Massachusetts Institute of Technology
6.5610: Applied Cryptography
Henry Corrigan-Gibbs and Yael Tauman Kalai

April 16, 2024
Due: March 15, 2024 at 4:59pm

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 LaTeX *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 LaTeX 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 $(\mathsf{Gen}, \mathsf{Com})$:

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

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

(a) Is this scheme statistically or computationally binding or neither. Explain your answer in a few sentences.

Solution: Computationally binding. In order for $\mathrm{Com}(A, x_0, r_0) = \mathrm{Com}(A, x_1, r_1)$, $(x_0, r_0)^T A = (x_1, r_1)^T A$ and $(x_0, r_0)^T A - (x_1, r_1)^T A = ((x_0, r_0) - (x_1, r_1))^T A = 0$. Since $x_0, r_0, x_1, r_1 \in \{0,1\}^m$ (they are small), SIS says that such a vector is hard to find. However, this only holds for PPT adversaries and an all powerful adversary can brute force such a vector, so it is not computationally binding (or cite that a scheme cannot be both statistically hiding and binding).

(b) Is this scheme statistically or computationally hiding or neither. Explain your answer in a few sentences.

Solution: Statistically hiding. Let $A' \in \mathbb{Z}^{m \times n}$ be the top half of A, and let $A'' \in \mathbb{Z}^{m \times n}$ be the bottom half of A. Then $(x, r)^T A = xA' + rA''$. We know rA'' is random, so it statistically hides xA' and therefore x.

(c) How do the properties of this scheme (binding and hiding) differ from those of the commitment scheme described in class (in Lecture 10).

Solution: In lecture: statistically binding and computationally hiding.

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 $i \in \{1, \ldots, n\}$, 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 μ 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 $M(i)$.

(a) 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.

Solution: Try all $\binom{n}{10}$ subsets of the shares in time $O(n^{10})$ to be the invalid shares. For each subset, create a polynomial of degree $\leq t - 1$ using the shares outside the subset and check that all points lie on the polynomial. The $n - 10$ valid points will always define a unique degree-$\leq t - 1$ so any invalid share will be off of this line. Accept once finding a good set of shares.

(b) 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?

Solution: Try all $\binom{n}{n/4}$ subsets of the shares. For each subset, check that all points lie on a polynomial of degree $\leq t - 1$. Accept once finding a good set of shares. The algorithm is still correct because the $3n/4$ points will define a polynomial of degree $n/4 - 1$, but it runs in at least $n^{n/4}$ time.

(c) 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?

Solution: The algorithm is not correct. The $n - 7n/8 = n/8$ shares cannot define a polynomial of degree $n/4 - 1$.

(d) Say that you are given n shares $(r_1, \ldots, r_n)$ 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 $i \in \{1, \ldots, n\}$, it holds that

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

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.

Solution: The equation holds because if i is invalid, then $E(i) = 0$ and the equation holds. If i is valid, then $M(i) = r_i$ and the equation holds as well.
Given E, for all i, we check if $E(i) = 0$. If $E(i) \neq 0$, then we know that it is valid because E has at most d roots. Then we use all valid shares to recover the secret.

(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.)

Solution: We have n linear equations $E(i) \cdot (M(i) - r_i) = E(i)M(i) - E(i)r_i = 0$. If we think of $E(i)M(i)$ as a degree $2d - 1$ polynomial, then we have $2d + d + 1 = 3n/4 + 1$ variables. We can solve this system of equations in polynomial time to obtain $E(i)$.

Problem 4-3. Sumcheck protocol

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

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

for a function $f : \mathbb{F}^m \to \mathbb{F}$. 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

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

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

Solution: If is suffice to find g_is of degree at most d such that

$$\begin{aligned}
\sum_{x \in H} g_1(x) &= \beta' \\
\sum_{x \in H} g_2(x) &= g_1(t_1) \\
&\vdots \\
\sum_{x \in H} g_m(x) &= g_{m-1}(t_{m-1}) \\
g_m(t_m) &= f(t_1, t_2, \ldots, t_m)
\end{aligned}$$

The malicious prover can first choose arbitrary values for the right hand side: $g_1(t_1), \ldots, g_{m-1}(t_{m-1})$. Now for each g_i, we only need to make sure that $\sum_{x \in H} g_i(x)$ and $g_i(t_i)$ are some fixed values. Such g_i can be easily constructed with lagrange interpolation.

(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.