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 poly(\(\lambda\))
- leads to the most compact SNARGs (pairing-based construction is 3 group elements \(\sim 128\) bytes)
  (basis of ZeroCash protocol)

Starting point: different information-theoretic proof system

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

\[(x, \omega) \xrightarrow{\text{proof}} (121) \]
\[\pi \in \mathbb{F}^n \]
\[g \in \mathbb{F}^n \]
\[\langle g, \pi \rangle \in \mathbb{F} \]

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

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

- **Completeness:** If \((x, \omega) \in R\), then if we set \(\pi \leftarrow P(x, \omega)\):
  \[\Pr[V(\pi, x) = 1] = 1\]

- **Soundness:** For all \(x \notin L\) and all \(\pi \in \mathbb{F}^n\)
  \[\Pr[V(\pi, x) = 1] \leq \varepsilon\]

Constructing linear PCPs (for Boolean circuit satisfiability - let \(C\) be the circuit)
  \[3 \text{ queries, query length } m = O(1C^3), \text{ soundness error } \varepsilon = 2/|\mathbb{F}|\]
- Can instantiate using quadratic span programs [Gennaro-Gentry-Papo-Rejksum, 2013]
  \[3 \text{ queries, query length } m = O(1C), \text{ soundness error } \varepsilon = O(\delta)/|\mathbb{F}|\]

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 quadratic relation over the linear PCP responses
  \[(\text{useful primarily for achieving public verifiability})\]
  \[f_r = \langle r_1, \phi_1 \rangle, \ldots, f_{\kappa} = \langle r_\kappa, \phi_\kappa \rangle, \text{ verifier's response is quadratic function in the variables } r_1, \ldots, r_\kappa\]
From linear PCPs to SNARGs: Suppose verifier in linear PCP system is oblivious. Then, we can generate the verifier's queries ahead of time (before the statement is known).

**Key idea:** generate the queries during setup:

1. Setup $(1^*): (\sigma, \tau) \text{ where } \sigma \text{ is the CRS and } \tau \text{ is the verification state.}$

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

2. Setup $(2^*): \text{ generate queries } g_1, ..., g_k \in \mathbb{F}^n \text{ and linear PCP verification state } \tau_{\text{PCP}}$

\[ \Rightarrow \sigma = (g_1, ..., g_k) \text{ and } \tau = \tau_{\text{PCP}} \]

**Prover's operation:**

1. Encode statement-witness $(x, w)$ as linear PCP $x \in \mathbb{F}^n$
2. Compute responses to verifier's queries
   \[ r_i = (\pi, q_i), ..., r_k = (\pi, q_k) \in \mathbb{F} \]
3. Proof consists of linear PCP responses $r_1, ..., r_k \in \mathbb{F}$

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

**Problem:** Prover can choose proof $x \in \mathbb{F}^n$ after seeing the verifier's queries $\Rightarrow$ cannot appear to soundness of linear PCP!

**Solution:** encrypt verifier's queries with additively-homomorphic encryption scheme [suffices for designated-verifier SNARGs]

1. Setup $(1^*): (pk, sk) \leftarrow \text{KeyGen}(1^*)$
2. Generate linear PCP queries $g_1, ..., g_k \in \mathbb{F}^n$ and $\textbf{Encrypt}$ each component of the query vector
3. Let $C_{ij}$ for $i \in [k]$ and $j \in [m]$ be $C_{ij} = \textbf{Encrypt}(pk, q_{ij})$, and let $C_i = (C_{i1}, ..., C_{im})$
4. Output $\sigma = (C_1, ..., C_k)$ and $\tau = (sk, \tau_{\text{PCP}})$

To construct a proof, prover homomorphically computes

\[ C'_i = \Sigma_{j=1}^m C_{ij} = \textbf{Encrypt}(pk, (\pi, q_i)) \]

Verifier takes $C'_1, ..., C'_k$, decrypts the ciphertexts to obtain responses $r_1, ..., r_k$ and applies linear PCP verification.
**Problem:** Prover need not apply same linear function \( \pi \) to construct its queries.

**Solution:** Introduce an additional consistency check.

Prover operates on queries \( g_1, \ldots, g_k \) and can compute:

\[
\begin{align*}
 r_1 &= \langle g_1, \pi \rangle \\
r_2 &= \langle g_2, \pi \rangle \\
&\vdots \\
r_k &= \langle g_k, \pi \rangle
\end{align*}
\]

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: random linearity check — verifies a linear function \( \alpha_1, \ldots, \alpha_k \in \mathbb{F} \) and submits a query \( g_{\text{in}} = \sum_{i \in \mathbb{Z}} \alpha_i g_i \in \mathbb{F}^n \)

Verifier additionally checks that \( r_{\text{in}} = \sum_{i \in \mathbb{Z}} \alpha_i r_i \).

**Observe:** if prover uses the same \( \pi \) for all queries:

\[
\sum_{i \in \mathbb{Z}} \alpha_i r_i = \sum_{i \in \mathbb{Z}} \alpha_i \langle g_i, \pi \rangle = \langle \sum_{i \in \mathbb{Z}} \alpha_i g_i, \pi \rangle = \langle g_{\text{in}}, \pi \rangle
\]

if prover does not use the same \( \pi \) for all queries, then

\[
\sum_{i \in \mathbb{Z}} \alpha_i r_i = \sum_{i \in \mathbb{Z}} \alpha_i \langle g_i, \pi_i \rangle \quad \text{and} \quad \langle g_{\text{in}}, \pi_{\text{in}} \rangle = \sum_{i \in \mathbb{Z}} \alpha_i \langle g_i, \pi_{\text{in}} \rangle
\]

Verifier accepts only if

\[
\left\langle \sum_{i \in \mathbb{Z}} \alpha_i, \langle q_i, \pi_i \rangle \right\rangle = \sum_{i \in \mathbb{Z}} \alpha_i \langle q_i, \pi_{\text{in}} \rangle
\]

\[\iff \sum_{i \in \mathbb{Z}} \alpha_i \langle q_i, \pi_i - \pi_{\text{in}} \rangle = 0\]

\(\quad\text{non-zero value since} \pi_i - \pi_{\text{in}} \neq 0 \text{ for some } i\)

\[\left\{ \begin{array}{l}
\text{if } g_i = 0 \text{ on all positions where } \pi_i \neq \pi_{\text{in}}, \\
\text{then can replace } \pi_i \text{ with } \pi_{\text{in}} \text{ and there is no longer an inconsistency} \\
\text{this relation is satisfied with probability at most } \frac{1}{|\mathbb{F}|} \quad \text{[Schwartz-Zippel lemma]}
\end{array} \right.\]

**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 \( A \), there exists an extractor \( E \) alone for any sequence of messages \( m_1, \ldots, m_k \):

\[
\begin{align*}
\text{ct}_i &\leftarrow \text{Encrypt}(pk, m_i) \quad \forall i \in \mathbb{Z} \\
\text{ct}' &\leftarrow A(pk, \text{ct}_1, \ldots, \text{ct}_k) \\
(\pi, b) &\leftarrow E(pk, \text{ct}_1, \ldots, \text{ct}_k)
\end{align*}
\]

it follows that

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

“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-falseifiable).

- 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 no efficient extractor (can be very challenging!)
Putting the pieces together:

Setup(1^n) : (pk, sk) ← KeyGen(1^n) (generate public/private key for a linear-only encryption scheme)

correct PCP queries g_1, ... , g_k, linear consistency check query g_{k+1} and linear PCP verification oracle Ttcp

encrypt queries g_1, ... , g_k (component-wise) using linear-only encryption scheme to obtain encrypted queries

\( c_{t_1}, ..., c_{t_k} \)

publish \( \sigma = (pk, c_{t_1}, ..., c_{t_k}) \) and \( T = Ttcp \)

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

- homomorphically compute \( c_{t_i} \leftarrow \text{Encrypt}(pk, g_i, \pi) \) from encrypted queries \( c_{t_1}, ..., c_{t_k} \)

output SNARG Tlsnaro = \( (c_{t_1}, ..., c_{t_k}) \)

Verify(\tau, Tlsnaro, x) : decrypt ciphertexts in Tlsnaro 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]

↓

appeal to soundness of linear PCP

Soundness: Proof consists of \((k+1)\) ciphertexts

\( 1 \rightarrow \) Existing linear PCPs are constant query (e.g., \( k=3 \)) \( \Rightarrow \) SNARG proof consists of \( 4 \) ciphertexts \( (20\) bit proofs! \)

Concrete instantiations:

- We can instantiate linear-only encryption with Paillier, ElGamal (over small fields), or Regev

\( 1 \rightarrow \) gives designated-verifier SNARGs

- Can also instantiate with pairing-based linear-only encodings

\( 1 \rightarrow \) 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)

\( 1 \rightarrow \) Most efficient instantiations based on quadratic gap/arithmetic programs

(3 group elements, ~81 bytes) [Groth 2016]

(Basis for privacy-preserving concurrencies like Zcash)

\( 1 \rightarrow \) 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 generate integrity for communication to general computations.

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

\( 1 \rightarrow \) More next semester!
A brief epilogue (constructing linear PCPs from quadratic arithmetic programs):

We will consider the language of rank-1 constraint satisfaction (RICs) — captures arithmetic circuit satisfiability:
- RICS instance is a Constraint Satisfaction problem where each constraint is a quadratic relation
- Variables are \( w_1, w_2, w_3, \ldots, w_{mn} \) and values are field elements \( w_i \in \mathbb{F} \) (e.g., integers modulo \( p \))
- Each constraint is a quadratic function:
  \[
  \left( a_0 + \sum_{i=1}^n a_i w_i \right) \left( b_0 + \sum_{i=1}^n b_i w_i \right) = \left( c_0 + \sum_{i=1}^n c_i w_i \right)
  \]

A single constraint can be defined as:
\[
\begin{align*}
\bar{a} &= (a_0, a_1, \ldots, a_n) \\
\bar{b} &= (b_0, b_1, \ldots, b_n) \\
\bar{c} &= (c_0, c_1, \ldots, c_n)
\end{align*}
\]

We can write this also in matrix form. An element \( \bar{w} = (1, w_1, \ldots, w_{mn}) \) satisfies the system if
\[
(A \bar{w}) \odot (B \bar{w}) = C \bar{w}
\]

where
\[
A = \begin{bmatrix}
\bar{a}_1 \\
\bar{a}_2 \\
\vdots \\
\bar{a}_n
\end{bmatrix} \\
B = \begin{bmatrix}
\bar{b}_1 \\
\bar{b}_2 \\
\vdots \\
\bar{b}_n
\end{bmatrix} \\
C = \begin{bmatrix}
\bar{c}_1 \\
\bar{c}_2 \\
\vdots \\
\bar{c}_n
\end{bmatrix}
\]

and \( u \odot v \) denotes the Hadamard product (component-wise product).

Let \( C = (A, B, C) \) be an RICS system over \( \mathbb{F} \). We say that a statement \( x \in \mathbb{F}^n \) satisfies \( C \) if there exist \( \bar{w} = (1, w_1, \ldots, w_{mn}) \) where \( (A \bar{w}) \odot (B \bar{w}) = C \bar{w} \).

- Not difficult to show that RICS captures Boolean circuit satisfiability (which is NP-complete):
  - Let \( C : \{0,1\}^m \times \{0,1\}^t \rightarrow \{0,1\}^f \) be a Boolean circuit with \( m \) gates (assume \( \text{XOR} \) and \( \text{AND} \) gates) and \( t \) wires
  - RICS instance will have \( t \) variables, corresponding to wire values of \( C \); and
  - \( m + t \) constraints:
    1) gate constraints: for \( \text{XOR} \) gate \((i,j,k)\):
       \( W_k = W_i + W_j \) [Constraint vectors for \( \bar{a}, \bar{b}, \bar{c} \) are just standard basis vectors]
    2) wire validity: for each wire in circuit, \( w_i (1, w_i) = 0 \)
      \( \Rightarrow w_i^2 = w_i \) [Quadratic constraint]

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

Key idea: Use polynomial magic!

\[
A = \begin{bmatrix}
a_{00} & a_{01} & \cdots & a_{0,mn} \\
a_{10} & a_{11} & \cdots & a_{1,mn} \\
\vdots & \vdots & \ddots & \vdots \\
a_{m0} & a_{m1} & \cdots & a_{mn}
\end{bmatrix}
\]

associate each row (i.e., each constraint with a point \( t : \mathbb{F}^1 \))

\[
A_0 \quad A_1 \quad \cdots \quad A_{mn}
\]

associate each column with a polynomial of degree \( m-1 \) where

\[
A_i(t) = a_{i0} \quad A_i(t) = a_{i1} \quad \cdots \quad A_i(t) = a_{im}
\]
Namely:

\[
A = \begin{bmatrix}
A_0(t_1) & \cdots & A_{tn}(t_1) \\
\vdots & \ddots & \vdots \\
A_0(t_n) & \cdots & A_{tn}(t_n)
\end{bmatrix}
\]

Define \(B_i\) and \(C_i\) accordingly

(using the same set of points \(t_1, \ldots, t_n\))

By construction,

\[
A \tilde{\omega} = \begin{bmatrix}
\omega_0 A_0(t_1) + \cdots + \omega_m A_{tn}(t_1) \\
\vdots \\
\omega_0 A_0(t_n) + \cdots + \omega_m A_{tn}(t_n)
\end{bmatrix}
= \begin{bmatrix}
\bar{A}(t_1) \\
\vdots \\
\bar{A}(t_n)
\end{bmatrix}
\]

where \(\bar{A} = \omega_0 A_0 + \cdots + \omega_m A_{mth}\)

Now \((A \tilde{\omega}) \cdot (B \tilde{\omega}) = C \tilde{\omega}\) if and only if

\[
\bar{A}(z) \cdot \bar{B}(z) = \bar{C}(z)
\]

for all \(z = t_1, t_2, \ldots, t_n\)

Equivalently:

\[
\bar{A}(z) \cdot \bar{B}(z) - \bar{C}(z) = 0
\]

for all \(z = t_1, t_2, \ldots, t_n\)

Let \(Z(z)\) be the polynomial

\[
Z(z) = (z-t_1)(z-t_2) \cdots (z-t_n)
\]

Then,

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

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

**Observe:** If \((A \tilde{\omega}) \cdot (B \tilde{\omega}) = C \tilde{\omega}\) then we can find \(H(z)\) of degree \(m-1\)

\[
\forall z \in F: \bar{A}(z) \bar{B}(z) = \bar{C}(z)
\]

If there is no \(z\) where \((A \tilde{\omega}) \cdot (B \tilde{\omega}) = C \tilde{\omega}\), then for all \(H(z)\) of degree \(m-1\):

\[
\bar{A}(z) \bar{B}(z) - \bar{C}(z) \neq H(z)
\]

since \(\bar{A}(z) \bar{B}(z) - \bar{C}(z) = 0\) 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 \(z \in F\) where \(\bar{A}(z) \bar{B}(z) - \bar{C}(z) = H(z) Z(z)\)

**Idea:** Verifier will check evaluation of \(\bar{A} \bar{B} - \bar{C}\) and \(H\) at a random \(T \in F\)

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

**Linear PCP construction:**

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

\[
\bar{A}(z) \bar{B}(z) - \bar{C}(z) \neq H(z) Z(z)
\]
- **Recall:**
  \[ \overline{A} = \sum_{i=0}^{m} \overline{w}_i A_i \]
  \[ \overline{B} = \sum_{i=0}^{m} \overline{w}_i B_i \Rightarrow \overline{A}(\tau), \overline{B}(\tau), \overline{C}(\tau) \] are linear functions of
  \[ \begin{align*}
    A_i(\tau), & \quad B_i(\tau), & \quad C_i(\tau) \\
    \text{known to verifier} & \quad \text{prover knows } w_i
  \end{align*} \]

  To compute \( H(\tau) \), we can write
  \[ H(\tau) = \sum_{i=0}^{m} h_i \tau^i \] where \( h_i \in \mathbb{F} \) are the coefficients of \( H \)
  \[ \text{known to verifier} \quad \text{prover knows } h_i \]

- **Quadratic arithmetic program:**
  - **LPCP proof:** \( \pi = [w_0, \ldots, w_m, h_0, \ldots, h_{m-1}] \)
  - **LPCP queries:** \( \tau \in \mathbb{F} \)
    - query for \( \overline{A}(\tau) \): \([A_0(\tau), \ldots, A_m(\tau), 0, \ldots, 0]\) \( \overline{B}(\tau) \): \([B_0(\tau), \ldots, B_m(\tau), 0, \ldots, 0]\)
    - query for \( \overline{C}(\tau) \): \([C_0(\tau), \ldots, C_m(\tau), 0, \ldots, 0]\)
  - \( H(\tau) = [0, \ldots, 0, \tau^0, \tau^1, \ldots, \tau^{m-1}] \) \( \text{verifier checks that} \)
    - \( \overline{A}(\tau)\overline{B}(\tau) - \overline{C}(\tau) = H(\tau)\tau(\tau) \)
  - **Note:** nothing here to check for statement
    - **Approach 1:** add 1 check that \( (w_0, \ldots, w_m, x_0, \ldots, x_n) \) can just take random linear combination
    - **Approach 2:** remove \( A_1(\tau), \ldots, A_m(\tau) \) from queries and just add in those components at verification time (since verifier knows statement)