Previously, we showed how to construct verifiable computation – this is suitable when the client knows the program and input

↳ This is a succinct non-interactive argument (SNARG) for a deterministic polynomial-time computation (SNARG for P)

But in many cases, the input might not be known to the verifier

↳ Example: Server publishes a hash of some database. Client performs a query and wants a proof that the query was performed correctly relative to the hash of the database.

↳ For this setting, we need a SNARG for NP. Namely, we consider the following NP relation:

- Statement: hash h of the database contents, output y of the query
- Witness: database D
- Relation checks that $h = \text{Hash}(D)$ and y is output of Query algorithm on D

We will construct a SNARG for NP where the size of the proof is a constant number of group elements, regardless of the complexity of the NP relation

↳ Construction will rely on random oracles

Starting point: Polynomial commitment scheme

We will work over $\mathbb{F}_p$ (the integers modulo p). A polynomial commitment scheme over $\mathbb{F}_p$ consists of four algorithms:

- $\text{Setup}(d)$: Takes the (max) degree of the polynomial and outputs a common reference string crs
- $\text{Commit}(\text{crs}, f) \to c$: Commits to the polynomial f (of degree at most d)
- $\text{Eval}(\text{crs}, c, f, x) \to \pi$: Computes an opening π to the evaluation $y = f(x)$
- $\text{Verify}(\text{crs}, c, x, y, \pi) \to 0/1$: Checks whether opening is valid or not

Correctness: Let $f(x) = f_0 + f_1 x + \cdots + f_d x^d$ be a polynomial and $x^* \in \mathbb{F}_p$ be a point.
If $\text{crs} \leftarrow \text{Setup}(d)$, $c \leftarrow \text{Commit}(\text{crs}, f)$, $\pi \leftarrow \text{Eval}(\text{crs}, c, x^*)$, then
$$\text{Verify}(\text{crs}, c, x^*, f(x^*), \pi) = 1$$

Binding: Given crs, difficult to come up with commitment c, a point x, and value/opening pairs (y_1, π_1), (y_2, π_2) where $y_1 \neq y_2$ and

$$\text{Verify}(crs, c, x, y_1, \pi_1) = 1 = \text{Verify}(crs, c, x, y_2, \pi_2)$$

For applications often need a stronger "soundness" property (will not be too important for understanding the construction)

Kate-Zaverucha-Goldberg (KZG) construction:

Setup (d): Sample $\alpha \xleftarrow{R} \mathbb{F}_p$

$$crs = (g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^d})$$

[p is the order of the group, scheme supports polynomials over $\mathbb{F}_p$]

Commit (crs, f): Commitment is the element $g^{f(\alpha)}$.

Suppose $f(X) = f_0 + f_1 X \cdots + f_d X^d$.

Let $crs = (g_0, g_1, \ldots, g_d)$ where $g_i = g^{\alpha^i}$. Then commitment is

$$c = \prod_{i=0}^{d} g_i^{f_i} = \prod_{i=0}^{d} g^{f_i \alpha^i} = g^{f(\alpha)}.$$

Eval (crs, c, f, x^*): Let $y = f(x^*)$. Goal is to construct a proof π that $y = f(x^*)$. where f is the polynomial associated with c.

Define the polynomial $\hat{f}(X) = f(X) - y$. Observe that $f(x) = y$ if and only if $\hat{f}(x^*) = 0$, or equivalently, if x^* is a root of $\hat{f}$. This means there exists a polynomial $q(X)$ such that

$$\hat{f}(X) = (X - x^*)\, q(X).$$

The opening will be a commitment to the polynomial $q(X) = \sum_{i=0}^{d-1} q_i X^i$

$$\pi = \prod_{i=0}^{d-1} g_i^{q_i} = \prod_{i=0}^{d-1} g^{q_i \alpha^i} = g^{q(\alpha)}$$

Verify (crs, c, x^*, y, π): Verifier will essentially check that the polynomials $\hat{f}$ and $(X - x^*)\, q(X)$ are equal at $X = \alpha$. Normally, we have that $c = g^{f(\alpha)}$ and $\pi = g^{q(\alpha)}$.

From $c = g^{f(\alpha)}$, verifier computes $c \cdot g^{-y} = g^{f(\alpha) - y} = g^{\hat{f}(\alpha)}$.

From $\pi = g^{q(\alpha)}$, verifier computes

$$e(g_1 g^{-x^*}, \pi) = e(g^{\alpha - x^*}, g^{q(\alpha)}) = e(g, g)^{(\alpha - x^*) q(\alpha)}.$$

Verification relation is thus

$$e(g, c \cdot g^{-y}) \stackrel{?}{=} e(g_1 g^{-x^*}, \pi)$$

Binding relies on the d-strong Diffie-Hellman assumption:

given $g, g^{\alpha}, g^{\alpha^2}, \ldots, g^{\alpha^d}$, hard to come up with $(c, g^{\frac{1}{\alpha+c}})$ for any $c \neq -\alpha$.

Suppose adversary produces a commitment $c = g^s$ and opens c to two different values y_1 and y_2 at x^* with proof $\pi_1 = g^{t_1}$ and $\pi_2 = g^{t_2}$.

Note: reduction does ~~not~~ know s, t_1, t_2

Then, by the verification relation:

$$e(g, c \cdot g^{-y_1}) = e(g^{\alpha - x^*}, \pi_1)$$
$$e(g, c \cdot g^{-y_2}) = e(g^{\alpha - x^*}, \pi_2)$$

In the exponent, this means

$$\begin{aligned} s - y_1 &= t_1(\alpha - x^*) \\ s - y_2 &= t_2(\alpha - x^*) \end{aligned} \quad \Rightarrow \quad t_1(\alpha - x^*) + y_1 - y_2 = t_2(\alpha - x^*)$$

$$\Rightarrow \quad (t_1 - t_2)(\alpha - x^*) = y_2 - y_1$$

$$\Rightarrow \quad \frac{t_1 - t_2}{y_2 - y_1} = \frac{1}{\alpha - x^*}$$

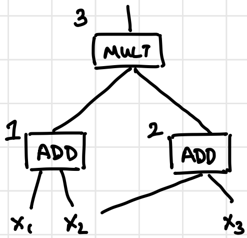
Thus $g^{\frac{1}{\alpha - x^*}} = g^{\frac{t_1 - t_2}{y_2 - y_1}} = \left(\frac{\pi_1}{\pi_2}\right)^{\frac{1}{y_2 - y_1}}$, which the reduction can compute.

Note: $y_1 \neq y_2$ since the adversary needs to open c two different ways. If the adversary outputs $x^* = \alpha$, then reduction trivially breaks the assumption.

We will develop protocols to prove additional properties on committed polynomials. To motivate this, we first sketch the ideas underlying the PLONK scheme.

For PLONK, the computational model will be <u>arithmetic circuits</u>

- Gates will be addition or multiplication
- Wires will be labeled by a field element ($\mathbb{F}_p$ element)



For any choice of input (x_1, x_2, x_3), can define an execution trace. Suppose $x_1 = 1$, $x_2 = 2$, $x_3 = 3$. Then the trace will be:

gate	left input	right input	output
1	1	2	3
2	2	3	5
3	3	5	15

Can be used to implement Boolean circuits

<u>The idea</u>: prover will choose a polynomial that interpolates the entire execution trace.

Take a point $\omega \in \mathbb{F}_p$. Let m be a bound on the number of gates in the circuit and let n be the number of public inputs (i.e., the statement) that is known to the verifier. We require $\text{ord}(\omega) > 3m + n$. Namely, the following elements are all distinct in $\mathbb{F}_p$: $\omega^{-n}, \omega^{-n+1}, \ldots, \omega^0, \omega^1, \ldots, \omega^{3m}$

The prover will interpolate the trace polynomial T where

$T(\omega^{-i})$ = value of i^{th} public input
$T(\omega^{3j})$ = value of left input to the j^{th} gate
$T(\omega^{3j+1})$ = value of right input to the j^{th} gate
$T(\omega^{3j+2})$ = value of output of j^{th} gate

The polynomial T encodes the entire execution of C. The prover commits to C using a polynomial commitment scheme. Now the prover needs to show the following:

1. Input consistency: $T(\omega^{-i})$ = value of i^{th} public input
2. Every gate is correctly implemented
3. Wires are labeled consistently: if output of gate j is left input of gate k, then $T(\omega^{3j+2}) = T(\omega^{3k})$.
4. Output gate has the correct value

Proving that the output gate has the correct value is just opening the polynomial commitment at $\omega^{3|C|-1}$.

Suffices to consider the other properties. All of these can be reduced to a "zero-testing" gadget: show that a polynomial $f(X)$ is zero on a set S.

First define the vanishing polynomial for S: $Z_S(X) = \prod_{t \in S} (X - t)$.

Then f is zero on S if and only if there exists a polynomial $q(X)$ such that $f(X) = Z_S(X) \cdot q(X)$.

Suppose the verifier has a commitment to f. To prove that f is zero on S, we can use the following protocol:

1. Prover commits to the polynomial q of degree at most $d - |S|$ where $f(X) = Z_S(X) \cdot q(X)$
2. Verifier samples a random $r \xleftarrow{R} \mathbb{F}_p$
3. Prover opens commitments to f and q at r
4. Verifier checks that $f(r) \stackrel{?}{=} Z_S(r) q(r)$

To see why this is sound. Suppose $f(X)$ is not zero on S. Then, there does not exist a polynomial $q(X)$ such that $f(X) = Z_S(X) \cdot q(X)$.

Consider the polynomial $h(X) := f(X) - Z_S(X) \cdot q(X)$. This is a polynomial of degree at most d and is not the zero polynomial. Thus, it has at most d roots. Then,

$$\Pr_{r \leftarrow \mathbb{F}_p} [h(r) = 0] = \frac{d}{p} = \text{negl.}$$

$$h(r) = 0 \iff f(r) = Z_S(r) q(r)$$

In the SNARG, the commitments are implemented using KZG polynomial commitments.

The randomness r is derived using the random oracle (by hashing the input) — as in Fiat-Shamir.

Proof then consists of three group elements: commitment to q, openings for f and q

Back to PLONK. Prover commits to trace polynomial T.

1. <u>Input consistency</u>: Suppose public input is $x = (x_1, \dots, x_n)$. Then, the prover should show that $T(\omega^{-i}) = x_i$ for all $i \in [n]$.

   To do so, prover (and verifier) interpolate polynomial $v(X)$ where $v(\omega^{-i}) = x_i$.

   Then, the polynomial $T(X) - v(X)$ is zero on the set $S = \{\omega^{-1}, \dots, \omega^{-n}\}$. Prover and the verifier now run the above zero-testing protocol.

   <u>Note</u>: In the zero-testing protocol, the prover needs to reveal $T(r) - v(r)$. It does so by revealing $T(r)$ and the verifier can then compute $T(r) - v(r)$ itself.

2. <u>Gate consistency</u>: Define a selector polynomial $v(X)$ where

   $v(\omega^{3\ell}) = 1$ if gate ℓ is an addition gate
   $v(\omega^{3\ell}) = 0$ if gate ℓ is a multiplication gate

   Suppose all the gates are implemented correctly:
   - If gate ℓ is an addition gate: $T(\omega^{3\ell}) + T(\omega^{3\ell+1}) = T(\omega^{3\ell+2})$
   - If gate ℓ is a multiplication gate: $T(\omega^{3\ell}) \cdot T(\omega^{3\ell+1}) = T(\omega^{3\ell+2})$

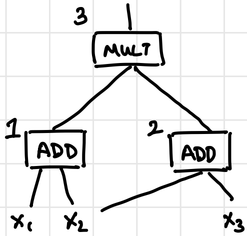
   This means for all $X \in \{\omega^0, \omega^3, \dots, \omega^{3|C|-3}\}$,

   $$v(X)\left[T(X) + T(\omega X)\right] + (1 - v(X))\left[T(X) \cdot T(\omega X)\right] = T(\omega^2 X)$$

   Reduces to zero-test protocol on the set $\{\omega^0, \omega^3, \dots, \omega^{3|C|-3}\}$.

   <u>Note</u>: To implement this protocol, verifier needs to evaluate polynomial
   $$v(X)\left[T(X) + T(\omega X)\right] + (1 - v(X))\left[T(X)\, T(\omega X)\right] - T(\omega^2 X)$$
   at a random point r. This can be implemented using KZG by having prover open T at r, ωr, and $\omega^2 r$. Verifier can compute $v(r)$ itself.

3. <u>Wire consistency</u>:



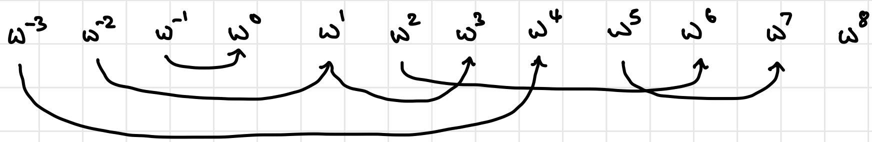
gate	left input		right input		output	
1	1	(ω^0)	2	(ω^1)	3	(ω^2)
2	2	(ω^3)	3	(ω^4)	5	(ω^5)
3	3	(ω^6)	5	(ω^7)	15	(ω^8)

inputs: 1 (ω^{-1}) 2 (ω^{-2}) 3 (ω^{-3})

In this example, we would require that

$$T(\omega^{-1}) = T(\omega^0)$$
$$T(\omega^{-2}) = T(\omega^1) = T(\omega^3)$$
$$T(\omega^{-3}) = T(\omega^4)$$
$$T(\omega^2) = T(\omega^6)$$
$$T(\omega^5) = T(\omega^7)$$

Whenever a wire value is used multiple times, we introduce a constraint. Every wire value participates in at most <u>one</u> constraint group:



We can view this "replication pattern" as inducing a permutation P on the set $\{\omega^{-3}, \omega^{-2}, \dots, \omega^8\}$. For each input, ω^i, $P(\omega^i)$ sends it to ω^j where j is the index of the next copy of the wire associated with index i.

P can be described by a polynomial of degree $3m+n$ (just like T). Checking equality of the wire constraints then boils down to checking $T(x) \stackrel{?}{=} T(P(x))$ for all $x \in \{\omega^{-n}, \dots, \omega^{3m}\}$. The polynomial P is known to the verifier so this can again be done using the zero-testing protocol.

<u>Summary</u>: To prove that $C(x, w) = 1$, prover commits to the execution trace $T(x)$ of C and then proves the following statements:

- Input consistency
- Gate consistency
- Wire consistency
- Output correctness

$\Big\}$ Each proof requires revealing a <u>constant</u> number of group elements (i.e., commitments + openings to the polynomial commitment scheme)

Soundness requires random oracle (to make the interactive protocol non-interactive) and the algebraic group model (or generic group model) to argue soundness of the KZG scheme

Many extensions: Can modify base protocol so prover complexity is quasi-linear in $|C|$ rather than quadratic

Can consider multivariate polynomials over $\mathbb{F}_2^t$ to support linear-time prover (HyperPlonk)

Can support more general gates by extending gate consistency checks