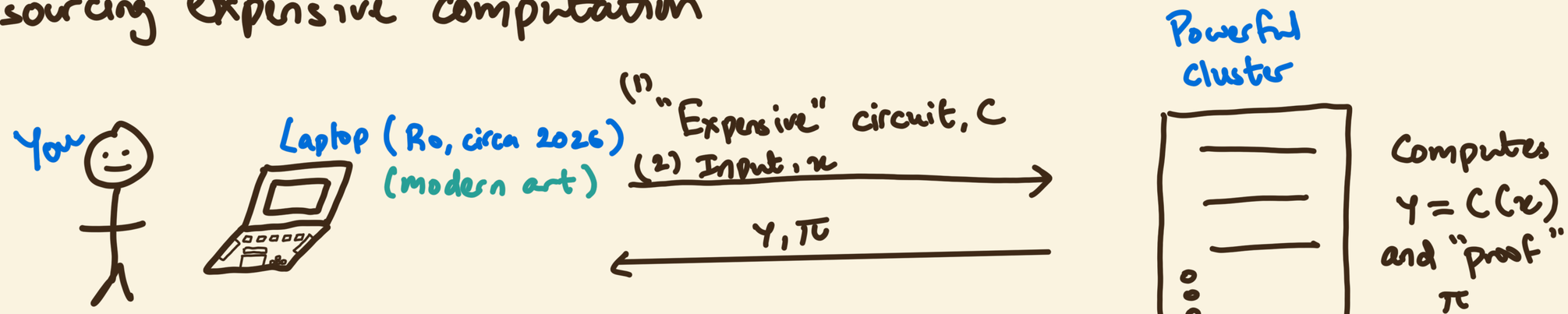
SUCCINT ARGUMENTS (SNARGs)

TODAY:
- Succinctness
- PCP Theorem
- Our First SNARG
- Interactive Oracle Proofs (IOP)
- Polynomial Commitments + KZG construction

Motivation: Verifiable Computation

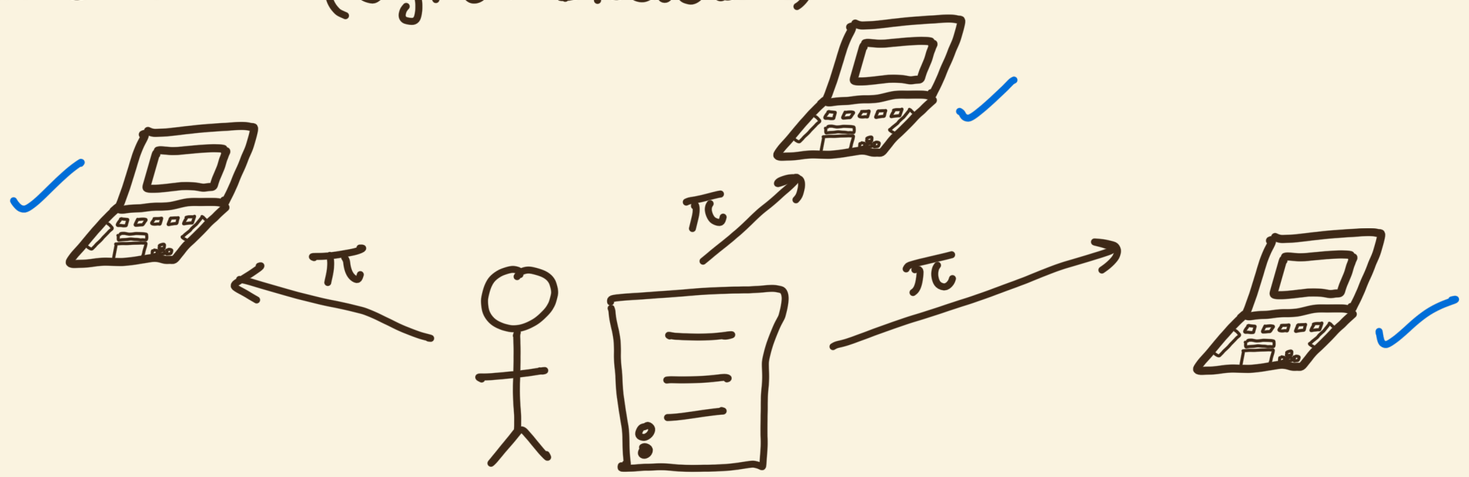
(1) Outsourcing expensive computation



C is polynomial-size circuit : π can be empty and You can verify y is correct Yourself. But, we want something that's "much more efficient".

Or maybe π can be the entire computational trace. But that's a huge proof (still polynomial) and we run into the same problem :(.

(2) Eliminating repetitive work (e.g. on Blockchains)



So, we want both the proof and its verification to be "succinct".

Succinctness

Slightly more formally: Let R be an NP-relation, ← $\{(x,w) : \text{Verify}_L(x,w) = 1\}$ i.e., set of instance-witness pairs
and let T denote the runtime of the associated NP-verifier.

An IP (P,V) is succinct if when ran on input $(x,w) \in R$,

(1) Total communication: $o(|w|)$, ← little-o

(2) V's runtime is $\text{poly}(|x|) + o(|w| + T)$

Polynomial in instance size, sub-linear in witness size and runtime of V

Succinct Non-interactive Arguments

A type of proof system. Let's break this down:

- **Succinct**: described above
- **Non-interactive**: The entire "interaction" consists of only a single message from the Prover to the Verifier ← we saw this last lecture!
- **Argument**: computational soundness, sound against PPT P^* (may be unsound if P^* unbounded)

We will use a very powerful tool to build our first SNARG:
the **PCP theorem** ← another kind of proof system, huge advance in computational complexity!

PCP Theorem

Informally, any NP language, say $\mathcal{L}$, has a **probabilistically checkable proof** (PCP)

$x \in \mathcal{L} \Rightarrow$ P can compute (long) proof $\pi \in \{0,1\}^*$
s.t. V only has to read very few bits (probabilistic check)
to be soundly convinced! ← as few as 3 bits!!

e.g. $\tilde{P}(x, w)$ — in the settings of "proofs", $\tilde{P}$ is unbounded so can compute the witness instead of being given as input.

$\xrightarrow{\tilde{\pi} \in \{0,1\}^*}$

$\tilde{V}(x; r)$

- Sample random bits to read.
  $(i_1, i_2, i_3) \leftarrow \text{Sample}(|\tilde{\pi}|; r)$
- $0/1 \leftarrow \text{Verify}^{\text{PCP}}_{\mathcal{L}}((\tilde{\pi}[i_1], \tilde{\pi}[i_2], \tilde{\pi}[i_3]); r)$
  (reject/accept)

Turns out given w, $\tilde{P}$ runs in poly-time to compute $\tilde{\pi}$. Useful!

Perfect completeness + statistical soundness (error $< 1/2$; amplified via parallel repetition)

As-is, this is not a SNARG: $\tilde{\pi}$ is not necessarily $o(|\omega|)$.

SNARGs From PCPs

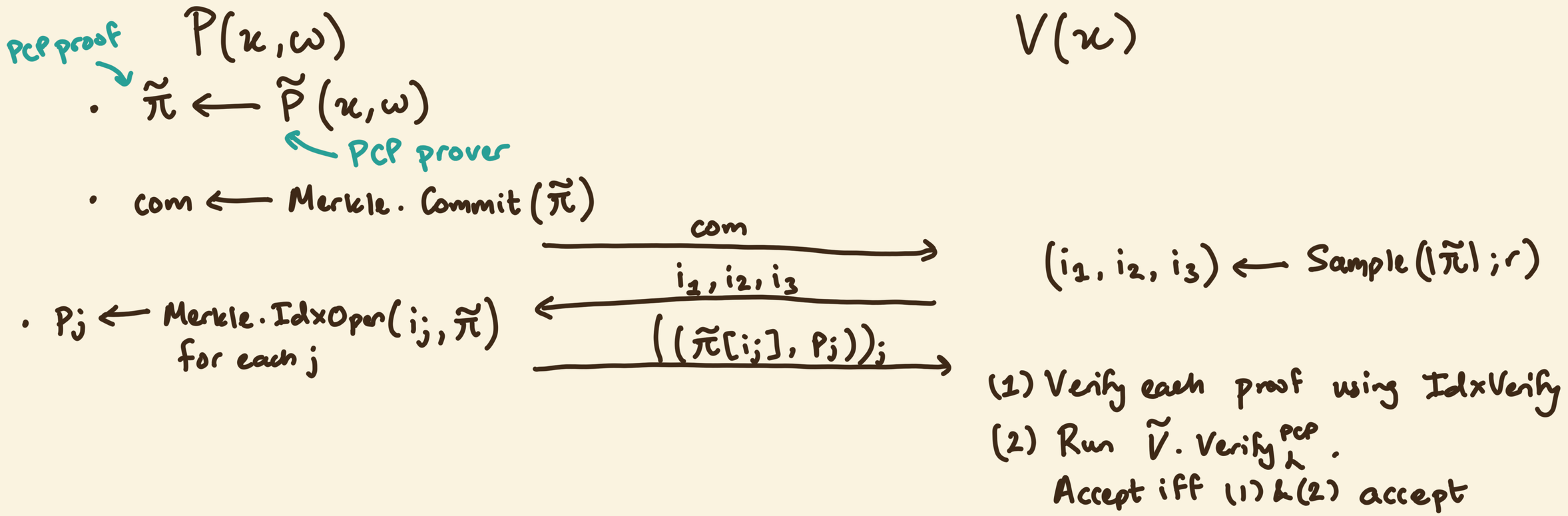
Think of $\tilde{\pi}$, a bit string, as a vector (over $\mathbb{F}_2$)

Have we seen a way to encode a vector succinctly such that P can reveal queried positions (in a verifiable manner)? (HW2)

Merkle trees!

Following construction due to ideas from Kilian (1992) and Micali (1994).

Essentially, commit to $\tilde{\pi}$ using a merkle tree. P can open $\tilde{\pi}[i_1], \tilde{\pi}[i_2], \tilde{\pi}[i_3]$.



Completeness. follows from completeness of Merkle commitments and PCP

Soundness. (informal) index-binding of commitment prevents P from cheating on randomly picked indices. From there, follows by soundness of PCP...

Non-interactivity. As stated, it is not non-interactive! Via parallel repetition, we can reduce soundness error to negl., then apply Fiat-Shamir to derive indices to check

Succinctness. $|com| = \lambda$, each opening is a path in Merkle tree, which is $O(\log|\tilde{\pi}|) = o(|w|)$

indices are λ-bits each

Total comm. $o(|w|)$ ✓

PCP thm.: $|\tilde{\pi}| = \text{poly}(|w|)$

V's runtime? Must reconstruct paths in Merkle tree : $O(\log|\tilde{\pi}|) = o(|w|)$

Knowledge? SNARGs that are also knowledge sound are known as **SNARKs**

arguments of knowledge

We won't see this today, but can be done...

Efficiency. Indeed succinct, but not concretely efficient. PCP prover is expensive...

Proof is asymptotically reasonable but too large to be practical

Still, a really cool feasibility result. We need more for concrete efficiency...

So, how do we get there?

Let's abstract the Kilian-Micali construction.

Broadly, we combined.

Recipe

(1) An interactive proof system for NP in an idealized world

+

(2) Succinct vector commitment scheme to compile IP into a succinct argument

(1) ← information-theoretic proof system

(1) ← the long proof vector $\tilde{\pi}$ can be thought of as an oracle, which can be queried for values at input indices $\tilde{\pi}[i]$. (just like ROM)

(2) ← cryptographic scheme

We just saw

(1) PCP + (2) Merkle commitment

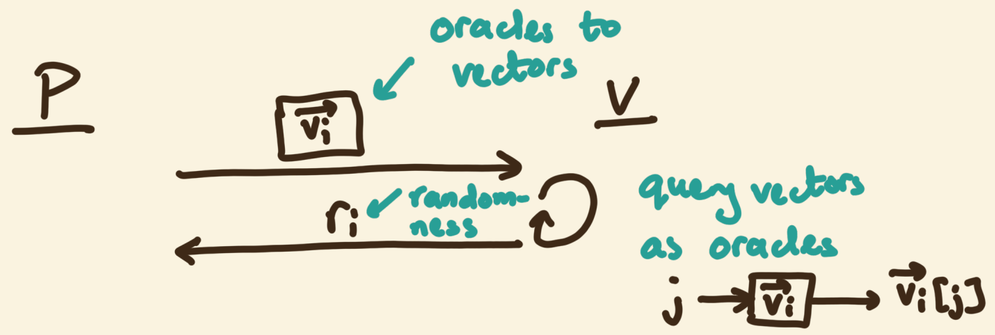
← but any succinct vector commitment would work

Suggests two possible paths to improve:

(a) stronger idealized world in (1)?

(b) more eff. cryptographic scheme in (2)?

Expanding on (a), we think of these IPs in an idealized world as **Interactive Oracle Proofs (IOPs)**



Approach we'll see in this class: Replace vector oracles with polynomials! The additional structure proves useful as we'll see

Replace IOPs with "Poly-IOPs" ← Polynomial IOPs

(We'll see more of this next class)

To instantiate PolyIOPs succinctly, we'll need something more powerful for (2)

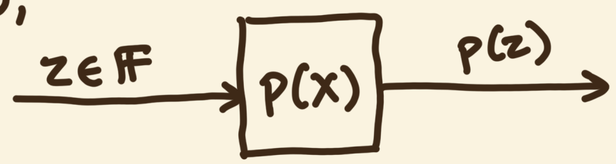
Polynomial Commitments !

← today's focus

Polynomial Commitment Schemes (PCS)

Commit to a polynomial over a finite field for example such that given input point, can open commitment to the polynomial evaluated at that point.

e.g. $P(X) \in \mathbb{F}[X]$,
(as an oracle)



A PCS is a tuple of four efficient algorithms,

PCS = (Setup, Commit, Open, Verify) s.t.

1. $\text{Setup}(1^\lambda, d) \to pp$

d: max degree $\in \mathbb{N}$; pp: public parameters

2. $\text{Commit}(pp, f) \to c$ ← commitment

f: polynomial in $\mathbb{F}[X]$, $\deg(f) \le d$

3. $\text{Open}(pp, f, z) \to (y, \pi)$

z: input point $\in \mathbb{F}$; y: alleged evaluation $y = f(z)$; π: "evaluation proof"

4. $\text{Verify}(pp, c, z, y, \pi) \to 0/1$

Check if evaluation proof is valid with respect to commitment c, point z, and claimed eval y.

Must satisfy

Correctness (perfect). $\forall d \in \mathbb{N}$, $d = \text{poly}(\lambda)$, $f \in \mathbb{F}^{\le d}[X]$ ← degree d bounded polynomials / $\mathbb{F}$, $z \in \mathbb{F}$,

$$\Pr\left[\text{Verify}(pp, c, z, f(z), \pi) = 1 \;\middle|\; \begin{array}{c} pp \leftarrow \text{Setup}(1^\lambda, d) \\ c \leftarrow \text{Commit}(pp, f) \\ (y, \pi) \leftarrow \text{Open}(pp, f, z) \end{array}\right] = 1$$

Evaluation Binding. $\forall d \in \mathbb{N}, d = \text{poly}(\lambda)$, $\forall \text{PPT } \mathcal{A}$,

$$\Pr\left[\begin{array}{c} \text{Verify}(pp, c, z, y, \pi) = 1, \text{ and} \\ \text{Verify}(pp, c, z, y', \pi') = 1, \text{ and} \\ y \neq y' \end{array} \;\middle|\; \begin{array}{c} pp \leftarrow \text{Setup}(1^\lambda, d) \\ (c, z, y, \pi, \\ y', \pi') \leftarrow \mathcal{A}(pp) \end{array}\right] \le \text{negl}(\lambda)$$

in words, cannot open commitment to two different evaluations of its choice of input.

(possibly maliciously generated) (similar to vector commitments' index binding)

But wait... isn't a polynomial a vector of evaluations? Why not just use a vector commitment?

Issue. vector is too big if $\mathbb{F}$ is large. Suppose $|\mathbb{F}| = p$.

e.g. if $p = O(2^\lambda)$, $\vec{f} = (f(0), f(1), \ldots, f(p-1))$ is exponential-size...

Later, we will indeed need $|\mathbb{F}| = O(2^\lambda)$ for the PolyIOP recipe to work, so this won't do!

How will we do better?

Intuition. degree-d polynomial is uniquely defined by $d+1$ evaluations or coefficients.

$d \ll p$.

First Idea. Then, commit to the coefficients! $f(X) = \sum_{i=0}^{d} f_i X^i$

We'll use Pedersen:

- $\text{Setup}(1^\lambda, d) \to pp = (g, h)$ $\quad h \xleftarrow{\$} G$, $G = \langle g \rangle$
- $\text{Commit}((g,h), f) \to \left(c_i = g^{r_i} h^{f_i}\right)_{i=0}^{d}$ (comm. to ith coeff. with fresh randomness r_i)
- $\text{Open}(pp, f, z) \to \pi = \sum_{i=0}^{d} r_i z^i$
- $\text{Verify}(pp, c, z, y, \pi) \to 1$ iff $g^\pi \cdot h^y = \prod_{i=0}^{d} c_i z^i$

(technical note: committer must keep state: r_i, to be able to open. Our more eff. construction won't require this!)

Correct because $\prod_{i=0}^{d} c_i z^i = \prod_{i=0}^{d} g^{r_i z^i} h^{f_i z^i} = g^\pi \cdot h^y$

Can show DLOG $\Rightarrow$ eval. binding

Problem? Not efficient :(Commitment and verification time linear in degree d. Won't work for us later...

KZG Polynomial Commitments

[Kate-Zaverucha-Goldberg 2010]

~ PCS using **Pairings** ← assume $(p, \mathbb{G}_1, \mathbb{G}_2, \mathbb{G}_T, g_1, g_2, e)$ is an asymmetric bilinear group. Denote $g_T = e(g_1, g_2)$.

$O_d(1)$ commitment, $O_d(1)$ verification time!

Constructed as follows:

- $\text{Setup}(1^\lambda, d)$:
  - sample $\alpha \xleftarrow{\$} \mathbb{Z}_p$.
  - Denote $u_i = g_1^{\alpha^i}$ for $i \in \{0, \dots, d\}$ ← "powers of α in exponent"
  - $v = g_2^{\alpha}$
  - Output $pp = ((u_i)_i, v)$

(α is secret to the committer) ←

Called "Trusted" Setup. Trusted to not leak α

- $\text{Commit}(((u_i)_i, v), f)$:
  - Output $c \leftarrow g_1^{f(\alpha)} = \prod_{i=0}^{d} u_i^{f_i}$ (evaluate f at α in the exponent using u_i)

- $\text{Open}(((u_i)_i, v), f, z)$:
  - Compute $h(X) = \frac{f(X) - f(z)}{X - z}$ ← why is this a polynomial?
  - Output $\pi \leftarrow g_1^{h(\alpha)} = \prod_i u_i^{h_i}$

- Verify $((u_i)_i, v), C, z, y, \pi)$:
  - Output 1 iff $e\left(\pi, \frac{v}{g_2^{z}}\right) = e\left(\frac{C}{g_1^{y}}, g_2\right)$

Correctness. $e\left(\pi, \frac{v}{g_2^{z}}\right) = e\left(g_1^{h(\alpha)}, \frac{g_2^{\alpha}}{g_2^{z}}\right) = e\left(g_1^{\frac{f(\alpha)-f(z)}{\alpha - z}}, g_2^{\alpha - z}\right) = g_T^{f(\alpha)-y}$

$$\| $$

$$e\left(g_1^{f(\alpha)-y}, g_2\right) = e\left(\frac{C}{g_1^{y}}, g_2\right) \checkmark$$

Eval. Binding. Requires new assumption: t-Bilinear Strong Diffie-Hellman (t-BSDH)

informally, given $\left(g_1, g_1^{\alpha}, g_1^{\alpha^2}, \ldots, g_1^{\alpha^t}, g_2^{\alpha}\right)$,
hard to compute constant $r \neq \alpha$ and $g_T^{\frac{1}{\alpha + r}}$

Suppose $\exists \mathcal{A}$ that breaks eval. binding.

i.e., $\mathcal{A}(pp) \rightarrow (C, z, y, \pi, y', \pi')$ s.t. ① $Verify(pp, C, z, y, \pi) = 1$, and ② $Verify(pp, C, z, y', \pi') = 1$

① $\Rightarrow e(\pi, v/g_2^{z}) = e(C/g_1^{y}, g_2) \Rightarrow \pi^{\alpha - z} \cdot g_1^{y} = C$

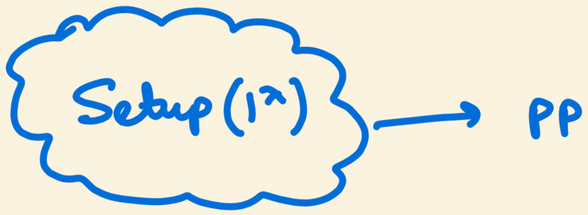
② $\Rightarrow e(\pi', v/g_2^{z}) = e(C/g_1^{y'}, g_2) \Rightarrow (\pi')^{\alpha - z} \cdot g_1^{y'} = C$

$\Rightarrow (\pi/\pi')^{\frac{1}{y'-y}} = g_1^{\frac{1}{\alpha - z}}$

So, reduction to d-BSDH outputs $r = -z$ and $e\left((\pi/\pi')^{\frac{1}{y'-y}}, g_2\right)$

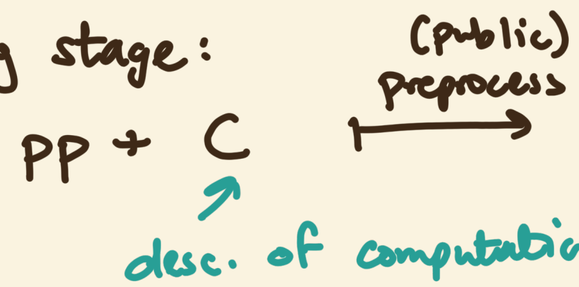
Building Efficient SNARGs/SNARKs In Practice

- Trusted Setup



performed by trusted party or multi-party computation

- Preprocessing stage:



$\{f_i\}_i$ "proving key" polynomial encoding of C

$\{c_i\}_i$ "verification key" commitments to polynomial encodings

- (P,V) system : PolyIOP + PCS

One way to describe computation is using arithmetic circuits. ← more on this next class.

We consider NP-relations indexed by arithmetic circuits:

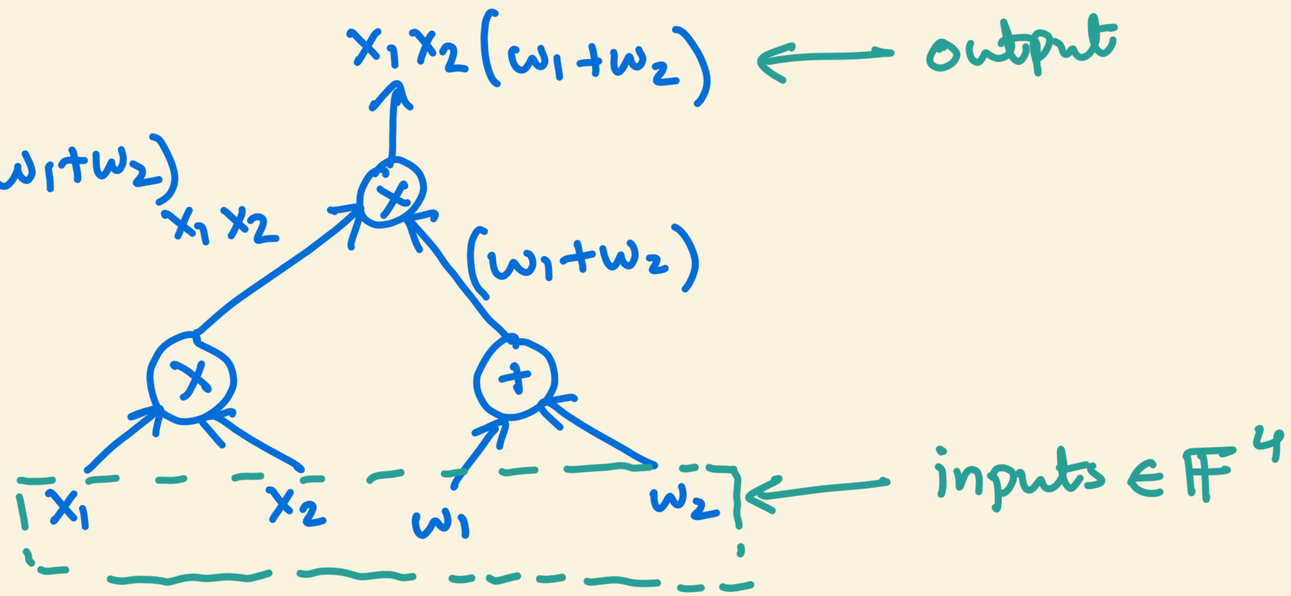
$$\mathcal{R}_C = \{(x,\omega) \mid C(x,\omega) = 0\}$$

Arithmetic Circuits (a computational model)

Directed acyclic graph where each vertex/node is an addition/multiplication gate with input arity 2 (fan-in) and output arity 1 (fan-out). The edges (wires) take on values in a specified field $\mathbb{F}$.

Wires must obey constraints defined by the gates and direction of edges.

E.g. circuit that computes $x_1x_2(w_1+w_2)$ given inputs $(x,w) \in \mathbb{F}^{2+2}$



Defining SNARGs (for AC)

Tuple of eff. algorithms SNARG = (Setup, Preprocess, P, V):

1. $\text{Setup}(1^\lambda) \longrightarrow pp$
2. $\text{Preprocess}(pp, C) \longrightarrow (pk, vk)$

   C: arithmetic circuit; pk: proving key; vk: verification key
3. $P(pk, x, w) \longrightarrow \pi$

   x: instance; w: witness; π: succinct
4. $V(vk, x, \pi) \longrightarrow 0/1$

A SNARG must satisfy the following properties:

Completeness. $\forall C, (x,w) \in R_C$,

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

Soundness. $\forall x \notin \mathcal{L}(R_C) := \{x \mid \exists w \text{ s.t. } (x,w) \in R_C\}$, $\forall$ PPT P^*

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

Succinctness. $|\pi| \in o(|w|)$ ← succinct proof

$|V| \in O(|x|) + o(|w| + |C|)$ ← succinct runtime

Non-interactive. (P, V) is non-interactive.