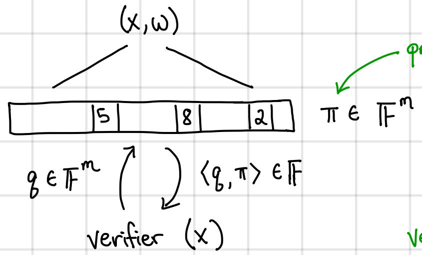
Today: another abstraction to construct succinct arguments

- does not rely on random oracles (needs a CRS instead)
- asymptotically shorter arguments: $\tilde{O}(\lambda)$ proofs for proving NP relations of size $\mathrm{poly}(\lambda)$
- leads to the most compact SNARGs (pairing-based construction is 3 group elements ~ 128 bytes) (basis of Zerocash protocol)

Starting point: different information-theoretic proof system

Linear PCPs [Ishai-Kushilevitz-Ostrovsky, 2007]:



proof is a vector of field elements (not necessarily binary field)

verifier is given oracle access to the linear PCP proof oracle (e.g., verifier submits a query vector $q \in \mathbb{F}^m$ and receives the response $\langle q, \pi \rangle \in \mathbb{F}$

Definition. A linear PCP for an NP language $\mathcal{L}$ (with corresponding relation $\mathcal{R}$) consists of two algorithms (P, V) with the following properties:

- **Completeness**: If $(x, w) \in \mathcal{R}$, then if we set $\pi \leftarrow P(x, w)$:
$$\Pr[V^{\langle \pi, \cdot \rangle}(x) = 1] = 1$$
- **Soundness**: For all $x \notin \mathcal{L}$ and all $\pi^* \in \mathbb{F}^m$ — proof dimension
$$\Pr[V^{\langle \pi^*, \cdot \rangle}(x) = 1] \leq \varepsilon$$
soundness error

Constructing linear PCPs (for Boolean circuit satisfiability – let C be the circuit)

- Can instantiate using Walsh-Hadamard code [Arora-Lund-Motwani-Sudan-Szegedy, 1992]
  [3 queries, query length $m = O(|C|^2)$, soundness error $\varepsilon = 2/|\mathbb{F}|$]
- Can instantiate using quadratic span programs [Gennaro-Gentry-Parno-Raykova, 2013]
  [3 queries, query length $m = O(|C|)$, soundness error $\varepsilon = O(s)/|\mathbb{F}|$]

Several useful properties of these linear PCP constructions:

- Verifier is oblivious (i.e., the queries do not depend on the choice of the statement)
- Verification algorithm is a <u>quadratic</u> relation over the linear PCP responses [useful primarily for achieving public verifiability]
  (if $r_1 = \langle \pi, q_1 \rangle, \ldots, r_k = \langle \pi, q_k \rangle$, verifier's response is quadratic function in the variables $r_1, \ldots, r_k$)

From linear PCPs to SNARGs: Suppose verifier in linear PCP system is <u>oblivious</u>. Then, we can generate the verifier's queries ahead of time (before the statement is known).

<u>Key idea:</u> generate the queries during setup:

$\quad \text{Setup}(1^\lambda) \longrightarrow (\sigma, \tau)$ where σ is the CRS and τ is the verification state

Suppose we have a k-query linear PCP. Then:

$\quad \text{Setup}(1^\lambda) \longrightarrow (\sigma, \tau)$: generate queries $q_1, \ldots, q_k \in \mathbb{F}^m$ and linear PCP verification state τ_{LPCP}

$\qquad \hookrightarrow \sigma = (q_1, \ldots, q_k)$ and $\tau = \tau_{\text{LPCP}}$

<u>Prover's operation</u>: 1. Encode statement-witness (x, w) as linear PCP $\pi \in \mathbb{F}^m$

2. Compute responses to verifier's queries $r_1 = \langle \pi, q_1 \rangle, \ldots, r_k = \langle \pi, q_k \rangle \in \mathbb{F}$

3. Proof consists of linear PCP responses $r_1, \ldots, r_k \in \mathbb{F}$

Verifier runs verification procedure for underlying linear PCP (using $r_1, \ldots, r_k$ and τ)

<u>Problem</u>: Prover can choose proof $\pi \in \mathbb{F}^m$ <u>after</u> seeing the verifier's queries $\Rightarrow$ cannot appear to soundness of linear PCP!

<u>Solution</u>: encrypt verifier's queries with additively-homomorphic encryption scheme [suffices for designated-verifier SNARGs]

$\quad \text{Setup}(1^\lambda)$: $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$

generate linear PCP queries $q_1, \ldots, q_k \in \mathbb{F}^m$ and τ_{LPCP}

let $ct_{i,j}$ for $i \in [k]$ and $j \in [m]$ be $ct_{i,j} \leftarrow \overbrace{\text{Encrypt}(pk, q_{ij})}^{\text{encrypt each component of the query vector}}$, and let $ct_i = (ct_{i,1}, \ldots, ct_{i,m})$

output $\sigma = (ct_1, \ldots, ct_k)$ and $\tau = (sk, \tau_{\text{LPCP}})$

To construct a proof, prover homomorphically computes

$$ct_i' \leftarrow \Sigma_{j \in [m]} \pi_j \cdot ct_{i,j} = \text{Encrypt}(pk, \langle \pi, q_i \rangle)$$

Verifier takes $ct_1', \ldots, ct_k'$, decrypts the ciphertexts to obtain responses $r_1, \ldots, r_k$ and applies linear PCP verification

Problem: Prover need not apply <u>same</u> linear function π to construct its queries

Solution: Introduce an additional consistency check

prover operates on queries $q_1, \dots, q_k$ and can compute

$$\left.\begin{aligned} r_1 &= \langle q_1, \pi_1 \rangle \\ r_2 &= \langle q_2, \pi_2 \rangle \\ &\vdots \\ r_k &= \langle q_k, \pi_k \rangle \end{aligned}\right\}$$

not quite accurate since prover can compute each response to be a linear function of all of the query components — but same idea applies to the general setting

useful trick: <u>random</u> linearity check — verifies chooses $\alpha_1, \dots, \alpha_k \xleftarrow{R} \mathbb{F}$ and submits a query $q_{k+1} = \sum_{i \in [k]} \alpha_i q_i \in \mathbb{F}^m$

verifier additionally checks that $r_{k+1} = \sum_{i \in [k]} \alpha_i r_i$

observe: if prover uses the same π for all queries:

$$\sum_{i \in [k]} \alpha_i r_i = \sum_{i \in [k]} \alpha_i \langle q_i, \pi \rangle = \left\langle \sum_{i \in [k]} \alpha_i q_i, \pi \right\rangle = \langle q_{k+1}, \pi \rangle$$

if prover does not use the same π for all queries, then

$$\sum_{i \in [k]} \alpha_i r_i = \sum_{i \in [k]} \alpha_i \langle q_i, \pi_i \rangle \quad \text{and} \quad \langle q_{k+1}, \pi_{k+1} \rangle = \sum_{i \in [k]} \alpha_i \langle q_i, \pi_{k+1} \rangle$$

verifier accepts only if

$$\sum_{i \in [k]} \alpha_i \langle q_i, \pi_i \rangle = \sum_{i \in [k]} \alpha_i \langle q_i, \pi_{k+1} \rangle$$

$$\iff \sum_{i \in [k]} \alpha_i \underbrace{\langle q_i, \pi_i - \pi_{k+1} \rangle}_{\substack{\text{non-zero value} \\ \text{since } \pi_i - \pi_{k+1} \neq 0 \text{ for some } i}} = 0$$

[if $q_i = 0$ on all positions where $\pi_i \neq \pi_{k+1}$, then can replace π_i with π_{k+1} and there is no longer an inconsistency]

↳ since $\alpha_i \xleftarrow{R} \mathbb{F}$, and independent of $\pi_1, \dots, \pi_{k+1}$, over the choice of α_i, this relation is satisfied with probability at <u>most</u> $\frac{1}{|\mathbb{F}|}$

[Schwartz-Zippel lemma]

Problem: To appeal to soundness of linear PCP, we need to ensure that prover only implements linear strategy

Solution: Assume encryption scheme is "linear-only" (i.e., only supports linear homomorphisms)

"Linear-only" (informally):

for all efficient adversaries $\mathcal{A}$, there exists an extractor $\mathcal{E}$ where for any sequence of messages $m_1, \dots, m_k$:

$$\left.\begin{aligned} ct_i &\leftarrow \mathsf{Encrypt}(pk, m_i) \quad \forall i \in [k] \\ ct' &\leftarrow \mathcal{A}(pk, ct_1, \dots, ct_k) \end{aligned}\right| \quad (\pi, b) \leftarrow \mathcal{E}(pk, ct_1, \dots, ct_k)$$

it follows that

$$\mathsf{Decrypt}(sk, ct') = \sum_{i \in [k]} \pi_i m_i + b \in \mathbb{F}$$

"any ciphertext that the adversary can compute can be explained by a linear function of the provided ciphertext"

Note: Not your typical cryptographic assumption (non-falsifiable)

- Typical cryptographic assumptions like factoring, DDH, LWE can be formulated as a game between a challenger and an adversary
- To break the linear-only assumption, need to exhibit some adversary such that there is <u>no efficient extractor</u> (can be very challenging!)

Putting the pieces together:

Setup(1^λ): $(pk, sk) \leftarrow \text{KeyGen}(1^\lambda)$ generate public/private key for a linear-only encryption scheme

compute linear PCP queries $q_1, \dots, q_k$, linear consistency check query q_{k+1} and linear PCP verification state τ_{LPCP}

encrypt queries $q_1, \dots, q_{k+1}$ (component-wise) using linear-only encryption scheme to obtain encrypted queries $ct_1, \dots, ct_{k+1}$

publish $\sigma = (pk, ct_1, \dots, ct_{k+1})$ and $\tau = \tau_{LPCP}$

Prove(σ, x, w): construct linear PCP proof $\pi \in \mathbb{F}^m$ from (x, w)

homomorphically compute $ct_i' \leftarrow \text{Encrypt}(pk, \langle q_i, \pi \rangle)$ from encrypted queries $ct_1, \dots, ct_{k+1}$

output SNARG $\pi_{SNARG} = (ct_1', \dots, ct_{k+1}')$

Verify(τ, π_{SNARG}, x): decrypt ciphertexts in π_{SNARG} and verify responses using linear PCP verifier

Completeness: Follows by correctness of encryption scheme + completeness of linear PCP

Soundness (Sketch): Prover's strategy can be explained by a linear function [linear-only encryption]

Consistent linear function used for all responses [linear consistency check]

Prover's linear function independent of the verifier's queries [semantic security]

$\Downarrow$

appeal to soundness of linear PCP

Succinctness: Proof consists of $(k+1)$ ciphertexts

↳ Existing linear PCPs are constant query (e.g., $k=3$) $\Longrightarrow$ SNARG proof consists of 4 ciphertexts $\underbrace{(\tilde{O}(\lambda) \text{ bit proofs!})}$ proof size independent of circuit size!

Concrete instantiations: We can instantiate linear-only encryption with Paillier, ElGamal (HW4) (over small fields), or Regev (lattice-based)

↳ gives designated-verifier SNARGs

Can also instantiate with pairing-based linear-only encodings

↳ if the underlying linear PCP has a quadratic verification relation, then can verify the SNARG publicly (by evaluating the check in the exponent using the pairing)

↳ Most efficient instantiations based on quadratic span/arithmetic programs (3 group elements, ~128 bytes) [Groth 2016]

(Basis for privacy-preserving concurrencies like Zcash)

↳ Most fancy crypto that has seen large-scale deployment!

Note: Techniques readily generalize to yield both zero-knowledge as well as proofs of knowledge (i.e. zkSNARKs)

To wrap up: In this course, we showed how to use cryptography to protect communication

- Confidentiality of communication (encryption)
- Integrity for communication (signatures)

Proof systems generalize integrity for communication to general computations

We can also perform general computations with strong confidentiality / integrity guarantees

↳ More next semester!

A brief epilogue (constructing linear PCPs from quadratic arithmetic programs):

We will consider the language of rank-1 constraint satisfiability (R1CS) — captures arithmetic circuit satisfiability:
- R1CS instance is a constraint satisfiability problem where each constraint is a quadratic relation
- Variables are $[w_1, \dots, w_n, w_{n+1}, \dots, w_{n+h}]$ and values are field elements $w_i \in \mathbb{F}$ (e.g., integers modulo p)
- Each constraint is a quadratic function:

$$\left(a_0 + \sum_{i \in [n]} a_i w_i\right)\left(b_0 + \sum_{i \in [n]} b_i w_i\right) = \left(c_0 + \sum_{i \in [n]} c_i w_i\right)$$

A single constraint can be defined as $\vec{a} = (a_0, a_1, \dots, a_n)$

$\vec{b} = (b_0, b_1, \dots, b_n)$

$\vec{c} = (c_0, c_1, \dots, c_n)$

We can write this also in matrix form. An element $\bar{w} = (1, w_1, \dots, w_{n+h})$ satisfies the system if

$(A\bar{w}) \circ (B\bar{w}) = C\bar{w}$

where $A = \begin{bmatrix} — \vec{a}_1 — \\ — \vec{a}_2 — \\ \vdots \\ — \vec{a}_m — \end{bmatrix}$ $B = \begin{bmatrix} — \vec{b}_1 — \\ — \vec{b}_2 — \\ \vdots \\ — \vec{b}_m — \end{bmatrix}$ and $C = \begin{bmatrix} — \vec{c}_1 — \\ — \vec{c}_2 — \\ \vdots \\ — \vec{c}_n — \end{bmatrix}$

and $u \circ v$ denotes the Hadamard product (component-wise product)
- Let $\mathcal{C} = (A, B, C)$ be an R1CS system over $\mathbb{F}$. We say that a statement $x \in \mathbb{F}^n$ satisfies $\mathcal{C}$ if there exist $\bar{w} = (1, x_1, \dots, x_n, w_{n+1}, \dots, w_{n+h})$ where $(A\bar{w}) \circ (B\bar{w}) = C\bar{w}$.
- Not difficult to show that R1CS captures Boolean circuit satisfiability (which is NP-complete):
  - Let $C: \{0,1\}^n \times \{0,1\}^h \to \{0,1\}$ be a Boolean circuit with m gates (assume XOR and AND gates) and t wires
  - R1CS instance will have t variables, corresponding to wire values of C; and

    $m + t$ constraints:

    indices of input wires / index of output wire

    1) gate constraints: for XOR gate (i, j, k): $w_k = w_i + w_j$ [constraint vectors for $\vec{a}, \vec{b}, \vec{c}$ are just standard basis vectors]

       for AND gate (i, j, k): $w_k = w_i \cdot w_j$

    2) wire validity: for each wire in circuit, $w_i(1 - w_i) = 0$ [ensures that each $w_i \in \{0,1\}$]

       $\hookrightarrow w_i^2 = w_i$ [quadratic constraint]

How do we quickly check that $(A\bar{w}) \circ (B\bar{w}) \stackrel{?}{=} C\bar{w}$?

Key idea: Use polynomial magic!

$$\begin{array}{cl} A = \begin{bmatrix} a_{1,0} & a_{1,1} & \cdots & a_{1,n+h} \\ a_{2,0} & a_{2,1} & \cdots & a_{2,n+h} \\ \vdots & \vdots & & \vdots \\ a_{m,0} & a_{m,1} & \cdots & a_{m,n+h} \end{bmatrix} & \begin{array}{l} \leftarrow t_1 \\ \leftarrow t_2 \\ \\ \leftarrow t_m \end{array} \\ \quad\;\; \begin{array}{cccc} \uparrow & \uparrow & & \uparrow \\ A_0 & A_1 & & A_{n+h} \end{array} & \end{array}$$

associate each row (i.e., each constraint with a point $t_i \in \mathbb{F}$)

associate each column with a polynomial of degree $m-1$ where

$A_i(t_1) = a_{1,i}$

$A_i(t_2) = a_{2,i}$

$\vdots$

$A_i(t_m) = a_{m,i}$

Namely:

$$A = \begin{bmatrix} A_0(t_1) & \cdots & A_{nth}(t_1) \\ \vdots & & \vdots \\ A_0(t_m) & \cdots & A_{nth}(t_m) \end{bmatrix}$$

Define B_i and C_i accordingly (using the same set of points $t_0, \dots, t_m$)

By construction,

$$A\bar{w} = \begin{bmatrix} w_0 A_0(t_1) + \cdots + w_{nth} A_{nth}(t_1) \\ \vdots \\ w_m A_0(t_m) + \cdots + w_{nth} A_{nth}(t_m) \end{bmatrix} = \begin{bmatrix} \bar{A}(t_1) \\ \vdots \\ \bar{A}(t_m) \end{bmatrix}$$

$$\text{where } \bar{A} = w_0 A_0 + \cdots + w_{nth} A_{nth}$$

Now $(A\bar{w}) \circ (B\bar{w}) = C\bar{w}$ if and only if

$$\bar{A}(z) \cdot \bar{B}(z) = \bar{C}(z) \quad \text{for all } z = t_1, t_2, \dots, t_m$$

Eqivalently:

$$\bar{A}(z) \cdot \bar{B}(z) - \bar{C}(z) = 0 \quad \text{for all } z = t_1, t_2, \dots, t_m$$

Let $Z(z)$ be the polynomial $Z(z) = (z-t_1)(z-t_2)\cdots(z-t_m)$. Then,

$$\bar{A}(z) \cdot \bar{B}(z) - \bar{C}(z) = Z(z) H(z)$$

for some polynomial $H(z)$ of degree at most $m-1$.

<u>Observe</u>: If $(A\bar{w}) \circ (B\bar{w}) = C\bar{w}$ then we can find $H(z)$ of degree $m-1$

$$\forall z \in \mathbb{F}: \quad \bar{A}\bar{B} - \bar{C} = HZ$$

If there is no $\bar{w}$ where $(A\bar{w}) \circ (B\bar{w}) = C\bar{w}$, then for all $H(z)$ of degree $m-1$:

$$\bar{A}\bar{B} - \bar{C} \neq HZ$$

since $(\bar{A}\bar{B} - \bar{C})(t_i) \neq 0$ but $Z(t_i) = 0$

Since $\bar{A}, \bar{B}, \bar{C}, H, Z$ have degree $m-1$, and $(\bar{A}\bar{B} - \bar{C}) \neq HZ$, there are at most $2m-2$ points $z \in \mathbb{F}$ where $\bar{A}(z)\bar{B}(z) - \bar{C}(z) = H(z)Z(z)$

<u>Idea</u>: Verifier will check evaluation of $\bar{A}\bar{B} - \bar{C}$ and H at a random $\tau \xleftarrow{R} \mathbb{F}$

- True statement: $(\bar{A}\bar{B} - \bar{C})(\tau) = H(\tau)Z(\tau)$
- False statement: $(\bar{A}\bar{B} - \bar{C})(\tau) \neq H(\tau)Z(\tau)$ with prob. $1 - \frac{2m-2}{|\mathbb{F}|}$.

Linear PCP construction:

- Polynomials $\bar{A}, \bar{B}, \bar{C}$ depends only on R1CS system (known to verifier)
- $Z(z)$ is a fixed polynomial - depends only on evaluation domain
- LPCP responses will be to compute $\bar{A}(\tau), \bar{B}(\tau), \bar{C}(\tau), H(\tau)$ and verifier checks that

$$\bar{A}(\tau)\bar{B}(\tau) - \bar{C}(\tau) \stackrel{?}{=} H(\tau)Z(\tau)$$

- Recall: $\bar{A} = \sum_{i=0}^{n+h} \bar{w}_i A_i$

$\bar{B} = \sum_{i=0}^{n+h} \bar{w}_i B_i \implies \bar{A}(\tau), \bar{B}(\tau), \bar{C}(\tau)$ are linear functions of $\underbrace{A_i(\tau),\ B_i(\tau),\ C_i(\tau)}_{\text{known to verifier}}$

$\bar{C} = \sum_{i=0}^{n+h} \bar{w}_i C_i$

prover knows w_i

To compute $H(\tau)$, we can write

$H(\tau) = \sum_{i=0}^{m-1} h_i \tau^i$ where $h_i \in \mathbb{F}$ are the coefficient of H

known to verifier

prover knows h_i

- Quadratic arithmetic program:
  - LPCP proof: $\pi = [w_0, \ldots, w_{n+h}, h_0, \ldots, h_{m-1}]$
  - LPCP queries: $\tau \xleftarrow{R} \mathbb{F}$

query for $\bar{A}(\tau)$: $[A_0(\tau), \ldots, A_{n+h}(\tau), 0, \ldots, 0]$

$\bar{B}(\tau)$: $[B_0(\tau), \ldots, B_{n+h}(\tau), 0, \ldots, 0]$

$\bar{C}(\tau)$: $[C_0(\tau), \ldots, C_{n+h}(\tau), 0, \ldots, 0]$

$H(\tau)$: $[0, \ldots, 0, \tau^0, \tau^1, \ldots, \tau^{m-1}]$

verifier checks that $\bar{A}(\tau)\bar{B}(\tau) - \bar{C}(\tau) = H(\tau)Z(\tau)$

  - Note: nothing here to check for statement
    - Approach 1: add 1 check that $(w_1, \ldots, w_n) = (x_1, \ldots, x_n)$ – can just take random linear combination
    - Approach 2: Remove $A_1(\tau), \ldots, A_n(\tau)$, $B_1(\tau), \ldots, B_n(\tau)$, $C_1(\tau), \ldots, C_n(\tau)$ from queries and just add in those components at verification time (since verifier knows statement)