Polynomial Commitments Wrap-UP

Lecture 21

And Linear PCP-Based SNARKs

Recall

Polynomial Commitment

- Prover wants to (succinctly) commit to a polynomial and later let the verifier (interactively) evaluate it on points of its choice
    - Generally, a multi-variate polynomial with a known number of variables and known degree
        - e.g., a multi-linear polynomial in GKR. In some other applications, univariate polynomial of a known degree
- Trivial solution: send the coefficients of the polynomial
    - But not succinct and evaluating the polynomial is expensive
    - Want verifier's computation/communication to be sub-linear in the size of the polynomial
- Non-trivial solutions: Using Merkle hashes and low-degree tests; from hardness of discrete logarithm; from bilinear pairings; using "IOPs"; ...

Polynomial Commitment

- 3 Approaches:
  - Hash-Based
    - Ligero, FRI and their variants
  - Discrete Log-Based
    - Bulletproofs
  - Pairings-Based
    - KZG, Dory
- Can be combined with public-coin Outsourced computation protocols, MIP or IOPs (covered later) that use polynomial commitments, to get SNARKs
- Other approaches to SNARKS:
  - From PCPs and Merkle hashes
  - From Linear PCPs and Linear function commitment (Today)

Recall

SNARKs

from Linear PCPs

- PCP with a very long (super-polynomial) but more structured proof
  - Proof π is the evaluation of a multi-variate linear polynomial (total degree is 1) with 0 as the constant term
    - $\pi[ax+by] = a\pi[x] + b\pi[y]$ for $x,y \in \mathbb{F}^k$ and $a,b \in \mathbb{F}$
- Idea: can commit to such a multi-variate linear polynomial efficiently [Later]
- Linear PCPs + non-interactive multi-variate linear polynomial commitment schemes yield practical SNARKs
  - e.g., "Groth16"

SNARKs

from Linear PCPs

- A Scheme for R1CS
  - m public vectors $a_i, b_i, c_i \in \mathbb{F}^n$ and a private vector $z \in \mathbb{F}^n$ s.t. for all $i \in [m]$, $\langle a_i,z\rangle \langle b_i,z\rangle = \langle c_i,z\rangle$
  - Will require $z_1 = 1$. May also require some more z_j to be fixed.
  - Generalizes constraints like $z_j z_{j''} = z_{j'''}$, $z_j + z_{j''} = z_{j'''}$
- Idea: Encode $\{a_i,b_i,c_i\}_{i\in[m]}$ as polynomials evaluated at m places, so that a single combined constraint can be checked
- For $j\in[n]$, let degree m-1 polynomials A_j, B_j, C_j be such that for all $i\in[m]$, $A_j(\sigma_i) = a_{ij}$, $B_j(\sigma_i) = b_{ij}$, $C_j(\sigma_i) = c_{ij}$
- Let $P_z(X) = [\ \Sigma_{j\in[n]} z_j A_j(X)\ ] \cdot [\ \Sigma_{j\in[n]} z_j B_j(X)\ ] - [\ \Sigma_{j\in[n]} z_j C_j(X)\ ]$
- $P_z(\sigma_i) = \langle a_i,z\rangle \langle b_i,z\rangle - \langle c_i,z\rangle$
- P_z is a degree 2(m-1) polynomial that evaluates to 0 in $\{\ \sigma_i\ \}_{i\in[m]}$ iff all the constraints (other than fixed values) satisfied

SNARKs

from Linear PCPs

- To prove P_z $\exists z$ such that P_z evaluates to 0 in $H = \{ \sigma_i \}_{i \in [m]}$
  - (Ignoring for now that some coordinates of z have to be fixed)
  - Fact: P(X) vanishes on H iff $Z_H(X) = \Pi_{\sigma \in H} (X-\sigma)$ divides P(X)
  - To prove $P_z(X) = Z_H(X) \cdot Q(X)$, where and Q(X) is some polynomial of degree 2(m-1)-m = m-2
  - Enough to check $P_z(\beta) = Z_H(\beta).Q(\beta)$ for random $\beta \leftarrow \mathbb{F}$ for large $\mathbb{F}$
- Linear PCP: Proof includes linear functions L_z and L_Q s.t. $L_z(x) = \langle x,z \rangle$ and $L_Q(1,x,..,x^{m-2)}) = Q(x)$. Verifier checks $Z_H(\beta) \cdot L_Q(1,\beta,..,\beta^{m-2}) = L_z(a)L_z(b) - L_z(c)$ where $a_j = A_j(\beta)$, $b_j = B_j(\beta)$, $c_j = C_j(\beta)$
- SNARK: Need to commit to L_z and L_Q succinctly

Linear Function Commitment

- Goal: Prover commits to a vector $D \in \mathbb{F}^n$, and on being queried with a vector $x \in \mathbb{F}^n$, opens to $\langle D,x \rangle$.
- Simple interactive solution
  - Commitment: Verifier picks $\beta \leftarrow \mathbb{F}^n$, uses an additively homomorphic encryption scheme to encrypt each β_i, and sends them. Prover homomorphically computes encryption of $\langle D,\beta \rangle$ and sends it back. Verifier decrypts to get $s = \langle D,\beta \rangle$
  - Evaluation: Verifier picks $\alpha \leftarrow \mathbb{F}$ and send x, $y=\alpha x+\beta$. Prover sends $a = \langle D,x \rangle$ and $b = \langle D,y \rangle$. Verifier checks $b = \alpha a + s$.
    - Batch evaluation: For $x_1, x_2, \ldots$, let $y = (\alpha_1 x_1+\alpha_2 x_2+\ldots) + \beta$
- Soundness: For any x, on challenges y,y' for α,α' (resp.), if two answers $a \neq a'$ then $b-b'=\alpha a-\alpha' a'$ and $y-y'=(\alpha-\alpha')x$ yield α,α'. But if β is hidden (as it should be), only $\alpha-\alpha'$ is revealed.
- Not public coin: Verifier keeps secrets: β, α and decryption key

Enough to get s as g^s

SNARKs

from Linear PCPs

- Linear PCP: Proof includes linear functions L_z and L_Q s.t. $L_z(x) = \langle x,z \rangle$ and $L_Q(1,x,..,x^{m-2)}) = Q(x)$. Verifier checks $Z_H(\beta) \cdot L_Q(1,\beta,..,\beta^{m-2}) = L_z(a)L_z(b) - L_z(c)$ where $a_j = A_j(\beta)$, $b_j = B_j(\beta)$, $c_j = C_j(\beta)$
- Interactive commitment involves verifier sends a homomorphic encryption of r and later random α
- SNARK: Need to commit to L_z and L_Q non-interactively
  - Cannot use non-public coin protocol with Fiat-Shamir
- Idea (a la KZG): Compute $Z_H(\beta)$, $L_Q(1,\beta,..,\beta^{m-2})$ and $L_z(x)$ for x=a,b,c in the exponent, using trusted setup $(g^{Z_H(\beta)}, g^{\gamma Z_H(\beta)})$, $(g, g^{\gamma}, g^{\beta}, g^{\gamma\beta}, g^{\beta^2}, g^{\gamma\beta^2}, \ldots)$, $(g^{x_1}, g^{\gamma x_1}, g^{x_2}, g^{\gamma x_1}, \ldots)$ for x=a,b,c. Verifier will use pairings (with $\mathbb{G}_1 = \mathbb{G}_2$)
  - Need to also ensure same z used for $L_z(x)$, x=a,b,c. Ask for $L_z(x^*)$ too, where $x^* = \delta_1 a + \delta_2 b + \delta_3 c$, $\delta_i \leftarrow \mathbb{F}$ and cross-check

SNARKs

from Linear PCPs

- Linear PCP: Proof includes linear functions L_z and L_Q s.t. $L_z(x) = \langle x,z \rangle$ and $L_Q(1,x,..,x^{m-2})) = Q(x)$. Verifier checks $Z_H(\beta) \cdot L_Q(1,\beta,..,\beta^{m-2}) = L_z(a)L_z(b) - L_z(c)$ where $a_j = A_j(\beta)$, $b_j = B_j(\beta)$, $c_j = C_j(\beta)$
- SNARK:

- Setup: $g^{Z_H(\beta)}$, $(g,g^{\gamma},g^{\beta},g^{\gamma\beta},g^{\beta^2},g^{\gamma\beta^2},\ldots)$, $\{(g^{x_1},g^{\gamma x_1},g^{x_2},g^{\gamma x_2},\ldots)\}_{x=a,b,c,x^*}$, where $x^* = \delta_1 a + \delta_2 b + \delta_3 c$, with $\beta, \gamma, \delta_i \leftarrow \mathbb{F}$
- Prover sends $(g_Q,h_Q) = (g^{Q(\beta)},g^{\gamma Q(\beta)})$, $(g_x,h_x) = (g^{\langle z,x \rangle},g^{\gamma\langle z,x \rangle})$ for $x=a,b,c,x^*$
- Verifier checks:

$$e(g^{Z_H(\beta)},g_Q) \cdot e(g,g_c) = e(g_a,g_b)$$

$$e(g,g_{x^*}) = e(g^{\delta_1},g_a)\ e(g^{\delta_2},g_b)\ e(g^{\delta_3},g_c)$$

$$e(g^{\gamma},g_T) = e(g,h_T) \text{ for } T=Q,a,b,c,x^*$$

SNARKs

from Linear PCPs

- Setup: $g^{Z_H(\beta)}$, $(g, g^\gamma, g^\beta, g^{\gamma\beta}, g^{\beta^2}, g^{\gamma\beta^2}, \ldots)$, $\{(g^{x_1}, g^{\gamma x_1}, g^{x_2}, g^{\gamma x_2}, \ldots)\}_{x=a,b,c,x^*}$, where $x^* = \delta_1 a + \delta_2 b + \delta_3 c$, with $\beta, \gamma, \delta_i \leftarrow \mathbb{F}$
- Prover sends $(g_Q, h_Q) = (g^{Q(\beta)}, g^{\gamma Q(\beta)})$, $(g_x, h_x) = (g^{\langle z,x\rangle}, g^{\gamma\langle z,x\rangle})$ for $x=a,b,c,x^*$
- Verifier checks:

  $e(g^{Z_H(\beta)}, g_Q)\cdot e(g, g_c) = e(g_a, g_b)$

  $e(g, g_{x^*}) = e(g^{\delta_1}, g_a)\ e(g^{\delta_2}, g_b)\ e(g^{\delta_3}, g_c)$

  $e(g^\gamma, g_T) = e(g, h_T)$ for $T=Q,a,b,c,x^*$
- Knowledge soundness based on KEA and "Strong" Discrete Log assumption in the source group
  - "Strong DL assumption" : Given $g, g^\beta, g^{\beta^2}, \ldots$ can't find β
- Groth16 is a more efficient version, but soundness relies on the Generic Group model (or the Algebraic Group model) heuristics

SNARKs

from Linear PCPs

- Saw PoK of z such that P_z evaluates to 0 in H = { σ_i }$_{i\in[m]}$
- Also need to check z equals known values at various coordinates
  - In particular need at least one such coordinate (z_1=1) to model constraints from general circuit satisfiability
- Let z = z′ || z″, where z′ is known. In the Linear PCP, to commit to L_z, prover should commit only to $L_{z''}$ and the verifier computes $L_z(x)$ = <z′,x′> + $L_{z''}(x'')$ where x = x′||x″ (for x=a,b,c)
  - In the SNARK, instead of sending $(g_x,h_x) = (g^{\langle z,x\rangle},g^{\gamma\langle z,x\rangle})$, prover sends $(g'_x,h'_x) = (g^{\langle z'',x''\rangle},g^{\gamma\langle z'',x''\rangle})$ (for x=a,b,c,x*). Verifier computes $g_x = g'_x \cdot g^{\langle z',x'\rangle}$ using $g^{x'}$ which is included in the setup.