Zero Knowledge Proofs

SNARKs via Interactive Proofs

Instructors: Dan Boneh, Shafi Goldwasser, Dawn Song, Justin Thaler, Yupeng Zhang
Recall: What is a SNARK?

- **SNARK**: a succinct proof that a certain statement is true

  Example statement: “I know an \( m \) such that \( \text{SHA256}(m) = 0 \)”

- **SNARK**: the proof is “short” and “fast” to verify
  [if \( m \) is 1GB then the trivial proof (the message \( m \) is neither]

- **zk-SNARK**: the proof “reveals nothing” about \( m \) (privacy for \( m \))
Interactive Proofs: Motivation and Model
Interactive Proofs

Cloud Provider

Business/Agency/Scientist
Interactive Proofs

Cloud Provider

Data

Business/Agency/Scientist
Interactive Proofs

Cloud Provider

Data

Business/Agency/Scientist

Data Summary
Interactive Proofs

Cloud Provider

Data

Business/Agency/Scientist

Data Summary

Question

Answer
Interactive Proofs

Cloud Provider

Data

Challenge

Response

Business/Agency/Scientist

Data Summary
Interactive Proofs

Cloud Provider

Data

Challenge

Response

Business/Agency/Scientist

Data Summary

Challenge

Response
Interactive Proofs

Cloud Provider

Data

Challenge

Response

Business/Agency/Scientist

Data Summary

Challenge

Response
Interactive Proofs

Cloud Provider

Data

Challenge

Response

Challenge

Response

Business/Agency/Scientist

Accept or Reject
Interactive Proofs

- $P$ solves problem, tells $V$ the answer.
  - Then they have a conversation.
  - $P$’s goal: convince $V$ the answer is correct.

- Requirements:
  1. Completeness: an honest $P$ can convince $V$ to accept.
  2. (Statistical) Soundness: $V$ will catch a lying $P$ with high probability.
Interactive Proofs

- P solves problem, tells V the answer.
  - Then they have a conversation.
  - P’s goal: convince V the answer is correct.

- Requirements:
  1. Completeness: an honest P can convince V to accept.
  2. (Statistical) Soundness: V will catch a lying P with high probability. This must hold even if P is computationally unbounded and trying to trick V into accepting the incorrect answer.
Interactive Proofs

- $P$ solves problem, tells $V$ the answer.
  - Then they have a conversation.
  - $P$’s goal: convince $V$ the answer is correct.
- Requirements:
  1. Completeness: an honest $P$ can convince $V$ to accept.
  2. (Statistical) Soundness: $V$ will catch a lying $P$ with high probability.
    If soundness holds only against polynomial-time provers, then the protocol is called an interactive argument.
Interactive Proofs and Arguments

- Compare soundness to knowledge soundness (last lecture) for circuit-satisfiability:

Public arithmetic circuit: \( C(x, w) \rightarrow \mathbb{F} \)

- public statement in \( \mathbb{F}^n \)
- secret witness in \( \mathbb{F}^m \)
Interactive Proofs and Arguments

- Compare **soundness** to **knowledge soundness** (last lecture) for circuit-satisfiability:
  
  - **Sound**: \( V \) accepts \( \Rightarrow \) There exists \( w \) s.t. \( C(x, w) = 0 \)
  - **Knowledge sound**: \( V \) accepts \( \Rightarrow P \) “knows” \( w \) s.t. \( C(x, w) = 0 \)

  - Knowledge soundness is stronger.
  - But standard soundness is meaningful even in contexts where knowledge soundness isn’t.
    - Because there’s no natural “witness”.
    - E.g., P claims the output of V’s program on x is 42.
Interactive Proofs and Arguments

- Compare **soundness** to **knowledge soundness** (last lecture) for circuit-satisfiability:

  - **Sound**: $V$ accepts $\Rightarrow$ There exists $w$ s.t. $C(x, w) = 0$
  - **Knowledge sound**: $V$ accepts $\Rightarrow P$ “knows” $w$ s.t. $C(x, w) = 0$

- Knowledge soundness is stronger.
  - But standard soundness is meaningful over in contexts where knowledge soundness isn’t.
    - Because there’s no natural “witness”.
    - E.g., $P$ claims the output of $V$’s program on $x$ is 42.
Interactive Proofs and Arguments

- Compare **soundness** to **knowledge soundness** (last lecture) for circuit-satisfiability:
  - **Sound**: $V$ accepts $\Rightarrow$ There exists $w$ s.t. $C(x, w) = 0$
  - **Knowledge sound**: $V$ accepts $\Rightarrow P$ “knows” $w$ s.t. $C(x, w) = 0$

- Knowledge soundness is stronger.
- But standard soundness is meaningful even in contexts where knowledge soundness isn’t.
  - Because there’s no natural “witness”.
  - E.g., $P$ claims the output of $V$’s program on $x$ is 42.
Interactive Proofs and Arguments

- Compare **soundness** to **knowledge soundness** (last lecture) for circuit-satisfiability:
  - **Sound**: $V$ accepts $\Rightarrow$ There exists $w$ s.t. $C(x, w) = 0$
  - **Knowledge sound**: $V$ accepts $\Rightarrow P$ “knows” $w$ s.t. $C(x, w) = 0$
  - Knowledge soundness is stronger.
  - Likewise, knowledge soundness is meaningful in contexts where standard soundness isn’t.
    - e.g., $P$ claims to know the secret key that controls a certain bitcoin wallet.
Interactive proofs and arguments only convince the party that is choosing/sending the random challenges.

This is bad if there are many verifiers (as in most blockchain applications).

- $P$ would have to convince each verifier separately.

For public coin protocols, we have a solution: Fiat-Shamir.

- Makes the protocol non-interactive + publicly verifiable.
SNARKs from interactive proofs: outline
Recall: The trivial SNARK is not a SNARK

(a) Prover sends $w$ to verifier,
(b) Verifier checks if $C(x, w) = 0$ and accepts if so.

Problems with this:

(1) $w$ might be long: we want a “short” proof

(2) computing $C(x, w)$ may be hard: we want a “fast” verifier

(3) $w$ might be secret: prover might not want to reveal $w$ to verifier
SNARKS from Interactive Proofs (IPs)

- Slightly less trivial: $P$ sends $w$ to $V$, and uses an IP to prove that $w$ satisfies the claimed property.
  - Fast $V$, but proof is still too long.

**Actual SNARK:** $P$ commits cryptographically to $w$.

- Uses an IP to prove that $w$ satisfies the claimed property.
- Reveals just enough information about the committed witness $w$ to allow $V$ to run its checks in the IP.
- Render the protocol non-interactive via Fiat-Shamir.
SNARKS from Interactive Proofs (IPs)

- Slightly less trivial: P sends $w$ to V, and uses an IP to prove that $w$ satisfies the claimed property.
  - Fast V, but proof is still too long.
- Actual SNARK: P commits cryptographically to $w$.
  - Uses an IP to prove that $w$ satisfies the claimed property.
  - Reveals just enough information about the committed witness $w$ to allow V to run its checks in the IP.
  - Render non-interactive via Fiat-Shamir.
Review of functional commitments
Recall: three important functional commitments

**Polynomial commitments:** commit to a univariate $f(X)$ in $\mathbb{F}_p^{(\leq d)}[X]$

**Multilinear commitments:** commit to multilinear $f$ in $\mathbb{F}_p^{(\leq 1)}[X_1, \ldots, X_k]$

\[ f(x_1, \ldots, x_k) = x_1x_3 + x_1x_4x_5 + x_7 \]

**Vector commitments (e.g., Merkle trees):**
- Commit to $\vec{u} = (u_1, \ldots, u_d) \in \mathbb{F}_p^d$. Open cells: $f_\vec{u}(i) = u_i$

**Inner product commitments (inner product arguments - IPAs):**
- Commit to $\vec{x} \in \mathbb{F}_p^d$. Open an inner product: $f_{\vec{x}}(\vec{y}) = \langle \vec{x}, \vec{y} \rangle$
Recall: three important functional commitments

**Polynomial commitments:** commit to a univariate \( f(X) \) in \( \mathbb{F}_p^{(\leq d)}[X] \)

**Multilinear commitments:** commit to multilinear \( f \) in \( \mathbb{F}_p^{(\leq 1)}[X_1, \ldots, X_k] \)

\[ f(x_1, \ldots, x_k) = x_1x_3 + x_1x_4x_5 + x_7 \]

**Vector commitments (e.g., Merkle trees):**
- Commit to \( \overrightarrow{u} = (u_1, \ldots, u_d) \in \mathbb{F}_p^d \).
- Open cells: \( f_{\overrightarrow{u}}(i) = u_i \)

**Inner product commitments (inner product arguments - IPA):**
- Commit to \( \overrightarrow{q} \in \mathbb{F}_p^d \).
- Open an inner product: \( f_{\overrightarrow{q}}(\overrightarrow{x}) = \langle \overrightarrow{x}, \overrightarrow{q} \rangle \)
Recall: three important functional commitments

**Polynomial commitments**: commit to a univariate $f(X)$ in $\mathbb{F}_p^{\leq d}[X]$

**Multilinear commitments**: commit to multilinear $f$ in $\mathbb{F}_p^{\leq 1}[X_1, \ldots, X_k]$

  e.g., $f(x_1, \ldots, x_k) = x_1 x_3 + x_1 x_4 x_5 + x_7$

**Vector commitments (e.g., Merkle trees)**:  
- Commit to $\vec{u} = (u_1, \ldots, u_d) \in \mathbb{F}_p^d$.  
  Open cells: $f_{\vec{u}}(i) = u_i$

**Inner product commitments** (inner product arguments — IPA):  
Commit to $\vec{v} \in \mathbb{F}_p^d$.  
Open an inner product: $f_{\vec{v}}(\vec{s}) = (\vec{v}, \vec{s})$
Merkle Trees: The Commitment

\[ m_1 = H(M, Y) \]
\[ h_1 = H(m_1, m_2) \]
\[ k_1 = H(h_1, h_2) \]
\[ m_2 = H(V, E) \]
\[ h_2 = H(m_3, m_4) \]
\[ m_3 = H(C, T) \]
\[ m_4 = H(O, R) \]
Merkle Trees: Opening Leaf T

\[ k_1 = H(h_1, h_2) \]

\[ h_1 = H(m_1, m_2) \]

\[ m_1 = H(M, Y) \]

\[ h_2 = H(m_3, m_4) \]

\[ m_2 = H(V, E) \]

\[ m_3 = H(C, T) \]

\[ m_4 = H(O, R) \]
Merkle Trees

- Commitment to vector is root hash.
- To open an entry of the committed vector (leaf of the tree):
  - Send sibling hashes of all nodes on root-to-leaf path.
  - $V$ checks these are consistent with the root hash.
  - “Opening proof” size is $O(\log n)$ hash values.
Merkle Trees

- Commitment to vector is root hash.
- To open an entry of the committed vector (leaf of the tree):
  - Send sibling hashes of all nodes on root-to-leaf path.
  - $V$ checks these are consistent with the root hash.
  - "Opening proof" size is $O(\log n)$ hash values.
- Binding: once the root hash is sent, the committer is bound to a fixed vector.
  - Opening any leaf to two different values requires finding a hash collision (assumed to be intractable).
A First Polynomial commitment: commit to a univariate $f(X)$ in $\mathbb{F}^{(\leq d)}_7[X]$.
Reveal $f(4)$

$$k_1 = H(h_1, h_2)$$

$$h_1 = H(m_1, m_2)$$

$$h_2 = H(m_3, m_4)$$

$$m_1 = H(f(0), f(1))$$

$$m_2 = H(f(2), f(3))$$

$$m_3 = H(f(4), f(5))$$

$$m_4 = H(f(6), \ast)$$

$f(0)$ $f(1)$ $f(2)$ $f(3)$ $f(4)$ $f(5)$ $f(6)$ $\ast$
Summary: commit to a univariate $f(X)$ in $\mathbb{F}^{(\leq d)}[X]$

- $P$ Merkle-commits to all evaluations of the polynomial $f$.
- When $V$ requests $f(r)$, $P$ reveals the associated leaf along with opening information.

Two problems:

1. The number of leaves is $|\mathcal{F}|$, which means the time to compute the commitment is as issues $|\mathcal{F}|$.
2. Big problem when working over large fields (say, $|\mathcal{F}| \approx 2^{65}$ or $|\mathcal{F}| \approx 2^{129}$). We get time proportional to the degree bound $d$.
3. $V$ does not know if $f$ has degree at most $d$.

We’ll explain how to address both issues later in the course.
Summary: commit to a **univariate** $f(X)$ in $\mathbb{F}^{(\leq d)}[X]$ 

- $P$ Merkle-commits to all evaluations of the polynomial $f$.  
- When $V$ requests $f(r)$, $P$ reveals the associated leaf along with opening information.  
- Two problems:  
  1. The number of leaves is $|\mathbb{F}|$, which means the time to compute the commitment is at least $|\mathbb{F}|$.  
     - Big problem when working over large fields (say, $|\mathbb{F}| \approx 2^{64}$ or $|\mathbb{F}| \approx 2^{128}$).  
     - Want time proportional to the degree bound $d$.  
  2. $V$ does not know if $f$ has degree at most $d$!  
     - We’ll explain how to address both issues later in the course.
Interactive proof design: Technical preliminaries
Recall **FACT:** Let \( p \neq q \) be univariate polynomials of degree at most \( d \). Then \( \Pr_{r \in \mathbb{F}}[p(r) = q(r)] \leq \frac{d}{|\mathbb{F}|} \).

The **Schwartz-Zippel-Demillo-Lipton lemma** is a multivariate generalization:

- Let \( p \neq q \) be \( \ell \)-variate polynomials of total degree at most \( d \). Then \( \Pr_{r \in \mathbb{F}^\ell}[p(r) = q(r)] \leq \frac{d}{|\mathbb{F}|} \).
- “Total degree” refers to the maximum sum of degrees of all variables in any term. E.g., \( x_1^2 x_2 + x_1 x_2 \) has total degree 3.
Low-Degree and Multilinear Extensions

Definition [Extensions]. Given a function \( f : \{0,1\}^\ell \rightarrow \mathbb{F} \), a \( \ell \)-variate polynomial \( g \) over \( \mathbb{F} \) is said to extend \( f \) if \( f(x) = g(x) \) for all \( x \in \{0,1\}^\ell \).

Definition [Multilinear Extensions]. Any function \( f : \{0,1\}^\ell \rightarrow \mathbb{F} \) has a unique multilinear extension (MLE), denoted \( \tilde{f} \).

- Multilinear means the polynomial has degree at most 1 in each variable.
- \( (1 - x_1)(1 - x_2) \) is multilinear, \( x_1^2 x_2 \) is not.
\( f: \{0,1\}^2 \rightarrow \mathbb{F} \)

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td></td>
<td>10</td>
</tr>
</tbody>
</table>
\[ \tilde{f} : \mathbb{F}^2 \rightarrow \mathbb{F} \]
\[ \tilde{f}(x_1, x_2) = (1 - x_1)(1 - x_2) + 2(1 - x_1)x_2 + 8x_1(1 - x_2) + 10x_1x_2 \]

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>10</td>
<td>12</td>
<td>14</td>
<td>16</td>
<td>18</td>
</tr>
<tr>
<td>15</td>
<td>18</td>
<td>21</td>
<td>24</td>
<td>27</td>
<td>30</td>
</tr>
<tr>
<td>22</td>
<td>26</td>
<td>30</td>
<td>34</td>
<td>38</td>
<td>42</td>
</tr>
<tr>
<td>29</td>
<td>34</td>
<td>39</td>
<td>44</td>
<td>49</td>
<td>56</td>
</tr>
</tbody>
</table>

Can check:
- \( \tilde{f}(0, 0) = 1 \)
- \( \tilde{f}(0, 1) = 2 \)
- \( \tilde{f}(1, 0) = 8 \)
- \( \tilde{f}(1, 1) = 10 \)
Another (non-multilinear) extension of $f$: $g(x_1, x_2) = -x_1^2 + x_1x_2 + 8x_1 + x_2 + 1$

Can check:
- $g(0, 0) = 1$
- $g(0, 1) = 2$
- $g(1, 0) = 8$
- $g(1, 1) = 10$
Fact: Given as input all \( 2^\ell \) evaluations of a function \( f : \{0,1\}^\ell \to \mathbb{F} \), for any point \( r \in \mathbb{F}^\ell \) there is an \( O(2^\ell) \)-time algorithm for evaluating \( \tilde{f}(r) \).

Method: Use Lagrange interpolation.

Define \( \delta_x(r) = \prod_{y \neq x} (1 - x_y)(1 - y_y) \). This is called the multilinear Lagrange basis polynomial corresponding to \( x \).

Fact: \( \tilde{f}(r) = \sum_{x \in \{0,1\}^\ell} f(x) \cdot \delta_x(r) \).

For each \( x \in \{0,1\}^\ell \), \( \delta_x(r) \) can be computed with \( O(\ell) \) field operations.

This yields an \( O(2^\ell) \)-time algorithm.
Evaluating multilinear extensions quickly

Fact: Given as input all $2^\ell$ evaluations of a function $f : \{0,1\}^\ell \to \mathbb{F}$, for any point $r \in \mathbb{F}^\ell$ there is an $O(2^\ell)$-time algorithm for evaluating $\tilde{f}(r)$.

Sketch: Use Lagrange interpolation.

Define $\delta_w(r) = \prod_{i=1}^{\ell} (r_i + (1 - r_i)(1 - w_i))$. This is called the multilinear Lagrange basis polynomial corresponding to $w$.

Fact: $f(r) = \sum_{w \in \{0,1\}^\ell} f(w) \cdot \delta_w(r)$.

For each $w \in \{0,1\}^\ell$, $\delta_w(r)$ can be computed with $O(\ell)$ field operations.

Yield a an $O(2^\ell)$-time algorithm.
Evaluating multilinear extensions quickly

Fact: Given as input all \(2^\ell\) evaluations of a function \(f : \{0,1\}^\ell \to \mathbb{F}\), for any point \(r \in \mathbb{F}^\ell\) there is an \(O(2^\ell)\)-time algorithm for evaluating \(\tilde{f}(r)\).

- **Sketch:** Use Lagrange interpolation.
- Define \(\tilde{\delta}_w(r) = \prod_{i=1}^\ell (r_iw_i + (1 - r_i)(1 - w_i))\).
  - This is called the **multilinear Lagrange basis polynomial** corresponding to \(w\).
- Fact: \(\tilde{f}(r) = \sum_{w \in \{0,1\}^\ell} f(w) \cdot \tilde{\delta}_w(r)\).

For each \(w \in \{0,1\}^\ell\), \(\tilde{\delta}_w(r)\) can be computed with \(O(\ell)\) field operations. Yields an \(O(2^\ell)\)-time algorithm. Can reduce to time \(O(2^\ell)\) via dynamic programming.
Evaluating multilinear extensions quickly

Fact: Given as input all $2^\ell$ evaluations of a function $f : \{0,1\}^\ell \rightarrow \mathbb{F}$, for any point $r \in \mathbb{F}^\ell$ there is an $O(2^\ell)$-time algorithm for evaluating $\tilde{f}(r)$.

- **Sketch:** Use Lagrange interpolation.
- Define $\tilde{\delta}_w(r) = \prod_{i=1}^\ell (r_i w_i + (1 - r_i)(1 - w_i))$.
  - This is called the **multilinear Lagrange basis polynomial** corresponding to $w$.
- Fact: $\tilde{f}(r) = \sum_{w \in \{0,1\}^\ell} f(w) \cdot \tilde{\delta}_w(r)$.
- For each $w \in \{0,1\}^\ell$, $\tilde{\delta}_w(r)$ can be computed with $O(\ell)$ field operations.
  - Yields an $O(\ell 2^\ell)$-time algorithm.
  - Can reduce to time $O(2^\ell)$ via dynamic programming.
The sum-check protocol
Sum-Check Protocol [LFKN90]

- **Input:** Given oracle access to a $\ell$-variate polynomial $g$ over field $\mathbb{F}$.
- **Goal:** Compute the quantity:

\[
\sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_\ell \in \{0,1\}} g(b_1, \ldots, b_\ell).
\]
Sum-Check Protocol [LFKN90]

**Start:** $P$ sends claimed answer $C_1$. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_\ell \in \{0,1\}} g(b_1, \ldots, b_\ell).$$

**Round 1:** $P$ sends univariate polynomial $s_\ell(x_\ell)$ claimed to equal:

$$B_\ell(x_\ell) = \sum_{b_\ell \in \{0,1\}} g(x_\ell, b_\ell).$$

$V$ checks that $C_1 = s_\ell(0) + s_\ell(1)$.

If this check passes, it is safe for $V$ to believe that $C_1$ is the correct answer, so long as $V$ believes that $s_\ell(x_\ell) = B_\ell(x_\ell)$.

How to check this? Just check that $s_\ell$ and $B_\ell$ agree at a random point $x_\ell$.

$V$ can compute $s_\ell(x_\ell)$ directly from $P$'s first message, but not $B_\ell(x_\ell)$. 
Sum-Check Protocol [LFKN90]

- **Start:** P sends claimed answer $C_1$. The protocol must check that:

  $$C_1 = \sum \sum \ldots \sum g(b_1, \ldots, b_\ell).$$

- **Round 1:** P sends **univariate** polynomial $s_1(X_1)$ claimed to equal:

  $$H_1(X_1) := \sum \ldots \sum g(X_1, b_2, \ldots, b_\ell)$$

  $V$ checks that $C_1 = s_1(0) + s_1(1)$.

  If this check passes, it is safe for $V$ to believe that $C_1$ is the correct answer, so long as $V$ believes that $s_1 = X_1$.

  How to check this? Just check that $s_1$ and $H_1$ agree at a random point $y_1$.

  $V$ can compute $s_1(y_1)$ directly from P's first message, but not $H_1(y_1)$.
Sum-Check Protocol [LFKN90]

- **Start:** $P$ sends claimed answer $C_1$. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(b_1, \ldots, b_\ell).$$

- **Round 1:** $P$ sends **univariate** polynomial $s_1(X_1)$ claimed to equal:

$$H_1(X_1) := \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \ldots, b_\ell)$$

- $V$ checks that $C_1 = s_1(0) + s_1(1)$.

If this check passes, it is safe for $V$ to believe that $C_1$ is the correct answer, so long as $V$ believes that $s_1(X_1)$.

How to check this? Just check that $s_1$ and $H_1$ agree at a random point $x_1$.

$V$ can compute $s_1(x_1)$ already from $P$‘s first message, but not $H_1(x_1)$. 
Sum-Check Protocol [LFKN90]

- **Start**: \( P \) sends claimed answer \( C_1 \). The protocol must check that:

\[
C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(b_1, \ldots, b_\ell).
\]

- **Round 1**: \( P \) sends **univariate** polynomial \( s_1(X_1) \) claimed to equal:

\[
H_1(X_1): = \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \ldots, b_\ell)
\]

- \( V \) checks that \( C_1 = s_1(0) + s_1(1) \).

- If this check passes, it is safe for \( V \) to believe that \( C_1 \) is the correct answer, so long as \( V \) believes that \( s_1 = H_1 \).

- How to check this? Just check that \( s_1 \) and \( H_1 \) agree at a random point \( r_1 \).

\( V \) can compute \( s_1(X_1) \) directly from \( P \)'s first message, but not \( H_1(X_1) \).
Sum-Check Protocol [LFKN90]

- **Start**: $P$ sends claimed answer $C_1$. The protocol must check that:
  \[
  C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(b_1, \ldots, b_\ell).
  \]

- **Round 1**: $P$ sends univariate polynomial $s_1(X_1)$ claimed to equal:
  \[
  H_1(X_1) := \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \ldots, b_\ell)
  \]

- $V$ checks that $C_1 = s_1(0) + s_1(1)$.
- If this check passes, it is safe for $V$ to believe that $C_1$ is the correct answer, so long as $V$ believes that $s_1 = H_1$.
- How to check this? Just check that $s_1$ and $H_1$ agree at a random point $r_1$.
- $V$ can compute $s_1(r_1)$ directly from $P$’s first message, but not $H_1(r_1)$. 

ZKP MOOC
Sum-Check Protocol [LFKN90]

- **Start**: P sends claimed answer $C_1$. The protocol must check that:
  \[
  C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(b_1, \ldots, b_\ell).
  \]

- **Round 1**: P sends univariate polynomial $s_1(X_1)$ claimed to equal:
  \[
  H_1(X_1):= \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \ldots, b_\ell)
  \]

- V checks that $C_1 = s_1(0) + s_1(1)$.
- V picks $r_1$ at random from $\mathbb{F}$ and sends $r_1$ to P.
- **Round 2**: They recursively check that $s_1(r_1) = H_1(r_1)$. 

\[\text{Is that } s_1(x) = x_{i_1} \cdot x_{i_2} \cdots x_{i_k} \cdot \theta(x_{i_1}, \ldots, x_{i_k}).\]
Sum-Check Protocol [LFKN90]

- **Start:** P sends claimed answer $C_1$. The protocol must check that:

$$C_1 = \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(b_1, \ldots, b_\ell).$$

- **Round 1:** P sends **univariate** polynomial $s_1(X_1)$ claimed to equal:

$$H_1(X_1) := \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \ldots, b_\ell)$$

- V checks that $C_1 = s_1(0) + s_1(1)$.
- V picks $r_1$ at random from $\mathbb{F}$ and sends $r_1$ to P.
- **Round 2:** They recursively check that $s_1(r_1) = H_1(r_1)$.

i.e., that $s_1(r_1) = \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(r_1, b_2, \ldots, b_\ell).$
Sum-Check Protocol [LFKN90]

- **Round \( \ell \) (Final round):** \( P \) sends univariate polynomial \( s_\ell(X_\ell) \) claimed to equal

\[
H_\ell := g(r_1, \ldots, r_{\ell-1}, X_\ell).
\]

- \( V \) checks that \( s_{\ell-1}(r_{\ell-1}) = s_\ell(0) + s_\ell(1) \).
- \( V \) picks \( r_\ell \) at random, and needs to check that \( s_\ell(r_\ell) = g(r_1, \ldots, r_\ell) \).
  - No need for more rounds. \( V \) can perform this check with one oracle query.
Analysis of the sum-check protocol
Completeness holds by design: If $P$ sends the prescribed messages, then all of $V$’s checks will pass.
Soundness

- If $P$ does not send the prescribed messages, then $V$ rejects with probability at least $1 - \frac{\ell \cdot d}{|F|}$, where $d$ is the maximum degree of $g$ in any variable.
- E.g. $|F| \approx 2^{128}$, $d = 3$, $\ell = 60$.
  - Then soundness error is at most $3 \cdot 60 / 2^{128} = 2^{-120}$. 
Soundness

- If $P$ does not send the prescribed messages, then $V$ rejects with probability at least $1 - \frac{\ell \cdot d}{|F|}$, where $d$ is the maximum degree of $g$ in any variable.

- Proof is by induction on the number of variables $\ell$.
  - Base case: $\ell = 1$. In this case, $P$ sends a single message $s_1(X_1)$ claimed to equal $g(X_1)$. $V$ picks $r_1$ at random, checks that $s_1(r_1) = g(r_1)$.
  - If $s_1 \neq g$, then $\Pr_{r_1 \in F}[s_1(r_1) = g(r_1)] \leq \frac{d}{|F|}$. 
Soundness

- Inductive case: $\ell > 1$.
  - Recall: P’s first message $s_1(X_1)$ is claimed to equal
    \[ H_1(X_1) := \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_\ell \in \{0,1\}} g(X_1, b_2, \ldots, b_\ell). \]
  - Then V picks a random $r_1$ and sends $r_1$ to P. They (recursively) invoke sum-check to confirm that $s_1(r_1) = H_1(r_1)$.

If $s_1 = X_1$, then $\Pr_{r_1 \leftarrow \{0,1\}}[s_1(r_1) = X_1(r_1)] < \frac{1}{m}$.

If $s_1(r_1) \neq X_1(r_1)$, P is left to prove a false claim in the recursive call.

The recursive call applies sum-check to $g(s_1, X_2, \ldots, X_\ell)$, which is $\ell-1$ variate.

By induction, P fails to convince V in the recursive call with probability at least $1 - \frac{1}{m}$. 

ZKP MOOC
Soundness

- Inductive case: $\ell > 1$.
  - Recall: $P$’s first message $s_1(X_1)$ is claimed to equal
    $$H_1(X_1) := \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_{\ell} \in \{0,1\}} g(X_1, b_2, \ldots, b_{\ell}).$$
  - Then $V$ picks a random $r_1$ and sends $r_1$ to $P$. They (recursively) invoke sum-check to confirm that $s_1(r_1) = H_1(r_1)$.
  - If $s_1 \neq H_1$, then $\Pr_{r_1 \in F}[s_1(r_1) = H_1(r_1)] \leq \frac{d}{|F|}$.
  - If $s_1(r_1) \neq H_1(r_1)$, $P$ is left to prove a false claim in the recursive call.
    - The recursive call applies sum-check to $g(r_1, X_2, \ldots, X_{\ell})$, which is $\ell$-1 variate.
    - By induction, $P$ convinces $V$ in the recursive call with probability at most $\frac{d(\ell-1)}{|F|}$. 
Soundness analysis: wrap-up

**Summary:** if $s_1 \neq H_1$, the probability $V$ accepts is at most:

$$\Pr_{r_1 \in \mathbb{F}}[s_1(r_1) = H(r_1)] + \Pr_{r_2, \ldots, r_\ell \in \mathbb{F}}[V \text{ accepts} | s_1(r_1) \neq H(r_1)]$$

$$\leq \frac{d}{|\mathbb{F}|} + \frac{d(\ell-1)}{|\mathbb{F}|} \leq \frac{d\ell}{|\mathbb{F}|}.$$
Costs of the sum-check protocol

- Total communication is $O(d \ell)$ field elements.
  - $P$ sends $\ell$ messages, each a univariate polynomial of degree at most $d$. $V$ sends $\ell - 1$ messages, each consisting of one field element.

$V$'s runtime is:

$O(d^2 \ell \cdot [\text{time required to evaluate } g \text{ at one point}])$.

$P$'s runtime is at most:

$O(d \cdot 2^\ell \cdot [\text{time required to evaluate } g \text{ at one point}])$. 
Costs of the sum-check protocol

- Total communication is $O(d\ell)$ field elements.
  - $P$ sends $\ell$ messages, each a univariate polynomial of degree at most $d$. $V$ sends $\ell - 1$ messages, each consisting of one field element.

- $V$’s runtime is:
  $$O(d\ell + [\text{time required to evaluate } g \text{ at one point}]).$$

- $P$’s runtime is at most:
  $$O(d \cdot 2^\ell \cdot [\text{time required to evaluate } g \text{ at one point}]).$$
A first application of the sum-check protocol: An IP for counting triangles with linear-time verifier
Costs of the sum-check protocol

- Total communication is $O(d\ell)$ field elements.
  - $P$ sends $\ell$ messages, each a univariate polynomial of degree at most $d$. $V$ sends $\ell - 1$ messages, each consisting of one field element.

- $V$’s runtime is:
  $$O(d\ell + [\text{time required to evaluate } g \text{ at one point}]).$$

- $P$’s runtime is at most:
  $$O(d \cdot 2^\ell \cdot [\text{time required to evaluate } g \text{ at one point}]).$$
Counting Triangles

- Input: \( A \in \{0,1\}^{n\times n} \), representing the adjacency matrix of a graph.
- Desired Output: \( \sum_{(i,j,k)\in[n]^3} A_{ij}A_{jk}A_{ik} \).
- Fastest known algorithm runs in matrix-multiplication time, currently about \( n^{2.37} \).
Counting Triangles

- Input: $A \in \{0,1\}^{n \times n}$, representing the adjacency matrix of a graph.
- Desired Output: $\sum_{(i,j,k)\in[n]^3} A_{ij}A_{jk}A_{ik}$.
- The Protocol:
  - View $A$ as a function mapping $\{0,1\}^{\log n} \times \{0,1\}^{\log n}$ to $\mathbb{F}$. 
$$A \in F^{4 \times 4}$$
Counting Triangles

- Input: $A \in \{0,1\}^{n \times n}$, representing the adjacency matrix of a graph.
- Desired Output: $\sum_{(i,j,k) \in [n]^3} A_{ij}A_{jk}A_{ik}$.
- The Protocol:
  - View $A$ as a function mapping $\{0,1\}^{\log n} \times \{0,1\}^{\log n} \to \mathbb{F}$.
  - Recall that $\tilde{A}$ denotes the multilinear extension of $A$.
  - Define the polynomial $g(X,Y,Z) = \tilde{A}(X,Y) \tilde{A}(Y,Z) \tilde{A}(X,Z)$
  - Apply the sum-check protocol to $g$ to compute:

$$\sum_{(a,b,c) \in \{0,1\}^{3\log n}} g(a,b,c)$$
Counting Triangles

- Costs:
  - Total communication is $O(\log n)$, $V$ runtime is $O(n^2)$, $P$ runtime is $O(n^3)$.
  - $V$’s runtime dominated by evaluating:
    \[ g(r_1, r_2, r_3) = \tilde{A}(r_1, r_2) \tilde{A}(r_2, r_3) \tilde{A}(r_1, r_3). \]
A SNARK for circuit-satisfiability
Recall: SNARKs for circuit-satisfiability

- Given: An arithmetic circuit $C$ over $\mathbb{F}$ of size $S$ and output $y$.
- $P$ claims to know a $w$ such that $C(x, w) = y$.
- For simplicity, let’s take $x$ to be the empty input.
Recall: SNARKs for circuit-satisfiability

- A **transcript** $T$ for $C$ is an assignment of a value to every gate.
  - $T$ is a **correct** transcript if it assigns the gate values obtained by evaluating $C$ on a valid witness $w$.

Circuit-SAT instance $C$  
Correct transcript for $C$ yielding output 5.
Viewing a transcript as a **function** with domain $\{0,1\}^{\log S}$

- Assign each gate in $C$ a $(\log S)$-bit label and view $T$ as a function mapping gate labels to $\mathbb{F}$.
The polynomial IOP underlying the SNARK
The start of the polynomial IOP

- Assign each gate in $C$ a $(\log S)$-bit label and view $T$ as a function mapping gate labels to $\mathbb{F}$.
- $P$’s first message is a $(\log S)$-variate polynomial $h$ claimed to extend a correct transcript $T$, which means:
  
  $$h(x) = T(x) \ \forall \ x \in \{0, 1\}^{\log S}.$$
The start of the polynomial IOP

- Assign each gate in $C$ a $(\log S)$-bit label and view $T$ as a function mapping gate labels to $\mathbb{F}$.
- $P$’s first message is a $(\log S)$-variate polynomial $h$ claimed to extend a correct transcript $T$, which means:
  \[ h(x) = T(x) \quad \forall \ x \in \{0, 1\}^{\log S}. \]
- $V$ needs to check this, but is only able to learn a few evaluations of $h$. 
Intuition for why $h$ is a useful object for $P$ to send

- Think of $h$ as a **distance-amplified encoding** of the transcript $T$.
- The domain of $T$ is $\{0, 1\}^{\log S}$. The domain of $h$ is $\mathbb{F}^{\log S}$, which is vastly bigger.
Intuition for why $h$ is a useful object for $P$ to send

- Think of $h$ as a **distance-amplified encoding** of the transcript $T$.
- The domain of $T$ is $\{0, 1\}^{\log S}$. The domain of $h$ is $\mathbb{F}^{\log S}$, which is vastly bigger.

All four evaluations of a function $T$ mapping $\{0,1\}^2$ to $F_5$

All 25 evaluations of the multilinear polynomial $h$ that extends $T$, one for each element of $F_5 \times F_5$
Intuition for why $h$ is a useful object for $P$ to send

- Think of $h$ as a **distance-amplified encoding** of the transcript $T$.
- The domain of $T$ is $\{0, 1\}^{\log S}$. The domain of $h$ is $\mathbb{F}^{\log S}$, which is vastly bigger.
- Schwartz-Zippel: If two transcripts $T, T'$ disagree at even a **single** gate value, their extension polynomials $h, h'$ disagree at **almost all** points in $\mathbb{F}^{\log S}$.
  - Specifically, a $1 - \log(S)/|\mathbb{F}|$ fraction.
- Distance-amplifying nature of the encoding will enable $V$ to detect even a single “inconsistency” in the entire transcript.
Reminder: the start of the polynomial IOP

- P’s first message is a \((\log S)\)-variate polynomial \(h\) claimed to extend a correct transcript \(T\), which means:
  \[ h(x) = T(x) \quad \forall \ x \in \{0, 1\}^{\log S}. \]
- \(V\) needs to check this, but is only able to learn a few evaluations of \(h\).
Two-step plan of attack

1. Given any \((\log S)\)-variate polynomial \(h\), identify a related \((3\log S)\)-variate polynomial \(g_h\) such that:

\[
\text{\(h\) extends a correct transcript } T \iff g_h(a, b, c) = 0 \ \forall (a, b, c) \in \{0,1\}^{3 \log S}.
\]

- Moreover, to evaluate \(g_h(r)\) at any input \(r\), suffices to evaluate \(h\) at only 3 inputs.

2. Design an interactive proof to check that \(g_h(a, b, c) = 0 \ \forall (a, b, c) \in \{0,1\}^{3 \log S}\).

- In which \(V\) only needs to evaluate \(g_h(r)\) at one point \(r\).
Step 1 of the plan

- Given \((\log S)\)-variate polynomial \(h\), identify a related \((3\log S)\)-variate polynomial \(g_h\) such that: 
  \[ h \text{ extends a correct transcript } T \iff g_h(a, b, c) = 0 \ \forall (a, b, c) \in \{0,1\}^{3\log S}. \]
- And to evaluate \(g_h(r)\) at any \(r\), suffices to evaluate \(h\) at only 3 inputs.

Proof sketch (simple functions): Define \(g_{a, b, c}(x, y, z)\) via:

\[
\begin{align*}
  g_{a, b, c}(x, y, z) &= \left( h(x) - (h(y) + h(z)) \right) + \left( h(a, b, c) - h(y) - h(z) \right) + \left( h(a) - h(b) - h(c) \right).
\end{align*}
\]

- \(g_{a, b, c}(x, y, z) = h(x) - (h(y) + h(z))\) if \(a\) is the label of a gate that computes the sum of gates \(y\) and \(z\).
- \(g_{a, b, c}(x, y, z) = h(a, b, c) - h(y) - h(z)\) if \(a\) is the label of a gate that computes the product of gates \(y\) and \(z\).
- \(g_{a, b, c}(x, y, z) = 0\) otherwise.
Step 1 of the plan

- Given \((\log S)\)-variate polynomial \(h\), identify a related \((3\log S)\)-variate polynomial \(g_h\) such that:
  
  \(h\) extends a correct transcript \(T \iff g_h(a, b, c) = 0 \forall (a, b, c) \in \{0,1\}^{3\log S}\).

- And to evaluate \(g_h(r)\) at any \(r\), suffices to evaluate \(h\) at only 3 inputs.

- Proof sketch (simplification): Define \(g_h(a, b, c)\) via:
  
  \[
  \overline{\text{add}}(a, b, c) \cdot \left( h(a) - (h(b) + h(c)) \right) + \overline{\text{mult}}(a, b, c) \cdot (h(a) - h(b) \cdot h(c)).
  \]

- \(\overline{\text{add}}(a, b, c) = \overline{\text{A}}(a) - (\overline{\text{B}}(a) - \overline{\text{C}}(a))\) if \(a\) is the label of a gate that computes the sum of gates \(b\) and \(c\).

- \(\overline{\text{mult}}(a, b, c) = \overline{\text{A}}(a) - \overline{\text{B}}(a) \cdot \overline{\text{C}}(a)\) if \(a\) is the label of a gate that computes the product of gates \(b\) and \(c\).

- \(\overline{\text{A}}(a) = 0\) otherwise.
Step 1 of the plan

- Given \((\log S)\)-variate polynomial \(h\), identify a related \((3\log S)\)-variate polynomial \(g_h\) such that:
  
  \[
  h \text{ extends a correct transcript } T \iff g_h(a, b, c) = 0 \forall (a, b, c) \in \{0,1\}^{3\log S}.
  \]

- And to evaluate \(g_h(r)\) at any \(r\), suffices to evaluate \(h\) at only 3 inputs.

- Proof sketch (simplification): Define \(g_h(a, b, c)\) via:

  \[
  \overline{\text{add}}(a, b, c) \cdot \left( h(a) - (h(b) + h(c)) \right) + \overline{\text{mult}}(a, b, c) \cdot (h(a) - h(b) \cdot h(c)).
  \]

  1. \(g_h(a, b, c) = h(a) - (h(b) + h(c))\) if \(a\) is the label of a gate that computes the sum of gates \(b\) and \(c\).

  2. \(g_h(a, b, c) = h(a) - h(b) \cdot h(c)\) if \(a\) is the label of a gate that computes the product of gates \(b\) and \(c\).

  3. \(g_h(a, b, c) = 0\) otherwise.
Step 2: A Hint

- How to check that $g_h(a, b, c) = 0 \forall (a, b, c) \in \{0,1\}^{3 \log S}$?
  - With $V$ only evaluating $g_h$ at a single point?
- Imagine for a moment that $g_h$ were a univariate polynomial $g_h(X)$.
  - And rather than needing to check that $g_h$ vanishes over input set $\{0,1\}^{3 \log S}$, we needed to check that $g_h$ vanishes over some set $H \subseteq \mathbb{F}$.

For $y_i(x) = 0$ for all $x \in H \Rightarrow g_h$ is divisible by $Z_H(x) = \Pi_{x \in H} (x - a)$.
$Z_H$ is called the vanishing polynomial for $H$.

Polynomial IOPs:
- $P$ sends a polynomial $q$ such that $g_h(H) = q(H) \cdot Z_H(H)$.
- $V$ checks this by picking a random $y \in \mathbb{F}$ and checking that $g_h(y) = q(y) \cdot Z_h(y)$. 

ZKP MOOC
Step 2: A Hint

- How to check that $g_h(a, b, c) = 0 \ \forall (a, b, c) \in \{0, 1\}^{3 \log S}$?
  - With $V$ only evaluating $g_h$ at a single point?

- Imagine for a moment that $g_h$ were a univariate polynomial $g_h(X)$.
  - And rather than needing to check that $g_h$ vanishes over input set $\{0, 1\}^{3 \log S}$, we needed to check that $g_h$ vanishes over some set $H \subseteq \mathbb{F}$.

- Fact: $g_h(x) = 0$ for all $x \in H \iff g_h$ is divisible by $Z_H(x) := \prod_{a \in H} (x - a)$.
  - $Z_H$ is called the vanishing polynomial for $H$.

- Polynomial IOP:
  - $P$ sends a polynomial $q$ such that $g_h(X) = q(X) \cdot Z_H(X)$.
  - $V$ checks this by picking a random $r \in \mathbb{F}$ and checking that $g_h(r) = q(r) \cdot Z_H(r)$.
Previous slide doesn’t actually work.
- $g_h$ is not univariate, it has $3 \log S$ variables.
- Also, having $P$ find and send the quotient polynomial is expensive.
  - In the final SNARK, this would mean applying polynomial commitment to additional polynomials.
  - This is what Marlin, PlonK, and Groth16 do.

Solution: use the sum-check protocol [LFKN90].
- Handles multivariate polynomials.
- Doesn’t require $P$ to send additional large polynomials.
Recall sum-check
Sum-check protocol: a reminder

- Goal: compute the quantity:

\[ \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \ldots \sum_{b_\ell \in \{0,1\}} g(b_1, \ldots, b_\ell). \]

- Proof length is roughly the total degree of \( g \).
- Number of rounds is \( \ell \).
- \( V \) time is roughly the time to evaluate \( g \) at a single randomly chosen input.
- To run the protocol, \( V \) doesn’t even need to “know” what polynomial \( g \) is being summed, so long as it knows \( g(r) \) for a randomly chosen input \( r \in \mathbb{F}^\ell \).
The polynomial IOP for circuit-satisfiability

- How to check that $g_h(a, b, c) = 0 \forall (a, b, c) \in \{0,1\}^{3 \log s}$?
  - With $V$ only evaluating $g_h$ at a **single** point?

- General idea (working over the integers instead of $\mathbb{F}$):
  - $V$ checks this by running sum-check protocol with $P$ to compute:
    $$\sum_{a,b,c\in\{0,1\}^{\log s}} g_h(a, b, c)^2.$$
  - If all terms in the sum are 0, the sum is 0.
  - If working over the integers, any non-zero term in the sum will cause the sum to be strictly positive.
The polynomial IOP for circuit-satisfiability

- How to check that $g_h(a, b, c) = 0 \forall (a, b, c) \in \{0,1\}^{3 \log S}$?
  - With $V$ only evaluating $g_h$ at a single point?
- General idea (working over the integers instead of $\mathbb{F}$):
  - $V$ checks this by running sum-check protocol with $P$ to compute:
    $$\sum_{a, b, c \in \{0,1\}^{\log S}} g_h(a, b, c)^2.$$  
  - At end of sum-check protocol, $V$ needs to evaluate $g_h(r_1, r_2, r_3)$.
    - Suffices to evaluate $h(r_1), h(r_2), h(r_3)$.
    - Outside of these evaluations, $V$ runs in time $O(\log S)$.
    - $P$ performs $O(S)$ field operations given a witness $w$. 
END OF LECTURE