

lec11: Polynomial Commitments

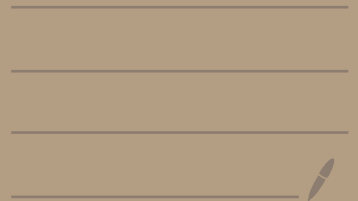
[1] Polynomial Commitment Schemes (PCSs)

[2] Pairings

- 3-party key exchange
- deterministic signatures

[3] The KZG PCS

- q -SDH assumption



① Recall: PCPs and Succinct Proofs

- $\phi(x, w)$, protocol (P, V)
- Succinctness: communication + V are: $O(|x|) + o(|p| + |w|)$
- Computational Soundness: $\forall x \notin \mathcal{L}_\phi, \forall \text{PPT } P^*, \Pr[\langle P^*, V \rangle(x) = 1] \leq \epsilon$ (negl.)
- PCP: Prove $(x, w) \rightarrow \pi$ checkable in 3 bits
- Vector Com: $\text{IdxOpen}(\vec{m}, i) \rightarrow p, \text{IdxVerify}(c, i, m_i, p) \rightarrow \{0, 1\}$
- Kilian-Micali: PCP + Vector Com = SNARG
crazy practical
 (Merkle)
- Since our PCP-approach is infeasible, we'll use stronger commitment, and weaker PCPs

1 Polynomial Commitment Schemes (PCSs)

• Goal: Commit to $f \in \mathbb{F}^d[x]$

Open $y = f(x) \quad \uparrow \quad \mathbb{F} = \mathbb{Z}_p, \quad p = \Theta(2^\lambda)$

• Syntax

• $\text{Setup}(d) \rightarrow \text{pp}$

• $\text{Commit}(\text{pp}, f) \rightarrow c$

• $\text{EOpen}(\text{pp}, f, x) \rightarrow \pi$

• $\text{Eval}(\text{pp}, c, y, x, \pi) \rightarrow \{0, 1\}$

• Properties

• Correctness: $\forall d, f \quad \Pr \left[\begin{array}{l} \text{pp} \leftarrow \text{Setup}(d) \\ c \leftarrow \text{Commit}(\text{pp}, f) \\ \pi \leftarrow \text{EOpen}(\text{pp}, f, x) \end{array} \quad \text{Eval}(\text{pp}, c, y, x, \pi) = 1 \right] = 1$

• Index Binding: $\forall d, \quad \Pr \left[\begin{array}{l} \text{pp} \leftarrow \text{Setup}(d) \\ c, x, y, y', \pi, \pi' \leftarrow A(\text{pp}) \\ \text{Eval}(\text{pp}, c, y, x, \pi) \wedge \text{Eval}(\text{pp}, c, y', x, \pi') \wedge y \neq y' \end{array} \right] = 0$

• Efficiency:

• $\text{EOpen} \quad O(d)$

• $\text{Eval}, |\pi| \quad o(d)$

• Open attempt: Merkle trees of evaluations

• $\text{pp} = \emptyset$

• $\text{Commit}(\text{pp}, f) = \text{Merkle.Commit}(f(0), f(1), \dots, f(p-1))$

• $\text{EOpen} @ x$: open $f(x)$

• Correctness + security are good!

• Problem: Commit takes $\Theta(p) = \Theta(2^\lambda)$ time.

2 Detour: Pairings

- Invented by André Weil (Fr) in 1940
 - In prison - refused to fight in WW2
- Improved by Tate, Ate, Miller - still alive and working
- Let G_1, G_2, G_T be order- p groups, generated by g_1, g_2, g_3
- A pairing is a det., eff. function $e: G_1 \times G_2 \rightarrow G_T$, with:
 1. Bilinearity: $e(u^a, v^b) = e(u, v)^{ab}$ (e is also called a bilinear map)
 2. Non-degeneracy: $e(g_1, g_2) = g_3$
 - (otherwise, $e(g_1, g_2)$ could be $1 \in G_T$)
- Pairings were 1st for cryptanalysis:

• Recall DDH for G :

$$\{ (g^a, g^b, g^{a \cdot b}) \}_{a,b} \approx_c \{ (g^a, g^b, g^r) \}_{a,b,r}$$

• this is false in G if $e: G \times G \rightarrow G_T$ exists!

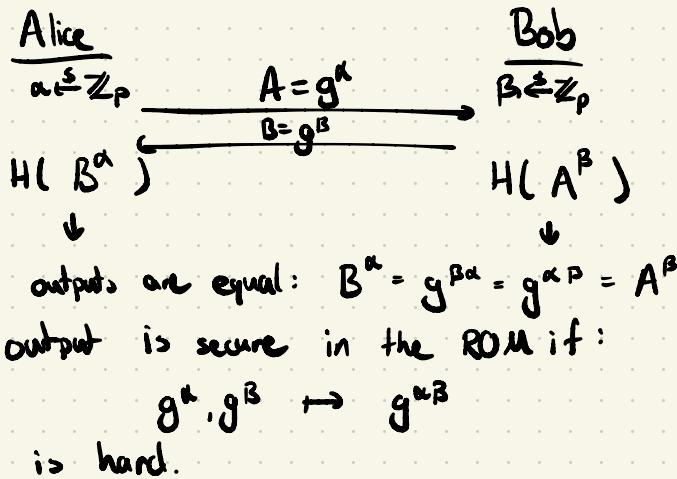
$$A(g^a, g^b, g^r) \stackrel{\Delta}{=} e(g^a, g^b) \stackrel{?}{=} e(g^r, g)$$

$$g_T^{a \cdot b} \stackrel{?}{=} g_T^r$$
$$a \cdot b \stackrel{?}{=} r$$

- a pairing with $G_1 = G_2$ is called symmetric.
- others: asymmetric
- given that symmetric pairing break DDH, many were scared.
- But, as we will see, bilinear structure can be good too
- you can use asymmetric pairings and DDH or more
- you can use symmetric pairings and more than DDH

Application 1: Key Exchange.

Diffie-Hellman:

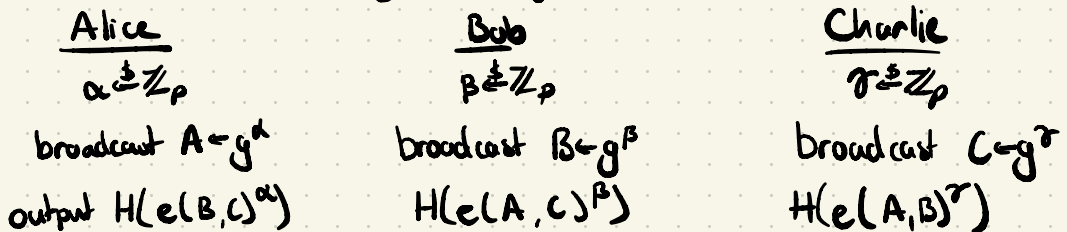


This is the Computational Diffie-Hellman (CDH) problem

• Diffie-Hellman powers the internet:

- Client $\leftrightarrow$ Server
- But what about 3 parties?
 - Example: group chats?
- Can they do a 1-round key exchange?
- Open for decades
- trivial with pairings:

• Joux tripartite key exchange [J04, Gödel Prize]



correctness:

$$\begin{aligned} &= H(e(g_1^\beta, g_2^\gamma)^\alpha) \\ &= H(e(g_1, g_2)^{\beta\gamma\alpha}) \end{aligned}$$

$$= H(g_1^{\alpha\beta\gamma})$$

$$= H(g_1^{\alpha\beta\gamma})$$

$$= H(g_T^{aBr})$$

security: requires bilinear CDH:

hard to compute:

$$g^a, g^b, g^c \mapsto g_T^{abc}$$

• Q: do pairings make 4-party key exchange easy?

• A: no.

$$e(g^a, g^b) = g_T^{ab}$$

$\uparrow \quad \uparrow \quad \quad \uparrow$
 in $G \quad \quad \quad$ in G_T

← now you can't use e ñ

• pairings multiply secret exponents and move to G_T

• intuition: you get 1 secret multiplication.

• just 1

Application 2: Short, deterministic signatures: [BLS]

BLS signatures:

• Let $e: G_1 \times G_2 \rightarrow G_T$ be a pairing, H be a RO

• Key Gen $() \rightarrow (sk, pk)$:

$$sk \xleftarrow{\$} \mathbb{Z}_p$$

$$pk \leftarrow g_2^{sk}$$

• Sign $(sk, m) \rightarrow \sigma$:

$$\sigma \leftarrow H(m)^{sk}$$

• Verify (pk, m, σ) :

$$e(\sigma, g_2) \stackrel{?}{=} e(H(m), pk)$$

• Correctness:

$$\text{Verify}(pk, m, \text{Sign}(sk, m)) \xleftarrow{\text{Verifier "exponentiates" by } sk \text{ itself, using } e \text{ \& } pk} e(H(m)^{sk}, g_2) \stackrel{?}{=} e(H(m), g_2^{sk})$$
$$e(H(m), g_2)^{sk} \stackrel{?}{=} e(H(m), g_2)^{sk}$$

• security: requires q -Strong Diffie Hellman (q -SDH)

• very well-studied

• we'll see it very soon.

Concrete costs

• BLS pairing family (asymmetric) is best right now.

• BLS12-381:

• $\lambda = 128$

• p : 255 bits

• G_1 : 381 bits (round up to 384)] $\sim$ pairings waste space in group

• G_2 : 762 bits (768) (2x size)

• G_T : $\sim 4.5k$ bits

• g_1^k : $\sim 80 \mu s$

• g_2^k : $\sim 240 \mu s$

• g_T^k : $\sim 400 \mu s$

• e : $\sim 600 \mu s$

For comparison, a normal group (Ristretto):

g_1^k : $50 \mu s$

G_1 : 256 bits

p : 253 bits

(a little faster / smaller)

[3] A PCS from pairings: [Kate, Zaverucha, Goldberg] 2012

- Let $e: G_1 \times G_2 \rightarrow G_T$ be a pairing
- Basic idea: poly evaluation reduces to
 - poly division
 - poly multiplication
 - checked in exponent with e

• That doesn't make sense yet, but it will.

• Construction:

• Setup(d):

• $\tau \xleftarrow{\$} \mathbb{Z}_p$

• output $(g, \tau^i)_{i=0}^{d-1}, g_2^\tau$

• DO NOT OUTPUT τ

• security will rely on τ being unknown

• this is called a trusted setup

• Listen to Radiolab "The Ceremony" to hear the story of how this was done, the 1st time, in practice.

• then, ask me in office hours about what went wrong...

• Commit($pp, f \in \mathbb{Z}_p^{<d}[x]$) $\rightarrow c$:

• let $f(x) = f_0 + f_1x + f_2x^2 + \dots + f_{d-1}x^{d-1}$

• output $c = g_1^{f(\tau)}$ as $c \leftarrow \prod_{i=0}^{d-1} \underbrace{(g_1^{\tau^i})^{f_i}}_{\text{in } pp}$

• EOpen(pp, f, x) $\rightarrow \pi$

• $q(x) = (f(x) - y) / (x - x)$ (poly division - discard remainder)

• output $c = g_1^{q(\tau)}$ as $c \leftarrow \prod_{i=0}^{d-1} (g_1^{\tau^i})^{q_i}$

• $\text{Ever}(pp, c, x, y, \pi) \rightarrow \{0, 1\}$

• check:

$$e(c, g_2) \stackrel{e \text{ in PP}}{=} e(\pi, g_2^\pi / g_2^x) \cdot e(g_1^y, g_2)$$

• What is going on?

• Fact: $y = f(x)$ iff x is a root of $f(X) - y$
 iff $X - x$ divides $f(X) - y$
 iff $\exists q(X)$ s.t. $f - y = q \cdot (X - x)$

• Correctness:

• First, since $y = f(x)$,

$$f - y = q \cdot (X - x)$$

and,

$$f(\tau) - y = q(\tau) \cdot (\tau - x)$$

• so:

$$\begin{aligned} e(c, g_2) &= g_2^{f(\tau)} \\ &= g_2^{q(\tau) \cdot (\tau - x) + y} \\ &= g_2^{q(\tau) \cdot (\tau - x)} \cdot g_2^y \\ &= e(g_1^{q(\tau)}, g_2^{(\tau - x)}) \cdot e(g_1^y, g_2) \\ &= e(\pi, g_2^\pi / g_2^x) \cdot e(g_1^y, g_2) \end{aligned}$$

(Commit, τ)

this is Ever.

• Evaluation binding requires a new assumption

• Assumption: q -Strong Diffie Hellman (q -SDH):

For all PPT A given

$$g_1^\tau, g_1^{\tau^2}, \dots, g_1^{\tau^{q-1}}, g_2^\tau$$

for hidden $\tau \in \mathbb{Z}_q$

A cannot output
 $(C \in \mathbb{Z}_p, g, \frac{1}{\pi+c})$

except w/ negl. probability

• This looks very similar to $[KZG'12]$'s construction, but it actually dates back to [BB04] signature

• Motivation:

• CDH says this is hard: $g^a, g^b \mapsto g^{ab}$

• reciprocals end up being equivalent:

$$g^a \mapsto g^{1/a}$$

• then, let A choose an offset

$$g^a \mapsto (g^{1/(a+c)}, c)$$

• then, give A more powers g^{a^i}

$$g^a, \dots, g^{a^2} \mapsto (g^{1/(a+c)}, c)$$

• so, q -SDH is stronger than SDH

• but still 20+ years old

• Evaluation binding.

• We'll reduce from $(d-1)$ -SDH.

• Let A be an eval. binding adversary

• Define a $(d-1)$ -SDH adversary B:

• gets $(g, \pi^i)_{i=0}^{d-1}, g_2^\pi$, gives to A (correct distribution)

• gets c, y, y', π, π'

• we'll show that if A wins, B can win

• have: $y \neq y'$

$$\begin{aligned} e(c, g_2) &= e(\pi, g_1^{\tau-x}) \cdot e(g_1^y, g_2) \\ &= e(\pi', g_1^{\tau-x}) \cdot e(g_1^y, g_2) \end{aligned}$$

$$\Rightarrow \pi^{\tau-x} g_1^y = \pi'^{\tau-x} g_1^{y'} \quad \text{goal: get } g_1^{\frac{1}{\tau-x}}$$

$$(\pi/\pi')^{\tau-x} = g_1^{y'-y}$$

$$(\pi/\pi')^{\frac{1}{y'-y}} = g_1^{\frac{1}{\tau-x}}$$

• so, B outputs

$$(-x, (\pi/\pi')^{\frac{1}{y'-y}})$$

• this solves $(d-1)$ -SDH by the deduction above.

• Efficiency:

• We'll count G -ops (time) and G -elements (size)

• Recall: $p = \Theta(2^\lambda)$, so $\log p = O(\lambda)$

• Sizes:

• $c \in G_1$

• $\pi \in G_1$

• Times:

• Eval: 3 pairings, $O(\lambda)$ G -ops (g_1^x, g_1^y) $O(\lambda)$

• Commit: naively $O(\lambda d)$ G -ops

• EOpen: naively $O(\lambda d)$ G -ops

Can be optimized to $O(\lambda d / \log d + d)$
via Pippenger's algorithm.