Zero Knowledge Proofs

Polynomial Commitments
based on error-correcting codes

Instructors: Dan Boneh, Shafi Goldwasser, Dawn Song, Justin Thaler, Yupeng Zhang
Recall: common paradigm for efficient SNARK

A polynomial commitment scheme

A polynomial interactive oracle proof (IOP)

SNARK for general circuits
Last time: KZG polynomial commitment

Univariate polynomials of degree $\leq d$

$gp = (g, g^\tau, g^{\tau^2}, ..., g^{\tau^d})$

$\begin{align*}
  f(x) - f(u) &= (x - u)q(x) \\
  \text{com}_f &= g^{f(\tau)} \\
  u \\
  \nu, \text{ proof } \pi &= g^{q(\tau)} \\
  e(\text{com}_f/g^\nu, g) &= e(g^{\tau-u}, \pi)
\end{align*}$
Last time: other PC based on discrete-log

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Prover</th>
<th>Proof size</th>
<th>Verifier</th>
<th>Trusted Setup</th>
<th>Crypto primitive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bullet -proofs</td>
<td>$O_\lambda(d)$</td>
<td>$O_\lambda(\log d)$</td>
<td>$O_\lambda(d)$</td>
<td>✗</td>
<td>discrete-log</td>
</tr>
<tr>
<td>Hyrax</td>
<td>$O_\lambda(d)$</td>
<td>$O_\lambda(\sqrt{d})$</td>
<td>$O_\lambda(\sqrt{d})$</td>
<td>✗</td>
<td>discrete-log</td>
</tr>
<tr>
<td>Dory</td>
<td>$O_\lambda(d)$</td>
<td>$O_\lambda(\log d)$</td>
<td>$O_\lambda(\log d)$</td>
<td>✗</td>
<td>pairing</td>
</tr>
<tr>
<td>Dark</td>
<td>$O_\lambda(d)$</td>
<td>$O_\lambda(\log d)$</td>
<td>$O_\lambda(\log d)$</td>
<td>✗</td>
<td>unknown order group</td>
</tr>
</tbody>
</table>
Poly-commit based on error-correcting codes

Motivations:
✓ Plausibly post-quantum secure
✓ No group exponentiations (prover only uses hashes, additions and multiplications)
✓ Small global parameters

Drawbacks:
✗ Large proof size
✗ Not homomorphic and hard to aggregate
Plan of this lecture

- Background on error-correcting codes
- Polynomial commitment based on error-correcting codes
- Linear-time encodable code based on expanders
Background
Error-correcting code

\([n, k, \Delta]\) code:

- \(\text{Enc}(m)\): Encode a message of size \(k\) to a codeword of size \(n\)
- Minimum distance (Hamming) between any two codewords is \(\Delta\)
Example: repetition code

Binary with $k = 2$ and $n = 6$

- $\text{Enc}(00) = 000000$, $\text{Enc}(01) = 000111$
- $\text{Enc}(10) = 111000$, $\text{Enc}(11) = 111111$
- Minimum distance $\Delta = 3$

Can correct 1 error during the transmission
e.g. $010111 \rightarrow 01$  $\text{Dec}(c)$: decode algorithm (not used in poly-commit)
Rate and relative distance

Rate: \( \frac{k}{n} \)

Relative distance: \( \frac{\Delta}{n} \)

E.g. repetition code with rate \( \frac{1}{a} \), \( \Delta = a \), relative distance: \( \frac{1}{k} \)

Trade-off between the rate and the distance of a code
Linear code

Any linear combination of codewords is also a codeword

$\Rightarrow$ Encoding can always be represented as vector-matrix multiplication between $m$ and the generator matrix

$\Rightarrow$ minimum distance is the same as the codeword with the least number of non-zeros (weight).
Example: Reed-Solomon Code

Encode: \( \mathbb{F}_p^k \rightarrow \mathbb{F}_p^n \)
- View the message as a unique degree \( k-1 \) univariate polynomial
- The codeword is the evaluations at \( n \) points
  E.g., \( (\omega, \omega^2, \ldots, \omega^n) \) for \( n \)-th root-of-unity \( \omega^n = 1 \) mod \( p \)
- Distance \( \Delta = n - k + 1 \)
  a degree \( k-1 \) polynomial has at most \( k-1 \) roots
  E.g, \( n = 2k \), rate is \( 1/2 \), and relative distance is \( 1/2 \)
- Encoding time: \( O(n \log n) \) using the fast Fourier transform (FFT)
Polynomial commitment based on linear codes
Recall: polynomial commitment

keygen(\(\lambda, F\)) \rightarrow gp

comm\(it(f) \rightarrow com_f\)

eval(gp,f,u) \rightarrow v, \pi
Polynomial coefficients in a matrix

\[ f(u) = \sum_{i=1}^{\sqrt{d}} \sum_{j=1}^{\sqrt{d}} f_{i,j} u^{i-1 + (j-1)\sqrt{d}} \]

\[
\begin{pmatrix}
  f_{1,1} & f_{1,2} & \cdots & f_{1,d} \\
  f_{2,1} & f_{2,2} & \cdots & f_{2,d} \\
  \vdots & \vdots & \ddots & \vdots \\
  f_{d,1} & f_{d,2} & \cdots & f_{d,d}
\end{pmatrix}
\]
Polynomial evaluation

\[
\begin{align*}
    f(u) &= \left[1, u, u^2, \ldots, u^{\sqrt{d}-1}\right] \times \left(\begin{array}{ccc}
    f_{1,1} & f_{1,2} & \cdots & f_{1,\sqrt{d}} \\
    f_{2,1} & f_{2,2} & \cdots & f_{2,\sqrt{d}} \\
    \vdots & \vdots & \ddots & \vdots \\
    f_{\sqrt{d},1} & f_{\sqrt{d},2} & \cdots & f_{\sqrt{d},\sqrt{d}}
    \end{array}\right) \times \left[\begin{array}{c}
    1 \\
    u^{\sqrt{d}} \\
    u^{2\sqrt{d}} \\
    \vdots \\
    u^{d-\sqrt{d}}
    \end{array}\right] \\
    f(u) &= \sum_{i=1}^{\sqrt{d}} \sum_{j=1}^{\sqrt{d}} f_{i,j} u^{i-1+(j-1)\sqrt{d}}
\end{align*}
\]
Reducing to Vec-Mat product

\[
\begin{bmatrix}
1, u, u^2, \ldots, u^{\sqrt{d}-1}
\end{bmatrix} \times \begin{pmatrix}
\begin{array}{cccc}
f_{1,1} & f_{1,2} & \cdots & f_{1,\sqrt{d}} \\
f_{2,1} & f_{2,2} & \cdots & f_{2,\sqrt{d}} \\
& & & \\
f_{\sqrt{d},1} & f_{\sqrt{d},2} & \cdots & f_{\sqrt{d},\sqrt{d}}
\end{array}
\end{pmatrix} = \begin{bmatrix}
\vdots \\
\sqrt{d}
\end{bmatrix}
\]

Argument for Vec-Mat product
→ Polynomial commitment with \( \sqrt{d} \) proof size
Encoding the polynomial

Encode each row with a linear code
Recall: Merkle tree commitment

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

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

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

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

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

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

\[ m_4 = H(O, R) \]
Recall: Merkle tree opening

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

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

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

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

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

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

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

\[ h_1 = H(m_1, m_2) \]
Committing the polynomial

Commit to each column of the encoded matrix using Merkle tree
Step 1: Proximity test

Test if the committed matrix indeed consists of $\sqrt{d}$ codewords

$$[r_1, r_2, r_3, \ldots, r_{\sqrt{d}}] \times$$

1. The vector is a codeword
2. Columns are as committed in Merkle tree
3. Inner product between $r$ and each column is consistent

Prover

Verifier

Send several random columns
Suppose the prover cheats

- If the vector is correctly computed $\rightarrow$ it is not a codeword $\rightarrow$ **check 1**
- If the vector is false $\rightarrow$ many different locations from the correct answer
  - By check 2, columns are as committed
  - Probability of passing check 3 is small
Ligero \cite{ahiv2017} and \cite{bcghj2017}

- Ligero \cite{ahiv2017} : Interleaved test. Reed-Solomon code

- \cite{bcghj2017} : Ideal linear commitment model. Linear-time encodable code $\rightarrow$ first SNARK with linear prover time
In the formal proof [AHIV’2017]

If the committed matrix $C$ is $e$-far from any codeword for $e < \frac{\Delta}{4}$

$\Rightarrow \Pr[w = r^T C \text{ is } e\text{-close to any codeword}] \leq \frac{e+1}{\mathcal{F}}$

If $w = r^T C$ is $e$-far from any codeword

$\Rightarrow \Pr[\text{check 3 is true for } t \text{ random columns}] \leq \left(1 - \frac{e}{n}\right)^t$
One optimization

\[
\begin{bmatrix}
    r_1, r_2, r_3, \ldots, r_{\sqrt{d}}
\end{bmatrix}
\times
\]

Prover

Send several random columns

Verifier

Encode

Message \( m \)

\[
= \quad \text{Send several random columns}
\]

H
Step 2: Consistency check

\[
\begin{bmatrix}
1, u, u^2, \ldots, u^{\sqrt{d}-1}
\end{bmatrix} \times \text{Send several random columns}
\]

1. The vector is a codeword
2. Columns are as committed in Merkle tree
3. Inner product between \( \vec{u} \) and each column is consistent

Prover

Verifyer

Encode message \( m \)
Soundness (intuition)

- By the proximity test, the committed matrix $C$ is close to a codeword.
- There exists an extractor that extracts $F$ by Merkle tree commitment and decoding $C$, s.t. $\vec{u} \times F = m$ with probability $1 - \epsilon$. 
Poly-commit based on linear code

- **Keygen:** sample a hash function
- **Commit:** encode the coefficient matrix of \( f \) row-wise with a linear code, compute the Merkle tree commitment
- **Eval and Verify:**
  - **Proximity test:** random linear combination of all rows, check its consistency with \( t \) random columns
  - **Consistency test:** \( \vec{u} \times F = m \), encode \( m \) and check its consistency with \( t \) random columns
  - \( f(u) = \langle m, \vec{u}' \rangle \)
Properties of the polynomial commitment

- **Keygen:** $O(1)$, transparent setup!
- **Commit:**
  - Encoding: $O(d \log d)$ field multiplications using RS code, $O(d)$ using linear-time encodable code
  - Merkle tree: $O(d)$ hashes, $O(1)$ commitment size
- **Eval:** $O(d)$ field multiplications
  (non-interactive via Fiat Shamir)
- **Proof size:** $O(\sqrt{d})$
- **Verifier time:** $O(\sqrt{d})$
Performance the poly-commit \([GLSTW'21]\)

- Degree \(d = 2^{25}\), linear-time encodable code
  - Commit: 36s
  - Eval: 3.2s
  - Proof size: 49MB
  - Verifier time: 0.7s
[Bootle-Chiesa-Groth’20] and Brakedown [GLSTW’21]

- [Bootle-Chiesa-Groth’20]: Tensor query IOP \( \langle f, (\tilde{u} \otimes \tilde{u}') \rangle \)
  - Generalizes to multiple dimensions with proof size \( O(n^\epsilon) \) for constant \( \epsilon < 1 \)

- Brakedown [GLSTW’21]: polynomial commitment based on tensor query
  - Knowledge soundness without efficient decoding algorithm
[Bootle-Chiesa-Liu’21] and Orion [Xie-Zhang-Song’22]

- [Bootle-Chiesa-Liu’21]
  - Proof size $\text{polylog}(n)$ with a proof composition of tensor IOP and PCP of proximity [Mie’09]

- Orion [Xie-Zhang-Song’22]
  - Proof size $O(\log^2 n)$ with a proof composition of the code-switching technique [Ron-Zewi-Rothblum’20]
    - (5.7MB for $d = 2^{25}$)
Linear-time encodable code
SNARKs with linear prover time

Ideal linear model

\[ O(\sqrt{d}) \]

proof size

Tensor IOP

\[ O(d^\epsilon) \]

Tensor IOP+PCPP

polylog(d)

Polynomial commitment

\[ O(d^\epsilon) \]

Code-switching proof composition

\[ O(\log^2 d) \]
Linear-time encodable code [Spielman’96][Druk-Ishai’14]
# Lossless Expander

- # left nodes = $|L|$, # right nodes = $\alpha |L|$ for a constant $\alpha$
- Degree of a left node = $g$
- For every subset $S$ of nodes on the left, # of neighbors $|\Gamma(S)| = g|S|$, for $|S| \leq \frac{\alpha |L|}{g}$
Lossless Expander

- # left nodes = $|L|$, # right nodes = $\alpha |L|$ for a constant $\alpha$
- Degree of a left node = $g$
- For every subset $S$ of nodes on the left, # of neighbors
  \[ |\Gamma(S)| \geq (1 - \beta) g |S|, \text{ for } |S| \leq \frac{\delta |L|}{g} \]
  $(\beta \to 0, \delta \to \alpha)$
Overview of the recursive encoding

- Message $k$
- Copy
- Message $k/2$
- Lossless expander $\alpha = \frac{1}{2}$
- Encode for $k/2$
- Codeword $c_1$
- Codeword $c_2$
- $2k$
- $k$
Encoding algorithm

- Message $m$ of size $k$, codeword size $4k$, rate is $1/4$
- Suppose there is an encoding algorithm from $k/2$ to $2k$ with good relative distance $\Delta$
- Suppose there are lossless expander graphs of size $k$ and $2k$, and $\alpha = 1/2$

1. Pass $m$ through lossless expander to get $m_1$ of size $k/2$
2. Encode $m_1$ to get $c_1$ of size $2k$
3. Pass $c_1$ through lossless expander to get $c_2$ of size $k$
4. Codeword $c = m || c_1 || c_2$
Recursive encoding

- Repeat for $k/2$, $k/4$ ... until a constant size

- Use any code with good distance for a constant-size message. E.g., Reed-Solomon code
Distance of the code

constant relative distance $\Delta' = \min\{\Delta, \frac{\delta}{4g}\}$
- # left nodes = $k$, # right nodes = $\alpha k$ for a constant $\alpha$
- Degree of a left node = $g$
- For every subset $S$ of nodes on the left, # of neighbors

$$|\Gamma(S)| \geq (1 - \beta)g|S|,$$  for $|S| \leq \frac{\delta |L|}{g}$

$(\beta \to 0, \delta \to \alpha)$
Proof of constant relative distance [Druk-Ishai’14]

constant relative distance $\Delta' = \min \{\Delta, \frac{\delta}{4g}\}$, codeword $c = m || c_1 || c_2$

1. If weight of $m$ is larger than $4k\Delta'$ → done
2. If (weight of $m$) $\leq 4k\Delta'$, the condition of lossless expander holds
   - Let $S$ be the set of nonzero nodes, $|\Gamma(S)| \geq (1 - \beta)g|S|$
   - At least 1 node in $|\Gamma(S)|$ have a unique neighbor in $S$
   - $m_1$ is nonzero $\rightarrow$ (weight of $c_1$) $\geq 2k\Delta$
3. If it is larger than $4k\Delta'$ → done
4. Else, weight of $c_2 \geq 2k\Delta'$ because of lossless expander
Sampling of the lossless expander

- [Capalbo-Reingold-Vadhan-Wigderson’2002]: Explicit construction of lossless expander (large hidden constant)

- Random sampling: $1/poly(n)$ failure probability
Improvements of the code

- Brakedown [Golovnev-Lee-Setty-Thaler-Wahby’21]: random summations with better concrete distance analysis

- Orion [Xie-Zhang-Song’22]: expander testing with a negligible failure probability via maximum density of the graph
Putting everything together

Polynomial commitment (and SNARK) based on linear code
✓ Transparent setup: $O(1)$
✓ Commit and Prover time: $O(d)$ field additions and multiplications
✓ Plausibly post-quantum secure
✓ Field agnostic

✗ Proof size: $O(\sqrt{d})$, MBs
End of Lecture

Next: FRI and Stark