0.1 Probabilistically Checkable Proofs

Before going further we should get acquainted with one more concept from the computational complexity theory, that have an important application in zk-SNARK and provides the theoretical backbone.

A Probabilistically Checkable Proof (PCP) is a type of proof system where the verifier can efficiently check the correctness of a proof by examining only a small, random portion of it, rather than verifying it entirely.

Definition 0.1. A language $\mathcal{L} \subseteq \Sigma^*$ (for some given alphabet Σ) is in the class $\mathsf{PCP}(r, q)$ (**probabilistically checkable proofs**), where r is the *randomness complexity* and q is the *query complexity*, if for a given pair of algorithms $(\mathcal{P}, \mathcal{V})$:

- *Syntax:* $\mathcal{P}$ calculates a proof (bit string) $\pi \in \Sigma^*$ in polynomial time $\mathsf{poly}(|x|)$ of the common input x. The prover $\mathcal{P}$ and verifier $\mathcal{V}$ interact, where the verifier has an oracle access to π (meaning, he queries it at any position).
- *Complexity:* $\mathcal{V}$ uses at most r random bits to decide which part of the proof to query and the verifier queries at most q bits of the proof.

Such pair of algorithms $(\mathcal{P}, \mathcal{V})$ should satisfy the following properties (with a security parameter $\lambda \in \mathbb{N}$):

- **Completeness**: If $x \in \mathcal{L}$, then $\Pr[\mathcal{V}^{\pi}(x) = 1] = 1$.
- **Soundness**: If $x \notin \mathcal{L}$, then for any possible (malicious) proof π^*,

$$\Pr\left[\mathcal{V}^{\pi^*}(x) = 1\right] = \mathsf{negl}(\lambda).$$

This allows a verification of huge statements with high confidence while using limited computational resources. See Figure 2.

Theorem 0.2. PCP theorem (PCP characterization theorem)
Any decision problem in NP has a PCP verifier that uses logarithmic randomness $O(\log n)$ and a constant number of queries $O(1)$, independent of n.

$$\mathsf{NP} = \mathsf{PCP}(O(\log n), O(1))$$

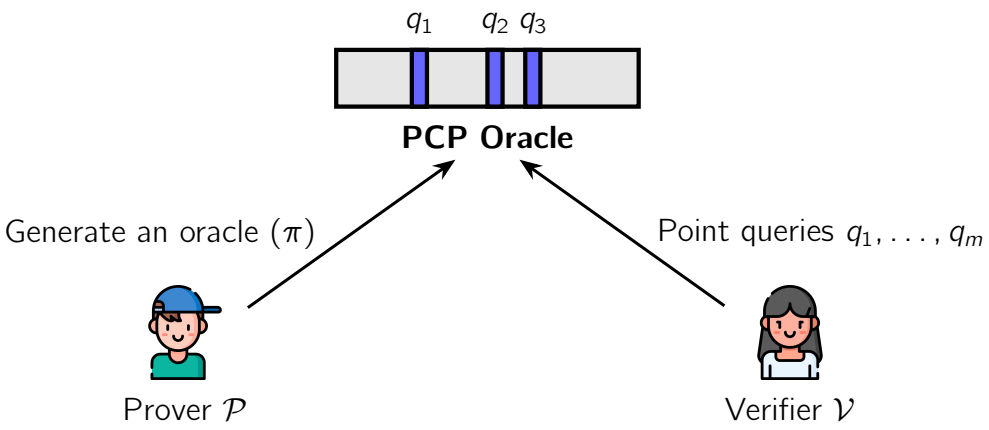

Figure 1: Illustration of a Probabilistically Checkable Proof (PCP) system. The prover $\mathcal{P}$ generates a PCP oracle π that is queried by the verifier $\mathcal{V}$ at specific points $q_1, \ldots, q_m$.

However, despite the fact that PCP is a very powerful tool, it is not used directly in zk-SNARKs. We need to extend it a bit to make it more suitable for our purposes.

Three main extensions of PCPs that are frequently used in SNARKs are:

- **IPCP** (**Interactive PCP**): The prover commits to the PCP oracle and then, based on the interaction between the prover and verifier, the verifier queries the oracle and decides whether to accept the proof.
- **IOP** (**Interactive Oracle Proof**): The prover and verifier interact and on each round, the prover commits to a new oracle. The verifier queries the oracle and decides whether to accept the proof.
- **LPCP** (**Linear PCP**): The prover commits to a linear function and the verifier queries the function at specific points.

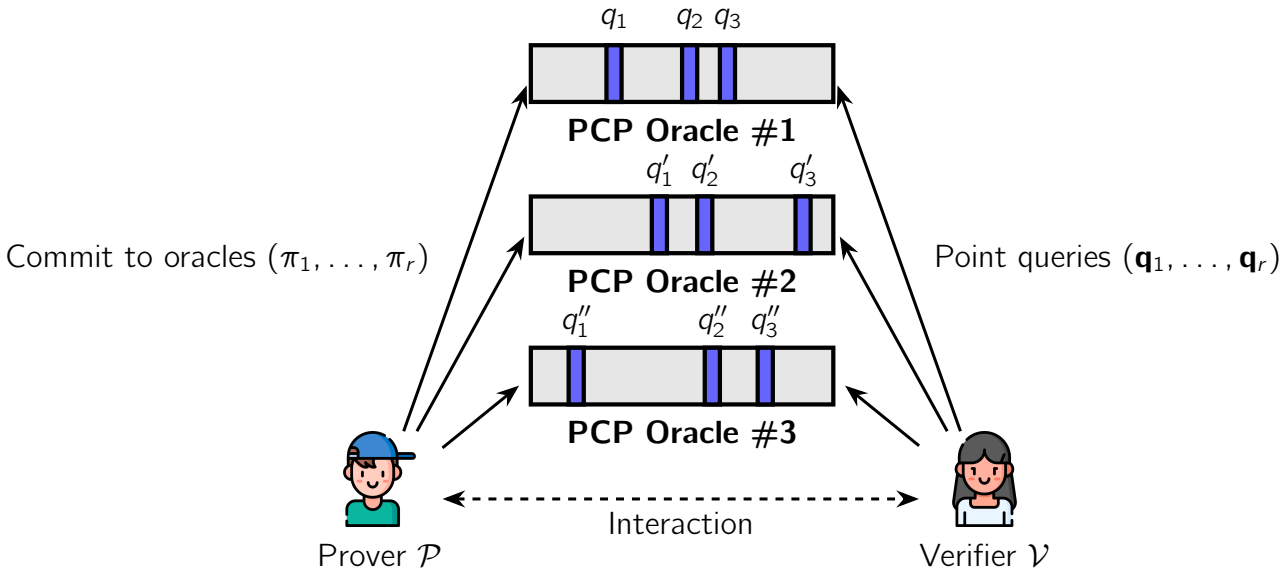

Figure 2: Illustration of an Interactive Oracle Proof (IOP). On each round i ($1 \leq i \leq r$), $\mathcal{V}$ sends a message m_i, and $\mathcal{P}$ commits to a new oracle π_i, which $\mathcal{V}$ can query at $\mathbf{q}_i = (q_{i,1}, \ldots, q_{i,m})$.

While IOPs will be later used for PLONK and zk-STARKs, we will focus on Linear PCPs in the context of Groth16 zk-SNARK. Let us define it below.

Definition 0.3 (Linear PCP)**.** A **Linear PCP** is a PCP where the prover commits to a linear function $\boldsymbol{\pi} = (\pi_1, \ldots, \pi_k)$ and the verifier queries the function at specific points $\boldsymbol{q}_1, \ldots, \boldsymbol{q}_r$. Then, the prover responds with the values of the function at these points:

$$\langle \boldsymbol{\pi}_1, \mathbf{q}_1 \rangle, \langle \boldsymbol{\pi}_2, \mathbf{q}_2 \rangle, \ldots, \langle \boldsymbol{\pi}_r, \mathbf{q}_r \rangle.$$

Example (QAP as a Linear PCP)**.** Recall that key QAP equation is:

$$\left(\sum_{i=1}^{n} w_i L_i(X)\right)\left(\sum_{i=1}^{n} w_i R_i(X)\right) - \left(\sum_{i=1}^{n} w_i O_i(X)\right) = Z_\Omega(X)H(X).$$

Now, the notation might be confusing, but firstly, we denote vectors of polynomials:

$$\begin{aligned} \boldsymbol{L}(X) &= (L_1(X), \ldots, L_n(X)), \\ \boldsymbol{R}(X) &= (R_1(X), \ldots, R_n(X)), \\ \boldsymbol{O}(X) &= (O_1(X), \ldots, O_n(X)). \end{aligned}$$

Now, consider the following **linear PCP for QAP**:

1. $\mathcal{P}$ commits to an extended witness $\mathbf{w}$ and coefficients $\mathbf{h} = (h_1, \ldots, h_n)$ of $H(x)$.
2. $\mathcal{V}$ samples $\gamma \xleftarrow{R} \mathbb{F}$ and sends $\boldsymbol{\gamma} = (\gamma, \gamma^2, \ldots, \gamma^n)$ to $\mathcal{P}$.
3. $\mathcal{P}$ reveals the following values:

$$\pi_1 \leftarrow \langle \mathbf{w}, \boldsymbol{L}(\gamma) \rangle, \quad \pi_2 \leftarrow \langle \mathbf{w}, \boldsymbol{R}(\gamma) \rangle, \quad \pi_3 \leftarrow \langle \mathbf{w}, \boldsymbol{O}(\gamma) \rangle, \quad \pi_4 \leftarrow Z_\Omega(\gamma) \cdot \langle \mathbf{h}, \boldsymbol{\gamma} \rangle.$$

4. $\mathcal{V}$ checks whether $\pi_1\pi_2 - \pi_3 = \pi_4$.

0.1.1 PCP application in QAP

When constructing a Quadratic Arithmetic Program (QAP) for a circuit $\mathcal{C}$, we represented the whole circuit's computation using the following relation:

$$L(X)R(X) - O(X) = Z_\Omega(X)H(X),$$

where by $L(X)$, $R(X)$, $O(X)$ we denote the polynomials that represent the left, right and output wires of the circuit, respectively. $Z_\Omega(X)$ is the target polynomial, while $H(X) := M(X)/Z_\Omega(X)$ for master polynomial $M(X) = L(X)R(X) - O(X)$ is the quotient polynomial.

We effectively managed to transform all the circuit's constraints, and computations in the short form. It still allows one to verify that each computational step is preserved by verifying the polynomial evaluation in specific (random) points, instead of recomputing everything. However, it is not quite clear why such a check is safe and how it can be used in a PCP. In other words, why checking that $L(s)R(s) - O(s) = Z_\Omega(s)H(s)$ for randomly selected s is enough to verify the circuit $\mathcal{C}$?

Soundness justification. Why is it safe to use such a check? As it was said early, we perform all the computations in some finite field $\mathbb{F}$. The polynomials $L(X)$, $R(X)$ and $O(X)$

are interpolated polynomials using $|\mathcal{C}|$ (number of gates) points, so

$$\deg(L) \leq |\mathcal{C}|\,, \quad \deg(R) \leq |\mathcal{C}|\,, \quad \deg(O) \leq |\mathcal{C}|$$

Thus, using properties of polynomials' degrees, we can estimate the degree of polynomial $M(X) = L(X)R(X) - O(X)$.

$$\deg(M) \leq \max\{\deg(A) + \deg(B), \deg(C)\} \leq 2\,|\mathcal{C}|$$

Now, using the Schwartz-Zippel Lemma (see **??**), we can deduce that if an adversary $\mathcal{A}$ does not know a valid witness $\mathbf{w}$, resolving the circuit $\mathcal{C}$, he can compute a polynomial $\widetilde{M}(X) \leftarrow \mathcal{A}(\cdot)$ that satisfies a verifier $\mathcal{V}$ with probability less than $2\,|\mathcal{C}|\,/|\mathbb{F}|$. To put it formally, we can write:

$$\Pr_{s \xleftarrow{R} \mathbb{F}}[\widetilde{M}(s) = M(s)] \leq \frac{2\,|\mathcal{C}|}{|\mathbb{F}|}$$

This probability becomes negligible as $|\mathbb{F}|$ grows large (which is typically the case), giving us soundness. In the same time, the verifier accepts the $M(X)$ generated using a valid witness with probability 1 giving us the completeness, so, we can categorize QAP as PCP.

We will modify the form of our proof with the next modifications, but still preserve the PCP properties.

In the following sections, we will introduce tools needed to succintly verify the equality above using the PCP properties. Since the overall proof is very complex from the very first glance, we will break it down into smaller parts and explain each of them in detail. Let us start with the first step.

0.1.2 Attempt #1: Encrypted Verification

Now, assume we have the cyclic group $\mathbb{G}$ of prime order q with a generator g. Typically, this is the group of points on an elliptic curve. Assume for simplicity that $e : \mathbb{G} \times \mathbb{G} \to \mathbb{G}_T$ is a symmetric pairing function, where $(\mathbb{G}_T, \times)$ is a target group.

Now, suppose during the setup phase, we have a trusted party that generated a random value $\tau \xleftarrow{R} \mathbb{F}$ and public parameters $g^{\tau}, g^{\tau^2}, \ldots, g^{\tau^d}$ for $d = 2|\mathcal{C}|$ — maximum degree of used polynomials (later we will use notation $\{g^{\tau^i}\}_{i\in[d]}$ for brevity). Then, party **deleted** τ. This way, we can now find the KZG commitment for each polynomial. Indeed, for example,

$$\mathsf{com}(L) \triangleq g^{L(\tau)} = g^{\sum_{i=0}^{d} L_i \tau^i} = \prod_{i=0}^{d} (g^{\tau^i})^{L_i},$$

and the same goes for $g^{R(\tau)}, g^{O(\tau)}, g^{H(\tau)}, g^{Z_\Omega(\tau)}$. Now, given these give points, how can we verify that the polynomial $M(X) = L(X)R(X) - O(X)$ is correct? Well, first notice that the check is equivalent to

$$L(\tau)R(\tau) = Z_\Omega(\tau)H(\tau) + O(\tau).$$

Notice that we transferred $O(\tau)$ to the right side of the equation to further avoid finding the inverse. Now, we can check this equality using encrypted values as follows:

$$e(\mathsf{com}(L), \mathsf{com}(R)) = e(\mathsf{com}(Z_\Omega), \mathsf{com}(H)) \cdot e(\mathsf{com}(O), g),$$

Remark. One might ask: why is the above equation correct? Well, let us see:

$$
\begin{aligned}
& e(\mathsf{com}(L), \mathsf{com}(R)) = e(\mathsf{com}(Z_\Omega), \mathsf{com}(H)) \cdot e(\mathsf{com}(O), g) && \textit{Initial statement} \\
\Leftrightarrow\ & e(g^{L(\tau)}, g^{R(\tau)}) = e(g^{Z_\Omega(\tau)}, g^{H(\tau)}) \cdot e(g^{O(\tau)}, g) && \textit{KZG Commitment Definition} \\
\Leftrightarrow\ & e(g,g)^{L(\tau)R(\tau)} = e(g,g)^{Z_\Omega(\tau)H(\tau)} e(g,g)^{O(\tau)} && \textit{Pairing bilinearity} \\
\Leftrightarrow\ & e(g,g)^{L(\tau)R(\tau)} = e(g,g)^{Z_\Omega(\tau)H(\tau)+O(\tau)} && \textit{Exponent product rule} \\
\Leftrightarrow\ & L(\tau)R(\tau) = Z_\Omega(\tau)H(\tau) + O(\tau) && \textit{QAP Check} \\
\Leftrightarrow\ & L(X)R(X) \equiv Z_\Omega(X)H(X) + O(X) && \textit{Schwarz-Zippel Lemma}
\end{aligned}
$$

So, sounds like we are done. Let us summarize what we have done so far:

Attempt #1 zk-SNARK Protocol

Suppose we are given a circuit $\mathcal{C}$ with a maximum degree d of polynomials used underneath. Thus, all parties additionally know the target polynomial $Z_\Omega(X)$.

Setup(1^λ)

The *trusted party* conducts the following steps:

- Picks a random value $\tau \xleftarrow{R} \mathbb{F}$.
- Calculates the public parameters $\{g^{\tau^i}\}_{i\in[d]}$.
- **Deletes** τ (toxic waste).
- **Outputs** verification key $\mathsf{vk} = \{\{g^{\tau^i}\}_{i\in[d]}, \mathsf{com}(Z_\Omega)\}$.

Prove(vk, **x**, **w**)

The prover $\mathcal{P}$ conducts the following steps:

- Runs the circuit with the statement **x** and witness **w**, obtains the intermediate constraint values, and calculates the polynomials $L(X), R(X), O(X)$ through Lagrange Interpolation.
- Calculates $H(X) \leftarrow (L(X)R(X) - O(X))/Z_\Omega(X)$.
- Calculates the KZG commitments as follows: $\pi_1 \leftarrow \mathsf{com}(L), \pi_2 \leftarrow \mathsf{com}(R), \pi_3 \leftarrow \mathsf{com}(O), \pi_4 \leftarrow \mathsf{com}(H)$ using powers of τ from the verification key vk.
- Publishes $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3, \pi_4)$ as a proof.

Verify(vk, **x**, $\boldsymbol{\pi}$)

Upon receiving $\boldsymbol{\pi} = (\pi_1, \pi_2, \pi_3, \pi_4)$, the verifier $\mathcal{V}$ checks:

$$e(\pi_1, \pi_2) = e(\mathsf{com}(Z_\Omega), \pi_4) \cdot e(\pi_3, g).$$

This sounds like an end to the story. However, there is a problem with this approach.

0.1.3 Proof of Exponent

The next step is to ensure that each of $\pi_1, \ldots, \pi_4$ was obtained through exponentiating the base g and not some other value (e.g., g^{100}). This is crucial since the prover could have cheated by using a different base. For that reason, we need some protocol that will check that. It is

called the **Proof of Exponent** (PoE). Let us define it below.

Definition 0.4 (Proof of Exponent)**.** A **Proof of Exponent** (PoE) is a protocol that allows the prover $\mathcal{P}$ to convince the verifier $\mathcal{V}$ that he obtained a value u through exponentiating a base g by a certain value x. The protocol works as follows:

1. **Setup:** The trusted party picks a random $\alpha \xleftarrow{R} \mathbb{Z}_q$, publishes g^α
2. **Prove:** $\mathcal{P}$ samples $\alpha \xleftarrow{R} \mathbb{F}$ and computes $u' \leftarrow u^\alpha$. Then, $\mathcal{P}$ publishes (u, u').
3. **Verify:** $\mathcal{V}$ checks whether $e(u, g^\alpha) = e(u', g)$.