Outline

- succinct arguments / SNARGs
- PCP Theorem
- Our first SNARGs
- Poly IOPs / IOPs
- Polynomial Commitments / KZG

Motivation - Verifiable Computation

1) Outsourcing computation

You → Laptop

| Run this expensive computation, C, on my input x! |

The output is y; here is a succinct proof \( \pi \) that \( \phi \) did do the computation correctly.

2) Eliminating repetitive work (Blockchain)

\( \pi \rightarrow \pi \rightarrow \pi \rightarrow \pi \rightarrow \pi \)
What does it mean to be succinct?

(Informally) Let $R$ be an NP relation, and let $T$ denote the runtime of the associated NP verifier. An interactive protocol, $(P,V)$, for $R$ is succinct if when ran on input $(x,w) \in R$:

1) Total Communication is $O(|w|)$.
2) $V$'s runtime is $\text{poly}(|x|) + O(|w| + T)$.

In particular, these quantities depend polynomially in the instance size, but sublinear in the witnesses size and runtime of the NP verifier, $T$.

What does it mean to be non-interactive?

The complete interaction consists of just a single message, $\Pi$, from the Prover to the Verifier.

PCP Theorem (informal)

A NP relation $R$, there exists an interactive protocol, $(\hat{P}, \hat{V})$, called a probabilistically checkable proof, such that $\hat{P}(x,w)$ outputs a long proof $\hat{\Pi} \in \{0,1\}^*$. To check this proof, the verifier $\hat{V}(\hat{\Pi}(x))$ surprisingly only has to read a small number of bits (can be just 3!). This protocol is both complete and sound.
From PCPs to SNARGs (Kilian & Micali construction)

\[ \overline{P}(x, w) \]

- Call the PCP Prover
  \[ \overline{\pi} \leftarrow \overline{P}(x, w) \]

- Commit to \( \overline{\pi} \) using a Merkle Tree
  \[ c \leftarrow \text{Merkle.Commit}(\overline{\pi}) \]

\[ \overline{V}(x; r) \]

- Sample Query Locations in \( \{i_1, \ldots, i_3\} \)
  \[ (i_1, i_2, i_3) \leftarrow \text{Sample}(r) \]

- Execute a decision procedure based on the queried bits
  \[ b \leftarrow \text{Decide}(\overline{\pi}_{i_1, i_2, i_3}, r) \]

- Output the decision bit \( b \)

\[ \text{Output} \]

\[ \text{Merkle.Verify}(c, (i_1, i_2, i_3), (\overline{\pi}_{i_1, i_2, i_3}, \overline{\pi}_M)) \]
Completeness: Follows from completeness of Merkle commitments and PCP

Soundness: Follows from index binding of MTs and soundness of PCP.

Non-Interactive: Can be made non-interactive in ROM with Fiat-Shamir.

Succinctness: \(O(1) + O(\log(1/\varepsilon))\)
\[ \frac{|c| + \text{bits}}{\text{poly in } |W|} \]

This SNARG can be viewed more abstractly as instantiating an Interactive Oracle Proof with a succinct vector commitment scheme.

\[ \text{IOP} \quad + \quad \text{Vector Commitment} \]

\[ P \quad \overset{\text{oracles to vectors}}{\quad \rightarrow} \quad V \]
\[ \overset{\text{can query vectors as oracles}}{\quad \leftarrow} \quad \text{randomness} \]

\[ \text{"Information Theoretic Proof } \quad + \quad \text{Concrete cryptographic Commitment to oracle} \]
In practice, the Kilian–Micali SNARG is wildly impractical due to the concrete cost of producing and committing to the PCP \( \Pi \).

However, by generalizing the types of oracles and increasing the rounds, we will be able to obtain practically efficient SNARKs by following this paradigm. In particular,

\[
\begin{align*}
\text{Info Theoretic Proof} & \quad + \quad \text{Cryptographic Commitment} \\
\text{IOP (vector oracles)} & \quad + \quad \text{Vector Commitments} \\
\downarrow & \quad \quad \quad \quad \downarrow \\
\text{Polynomial IOP} & \quad + \quad \text{Polynomial Commitments} \\
(\text{polynomial oracles}) & \quad \quad \quad \quad \ast \text{ Today}
\end{align*}
\]

\[\text{Polynomial Commitment Schemes (PCS)}\]

A PCS enables a commiter to commit to a polynomial over a finite field \( \mathbb{F}[X] \) such that they can open the polynomial at any point in the field. A PCS is a tuple of 4 efficient algs:

- \( \text{Setup}(1^\lambda, d) \rightarrow pp : \) takes in sec param and degree bound \( d \), and outputs \( pp \).
- \( \text{Commit}(pp, f) \rightarrow c : \) takes in \( pp \), poly \( \in \mathbb{F}[X] \) of degree \( \leq d \) and outputs a commitment.
- **Open** \((pp, f, z) \rightarrow \Pi\): takes in \(pp, f \in \mathbb{F}[x], z \in \mathbb{F}\), and outputs a proof \(\Pi\) that attests to the eval \(y = f(z)\).

- **Verify** \((pp, c, z, y, \Pi) \rightarrow 0/1\): Checks the evaluation proof \(\Pi\) with respect to commitment \(c\), point \(z\), and claimed eval \(y\).

**Properties:**

**Correctness:** \(\forall\) poly-bounded \(d \in \mathbb{N}\), \(f \in \mathbb{F}[x]\), \(z \in \mathbb{F}\),

\[
\Pr\left[\text{Verify}(pp, c, z, f(z), \Pi) = 1 \mid pp \leftarrow \text{Setup}(1^\lambda, d), c \leftarrow \text{Commit}(pp, f), \Pi \leftarrow \text{Open}(pp, f, z)\right] = 1
\]

**Evaluation Binding:** \(\forall\) poly-bounded \(d \in \mathbb{N}\), \(\forall\) PPT \(A\),

\[
\Pr\left[\begin{array}{l}
\text{Verify}(pp, c, z, y, \Pi) = 1 \\
\land \\
\text{Verify}(pp, c, z, y', \Pi') = 1 \\
\land \\
y \neq y'
\end{array} \right] \leq \text{negl}(\lambda)
\]

**KZG Commitments:**

A polynomial commitment scheme from pairings!
Trusted Setup:

\[
\text{Setup}(1^\lambda, \mathsf{d}) : \quad \text{Let's assume the group elts come with the description of the BG.}
\]

1) Sample a bilinear group,

\[
\mathsf{IB} := (G_1, G_2, G_T, p, g_1, g_2, e) \leftarrow \mathsf{BG \ Sample}(1^\lambda)
\]

2) Sample a secret value \( \alpha \leftarrow \mathbb{Z}_p \). \text{Toxic waste - private randomness}

3) Output \( pp := (\sum_{i \in \{0, \ldots, d-3\}} g_1^{\alpha^i} g_2^{\alpha^i}, g_2^{\alpha}) \)

\[
\text{Commit}(pp, f) : \quad \text{Commit}(pp, f) : \quad \text{Commit}(pp, f) :
\]

1) Parse \( (\sum_{i \in \{0, \ldots, d-3\}} g_1^{\alpha^i} g_2^{\alpha^i}, g_2^{\alpha}) \leftarrow pp \).

2) Output \( C := \prod_{i=0}^{d-1} (g_1^{\alpha^i})^{f_i} \) where \( f_i \) is the \( i \)-th coeff of \( f \). \text{Notice } C = g_1^{f(\alpha)}

\[
\text{Open}(pp, f, z) : \quad \text{Open}(pp, f, z) : \quad \text{Open}(pp, f, z) :
\]

1) Parse \( (\sum_{i \in \{0, \ldots, d-3\}} g_1^{\alpha^i} g_2^{\alpha^i}, g_2^{\alpha}) \leftarrow pp \).

2) Compute the quotient \( \frac{f(X) - f(z)}{X - z} = q(x) \)

3) Output \( \Pi := \text{Commit}(pp, q) \) \( \Pi := \text{Commit}(pp, q) \) \( \Pi := \text{Commit}(pp, q) \) \( = g_1^{q(\alpha)} \)
Verify \((pp, c, z, y, \Pi)\):

1) Parse \(pp\) to obtain \(g^\alpha_2\).

2) Output accept iff
\[
e(\Pi, g^\alpha_2/g^z_2) = e(c/g^y_1, g_2)
\]

**Correctness:**
\[
e(\Pi, g^\alpha_2/g^z_2) = e(g^q(\alpha)_1, g^{\alpha - z}_2) = e(g^{q(\alpha)}_1(\alpha - z), g_2)
\]
\[
= e(g^{f(\alpha) - y}_1, g_2) = e(g^{f(\alpha)}_1/g^y_1, g_2)
\]
\[
= e(c/g^y_1, g_2)
\]

**Eval Binding:** Reduces to breaking the \(d\)-BDH assumption: "Given \((E g^\alpha_1, z_1 e_{z_0}, ..., a_3, g^x_2)\) it is hard to come up with \((y \in F, h \in G_1)\) such that \(y \neq \alpha\) and \(h = e(g_1, g_2)^{(\alpha - y)}\).

Consider \(A(pp) \rightarrow (c, z, y, \Pi, y', \Pi')\) that breaks eval binding. Since both verification equations accept we know,
\[
e(\Pi, g^\alpha_2/g^z_2) = e(c/g^y_1, g_2) \Rightarrow \Pi^{\alpha - z} \cdot g^y_1 = c
\]
\[
e(\Pi', g^\alpha_2/g^z_2) = e(c/g^{y'}_1, g_2) \Rightarrow (\Pi')^{\alpha - z} \cdot g^{y'}_1 = c
\]
Thus, we must have
\[(\frac{\pi}{\pi'})^{x-z} x_{\pi' - \gamma'} = 1 \Rightarrow (\frac{\pi}{\pi'})^{\gamma - \gamma'} = g_1^{\frac{1}{x-z}}\]

Therefore, we output \(x = z\) and \(h = e((\frac{\pi}{\pi'})^{\frac{1}{\gamma - \gamma'}}, g_2)\).

Now we can commit to polynomials!

Building Efficient SNARKs in Practice

- Trusted Setup

\[\text{Setup}(1^n) \rightarrow \text{PP}\]

\[\uparrow \text{performed by trusted party or multiparty computation}\]

- Preprocessing Stage

\[\text{Preprocess} \quad \text{desc of computation}\]

\[\text{PP} + C \rightarrow \text{Preprocess} + C\]

- Prover / Verifier

\[\text{PolyIOP} + \text{PCS}\]
Arithmetic Circuits (a computational model)

- An arithmetic circuit is a directed acyclic graph where each node is an addition or multiplication gate with input-arity 2 and output-arity 1. The edges (wires) take on values from a specified field $\mathbb{F}$.
- These wires must satisfy the constraints enforced by the constraints.

Here is a simple example of an Arithmetic Circuit $C$ with inputs $(x,w) \in \mathbb{F}^4$ that computes $x_i x_2 (w_1 + w_2)$.

![Circuit Diagram]

- Copy Constraints (optional): Field values can instead be assigned to gate pins (black) and wiring (blue) enforce equality. (For example, here we enforce $a = b = c$, $d = e$)
- We will consider the family of NP-Relations $\{ R^C \}_C$ determined by Arithmetic Circuits $C$, $R^C := \{(x;w) \mid C(x,w) = 0\}$
Succinct Argument (SNARG)

A SNARG is a tuple of efficient algs

\((\text{Setup}, \text{Preprocess}, P, V)\)

- \(\text{Setup}(1^\lambda) \rightarrow pp\): Takes in as input the security parameter, \(1^\lambda\), outputs public parameters, pp.
- \(\text{Preprocess}(pp, C) \rightarrow (pk, vk)\): Takes in as input pp and a description of a circuit \(C\), outputs a proving key and succinct verification key.
- \(P(pk, x, w) \rightarrow \Pi\): Takes in the proving key and instance \(x\), witness \(w\), outputs a succinct proof \(\Pi\).
- \(V(vk, x, \Pi) \rightarrow 0/1\): Takes in the succinct verification key, vk, instance \(x\), and proof \(\Pi\).

Completeness: \(\forall C, (x, w) \in R_C,\)

\[ Pr \left[ V(vk, x, \Pi) = 1 : \begin{array}{l} pp \leftarrow \text{Setup}(1^\lambda) \\ (pk, vk) \leftarrow \text{Preprocess}(pp, C) \\ \Pi \leftarrow P(pk, x, w) \end{array} \right] \geq 1 - \text{negl}(\lambda) \]

Soundness: \(\forall x \notin L(R_C), \forall \text{PPT } P^*\)

\[ Pr \left[ V(vk, x, \Pi) = 1 : \begin{array}{l} pp \leftarrow \text{Setup}(1^\lambda) \\ (pk, vk) \leftarrow \text{Preprocess}(pp, C) \\ \Pi \leftarrow P^*(pp) \end{array} \right] \leq \text{negl}(\lambda) \]
Succinctness:

Succinct Proof: $|T| \in o(1w1)$

Succinct Verifier: $|V| \in O(1x1) + O(1w1 + 1c1)$

Verifier runtime

Non-interactive: ✓ by definition of P,V algs.