Decision:** Accept iff both checks pass.

Completeness. If ϕ is satisfiable, fix a satisfying assignment $a^\star \in \{0,1\}^n$. The honest strategy is:

$$\mathcal{P}_1(j) \;=\; (a^\star_{i_{j,1}}, a^\star_{i_{j,2}}, a^\star_{i_{j,3}}), \qquad \mathcal{P}_2(j,t) \;=\; a^\star_{i_{j,t}}.$$

Then C_j is satisfied and the answers are consistent for all (j, t), so $\mathcal{V}$ accepts with probability 1.

Soundness. Suppose ϕ is unsatisfiable. Consider arbitrary (possibly cheating) strategies for $\mathcal{P}_1, \mathcal{P}_2$. Define a *derived global assignment* $\tilde{a} \in \{0,1\}^n$ as follows: for each variable x_i, let $\tilde{a}_i$ be the bit that $\mathcal{P}_2$ would most commonly answer when asked for x_i (break ties arbitrarily). Since ϕ is unsatisfiable, $\tilde{a}$ falsifies at least one clause; let $\mathcal{B} \subseteq [m]$ be the set of clauses falsified by $\tilde{a}$, so $\mathcal{B} \neq \emptyset$.

Fix any $j \in \mathcal{B}$. For clause C_j, one of the following must happen:

1. **(Clause failure)** $\mathcal{P}_1$'s triple $(b_{j,1}, b_{j,2}, b_{j,3})$ fails to satisfy C_j. Then the verifier rejects regardless of t.
2. **(Inconsistency)** $\mathcal{P}_1$ outputs a triple that *does* satisfy C_j. Since C_j is false under $\tilde{a}$, this triple must disagree with $\tilde{a}$ on at least one of the three variables of C_j. When the verifier's random $t \in \{1, 2, 3\}$ hits any position where $b_{j,t} \neq \tilde{a}_{i_{j,t}}$, the consistency check tends to fail because $\mathcal{P}_2$ answers (with high bias) according to $\tilde{a}_{i_{j,t}}$. In particular, conditioned on clause j, the probability (over t) of catching a disagreement is at least $1/3$.

Therefore, when j is chosen uniformly from $[m]$ and t uniformly from $\{1, 2, 3\}$, the verifier rejects with probability at least

$$\Pr[j \in \mathcal{B}] \cdot \frac{1}{3} \;\geq\; \frac{1}{m} \cdot \frac{1}{3} \;=\; \frac{1}{3m},$$

so the acceptance probability is at most $1 - \frac{1}{3m}$. By independent parallel repetition (running the one-round protocol k times and accepting only if all copies accept), the soundness error drops exponentially:

$$\Pr[\text{accept all } k \text{ copies}] \;\leq\; \left(1 - \frac{1}{3m}\right)^k.$$

Note: This is an illustrative example. We will discuss performance considerations in future sections.

Remark. This is the standard "oracularization" of an NP witness: $\mathcal{P}_1$ pretends to be a local view of the assignment on a random clause; $\mathcal{P}_2$ pretends to be a consistent oracle for single variables. The second prover enforces non-adaptivity/consistency, which is what you cannot get in the single-prover version without cryptographic tools. Two provers, one round already suffices for the usual MIP power phenomena (and is the canonical gateway to PCP).

1.3 Probabilistically Checkable Proofs (PCP)

In a PCP setting, the prover writes down a long proof string $\pi \in \{0,1\}^\ell$, and the verifier is allowed only *oracle access* to π (i.e., it may query a few positions). The verifier uses randomness to decide which positions of π to read, and must decide whether to accept or reject based on those few symbols.

In MIP, if a prover is asked multiple questions (multiple rounds), then the prover can behave adaptively. This adaptivity is potentially bad for soundness since the ability to behave adaptively makes it harder to "pin down" the prover(s) in a lie. PCP mitigates this (although at the cost of efficiency).

A *probabilistically checkable proof system (PCP)* for a language $\mathcal{L} \subseteq \{0,1\}^n$ consists of a probabilistic polynomial-time verifier $\mathcal{V}$ that is given access to a common input x and *oracle access* to a proof string $\pi \in \Sigma^\ell$ (where Σ is the proof alphabet and ℓ is the length of the proof).

Definition 1.4 (Probabilistically Checkable Proof System)**.** Let $\mathcal{V}^\pi$ be a probabilistically checkable proof system for a language $\mathcal{L} \subseteq \{0,1\}^n$. Then:

- **Completeness:** For every $x \in \mathcal{L}$, there exists a proof string $\pi \in \Sigma^\ell$ such that

$$\Pr[\mathcal{V}^\pi(x) = \text{accept}] \geq 1 - \delta_c$$

- **Soundness:** For every $x \notin \mathcal{L}$ and every proof string $\pi \in \Sigma^\ell$,

$$\Pr[\mathcal{V}^\pi(x) = \text{accept}] \leq \delta_s$$

Here, δ_c denotes the completeness error and δ_s denotes the soundness error.

Remark. The system is considered valid if δ_c, δ_s are sufficiently small.

Example: PCP for SAT

Problem Definition

Same as above.

Proof Generation

Suppose ϕ is satisfiable and let $a^\star \in \{0,1\}^n$ be a satisfying assignment. The prover constructs a proof string π that encodes $a^\star$ redundantly so that local consistency checks are possible. A canonical choice is to encode the *truth table of the assignment as a linear error-correcting code* (e.g., Hadamard or Reed–Muller code).

Intuitively: - Each variable assignment is spread across many positions of π. - Local queries to π allow the verifier to check both *consistency* (different parts of the proof agree on the same variable) and *clause satisfaction*.

Description of Protocol

1. **Verifier $\mathcal{V}$:** Pick a random clause index $j \xleftarrow{\$} [m]$.
2. **Verifier $\mathcal{V}$:** Query π at a small number of locations sufficient to decode the purported values of the three variables in clause C_j. Denote these values $(b_{j,1}, b_{j,2}, b_{j,3})$.
3. **Verifier $\mathcal{V}$:** Check that the tuple $(b_{j,1}, b_{j,2}, b_{j,3})$ is consistent with the error-correcting code used for π (this enforces that all queried bits come from some global assignment).
4. **Verifier $\mathcal{V}$:** Evaluate $C_j(b_{j,1}, b_{j,2}, b_{j,3})$. Accept if it evaluates to 1 and the consistency check passes; otherwise reject.

Completeness. If $\phi \in$ SAT with satisfying assignment $a^\star$, the honest prover encodes $a^\star$ into π using the agreed-upon code. Then: - Every local view of π is consistent with some assignment. - For every clause C_j, the tuple $(b_{j,1}, b_{j,2}, b_{j,3})$ derived from π makes C_j true. Thus, the verifier always accepts.

Soundness. Suppose $\phi \notin$ SAT. Then no assignment satisfies all clauses. Any proof string π necessarily encodes information corresponding (at best) to some assignment $\tilde{a}$. Since $\tilde{a}$ falsifies at least one clause, there exists some $j \in [m]$ for which $C_j(\tilde{a}) = 0$.

When the verifier samples this clause and decodes the corresponding tuple from π, one of two things happens:

- The decoded triple fails the clause check $\implies$ verifier rejects.
- The decoded triple passes the clause check but is inconsistent with the purported codeword $\implies$ verifier rejects in the consistency check.

Thus, with non-negligible probability (at least $1/m$), the verifier rejects.

By repeating with fresh randomness, the soundness error decreases exponentially.

Remark. This toy construction illustrates the PCP idea:

1. The prover commits to a long, redundant proof string π.
2. The verifier uses only a few random queries to π to check both consistency and clause satisfaction.

In practice, real PCP constructions (e.g., using low-degree tests and robust codes) achieve stronger guarantees: *constant query complexity* and *constant soundness error*, independent of n or m.

Linear PCP

A linear PCP is a special case of a PCP where the proof π is interpreted as a vector over a finite field $\mathbb{F}$ and the verifier's queries are linear functionals of the proof. This structure enables algebraic manipulation and is a building block for many efficient cryptographic protocols.

Definition 1.5 (Linear PCP)**.** A *linear probabilistically checkable proof* for a language $\mathcal{L} \subseteq \{0,1\}^n$ consists of a proof vector $\pi \in \mathbb{F}^\ell$ and a verifier $\mathcal{V}$ such that:

- The verifier chooses a random linear query $q \in \mathbb{F}^\ell$ and computes the inner product $\langle q, \pi \rangle$.
- **Completeness:** If $x \in \mathcal{L}$, the honest proof π satisfies $\mathcal{V}$'s checks with probability $\geq 1 - \delta_c$.
- **Soundness:** If $x \notin \mathcal{L}$, any π passes the verifier's checks with probability $\leq \delta_s$.

Remark (Intuition). Linear PCPs are useful because linear operations commute with many algebraic encodings (like error-correcting codes). The verifier can check consistency and correctness of the proof using only a few inner products, enabling succinct, locally checkable proofs.

1.4 Interactive Oracle Proofs (IOP)

An Interactive Oracle Proof is a unifying abstraction that generalizes both interactive proofs and PCPs. As in an IP, the verifier and prover exchange messages in multiple rounds. As in a PCP, the verifier can access messages from the prover non-adaptively via *oracle queries* rather than reading them in full.

Formally, an IOP proceeds in t rounds. In each round i:

1. The verifier $\mathcal{V}$ sends a message m_i.
2. The prover $\mathcal{P}$ replies with a string $\pi_i \in \Sigma^{\ell_i}$ (which is treated as an oracle).
3. The verifier may query at most q_i positions of π_i, (with queries chosen adaptively based on answers received so far in the round).

At the end of the protocol, the verifier outputs either *accept* or *reject*.

Definition 1.6 (Interactive Oracle Proof System)**.** Let $(\mathcal{P}, \mathcal{V})$ be a *interactive oracle proof system* for a language. Then:

- **Completeness:** For every $x \in \mathcal{L}$,

$$\Pr_r[\mathcal{V}^{\pi_1,\ldots,\pi_t}(x) = \text{accept}] \geq 1 - \delta_c,$$

  where $\pi_1, \ldots, \pi_t$ are the oracles provided by the honest prover strategy $\mathcal{P}$.

- **Soundness:** For every $x \notin \mathcal{L}$ and every (possibly cheating) prover strategy $\mathcal{P}'$ producing oracle strings $\pi'_1, \ldots, \pi'_t$,

$$\Pr_r[\mathcal{V}^{\pi'_1,\ldots,\pi'_t}(x) = \text{accept}] \leq \delta_s.$$

Here, δ_c denotes the completeness error and δ_s denotes the soundness error.

Remark. The system is considered valid if δ_c, δ_s are sufficiently small.

Observe that the IOP model reduces to an IP if the verifier reads each oracle in full and to a PCP if there is only one round and the verifier sends no messages.

IOP for SAT

Problem Definition

Same SAT setup as before: input ϕ is a CNF formula with clauses $C_1, \ldots, C_m$, and the prover wishes to convince the verifier that $\phi \in$ SAT.

Description of Protocol

1. **Round 1:** $\mathcal{P}$ sends π_1, which is an error-correcting encoding of a candidate assignment $a \in \{0,1\}^n$. $\mathcal{V}$ commits to a random clause index $j \in [m]$, but does not reveal it yet.
2. **Round 2:** $\mathcal{P}$ sends π_2, designed so that the verifier can query the three variables appearing in clause C_j using only a few locations of π_2.
3. **Verifier Queries:** $\mathcal{V}$ makes a small number of queries to π_1 and π_2:
   - From π_1: check that queried symbols are consistent with a valid codeword (linear consistency check).
   - From π_2: decode the tuple $(b_{j,1}, b_{j,2}, b_{j,3})$ corresponding to the three literals in C_j.
4. **Check and Decision:** $\mathcal{V}$ verifies that the tuple from π_2 is consistent with π_1, and that $C_j(b_{j,1}, b_{j,2}, b_{j,3}) = 1$. Accept if both checks pass; otherwise reject.

Completeness. If ϕ is satisfiable with assignment $a^\star$, the honest prover encodes $a^\star$ into both π_1 and π_2. - Consistency queries succeed since both are encodings of the same assignment. - Clause queries succeed since $a^\star$ satisfies every clause. Thus the verifier always accepts.

Soundness. If ϕ is unsatisfiable, then for any encoding π_1, π_2, there exists a clause C_j that is falsified by the derived assignment. With probability at least $1/m$, the verifier selects such a clause.

- If π_2 encodes values that falsify C_j, the clause check fails.
- If π_2 encodes values that make C_j true but are inconsistent with π_1, the consistency check fails.

Hence the verifier rejects with noticeable probability. By independent repetition, the soundness error decreases exponentially:

$$\Pr[\text{accept all } k \text{ repetitions}] \;\le\; \left(1 - \tfrac{1}{m}\right)^k.$$

1.5 Linear IOP

A linear IOP generalizes linear PCPs to multiple rounds with interactive messages. The verifier can still query the prover's messages as linear functions over a finite field. This allows for interaction while preserving algebraic structure.

Definition 1.7 (Linear Interactive Oracle Proof)**.** A *linear IOP* for a language $\mathcal{L} \subseteq \{0,1\}^n$ consists of:

- A multi-round interaction between a prover $\mathcal{P}$ and verifier $\mathcal{V}$.
- In each round i, the prover sends an oracle string $\pi_i \in \mathbb{F}^{\ell_i}$.
- The verifier makes a bounded number of *linear queries* $q_i \in \mathbb{F}^{\ell_i}$ and checks $\langle q_i, \pi_i \rangle$.
- **Completeness:** If $x \in \mathcal{L}$, the honest prover ensures $\mathcal{V}$ accepts with probability $\ge 1 - \delta_c$.
- **Soundness:** If $x \notin \mathcal{L}$, no prover strategy passes with probability more than δ_s.

Remark (Intuition)*.* Linear IOPs combine the algebraic structure of linear PCPs with interaction. This allows the verifier to check complex claims efficiently over multiple rounds while maintaining local linear checks. Many modern SNARK/STARK constructions are based on linear IOP frameworks.

1.6 Knowledge Properties

Beyond completeness and soundness, cryptographic applications often require stronger guarantees about what the prover "knows" and what the verifier "learns." Two fundamental refinements are *knowledge soundness* (also called *proof of knowledge*) and *zero-knowledge*.

We first formalize what the verifier "sees" during an execution.

Definition 1.8 (View)**.** Let $(\mathcal{P}, \mathcal{V})$ be an interactive proof. Let x be a common input. The *view* of the verifier $\mathcal{V}$ on the input x in an interaction with the prover $\mathcal{P}$ consists of the transcript (sequence of messages exchanged) and its internal randomness r, and internal state. We denote this distribution by:

$$\text{view}_{\mathcal{V}}(\langle \mathcal{P}, \mathcal{V} \rangle(x))$$

Definition 1.9 (Indistinguishability)**.** Some notions of indistinguishability:

- *Perfect Indistinguishability*: $D_1 = D_2$.
- *Statistical Indistinguishability*: for all n,

$$\Delta(D_1(n), D_2(n)) = \tfrac{1}{2} \sum_z \left| \Pr[D_1(n) = z] - \Pr[D_2(n) = z] \right| \leq \text{negl}(n).$$

- *Computational Indistinguishability*: for every probabilistic polynomial-time distinguisher $\mathcal{D}$ there exists a negligible function $\mu(\cdot)$ such that for all n,

$$\left| \Pr[\mathcal{D}(D_1(n)) = 1] - \Pr[\mathcal{D}(D_2(n)) = 1] \right| \leq \mu(n).$$

Definition 1.10 (Knowledge Extractor)**.** Let $(\mathcal{P}, \mathcal{V})$ be an interactive proof system for a relation R. A *knowledge extractor* $\mathcal{K}$ is a probabilistic polynomial-time algorithm with (black-box) oracle access to a possibly malicious prover $\mathcal{P}^*$, such that for any input $x \in L$ and any $\mathcal{P}^*$ that convinces $\mathcal{V}$ to accept x with non-negligible probability $\varepsilon(n)$, the extractor $\mathcal{K}^{\mathcal{P}^*}(x)$ outputs a witness w satisfying $(x, w) \in R$ with probability at least $\varepsilon(n) - \text{negl}(n)$, where $\text{negl}(n)$ is a negligible function in the security parameter n.

Remark (Intuition)*.* Intuitively, the extractor tries to "observe" the prover's strategy to obtain enough information to reconstruct a valid witness. In many protocols, this is accomplished via *rewinding*: the extractor runs the prover multiple times with different verifier randomness and uses the differences between accepting transcripts to efficiently compute a witness. This technique captures the idea that a prover cannot convince the verifier without actually "knowing" a valid witness.

Definition 1.11 (Simulator)**.** Let $(\mathcal{P}, \mathcal{V})$ be an interactive proof system for a language L. A *simulator* $\mathcal{S}$ is a probabilistic polynomial-time algorithm such that for every input $x \in L$, the distribution $\mathcal{S}(x)$ is (perfectly / statistically / computationally) indistinguishable from the view of the verifier when interacting with the honest prover:

$$\mathcal{S}(x) \approx \text{view}_{\mathcal{V}}(\langle \mathcal{P}, \mathcal{V} \rangle(x)).$$

Intuitively, $\mathcal{S}$ generates what the verifier would see in the real protocol *without any interaction with the prover*, thus capturing the idea that the protocol leaks no additional information.

Ordinary soundness only guarantees that if the verifier accepts, the statement is true. It does not imply that the prover "knows" a witness. Knowledge soundness strengthens this by requiring that any prover that convinces the verifier can be used to extract a valid witness.

Definition 1.12 (Proof of Knowledge / Knowledge Soundness)**.** An interactive proof system $(\mathcal{P}, \mathcal{V})$ for a relation R is a *proof of knowledge* if there exists a probabilistic polynomial-time algorithm $\mathcal{K}$ (the *knowledge extractor*) such that for every (possibly malicious) prover $\mathcal{P}^*$ and input x, if

$$\Pr[\langle \mathcal{P}^*(x), \mathcal{V}(x) \rangle = \text{accept}] \geq \varepsilon,$$

then $\mathcal{K}^{\mathcal{P}^*}(x)$ outputs a witness w such that $(x, w) \in R$ with probability at least $\varepsilon - \text{negl}(n)$.

Remark. The extractor $\mathcal{K}$ is given black-box access to the prover $\mathcal{P}^*$ and may use techniques such as *rewinding* to obtain multiple executions of the interaction. Intuitively: if a prover can convince the verifier, then it must "know" a valid witness, because the extractor can efficiently obtain one.

Zero-knowledge ensures that the verifier learns nothing beyond the validity of the statement. This is formalized by the existence of a simulator that can reproduce the verifier's view without interacting with the prover.

Definition 1.13 (Zero-Knowledge)**.** An interactive proof system $(\mathcal{P}, \mathcal{V})$ for a language L is *zero-knowledge* if for every probabilistic polynomial-time verifier $\mathcal{V}^*$ there exists a simulator $\mathcal{S}$ such that for all $x \in L$,

$$\text{view}_{\mathcal{V}^*}(\langle \mathcal{P}, \mathcal{V}^* \rangle(x)) \approx \mathcal{S}(x),$$

where $\approx$ denotes either perfect, statistical, or computational indistinguishability.

Combining both notions yields zero-knowledge proofs of knowledge (ZKPoKs), where the prover convinces the verifier of a statement, while simultaneously ensuring the verifier learns nothing beyond validity, and the prover must "know" a witness.

Definition 1.14 (Zero-Knowledge Proof of Knowledge)**.** An interactive proof system $(\mathcal{P}, \mathcal{V})$ for a relation R is a *zero-knowledge proof of knowledge* if it is both zero-knowledge and a proof of knowledge.

Remark. ZKPoKs are the foundation of many modern cryptographic protocols, including succinct arguments (SNARGs) and their efficient variants (SNARKs, STARKs).

2 NIZK Transforms

2.1 Fiat-Shamir Transform

Definition 2.1 (Public-Coin Interactive Proof)**.** A *public-coin interactive proof* for a language $L \subseteq \{0,1\}^*$ is an IP in which, for every round that the verifier sends a message, that message consists solely of the outcome of tossing a sequence of uniformly random and independently distributed public coins, i.e. $\mathcal{V}$ samples $r_i \xleftarrow{\$} \{0,1\}^*$ for every round i.

Example 2.1. *A classic example of a public-coin IP is the Sum-Check Protocol (described in Section 1.1).*

The purpose of the Fiat-Shamir transformation is to transform any public-coin IP I into a non-interactive publicly-verifiable protocol Q in the random oracle model.

A naive approach would be to use a random oracle to sample challenges (independent of $\mathcal{P}$); however, if these random oracle queries are fixed in advance, i.e., $r_i = \mathsf{RO}(i)$, then the prover could precompute all verifier challenges before sending any messages, potentially violating soundness.

Remark. This is a consequence of the adaptivity mentioned in Section 1.

The Fiat-Shamir transformation addresses this by making the i-th random challenge depend on all prover messages thus far, binding the challenge to the transcript and preventing the prover from gaining access to r_i before sending m_i.

Definition 2.2 (Fiat–Shamir Transformation)**.** Let $I = (\mathcal{P}, \mathcal{V})$ be an m-round public-coin interactive proof or argument for a language $L \subseteq \{0,1\}^*$, where in round i the challenge r_i is drawn uniformly from $\{0,1\}^{k_i}$. The *Fiat–Shamir transformation* produces a non-interactive protocol $Q = (P^{\mathsf{FS}}, V^{\mathsf{FS}})$ in the *random oracle model* as described in the protocol below.

Description of Protocol

Common input: $x \in \{0,1\}^*$

Prover P^{FS}:

1. Initialize an empty transcript $T_0 := \epsilon$.
2. For $i = 1$ to m:
   (a) Compute the next prover message g_i as in P on input (x, T_{i-1}).
   (b) Query the random oracle on $(x, g_1, \ldots, g_i)$ to obtain $r_i \in \{0,1\}^{k_i}$.
   (c) Update the transcript: $T_i := T_{i-1} \,\|\, g_i \,\|\, r_i$.
3. Output $\pi := (g_1, r_1, g_2, r_2, \ldots, g_m, r_m)$ as the non-interactive proof.

Verifier V^{FS}:

1. Parse π into $(g_1, r_1, \ldots, g_m, r_m)$.
2. For each $i \in [m]$, recompute r_i as $\mathsf{RO}(x, g_1, \ldots, g_i)$ and verify it matches the value in π.
3. If all checks pass, run V on (x, π) using the included (g_i, r_i) values, and output V's decision.

Remark. In the *hash chaining* optimization, the query in round i may instead be (x, i, r_{i-1}, g_i), with $r_0 := 0^{k_0}$, to reduce the size of hash inputs.

Remark. For *adaptive soundness* against adversaries that choose x, the input x must be included in every hash query.

3 Commitment Schemes

Definition 3.1 (Commitment Scheme)**.** A *commitment scheme* for a message space $\mathcal{M}$ is a pair of probabilistic polynomial-time algorithms $(\mathsf{Com}, \mathsf{Ver})$ where:

- $\mathsf{Com}(m; r) \to c$: produces a commitment c for a message $m \in \mathcal{M}$ (using randomness r)
- $\mathsf{Ver}(m, c, r) \to$ *accept* or *reject*: outputs *accept* if and only if $c \stackrel{?}{=} \mathsf{Com}(m; r)$ (and *reject* otherwise)

Such a scheme must satisfy the security properties of *binding* and *hiding*.

Definition 3.2 (Binding)**.** Let Com be a commitment algorithm. Then:

- *Perfect Binding:* For all messages $m_0 \neq m_1$ with any randomness r_0, r_1,
$$c = \mathsf{Com}(m_0; r_0) \implies \mathsf{Ver}(m_1, r_1, c) = \textit{reject}$$
- *Statistical Binding:* For all (even unbounded) adversaries $\mathcal{A}$,
$$\Pr\left[(m_0, r_0, m_1, r_1, c) \leftarrow \mathcal{A}(1^\lambda) : m_0 \neq m_1 \text{ and } \mathsf{Ver}(m_0, r_0, c) = \mathsf{Ver}(m_1, r_1, c) = \textit{accept}\right] \leq \mathrm{negl}(\lambda)$$
- *Computational Binding:* For all probabilistic polynomial-time adversaries $\mathcal{A}$,
$$\Pr\left[(m_0, r_0, m_1, r_1, c) \leftarrow \mathcal{A}(1^\lambda) : m_0 \neq m_1 \text{ and } \mathsf{Ver}(m_0, r_0, c) = \mathsf{Ver}(m_1, r_1, c) = \textit{accept}\right] \leq \mathrm{negl}(\lambda)$$

Intuitively: It should not be possible to change the committed value.

Definition 3.3 (Hiding)**.** Let Com be a commitment algorithm (with randomness space $\mathcal{R}$). Then:

- *Perfect Hiding:* For all messages m_0, m_1 with any randomness $r \in \mathcal{R}$,
$$\mathsf{Com}(m_0; r) \stackrel{d}{=} \mathsf{Com}(m_1; r),$$
- *Statistical Hiding:* For all (even unbounded) distinguishers $\mathcal{D}$,
$$\left| \Pr_{r \leftarrow \mathcal{R}}[\mathcal{D}(\mathsf{Com}(m_0; r)) = 1] - \Pr_{r \leftarrow \mathcal{R}}[\mathcal{D}(\mathsf{Com}(m_1; r)) = 1] \right| \leq \mathrm{negl}(\lambda),$$
- *Computational Hiding:* For all probabilistic polynomial-time distinguishers $\mathcal{D}$,
$$\left| \Pr_{r \leftarrow \mathcal{R}}[\mathcal{D}(\mathsf{Com}(m_0; r)) = 1] - \Pr_{r \leftarrow \mathcal{R}}[\mathcal{D}(\mathsf{Com}(m_1; r)) = 1] \right| \leq \mathrm{negl}(\lambda).$$

Intuitively: The commitment should not reveal anything about the committed value.

Remark. If a commitment scheme is *perfectly or statistically binding*, then it can be at most *computationally hiding*, since perfect/statistical binding requires that the committed value is essentially fixed and therefore some information about the message may be leaked to an unbounded adversary. Conversely, if a commitment scheme is *perfectly or statistically hiding*, then it can be at most *computationally binding*, since perfect/statistical hiding requires that multiple messages could produce the same commitment, making binding impossible to enforce against an unbounded adversary.

Example 3.1 (Hash-based Commitment with Nonce)**.** *Let $H : \{0,1\}^* \to \{0,1\}^\lambda$ be a collision-resistant hash function. We define a commitment scheme for messages $m \in \{0,1\}^*$ as follows:*

- $\mathsf{Com}(m; r) := H(m \,\|\, r)$ *(where r is a random nonce)*
- $\mathsf{Ver}(m, r, c) :=$ accept *if and only if* $c = H(m \,\|\, r)$.

Hiding. *Since r is chosen uniformly at random, the commitment c looks pseudorandom (assuming H behaves like a random oracle). Thus, c reveals no information about m.*

Binding. *By the collision resistance of H, it is infeasible to find $(m_0, r_0) \neq (m_1, r_1)$ such that $H(m_0 \,\|\, r_0) = H(m_1 \,\|\, r_1)$. Thus, the sender cannot later open the commitment to a different message.*

Remark (Security of Hash-based Commitments). The hash-based scheme given above is *computationally binding*, assuming the hash function H is collision-resistant, and *computationally hiding*, assuming H behaves like a random oracle. In particular, it is not statistically or perfectly hiding, since an unbounded adversary could always brute-force over the randomness r.

3.1 Polynomial Commitment Schemes

Definition 3.4 (Polynomial Commitment Scheme)**.** A *polynomial commitment scheme* for polynomials over a field $\mathbb{F}$ and degree bound d is a tuple of probabilistic polynomial-time algorithms $(\mathsf{Com}, \mathsf{Open}, \mathsf{Ver})$ where:

- $\mathsf{Com}(f) \to c$: produces a commitment c for a polynomial $f \in \mathbb{F}[X]$ with $\deg(f) \leq d$
- $\mathsf{Open}(f, x) \to (y, \pi)$: outputs an evaluation $y = f(x)$ and a proof π
- $\mathsf{Ver}(c, x, y, \pi) \to$ *accept* or *reject*: outputs *accept* if and only if $y = f(x)$

Again, such a scheme must satisfy the security properties of *binding* and optionally *hiding*. Additionally, such a scheme must satisfy the property of *correctness*.

Definition 3.5 (Correctness)**.** For any polynomial $f \in \mathbb{F}[X]$ of degree at most d and for all $x \in \mathbb{F}$,

$$c \leftarrow \mathsf{Com}(f) \text{ and } (y, \pi) \leftarrow \mathsf{Open}(f, x) \quad \Longrightarrow \quad \mathsf{Ver}(c, x, y, \pi) = \textit{accept}$$

Remark (Commitments vs. Polynomial Commitments). A standard commitment scheme allows one to commit to a *single value* $m \in \mathcal{M}$ in a way that is both binding and hiding. A polynomial commitment scheme extends this idea: here the committed object is an entire polynomial $f \in \mathbb{F}[X]$, and the goal is to efficiently prove and verify claims of the form "$f(x) = y$" without revealing the whole polynomial.

Remark (Setup Algorithms). Some commitment schemes (in particular, polynomial commitments such as KZG or IPA) additionally require a setup or parameter generation algorithm $\mathsf{KeyGen}(1^\lambda) \to \mathsf{pp}$ that outputs public parameters pp. For simplicity, we omit KeyGen from the abstract definition above, and assume that any necessary public parameters are fixed and available to all parties.

3.1.1 KZG

Problem Definition

Suppose we are given a polynomial $f(X) \in \mathbb{F}[X]$ of degree at most d, where $\mathbb{F}$ is a finite field. The goal is to allow a prover $\mathcal{P}$ to commit to f compactly and later convince a verifier $\mathcal{V}$ that $f(z) = y$ at some evaluation point $z \in \mathbb{F}$.

Description of Protocol

1. **Setup:** A trusted party samples $\tau \in \mathbb{F}$ uniformly at random and publishes public parameters:

$$\mathrm{PP} = \{g^{s^i} \mid i = 0, 1, \ldots, d\}$$

for a fixed generator g of a cyclic group $\mathbb{G}$ of prime order p, where a bilinear pairing $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is available. The value τ is discarded.

2. **Commit:** To commit to $f(X) = \sum_{i=0}^{d} a_i X^i$, compute

$$\mathsf{Com}(f) = g^{f(s)} = \prod_{i=0}^{d} \left(g^{s^i}\right)^{a_i}.$$

3. **Open:** To prove that $f(z) = y$, the prover computes the quotient polynomial

$$q(X) = \frac{f(X) - y}{X - z},$$

and outputs the proof

$$\pi = g^{q(s)}.$$

4. **Verify:** The verifier checks the pairing equation:

$$e(\mathsf{Com}(f)/g^y, g) \stackrel{?}{=} e(\pi, g^{s-z}).$$

Accept if the equation holds.

Correctness. If $f(z) = y$, then $(X - z)$ divides $f(X) - y$, so $f(s) - y = q(s) \cdot (s - z)$. Thus:

$$e(\mathsf{Com}(f)/g^y, g) = e(g^{f(s)-y}, g) = e(g^{q(s)(s-z)}, g) = e(g^{q(s)}, g^{s-z}) = e(\pi, g^{s-z}),$$

and the verification succeeds.

Binding. The scheme is *computationally binding*. If a prover could open the same commitment $\mathsf{Com}(f)$ to two different evaluations (z, y) and (z, y') with $y \neq y'$, this would yield a nontrivial relation among the public parameters $\{g^{s^i}\}$ and break the *strong q-SDH assumption*. Thus, except with negligible probability, $\mathcal{P}$ cannot equivocate.

Remark. KZG is *perfectly binding* given the structured reference string: each polynomial f maps to a unique group element $g^{f(s)}$. The computational assumption ensures that $\mathcal{P}$ cannot forge different polynomials that collide under evaluation at s.

Hiding. The basic KZG commitment is *not hiding*: $\mathsf{Com}(f) = g^{f(s)}$ directly reveals information about $f(s)$. To obtain hiding, one can add randomness:

$$\mathsf{Com}(f; r) = g^{f(s)} \cdot h^r,$$

for independent generator h, where $r \stackrel{\$}{\leftarrow} \mathbb{F}$. This makes the scheme *perfectly hiding* while preserving binding.

Remark. The addition of a random blinding factor h^r is analogous to Pedersen commitments: it ensures that the commitment leaks no information about f beyond what is later revealed.

3.1.2 IPA-Based

Overview

The *inner product argument* (IPA) is not a standalone polynomial commitment scheme, but rather a **building block** used in Bulletproofs-style constructions. The idea is to commit to the coefficient vector $a = (a_0, \ldots, a_{n-1})$ of a polynomial $f(X)$ and then use IPA to prove evaluation claims of the form $f(z) = y$ efficiently. This achieves $O(\log n)$ communication complexity without requiring a trusted setup.

—

Inner Product Argument (IPA)

Suppose a prover $\mathcal{P}$ has two vectors $a, b \in \mathbb{F}^n$ and wants to convince a verifier $\mathcal{V}$ that

$$c = \langle a, b \rangle = \sum_{i=1}^{n} a_i b_i$$

without revealing the vectors themselves.

Description of Protocol

1. **Setup:** Let $G = (g_1, \ldots, g_n)$ and $H = (h_1, \ldots, h_n)$ be public group generators in a cyclic group $\mathbb{G}$ of prime order q. The prover commits to the vectors:

$$P = \prod_{i=1}^{n} g_i^{a_i} \cdot h_i^{b_i} \in \mathbb{G}.$$

2. **Recursive Folding:** The prover and verifier recursively reduce the vectors in $\log_2(n)$ rounds. At each round, the vectors are split into left/right halves and folded using a verifier-chosen challenge $x \in \mathbb{F}^\times$. The prover sends intermediate group elements $L, R \in \mathbb{G}$; these maintain consistency of the commitment under folding.

3. **Final Step:** After $\log n$ rounds, the vectors reduce to length 1: $a = (a_1)$, $b = (b_1)$. The prover reveals (a_1, b_1), and the verifier checks both:

$$c \stackrel{?}{=} a_1 b_1,$$

and the consistency of all folded commitments.

—

Correctness. If the prover is honest, then at each folding step the inner product is preserved:

$$\langle a, b \rangle = \langle a_L, b_L \rangle + \langle a_R, b_R \rangle \quad \rightsquigarrow \quad \langle a', b' \rangle$$

after folding with challenge x. By induction, the final scalar check $c = a_1 b_1$ succeeds, ensuring the verifier always accepts.

—

Binding. The scheme is *computationally binding*. Once P is fixed, a malicious prover cannot equivocate and open it to a different inner product $c' \neq c$ without breaking the discrete logarithm assumption (or the knowledge-of-exponent assumption) in $\mathbb{G}$. The recursive L, R commitments enforce consistency of the opening across all rounds, preventing equivocation.

—

Hiding. The basic IPA commitment

$$P = \prod_{i=1}^{n} g_i^{a_i} \cdot h_i^{b_i}$$

is not hiding, since P may leak information about the vectors (a, b). To achieve hiding, one can add a blinding factor:

$$P' = P \cdot u^r, \quad r \xleftarrow{\$} \mathbb{F},\ u \in \mathbb{G}.$$

This yields *computationally hiding* commitments while retaining binding.

—

IPA in Polynomial Commitment Schemes

To commit to a polynomial $f(X) = \sum_{i=0}^{n-1} a_i X^i$, the prover commits to its coefficient vector $a = (a_0, \ldots, a_{n-1})$ using the IPA commitment scheme. To prove that $f(z) = y$, observe:

$$f(z) = \langle a, (1, z, z^2, \ldots, z^{n-1}) \rangle.$$

Thus, proving $f(z) = y$ reduces to running an IPA with a and $b = (1, z, \ldots, z^{n-1})$.

Properties.

- **Correctness:** Honest proofs always verify since the inner product argument preserves $f(z)$.
- **Binding:** Computational binding holds under the discrete logarithm assumption in $\mathbb{G}$.
- **Hiding:** Achieved by adding a Pedersen-style blinding factor.
- **Efficiency:** Communication is $O(\log n)$, making this approach scalable.
- **Setup:** No trusted setup is required (transparent setup).

3.1.3 FRI-Based

Overview

The *Fast Reed–Solomon Interactive Oracle Proof of Proximity (FRI)* protocol is the basis for modern transparent polynomial commitment schemes (e.g., STARKs). FRI enables a prover to commit to a polynomial $f(X)$ of bounded degree and then prove, with logarithmic communication, that f evaluates to a given value at some point $z \in \mathbb{F}$, without revealing f.

Unlike KZG, FRI does not rely on pairings or structured reference strings. Instead, it exploits the fact that Reed–Solomon codes have good proximity properties that can be checked via low-degree tests.

—

Description of Protocol

1. **Commit:** The prover commits to the polynomial $f(X)$ by publishing evaluations of f over a large evaluation domain $D \subseteq \mathbb{F}$ (via a Merkle tree). The Merkle root acts as the commitment.
2. **Low-Degree Test (LDT):** To prove that f has degree at most d, the prover and verifier engage in the FRI protocol: the verifier samples random points and the prover responds with evaluations on successively folded polynomials. Each round halves the degree bound.
3. **Open:** To prove that $f(z) = y$, the prover reveals $f(z) = y$ along with Merkle authentication paths and FRI queries demonstrating both consistency of the evaluations and the low-degree property.
4. **Verify:** The verifier checks:
   - Merkle proofs for consistency with the committed codeword,
   - Correctness of the folding steps in the FRI protocol,
   - That the claimed evaluation $f(z) = y$ is consistent with the committed codeword.

—

Correctness. If the prover is honest and f is a polynomial of degree at most d, then:

- All Merkle openings are consistent with the original commitment,
- The recursive folding steps preserve the low-degree property,

- The revealed value $f(z) = y$ matches the committed evaluation.

Thus, the verifier accepts with overwhelming probability.

—

Binding. The scheme is *computationally binding*, assuming the collision resistance of the Merkle hash function. Once the Merkle root is fixed, the prover cannot later equivocate to two different polynomials (or two different evaluations at the same point) without finding a Merkle collision.

—

Hiding. The basic FRI-based commitment (Merkle root of evaluations) is not hiding, since the committed codeword may leak information about f. To achieve hiding, the prover can add random blinding polynomials to the evaluation table before committing, in a manner analogous to adding randomness in Pedersen commitments. This yields *computationally hiding* commitments, assuming the hash function behaves as a random oracle.

—

Properties

- **Correctness:** Honest proofs always verify.
- **Binding:** Computationally binding under collision resistance of the hash function.
- **Hiding:** Achieved by randomizing/blinding the evaluation table before commitment.
- **Efficiency:** Verification complexity is polylogarithmic in the degree; proof size is $O(\log^2 n)$ in practice.
- **Setup:** Transparent — requires only a hash function, no trusted setup.

Remark. FRI-based polynomial commitments form the foundation of STARK proof systems. They are highly scalable, post-quantum secure (hash-based), and transparent, though proof sizes are larger than in pairing-based schemes such as KZG.

4 Argument Systems

4.1 Σ-Protocols

Definition 4.1 (Σ-Protocol)**.** A Σ-protocol is a three-move public-coin interactive proof $(\mathcal{P}, \mathcal{V})$ for a language $L \subseteq \{0,1\}^*$ (with common input x). It satisfies the following structure:

1. **Commitment:** The prover $\mathcal{P}$ sends a message a to the verifier $\mathcal{V}$.
2. **Challenge:** The verifier $\mathcal{V}$ replies with a uniformly random challenge c.
3. **Response:** The prover sends a response z such that the verifier can check $\phi(x, a, c, z) \stackrel{?}{=} \textit{accept}$.

Remark. In a Σ-protocol, we use *special soundness* rather than the previously-defined notion of soundness. Soundness only guarantees that a prover without a witness cannot convince the verifier of a false statement, except with negligible probability. Special soundness, on the other hand, is an *extractability* property: given two accepting transcripts with the same first message but different challenges, one can efficiently compute a valid witness. This stronger notion is what allows Σ-protocols to serve as *proofs of knowledge*, and it underlies transformations such as the Fiat–Shamir heuristic.

Definition 4.2 (Special Soundness)**.** A Σ-protocol for a relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ has *special soundness* if there exists a polynomial-time algorithm Ext such that for every $x \in \{0,1\}^*$, and for any two accepting transcripts (a, e, z), (a, e', z') with the same first message a, distinct challenges $e \neq e'$, and both accepted by the verifier on input x, the extractor outputs a witness $w \leftarrow \mathsf{Ext}(x, a, e, z, e', z')$ such that $(x, w) \in R$.

A Σ-protocol also satisifies the following zero-knowledge property:

Definition 4.3 (Honest-Verifier Zero-Knowledge)**.** A Σ-protocol for a relation R has *honest-verifier zero-knowledge* (HVZK) if there exists a probabilistic polynomial-time simulator Sim such that for every $(x, w) \in R$, the distribution of transcripts produced by the simulator $\mathsf{Sim}(x)$ is identically distributed (or *computationally indistinguishable*) from the distribution of transcripts generated in a real execution of the protocol between $P(x, w)$ and an honest verifier $V(x)$. In other words, an honest verifier learns nothing beyond the validity of the statement $x \in L = \{\, x : \exists w \ (x, w) \in R \,\}$.

Remark. Intuitively, this shows an honest verifier learns nothing beyond the fact that the prover knows the discrete logarithm w.

4.1.1 Schnorr Protocol

Premise The Schnorr protocol is a classic example of a Σ-protocol, used to prove knowledge of a discrete logarithm. Let $\mathbb{G}$ be a cyclic group of prime order q with generator g. The public input is $x = g^w \in \mathbb{G}$, and the witness is the secret exponent $w \in \mathbb{Z}_q$ such that $(x, w) \in R = \{(g^w, w)\}$.

Description of Protocol

1. **Commitment:** The prover samples $r \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ uniformly at random and sends $a = g^r \in \mathbb{G}$ to the verifier.
2. **Challenge:** The verifier samples $c \stackrel{\$}{\leftarrow} \mathbb{Z}_q$ uniformly at random and sends c to the prover.
3. **Response:** The prover computes $z = r + c \cdot w \pmod q$ and sends z to the verifier.
4. **Verification:** The verifier accepts if $g^z \stackrel{?}{=} a \cdot x^c$.

Completeness If both prover and verifier follow the protocol honestly, then verification always holds:

$$g^z = g^{r+cw} = g^r \cdot (g^w)^c = a \cdot x^c$$

Special Soundness Suppose a cheating prover produces two valid accepting transcripts (a, c, z) and (a, c', z') with the same a but different challenges $c \neq c'$. Then from the responses:

$$z = r + cw \pmod{q}, \quad z' = r + c'w \pmod{q},$$

we can subtract to obtain

$$(z - z') \equiv (c - c')w \pmod{q}$$

Since $c \neq c'$, this gives

$$w \equiv (z - z') \cdot (c - c')^{-1} \pmod{q}.$$

Thus the witness w can be efficiently extracted, demonstrating *special soundness*.

Honest-Verifier Zero-Knowledge To show HVZK, we define a simulator $\mathsf{Sim}(x)$: sample $c \xleftarrow{\$} \mathbb{Z}_q$, sample $z \xleftarrow{\$} \mathbb{Z}_q$, and set

$$a = g^z \cdot x^{-c}.$$

Then output the transcript (a, c, z). This distribution is identical to that of real transcripts between an honest prover and an honest verifier, so the protocol is HVZK.

References

- Ronald Cramer. *Modular design of secure yet practical cryptographic protocols.* PhD thesis, University of Amsterdam, 1996.
- Claus-Peter Schnorr. *Efficient identification and signatures for smart cards.* In Advances in Cryptology—CRYPTO '89, Lecture Notes in Computer Science, vol. 435, pages 239–252. Springer, 1990.

4.2 SNARKs

Definition 4.4 (Circuit Satisfiability)**.** Let $C : \{0,1\}^n \to \{0,1\}^m$ be a Boolean circuit. A *satisfying assignment* for C is an input $w \in \{0,1\}^n$ such that $C(w) = y$ for a given public output $y \in \{0,1\}^m$. The *circuit satisfiability problem* asks: given a circuit C and output y, determine whether there exists a witness w such that $C(w) = y$.

Remark. Circuit satisfiability is an NP-complete problem.

Remark. While circuit satisfiability on its own is a well-defined computational problem, proving that a witness exists is trivial in the sense that anyone who knows the witness can simply reveal it. The power of SNARKs comes from adding *zero-knowledge*: the prover can convince the verifier that a satisfying assignment exists without revealing any information about the witness itself.

4.2.1 Groth16

Groth16 [**?**] is one of the most widely deployed succinct non-interactive arguments of knowledge (SNARKs), used in systems such as Zcash and numerous blockchain applications. It achieves extremely short proofs (just three group elements) and fast verification, making it particularly well-suited for practical deployment.

Arithmetization

R1CS The first step is to represent the computation as a *Rank-1 Constraint System* (R1CS). An R1CS consists of a sequence of constraints over a finite field $\mathbb{F}$, each of the form:

$$\langle \mathbf{a}_i, \mathbf{z} \rangle \cdot \langle \mathbf{b}_i, \mathbf{z} \rangle \;=\; \langle \mathbf{c}_i, \mathbf{z} \rangle \quad \forall i \in [m],$$

where $\mathbf{a}_i, \mathbf{b}_i, \mathbf{c}_i \in \mathbb{F}^n$ are fixed vectors, and $\mathbf{z} \in \mathbb{F}^n$ is the vector of variables, which includes both the witness and auxiliary variables.

Equivalently, each constraint enforces that the componentwise linear combinations $\langle \mathbf{a}_i, \mathbf{z} \rangle$ and $\langle \mathbf{b}_i, \mathbf{z} \rangle$ multiply to produce $\langle \mathbf{c}_i, \mathbf{z} \rangle$. A satisfying assignment $\mathbf{z}$ simultaneously satisfies all constraints.

R1CSs are expressive enough to capture any computation in NP. Indeed, given a Boolean or arithmetic circuit, one can systematically translate each gate into a corresponding constraint. For example, an addition gate $z_k = z_i + z_j$ can be encoded as

$$(z_i + z_j - z_k) \cdot 1 = 0,$$

and a multiplication gate $z_k = z_i \cdot z_j$ can be encoded as

$$z_i \cdot z_j = z_k.$$

Quadratic Arithmetic Programs (QAPs) To enable succinct proofs, R1CS constraints are further encoded into a *Quadratic Arithmetic Program* (QAP). A QAP represents the sequence of constraints as polynomial identities. Specifically, one constructs three families of polynomials $\{A_i(X)\}, \{B_i(X)\}, \{C_i(X)\}$ over $\mathbb{F}$, such that a valid witness assignment $\mathbf{z}$ corresponds to a polynomial relation:

$$\Big(\sum_{i=0}^{n} z_i A_i(X)\Big) \cdot \Big(\sum_{i=0}^{n} z_i B_i(X)\Big) - \Big(\sum_{i=0}^{n} z_i C_i(X)\Big)$$

is divisible by a fixed target polynomial $T(X)$ that encodes the set of constraints.

This divisibility condition ensures that all R1CS constraints are satisfied simultaneously. The QAP representation is the foundation on which Groth16 builds its succinct polynomial commitment scheme using pairings.

Target Polynomial and Lagrange Basis Let $\tau_1, \ldots, \tau_m \in \mathbb{F}$ be the evaluation domain. These are distinct points associated with the m constraints of the R1CS.
To express the constraint polynomials, one often uses the *Lagrange basis polynomials*:

$$L_j(X) = \prod_{\substack{1 \le k \le m \\ k \ne j}} \frac{X - \tau_k}{\tau_j - \tau_k},$$

which satisfy $L_j(\tau_k) = \delta_{jk}$. Then, for each variable z_i, the polynomials $A_i(X), B_i(X), C_i(X)$ can be written as

$$A_i(X) = \sum_{j=1}^{m} a_{i,j} L_j(X), \quad B_i(X) = \sum_{j=1}^{m} b_{i,j} L_j(X), \quad C_i(X) = \sum_{j=1}^{m} c_{i,j} L_j(X),$$

where $(a_{i,j}, b_{i,j}, c_{i,j})$ are the R1CS coefficients for the j-th constraint.

The target polynomial $T(X)$ is typically defined as

$$T(X) = \prod_{j=1}^{m} (X - \tau_j),$$

Each constraint is enforced at its corresponding evaluation point τ_j.

This Lagrange interpolation ensures that evaluating (A_i, B_i, C_i) at τ_j enforces the j-th constraint, while divisibility by $T(X)$ guarantees that all constraints hold simultaneously.

Protocol Intuition

Trace Table To better understand Groth16 in the context of interactive oracle proofs (IOPs), it is helpful to describe circuit satisfiability in terms of a *trace table*. The trace records the inputs, intermediate values, and outputs of each gate in the circuit.

Example 4.1 (Trace Table for a Small Circuit)**.** *Consider a circuit over a field $\mathbb{F}$ with inputs $x_1, x_2 \in \mathbb{F}$, and gates defined as:*

$$z_3 = x_1 + x_2, \quad z_4 = z_3 \cdot x_1, \quad z_5 = z_4 - 1.$$

A satisfying witness (x_1, x_2) induces the following table:

Gate	*Type*	*Inputs*	*Output*
z_1	*input*	–	x_1
z_2	*input*	–	x_2
z_3	*add*	z_1, z_2	$x_1 + x_2$
z_4	*mul*	z_3, z_1	$(x_1 + x_2) \cdot x_1$
z_5	*sub*	$z_4, 1$	$(x_1 + x_2) \cdot x_1 - 1$

Correctness Conditions For the table to represent a valid computation, the following checks must hold:

1. **Input check:** The initial rows encode the given public inputs correctly.
2. **Gate check:** Each gate's output is computed correctly from its inputs (e.g., addition, multiplication).
3. **Wiring check:** The wiring of the circuit is respected; i.e., whenever an input to a gate is supposed to be a copy of some previous output, the table entries agree.
4. **Output check:** The final designated output gate equals the claimed value (e.g., $z_5 = 0$ in the above example).

From Trace Checks to Polynomial Identities Groth16 enforces all of these conditions using a single type of algebraic constraint: a *zero test*. To require that a polynomial $P(X)$ vanishes on a set of evaluation points $\{\tau_1, \ldots, \tau_m\}$, one checks that

$$T(X) = \prod_{j=1}^{m} (X - \tau_j) \quad \Big| \quad P(X).$$

Here $T(X)$ is the vanishing polynomial of the constraint domain.

Definition 4.5 (Zero Test)**.** Let $D = \{\tau_1, \ldots, \tau_m\} \subseteq \mathbb{F}$ be a finite set of evaluation points, and let $T(X) = \prod_{j=1}^{m} (X - \tau_j)$ be the corresponding vanishing polynomial.

A polynomial $P(X) \in \mathbb{F}[X]$ passes the *zero test on* D if

$$P(\tau_j) = 0 \quad \text{for all } \tau_j \in D.$$

Equivalently, $P(X)$ passes the zero test on D if and only if $T(X)$ divides $P(X)$ in $\mathbb{F}[X]$:

$$T(X) \ \Big| \ P(X).$$

In other words, the zero test certifies that the polynomial identity $P(X) = H(X) \cdot T(X)$ holds for some $H(X) \in \mathbb{F}[X]$, which guarantees that all constraints indexed by D are satisfied simultaneously.

How Groth16 Implements These Checks Groth16 begins with an R1CS encoding of the computation, then transforms it into a QAP. Each constraint corresponds to an evaluation point τ_j, and the prover's assignment produces polynomials

$$A(X) = \sum_i z_i A_i(X), \quad B(X) = \sum_i z_i B_i(X), \quad C(X) = \sum_i z_i C_i(X).$$

The central QAP condition requires that

$$A(X) \cdot B(X) - C(X) = H(X) \cdot T(X),$$

for some polynomial $H(X)$.

Divisibility by $T(X)$ certifies that every constraint is satisfied at its designated point. Wiring is enforced directly by reusing the same variable index z_i wherever a wire appears, so no additional permutation argument is required.

Thus, Groth16 reduces all correctness conditions for circuit satisfiability to a single zero-test polynomial identity.

Setup

CRS. Let C be an arithmetic circuit with n variables, m multiplication gates, and ℓ public inputs. From C we derive a QAP $(A_i(X), B_i(X), C_i(X))_{i=0}^n$ over a field $\mathbb{F}$ with vanishing polynomial $Z(X) = \prod_{k=1}^m (X - r_k)$ for some evaluation points $\{r_1, \ldots, r_m\}$.

A trusted setup samples trapdoors

$$\tau, \alpha, \beta, \gamma, \delta \xleftarrow{\$} \mathbb{F},$$

and encodes them into group elements in bilinear groups $(\mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T)$ with pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$.

The Common Reference String (CRS) is divided into:

- **Verification key (VK):** Contains the group elements $[\alpha] \in \mathbb{G}_1$, $[\beta] \in \mathbb{G}_2$, $[\gamma] \in \mathbb{G}_2$, $[\delta] \in \mathbb{G}_2$, and the input consistency elements

$$\mathrm{IC}_j = \Big[\frac{\beta A_j(\tau) + \alpha B_j(\tau) + C_j(\tau)}{\gamma}\Big] \in \mathbb{G}_1 \quad \text{for } j = 0, \ldots, \ell.$$

- **Proving key (PK):** Contains $\{[A_i(\tau)] \in \mathbb{G}_1,\ [B_i(\tau)] \in \mathbb{G}_2,\ [C_i(\tau)] \in \mathbb{G}_1\}_{i=0}^n$, together with

$$\Big[\frac{\beta A_i(\tau) + \alpha B_i(\tau) + C_i(\tau)}{\delta}\Big] \in \mathbb{G}_1 \quad \text{for } i = 0, \ldots, n,$$

  as well as encodings of the evaluation basis $[1], [\tau], [\tau^2], \ldots, [\tau^d] \in \mathbb{G}_1$ sufficient for committing to degree-d polynomials.

This setup is *circuit-specific*, since the CRS explicitly encodes the QAP selector polynomials A_i, B_i, C_i of C evaluated at τ. Any change in the circuit requires generating a fresh CRS.

Remark. A useful way to view Groth16 is as a polynomial commitment scheme tailored to quadratic arithmetic programs (QAPs). The common reference string encodes powers of a secret value τ, allowing the prover to commit to polynomials "at the point τ" using only linear combinations of CRS elements. The verifier then uses bilinear pairings to check divisibility relations between these committed polynomials, without ever learning τ. This perspective highlights the role of the structured reference string: it enables succinct commitments (just three group elements) and enforces knowledge soundness through trapdoor scalars $\alpha, \beta, \gamma, \delta$.

Description of Protocol

Let C be a circuit with an associated R1CS instance of size m and variables $(z_1, \ldots, z_n)$ over a field $\mathbb{F}$. Let $\{A_i(X)\}, \{B_i(X)\}, \{C_i(X)\}$ be the QAP polynomials, and let $T(X)$ be the target polynomial. The Groth16 protocol consists of three phases:

1. **Setup:**
   - Choose a secret evaluation point $\tau \in \mathbb{F}$ (known only during setup).
   - Generate structured reference strings (SRS) containing group encodings of τ^j, and additionally encodings of $\alpha, \beta, \gamma, \delta \in \mathbb{F}$ and their multiples by τ^j.
   - Publish the proving key pk and verification key vk.
2. **Proving:** Given a satisfying assignment $\mathbf{z} = (z_1, \ldots, z_n)$, the prover computes

$$A(X) = \sum_{i=0}^n z_i A_i(X), \quad B(X) = \sum_{i=0}^n z_i B_i(X), \quad C(X) = \sum_{i=0}^n z_i C_i(X).$$

   By the QAP construction, if $\mathbf{z}$ is valid then

$$A(X) \cdot B(X) - C(X) = H(X) \cdot T(X)$$

for some polynomial $H(X)$. Using the proving key, the prover commits (in the exponent) to $A(\tau), B(\tau), C(\tau)$ and $H(\tau)$, and applies randomizers to ensure zero-knowledge. The resulting proof $\pi = (\pi_A, \pi_B, \pi_C)$ consists of three group elements.

3. **Verification:** The verifier checks a single pairing-based equation:

$$e(\pi_A, \pi_B) \stackrel{?}{=} e(\alpha, \beta) \cdot e(\mathsf{vk}_x, \gamma) \cdot e(\pi_C, \delta),$$

where vk_x is the verification key component that encodes the public inputs. If the equality holds, the verifier accepts.

Remark. Although Groth16 superficially resembles polynomial commitment schemes such as KZG — both rely on a structured reference string encoding $(1, \tau, \tau^2, \dots)$ — Groth16 does not provide a reusable polynomial commitment. Instead, the CRS is circuit-specific and only enables evaluation of the QAP polynomials at the hidden trapdoor τ. In contrast, KZG yields a universal, binding commitment that can be opened at arbitrary points.

Remark (Circuit-specific trusted setup). Let $\{A_i(X)\}_{i=0}^n, \{B_i(X)\}_{i=0}^n, \{C_i(X)\}_{i=0}^n$ be the QAP selector polynomials for a *fixed* circuit (R1CS instance), and let $\tau, \alpha, \beta, \gamma, \delta \xleftarrow{\$} \mathbb{F}$ be the trapdoors used in setup. The Groth16 CRS (split into proving and verification material) contains group encodings of *evaluations that depend on these polynomials at the hidden point* τ. Schematic examples (omitting routine powers of τ and degree bounds) include:

$$\text{VK:} \quad [\alpha] \in \mathbb{G}_1,\ [\beta] \in \mathbb{G}_2,\ [\gamma] \in \mathbb{G}_2,\ [\delta] \in \mathbb{G}_2, \text{ and } \mathrm{IC}_j \ = \ \left[\tfrac{\beta A_j(\tau) + \alpha B_j(\tau) + C_j(\tau)}{\gamma}\right] \in \mathbb{G}_1,$$

for $j = 0, \dots, \ell$ (public-input selectors), and in the proving key,

$$\left\{[A_i(\tau)] \in \mathbb{G}_1,\ [B_i(\tau)] \in \mathbb{G}_2,\ [C_i(\tau)] \in \mathbb{G}_1,\ \left[\tfrac{\beta A_i(\tau) + \alpha B_i(\tau) + C_i(\tau)}{\delta}\right] \in \mathbb{G}_1\right\}_{i=0}^n,$$

together with encodings of $[1], [\tau], \dots, [\tau^d]$ (for the relevant degree d).

Crucially, these CRS elements *embed the specific polynomials of this circuit* via the values $A_i(\tau), B_i(\tau), C_i(\tau)$ and their linear combinations with $\alpha, \beta, \gamma, \delta$. If one changes the circuit to a different instance with selector families $\{A'_i, B'_i, C'_i\}$, then the previously published encodings no longer match:

$$\mathrm{IC}'_j \ = \ \left[\tfrac{\beta A'_j(\tau) + \alpha B'_j(\tau) + C'_j(\tau)}{\gamma}\right] \ \neq \ \mathrm{IC}_j \quad \text{in general,}$$

and the pairing equation used in verification will not hold for the new circuit. Therefore a fresh trusted setup (new CRS tied to $\{A'_i, B'_i, C'_i\}$) is required.

In short, Groth16's CRS is *circuit-/QAP-specific* because it contains group elements that are linear combinations of the circuit's selector polynomials evaluated at the hidden point τ. This contrasts with universal/updatable-CRS schemes (e.g., Plonk/Marlin), where the CRS does not embed circuit-specific selector evaluations and can be reused across many circuits.

Remark (On the role of $\alpha, \beta, \gamma, \delta$). The trapdoor elements α, β are used in the CRS to tie the polynomials A, B, C together in the final pairing equation, ensuring the prover cannot manipulate them independently. The elements γ, δ serve a different purpose: they enforce knowledge soundness and provide slots for blinding, so that the proof reveals nothing about the witness beyond the statement's validity.

Remark (Succinctness). A Groth16 proof consists of only three group elements, regardless of the circuit size. Verification requires a constant number of pairings, which makes Groth16 one of the most efficient SNARKs in terms of proof size and verification complexity.

Completeness If the prover follows the protocol honestly with a satisfying witness $\mathbf{z}$, then the QAP divisibility condition

$$A(X) \cdot B(X) - C(X) = H(X) \cdot T(X)$$

holds. Since the prover evaluates these polynomials at τ consistently, the commitments π_A, π_B, π_C satisfy the pairing equation used in verification. Thus the verifier always accepts an honest proof.

Knowledge Soundness Suppose a malicious prover produces a proof π that passes verification. By the structure of the pairing equation, this implies that the committed values satisfy

$$A(\tau) \cdot B(\tau) - C(\tau) = H(\tau) \cdot T(\tau)$$

for some polynomial $H(X)$. Since τ is chosen randomly during setup and hidden in the SRS, the prover cannot force this equality to hold at τ without ensuring that the divisibility condition holds as a polynomial identity. Therefore, if a prover can produce valid proofs with non-negligible probability, there exists an efficient extractor that, given oracle access to the prover, can extract a valid witness $\mathbf{z}$ for the R1CS instance. This establishes the *knowledge soundness* of Groth16.

Zero-Knowledge To ensure zero-knowledge, the prover randomizes the proof with fresh blinding terms during the commitment phase. Concretely, the prover adds random multiples of δ (and related trapdoor elements) when forming π_A, π_B, π_C. These randomizers hide any dependence on the witness beyond what is revealed through the public inputs.

Formally, one can construct a simulator Sim that, given only the verification key and the public inputs, samples random blinding values and outputs a simulated proof π^* distributed identically to a real proof. Hence the verifier learns nothing about the witness $\mathbf{z}$ beyond the validity of the statement.

Succinctness. A Groth16 proof consists of only three group elements (π_A, π_B, π_C), independent of the circuit size. Verification requires a constant number of pairings, regardless of the circuit's complexity, making Groth16 one of the most efficient known SNARK constructions in terms of both proof size and verification cost.

4.2.2 PLONK

PLONK encodes circuits using a *universal arithmetization* based on selector and permutation polynomials, rather than circuit-specific QAPs. Let n be the number of gates in the circuit and let $H \subset \mathbb{F}$ be a multiplicative subgroup of size n with vanishing polynomial

$$Z_H(X) = \prod_{\omega \in H} (X - \omega).$$

The prover represents the witness with three polynomials $a(X), b(X), c(X)$ such that for every $\omega \in H$,

$$a(\omega), b(\omega), c(\omega)$$

are the left, right, and output wire values at gate ω.

Each gate constraint is encoded via *selector polynomials* $q_L(X), q_R(X), q_O(X), q_M(X), q_C(X)$, fixed by the circuit, such that for all $\omega \in H$,

$$q_L(\omega)\, a(\omega) \;+\; q_R(\omega)\, b(\omega) \;+\; q_O(\omega)\, c(\omega) \;+\; q_M(\omega)\, a(\omega) b(\omega) \;+\; q_C(\omega) \;=\; 0.$$

Equivalently, the constraint polynomial

$$C(X) := q_L(X)a(X) \;+\; q_R(X)b(X) \;+\; q_O(X)c(X) \;+\; q_M(X)a(X)b(X) \;+\; q_C(X)$$

must vanish on all of H, i.e.

$$Z_H(X) \;\big|\; C(X).$$

To enforce *copy constraints*, PLONK uses a *permutation argument*. Let $\sigma_a(X), \sigma_b(X), \sigma_c(X)$ be fixed permutation polynomials encoding how wires are connected across gates. The prover constructs the *grand product polynomial* $Z(X)$ with:

$$Z(\omega_{i+1}) = Z(\omega_i) \frac{a(\omega_i) + \beta\omega_i + \gamma}{a(\sigma_a(\omega_i)) + \beta\sigma_a(\omega_i) + \gamma} \cdot \frac{b(\omega_i) + \beta\omega_i + \gamma}{b(\sigma_b(\omega_i)) + \beta\sigma_b(\omega_i) + \gamma} \cdot \frac{c(\omega_i) + \beta\omega_i + \gamma}{c(\sigma_c(\omega_i)) + \beta\sigma_c(\omega_i) + \gamma},$$

starting from $Z(\omega_0) = 1$. This enforces that the tuples $(a(\omega), b(\omega), c(\omega))$ are consistent under the wiring permutation.

Finally, the prover combines all constraints into a single *quotient polynomial*:

$$Q(X) = \frac{C(X) \ + \ (\text{permutation checks})}{Z_H(X)},$$

which must have degree at most $d = O(n)$ if all constraints are satisfied.

The IOP thus reduces to proving: 1. $\deg a, b, c, Z \leq d$, 2. $Z(\omega_0) = 1$, 3. $Z(X)$ satisfies the grand product relation, 4. $Q(X)$ is a polynomial of degree $\leq d$.

IOP Description

Trace Table and Wire Polynomials PLONK organizes a computation as an R1CS and encodes it using a *trace table*, similar to Groth16. However, instead of working directly with QAP polynomials, PLONK introduces *wiring polynomials* to represent the values on each "wire" of the circuit.

Suppose the circuit has n gates and three wires per gate (left, right, and output). PLONK defines three polynomials $a(X), b(X), c(X) \in \mathbb{F}[X]$ of degree less than n, where the evaluation at a point τ_j encodes the values of the left, right, and output wires of the j-th gate:

$$a(\tau_j) = \text{left wire of gate } j, \quad b(\tau_j) = \text{right wire of gate } j, \quad c(\tau_j) = \text{output wire of gate } j.$$

Correctness Conditions For the table to represent a valid computation, the following checks must hold:

1. **Input check:** The initial rows encode the given public inputs correctly.
2. **Gate check:** Each gate's output is computed correctly from its inputs (e.g., addition, multiplication).
3. **Wiring check:** The wiring of the circuit is respected; i.e., whenever an input to a gate is supposed to be a copy of some previous output, the table entries agree.
4. **Output check:** The final designated output gate equals the claimed value (e.g., $z_5 = 0$ in the above example).

Selector Polynomials and Gate Constraints For each gate, PLONK defines a low-degree polynomial $q_j(X)$ that encodes the gate's operation. For instance, for a multiplication gate, the constraint is

$$a(\tau_j) \cdot b(\tau_j) - c(\tau_j) = 0.$$

PLONK introduces *selector polynomials* $q_M(X)$, $q_L(X)$, $q_R(X)$, $q_O(X)$, $q_C(X)$ to encode the circuit constraints: multiplication, linear terms, and constants. At each gate j, the constraint is

$$q_M(\tau_j) \cdot a(\tau_j) \cdot b(\tau_j) + q_L(\tau_j) \cdot a(\tau_j) + q_R(\tau_j) \cdot b(\tau_j) + q_O(\tau_j) \cdot c(\tau_j) + q_C(\tau_j) = 0.$$

This single linear combination allows PLONK to express any arithmetic gate type uniformly. Interpolating these values across all gates gives a single polynomial identity over the domain:

$$Q(X) := q_M(X) \cdot a(X) \cdot b(X) + q_L(X) \cdot a(X) + q_R(X) \cdot b(X) + q_O(X) \cdot c(X) + q_C(X) \equiv 0 \mod T(X),$$

where $T(X) = \prod_{j=1}^{n}(X - \tau_j)$ is the vanishing polynomial of the evaluation domain.

Permutation (Copy) Argument In addition to enforcing local gate constraints, PLONK must also ensure that wires referencing the same variable (across different gates) are assigned the same value. For example, if the output of one gate is reused as the input to another, these values must be consistent. This is called the *wiring constraint*.

PLONK enforces wiring consistency via a *permutation argument*. Let the evaluation domain be $H = \{\tau_1, \ldots, \tau_n\}$. Each row of the trace consists of three wire values: left, right, and output. We index these wires as positions $[3n]$. The wiring of the circuit induces a permutation

$$\sigma : [3n] \to [3n],$$

which maps each wire position to the wire position of the variable it is constrained to equal. For example, if the left input of row i is the same variable as the output of row j, then $\sigma(i_{\text{left}}) = j_{\text{out}}$.

To check consistency, PLONK uses a *grand product polynomial* $Z(X)$. Intuitively, $Z(X)$ accumulates a running product that compares wire values against their permuted copies. Let $\beta, \gamma \in \mathbb{F}$ be verifier-supplied random challenges. For each row j, define the multiplicative update:

$$Z(\tau_{j+1}) = Z(\tau_j) \cdot \frac{a(\tau_j) + \beta \cdot \sigma_a(\tau_j) + \gamma}{a(\tau_j) + \beta \cdot \tau_j + \gamma} \cdot \frac{b(\tau_j) + \beta \cdot \sigma_b(\tau_j) + \gamma}{b(\tau_j) + \beta \cdot \tau_j + \gamma} \cdot \frac{c(\tau_j) + \beta \cdot \sigma_c(\tau_j) + \gamma}{c(\tau_j) + \beta \cdot \tau_j + \gamma},$$

with initial condition $Z(\tau_0) = 1$. Here, $\sigma_a, \sigma_b, \sigma_c$ are the images of the left, right, and output wires under the permutation σ.

The key idea is that, if all copy constraints are satisfied, then the product telescopes and we obtain the invariant

$$Z(\tau_n) = 1.$$

If any wire assignment violates a copy constraint, then some factor in the numerator and denominator will not cancel, and the product deviates from 1.

Verifier's Check. The verifier does not check all intermediate values of $Z(\tau_j)$ directly. Instead, the prover commits to the polynomial $Z(X)$. The verifier samples a random evaluation point $\zeta \notin H$ and requests the prover to open $Z(\zeta)$. Consistency of the permutation argument is then reduced to checking the polynomial identity

$$Z(X)\cdot\big(a(X)+\beta X+\gamma\big)\big(b(X)+\beta X+\gamma\big)\big(c(X)+\beta X+\gamma\big) \;=\; Z(\sigma(X))\cdot\big(a(X)+\beta\sigma_a(X)+\gamma\big)\big(b(X)+\beta\sigma_b(X)+\gamma\big)\big(c(X)+\beta\sigma_c(X)+\gamma\big)$$

which holds over all $X \in H$. Since the prover cannot predict the verifier's challenge ζ, the probability of cheating without detection is negligible.

Thus, the permutation argument ensures that all wire values are consistent with the wiring of the circuit, while requiring only succinct commitments and a few evaluations.

Quotient Polynomial and Consolidated Identity PLONK combines all gate and permutation constraints into a single *quotient polynomial* $Q(X)$, defined such that

$$Q(X) = \frac{a(X) \cdot b(X) - c(X) + \text{copy terms} + \text{public input terms}}{T(X)}.$$

The key property is that $Q(X)$ has degree at most n, and its vanishing modulo $T(X)$ ensures all circuit and wiring constraints are satisfied simultaneously.

Why does the product telescope? Fix a row j. The update rule for $Z(\tau_{j+1})$ introduces a ratio with a numerator term

$$a(\tau_j) + \beta\sigma_a(\tau_j) + \gamma$$

and a denominator term

$$a(\tau_j) + \beta\tau_j + \gamma.$$

Suppose the wiring permutation is respected, i.e., the variable sitting in position $a(\tau_j)$ is *the same* as the variable assigned to position $\sigma_a(\tau_j)$. Then across all rows, every wire value appears exactly once in a denominator (at its original location) and exactly once in a numerator (at its permuted location). As a result, the product

$$\prod_{j=1}^{n} \frac{a(\tau_j) + \beta\sigma_a(\tau_j) + \gamma}{a(\tau_j) + \beta\tau_j + \gamma}$$

cancels perfectly, since each denominator term is matched by an identical numerator term from another row. The same holds for the b and c wires.

Thus, if all copy constraints are satisfied, we obtain

$$Z(\tau_n) = Z(\tau_0) \cdot \prod_{j=1}^{n} \frac{\cdots}{\cdots} = 1.$$

On the other hand, if some copy constraint is violated, then there exists at least one wire whose value differs from its supposed copy. In that case, the corresponding denominator term and numerator term are *not* equal, and cancellation fails. The grand product then deviates from 1, i.e., $Z(\tau_n) \neq 1$.

This telescoping property is what certifies that the prover's assignment is consistent with the permutation constraints.

Setup

PLONK uses a universal and updatable structured reference string (SRS) based on the KZG polynomial commitment scheme.

The SRS encodes powers of a secret trapdoor τ in a bilinear group:

$$\{[1], [\tau], [\tau^2], \ldots, [\tau^d]\} \subset \mathbb{G}_1$$

sufficient to commit to polynomials of degree at most d, independent of any particular circuit. Unlike Groth16, the SRS is circuit-independent; the same CRS can be reused for any circuit up to size d.

The KZG commitments allow the prover to commit to polynomials encoding the witness and the grand product, and later open these commitments at random points chosen by the verifier to check consistency.

Description of Protocol

Let C be a circuit of size m with witness vector $\mathbf{z}$, permutation σ, and public inputs $\mathbf{x}$. Let $\mathcal{S}$ be the universal KZG SRS over a sufficiently large domain.

1. **Prover polynomial construction:** Construct polynomials encoding the witness $(A(X), B(X), C(X))$, the selector polynomials (gate constraints), and the grand product polynomial $Z(X)$ enforcing wiring.
2. **Commitment:** Commit to each polynomial using the KZG SRS:

$$\mathsf{Com}(A), \mathsf{Com}(B), \mathsf{Com}(C), \mathsf{Com}(Z), \ldots$$

3. **Challenge sampling (Fiat–Shamir):** Convert the interactive IOP into a non-interactive proof by deriving random challenges $u, v, \ldots$ from the commitments via a hash function.

4. **Polynomial evaluations and openings:** Prover opens committed polynomials at challenge points to check:
   (a) Gate constraints are satisfied,
   (b) Wiring is correct via the grand product polynomial,
   (c) Public inputs are correctly embedded.
5. **Verification:** The verifier checks the KZG openings and pairing equations to ensure all polynomial relations hold.

The resulting proof size is a small constant number of group elements, independent of circuit size.

Completeness If the prover follows the protocol honestly with a valid witness $\mathbf{z}$, then the committed polynomials $A(X), B(X), C(X)$ satisfy all gate constraints, the grand product polynomial $Z(X)$ encodes the wiring correctly, and the public input polynomial matches the claimed inputs. All KZG openings at verifier challenge points will check out, so the verifier accepts.

Knowledge Soundness The KZG commitment scheme is binding under the discrete logarithm assumption. Hence, if a malicious prover produces a proof that passes verification, there exists a unique set of polynomials consistent with the commitments. Because the grand product polynomial enforces the wiring and the gate constraints are verified at random evaluation points, this implies the prover must know a witness $\mathbf{z}$ satisfying the circuit. This gives a **proof of knowledge** guarantee, analogous to special soundness in Σ-protocols.

Zero-Knowledge To achieve zero-knowledge, the prover adds random blinding terms to the polynomials before committing via KZG. These blinding factors hide any information about the witness beyond what is revealed by the public inputs. A simulator can produce commitments and polynomial openings at verifier challenges without knowledge of the witness, showing that the verifier learns nothing extra.

Succinctness PLONK proofs consist of only a few group elements (typically 4–5 in practice), independent of the size of the circuit. Verification requires a constant number of pairings and KZG openings, regardless of the circuit complexity. This makes PLONK a succinct and highly practical SNARK construction.

4.2.3 Marlin

Marlin is a universal and updatable SNARK that builds on the IOP+PCS paradigm. It achieves succinct proofs and efficient verification over pairing-friendly curves, while supporting arbitrary circuits via a universal SRS. Compared to Groth16, Marlin avoids the need for circuit-specific preprocessing, and compared to PLONK, it offers a more modular design with simpler constraint encoding.

Arithmetization

We work over a finite field $\mathbb{F}$.

- Let $H \subseteq \mathbb{F}$ be a multiplicative subgroup of size n (the row domain).
- Let $K \subseteq \mathbb{F}$ be a subgroup of size m (the index domain).
- Define the vanishing polynomial of a set S as

$$v_S(X) = \prod_{\gamma \in S} (X - \gamma).$$

- For $S \subseteq \mathbb{F}$, define

$$u_S(X, Y) = \frac{v_S(X) - v_S(Y)}{X - Y}.$$

An R1CS instance consists of sparse matrices $(A, B, C) \in \mathbb{F}^{H \times H}$ and a vector $z \in \mathbb{F}^H$ such that

$$(Az) \circ (Bz) = Cz.$$

Let $\hat{f}$ denote the unique degree-$(< |H|)$ interpolation of $f : H \to \mathbb{F}$.

For a sparse matrix M, the indexer defines maps $\text{row}, \text{col} : K \to H$ and $\text{val} : K \to \mathbb{F}$ describing the nonzero entries. Their interpolants

$$\widehat{\text{row}},\ \widehat{\text{col}},\ \widehat{\text{val}}$$

are univariate polynomials of degree $< |K|$. The low-degree extension of M is:

$$\hat{M}(X, Y) = \sum_{\kappa \in K} \frac{v_H(X)}{X - \widehat{\text{row}}(\kappa)} \cdot \frac{v_H(Y)}{Y - \widehat{\text{col}}(\kappa)} \cdot \widehat{\text{val}}(\kappa).$$

Protocol Intuition

The proof system reduces to two types of checks:

1. **Hadamard check:** ensures $(Az)[h](Bz)[h] = (Cz)[h]$ for all $h \in H$. This is enforced by testing whether
$$p(X) = \hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X)$$
is divisible by $v_H(X)$.
2. **Lincheck:** ensures that $\hat{z}_M$ is consistent with Mz for $M \in \{A, B, C\}$. This is achieved via a two-round sumcheck:
   - The verifier folds the H-sum into a single evaluation at a random point β_1.
   - The holographic encoding $\hat{M}$ is then verified via the index polynomials at a random (β_2, β_1).

Polynomial commitments (KZG) enforce degree bounds and allow succinct opening proofs at the verifier's sampled points.

Setup

- The *indexer* commits to $\widehat{\text{row}}_M, \widehat{\text{col}}_M, \widehat{\text{val}}_M$ for each $M \in \{A, B, C\}$.
- The *prover* commits to $\hat{z}, \hat{z}_A, \hat{z}_B, \hat{z}_C$ and to auxiliary polynomials arising in quotient and sumcheck steps.
- The *verifier* samples random challenges $(\alpha, \beta, \beta_1, \beta_2)$ and verifies the identities at those points.

Description of Protocol

Input: R1CS instance (A, B, C), public input x, witness w.

1. **Commitments:**
   - Prover commits to $\hat{z}, \hat{z}_A, \hat{z}_B, \hat{z}_C$.
   - Indexer provides commitments to $\widehat{\text{row}}_M, \widehat{\text{col}}_M, \widehat{\text{val}}_M$ for $M \in \{A, B, C\}$.
2. **Hadamard Check:**
   - Prover computes $t(X)$ such that
$$\hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X) = t(X) \cdot v_H(X).$$

- Verifier samples β and checks

$$\hat{z}_A(\beta)\hat{z}_B(\beta) - \hat{z}_C(\beta) \stackrel{?}{=} t(\beta)v_H(\beta).$$

3. **Lincheck for each $M \in \{A, B, C\}$:**

   - Verifier samples α, defines $q_1(X)$ as

$$q_1(X) = r(\alpha, X)\hat{z}(X) - r_M(\alpha, X)\hat{z}_M(X).$$

   - Prover demonstrates $\sum_{h\in H} q_1(h) = 0$ by sending g_1, h_1 such that

$$q_1(X) = h_1(X)u_H(X, \beta_1) + Xg_1(X),$$

     checked at a random β_1.

   - Verifier samples β_2 and checks holographic consistency:

$$\hat{M}(\beta_2, \beta_1) \stackrel{?}{=} \sum_{\kappa\in K} \frac{v_H(\beta_2)v_H(\beta_1)}{(\beta_2 - \widehat{\mathrm{row}}(\kappa))(\beta_1 - \widehat{\mathrm{col}}(\kappa))} \cdot \widehat{\mathrm{val}}(\kappa).$$

Output: Accept if all checks pass; otherwise reject.

Completeness If the prover holds a valid witness z such that $(Az) \circ (Bz) = Cz$, then:

- The Hadamard check passes since $p(X) = \hat{z}_A(X)\hat{z}_B(X) - \hat{z}_C(X)$ vanishes on H, hence is exactly divisible by $v_H(X)$, so identity (3) holds.
- For each $M \in \{A, B, C\}$, the lincheck passes since $\hat{z}_M$ is the interpolation of Mz, and thus the folded relation $q_1(X)$ has zero sum over H and admits a valid decomposition (6).
- The holographic evaluation of $\hat{M}$ at (β_2, β_1) is consistent with the committed index polynomials by construction.

Therefore an honest prover always convinces the verifier.

Soundness Suppose the prover commits to polynomials that are not consistent with any valid witness.

- If the Hadamard relation fails for some row of H, then $p(X)$ is a nonzero polynomial not divisible by $v_H(X)$. Equation (4) will fail except with probability at most $\deg(p)/|\mathbb{F}| = O(|H|/|\mathbb{F}|)$ over the verifier's random choice of β.
- If $\hat{z}_M \neq \widehat{Mz}$ for some M, then $q_1(X)$ has nonzero sum over H, and the sumcheck decomposition (6) cannot hold identically. The probability of passing a random check at β_1 is at most $O(|H|/|\mathbb{F}|)$.
- If the prover tries to forge $\hat{M}$, the holographic check (8)–(9) reduces to verifying a polynomial identity on K; a false statement passes with probability at most $O(|K|/|\mathbb{F}|)$.

Overall, the soundness error is negligible if $|\mathbb{F}| \gg |H|, |K|$.

Zero-Knowledge Marlin is compiled with a polynomial commitment scheme (e.g., KZG) augmented with *random masking* of committed polynomials:

- The prover blinds $\hat{z}$ and related polynomials with random multiples of v_H before committing.
- The auxiliary polynomials from the lincheck and quotient steps are similarly randomized.

These masks ensure that committed polynomials are information-theoretically hiding, and that all verifier queries reveal only evaluations at random points masked by fresh randomness. As a result, the verifier's view can be efficiently simulated given only the public input and the statement validity, ensuring zero-knowledge.

Succinctness The verifier's work is logarithmic in the circuit size:

- It samples a constant number of random field elements $(\alpha, \beta, \beta_1, \beta_2)$.
- It verifies a constant number of polynomial identities at those points.
- All heavy work (matrix-vector multiplications, polynomial interpolation) is done by the prover.

Commitments and openings are of constant size (a few group elements under KZG), and the verifier's checks reduce to a constant number of pairing evaluations. Thus the proof is succinct: verifier time is polylogarithmic in n, while the proof size and verification cost are independent of the circuit size.