CS 346: Cryptography **Spring 2023**

Homework 4: Public-Key Cryptography

Due: March 31, 2023 at 11:59pm (Submit on Gradescope) **Instructor:** David Wu

Instructions. You **must** typeset your solution in LaTeX using the provided template:

`https://www.cs.utexas.edu/~dwu4/courses/sp23/static/homework.tex`

You must submit your problem set via Gradescope (accessible through Canvas).

Collaboration Policy. You may discuss your general *high-level* strategy with other students, but you may not share any written documents or code. You should not search online for solutions to these problems. If you do consult external sources, you must cite them in your submission. You must include the names of all of your collaborators with your submission. Refer to the official course policies for the full details.

Problem 1: CCA-Security and Public-Key Encryption [18 points]. Suppose $(\mathsf{KeyGen}, \mathsf{Encrypt}, \mathsf{Decrypt})$ is a CCA-secure public-key encryption scheme with message space $\mathcal{M}$ (where $|\mathcal{M}| > 1$). We construct a new public-key encryption scheme $(\mathsf{KeyGen}', \mathsf{Encrypt}', \mathsf{Decrypt}')$ with message space $\mathcal{M}^2$ as follows:

- KeyGen': Sample $(\mathsf{pk}, \mathsf{sk}) \leftarrow \mathsf{KeyGen}$. Output the public key pk and the secret key sk.
- $\mathsf{Encrypt}'(\mathsf{pk}, (m_1, m_2))$: Output $\mathsf{ct} = (\mathsf{ct}_1, \mathsf{ct}_2)$ where $\mathsf{ct}_1 \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_1)$ and $\mathsf{ct}_2 \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_2)$.
- $\mathsf{Decrypt}'(\mathsf{sk}, (\mathsf{ct}_1, \mathsf{ct}_2))$: Output $(\mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}_1), \mathsf{Decrypt}(\mathsf{sk}, \mathsf{ct}_2))$.

Prove or disprove: $(\mathsf{KeyGen}', \mathsf{Encrypt}', \mathsf{Decrypt}')$ is CCA-secure.

Problem 2: Commitment Schemes from Discrete Log [20 points]. A commitment scheme is a digital analog of a "sealed envelope." Specifically, a sender can *commit* to a message m and send the resulting commitment c to a receiver (i.e., seal the message in an envelope). The commitment c should not reveal anything about the committed value m. Later on, the sender can *open* up the commitment and convince the receiver that c is indeed a commitment to the message m (i.e., open up the envelope and recover the original message). The commitment scheme is *hiding* if c hides the message m and is *binding* if the sender cannot open the commitment c to any message $m' \neq m$. In this problem, we will construct a commitment scheme from the discrete log assumption:

- **Public parameters:** Let $\mathbb{G}$ be a group of prime order p and let $g, h \in \mathbb{G}$ be arbitrary elements of $\mathbb{G}$ (that are not the identity element).
- **Commitment:** To commit to a message $m \in \mathbb{Z}_p$, sample $r \xleftarrow{\text{R}} \mathbb{Z}_p$ and output the commitment $c \leftarrow g^m h^r$.
- **Open:** To open the commitment c to the message m, the sender gives (m, r) to the receiver and the receiver checks that $c = g^m h^r$.

(a) Show that the above commitment scheme is *perfectly hiding* (i.e., the commitment c does not leak *any* information about the committed message m). Namely, show that given the commitment $c \in \mathbb{G}$, every candidate message $m' \in \mathbb{Z}_p$ is *equally likely* (over the randomness of r). One way to show this is that for every $m' \in \mathbb{Z}_p$, there is a *unique* $r' \in \mathbb{Z}_p$ such that $c = g^{m'} h^{r'}$.

(b) Show that the above commitment scheme is *computationally binding* assuming hardness of discrete log in $\mathbb{G}$. Namely, show that if an efficient adversary can output a commitment c together with openings (m, r) and (m', r') such that $g^m h^r = c = g^{m'} h^{r'}$ and $m \neq m'$, then the adversary can also compute the discrete log of h base g. In other words, if the sender can open the commitment in two different ways, then it can also compute the discrete log of h in $\mathbb{G}$.

Remember to give a brief explanation why any inverses you take actually exist.

Problem 3: Hash Functions from Discrete Log [20 points]. Let $\mathbb{G}$ be a group of prime order p with generator g. Sample $h_1, \dots, h_n \xleftarrow{\text{R}} \mathbb{G}$ and define the hash function $H_{h_1,\dots,h_n} : \mathbb{Z}_p^n \to \mathbb{G}$ as follows:

$$H_{h_1,\dots,h_n}(x_1, \dots, x_n) := h_1^{x_1} h_2^{x_2} \cdots h_n^{x_n} \in \mathbb{G}.$$

(a) Show that $H_{h_1,\dots,h_n}$ is collision resistant under the discrete log assumption in $\mathbb{G}$. Specifically, in the (keyed) collision-resistant hashing security game, the adversary is first given $h_1, \dots, h_n \xleftarrow{\text{R}} \mathbb{G}$ and it succeeds if it outputs $(x_1, \dots, x_n) \neq (x_1', \dots, x_n')$ such that $H_{h_1,\dots,h_n}(x_1, \dots, x_n) = H_{h_1,\dots,h_n}(x_1', \dots, x_n')$. In the discrete log security game, the adversary is given $h \xleftarrow{\text{R}} \mathbb{G}$, and it wins if it outputs $x \in \mathbb{Z}_p$ such that $h = g^x$. **Hint:** Consider a reduction algorithm that starts by guessing the index $i^* \in [n]$ (uniformly at random) where $x_{i^*} \neq x_{i^*}'$. Show that your reduction algorithm succeeds whenever the guess is correct. Remember to compute the advantage of your reduction algorithm (for breaking the discrete log assumption).

(b) Show that the function $H_{h_1,\dots,h_n}$ has a *trapdoor* that can be used to sample pre-images. Specifically, show that if someone knew the discrete logs of $h_1, \dots, h_n$ (i.e., $z_i \in \mathbb{Z}_p$ where $h_i = g^{z_i}$ for each $i \in [n]$), then on input any $t \in \mathbb{Z}_p$, they can find a pre-image $(x_1, \dots, x_n) \in \mathbb{Z}_p^n$ such that $H_{h_1,\dots,h_n}(x_1, \dots, x_n) = g^t$.

Problem 4: Time Spent [2 points]. How long did you spend on this problem set? This is for calibration purposes, and the response you provide does not affect your score.

Optional Feedback. Please answer the following *optional* questions to help us design future problem sets. You do not need to answer these questions. However, we do encourage you to provide us feedback on how to improve the course experience.

(a) What was your favorite problem on this problem set? Why?

(b) What was your least favorite problem on this problem set? Why?

(c) Do you have any other feedback for this problem set?

(d) Do you have any other feedback on the course so far?