

## Problem Set 4

**Problem sets are due at 4:59pm on the due date.**

Please submit your problem set, in PDF format, on Gradescope. *Each problem should be in a separate page.*

You are to work on this problem set in groups of your choice. Each group member must independently write up and submit their own solutions.

*You must typeset your homework in L<sup>A</sup>T<sub>E</sub>X and submit it electronically!* Each problem answer must be provided as a separate page. Mark the top of each page with your group member names, the course number (6.5610), the problem set number and question, and the date. We have provided a template for L<sup>A</sup>T<sub>E</sub>X on the course website (see the *Psets* tab at the top of the page).

### Problem 4-1. SIS-based commitment

Recall that in class we defined the notion of a commitment scheme (see Lecture 10), and we showed a construction based on the LWE. That scheme is a bit commitment scheme where we commit to a single bit.

In this problem we consider a commitment scheme where we commit to long  $m$ -bit strings, and where the commitment is shrinking (in the sense the the commitment string is shorter than  $m$ ). The scheme is associated with the following algorithms (Gen, Com):

Gen generates a random matrix  $A \xleftarrow{\mathbb{R}} \mathbb{Z}_q^{2m \times n}$  where  $m > 5n \log q$ .

Com( $A, x, r$ ) =  $(x, r)^T A$  where  $r \xleftarrow{\mathbb{R}} \{0, 1\}^m$  is randomness used by the algorithm and  $x \in \{0, 1\}^m$  is the string we wish to commit to.

- Is this scheme statistically or computationally binding or neither. Explain your answer in a few sentences.
- Is this scheme statistically or computationally hiding or neither. Explain your answer in a few sentences.
- How do the properties of this scheme (binding and hiding) differ from those of the commitment scheme described in class (in Lecture 10).

### Problem 4-2. Recovery in Shamir's secret-sharing scheme

In lecture, we saw Shamir's  $t$ -of- $n$  secret-sharing scheme over the finite field  $\mathbb{F} = \mathbb{F}_p$ , where  $p > n$  is a prime.

To secret-share a message  $\mu \in \mathbb{F}$ , the dealer chooses a random polynomial  $M(x)$  of degree  $\leq t - 1$  with coefficients in  $\mathbb{F}$ , such that  $M(0) = \mu$ . To party  $i$ , for  $i \in \{1, \dots, n\}$ , the dealer gives the secret share  $(i, M(i)) \in \mathbb{F}^2$ .

We saw in class that any  $t$  shares are enough to recover the secret message  $\mu$  via polynomial interpretation.

In this problem, we consider the more interesting case, in which some parties give valid shares  $(i, M(i))$  and other parties give invalid shares  $(i, r)$ , where  $r \in \mathbb{F}$  is some arbitrary value not equal to  $M(i)$ .

- Say that  $t = n/4$ . You are given  $n$  shares. Of the shares,  $n - 10$  are valid shares, and 10 are invalid shares. (Assume that  $n, t \gg 10$ .)  
Give a polynomial-time algorithm for recovering the secret that has zero probability of error. Argue why your algorithm is correct and analyze its runtime.
- Explain how your algorithm behaves when you are given  $n$  shares and  $n/4$  of them are invalid. (As in part (a)), assume that  $t = n/4$ .) Is your algorithm still correct? Does it still run in polynomial time?
- Explain how your algorithm behaves when you are given  $n$  shares and  $7n/8$  of them are invalid. (As in part (a)), assume that  $t = n/4$ .) Is your algorithm still correct? Does it still run in polynomial time?

- (d) Say that you are given  $n$  shares  $(r_1, \dots, r_n)$  with  $d = n/4$  of them being invalid. Let  $E(x)$  be a polynomial of degree at most  $d$  such that if  $r_i$  is an invalid share, then  $E(i) = 0$ . Explain why, for all  $i \in \{1, \dots, n\}$ , it holds that

$$E(i) \cdot (M(i) - r_i) = 0 \quad \in \mathbb{F}. \quad (1)$$

Given such a polynomial  $E$ , explain how to recover the secret in polynomial time with zero correctness error. The general recovery algorithm for the Shamir secret-sharing scheme works by finding  $E$  and then recovering as you do in this problem. The cleverness comes in finding  $E$  efficiently.

- (e) [Extra credit] Explain how to find the polynomial  $E$  in polynomial time. (*Hint*: View each equation of the form Eq. (1) as a linear relation on some set of variables.)

### Problem 4-3. Sumcheck protocol

Following the notation in the lecture note, in the sumcheck protocol, the prover wants to prove that

$$\sum_{h_1, h_2, \dots, h_m \in H} f(h_1, h_2, \dots, h_m) = \beta$$

for a function  $f : \mathbb{F}^m \rightarrow \mathbb{F}$ . There will be  $m$  rounds of communication. In  $i$ -th round, the prover sends a polynomial  $g_i(x)$ , while the verifier performs the checks and sends a uniformly random  $t_i$  back.

- (a) Describe how a malicious prover can convince the verifier that

$$\sum_{h_1, h_2, \dots, h_m \in H} f(h_1, h_2, \dots, h_m) = \beta'$$

for some  $\beta \neq \beta'$ , **if all  $t_i$ s are known beforehand**. In other words, the protocol has only two steps: the verifier sends all  $t_i$ s, and the malicious prover sends back all  $g_i$ s.

- (b) Implement the prover in the sumcheck protocol by filling out `prover_template.sage`. You can find a quick reference here. Make sure that
- You submit a sage file on gradescope (with `.sage` file extension).
  - The code does not use any outside library such as numpy. You can double-check it by running on sagecell.
  - No hacking the auto-grader.

`verifier.sage` is the actual code running on the auto-grader, so you can use it to debug.