CS 6222: Introduction to Cryptography **Spring 2020**

Homework 5: Cryptographic Protocols

Due: April 29, 2020 at 5pm (Submit on Gradescope) **Instructor:** David Wu

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

https://www.cs.virginia.edu/dwu4/courses/sp20/static/homework.tex

You must submit your problem set via Gradescope. Please use course code **MB84NW** to sign up.

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 computing IDs of all of your collaborators with your submission. Refer to the official course policies for the full details.

Problem 1: Schnorr's Protocol and Signatures [20 points].

(a) We saw in lecture that in the Schnorr signature scheme, signing different messages using the same randomness leaks the secret key. Show that this is also the case for ECDSA. **Remark:** This attack was used to break security on the Sony Playstation 3. It turns out that Sony had used ECDSA with a *fixed* string as the randomness to sign software for the PS3. This attack allowed hackers to extract Sony's ECDSA signing key and in turn, issue arbitrary firmware updates for the PS3. Thus, when using ECDSA, it is also critical to derandomize the signing algorithm.

(b) Suppose in Schnorr's protocol, the challenger sampled the challenge from a set S where $|S| = k$. Show that this protocol has soundness error $1/k$. **Remark:** This shows that to achieve negligible soundness error in Schnorr's protocol, the challenge must be samples from a set of super-polynomial size.

(c) Let $\mathbb{G}$ be a cyclic group of order $n = kq$, where $k > 1$ is a small constant that is relatively prime with q (e.g., $k = 8$) and q is prime. Let g be a generator of $\mathbb{G}$, and let $\hat{g} = g^k \in \mathbb{G}$. In this case, $\hat{g}$ generates a subgroup of $\mathbb{G}$ of prime order q. Consider Schnorr's protocol defined in the prime order group generated by $\hat{g}$ (this coincides with the typical setting in cryptography where we work over a group of prime order). Namely, on input an instance $(\hat{g}, \hat{h} = \hat{g}^x)$, the prover's first message is $\hat{g}^r$ where $r \xleftarrow{R} \mathbb{Z}_q$, the verifier's challenge is $t \xleftarrow{R} \mathbb{Z}_q$, and the prover's response is $r + xt \in \mathbb{Z}_q$. Show that even if $\hat{h} \notin \langle \hat{g} \rangle$, then a (malicious) prover is still able to convince the verifier to accept with probability that is negligibly close to $1/k$ (you can assume that the ratio k/q is negligible). Namely, even if $\hat{h}$ is not in the group generated by $\hat{g}$, the prover can nonetheless convince the verifier that it knows the discrete log of $\hat{h}$ base $\hat{g}$ (even though the discrete log does *not* exist in this case).

Remark: This show that Schnorr's protocol is no longer sound if the base $\hat{g}$ is not a generator of the full group $\mathbb{G}$ and $\mathbb{G}$ has small prime factors. To prevent this attack and recover soundness, the verifier must *additionally* check that $\hat{h}$ is indeed in the group generated by $\hat{g}$. This setting is very common in practice (and should not be glossed over when designing cryptographic systems). For instance, if we consider groups over the integers, we typically work over a prime-order subgroup of $\mathbb{Z}_p^*$ (e.g., the subgroup of prime order q when $p = kq + 1$ for some constant k) or over an elliptic-curve group with order kq and small k (in this case, k is referred to as the "cofactor" of the curve).

Problem 2. Graph Isomorphism [16 points]. Let $G = (V, E)$ be a graph, where V is the set of vertices and E is the set of edges. We say that two graphs $G_1 = (V, E_1), G_2 = (V, E_2)$ are *isomorphic* if there exists a permutation $\pi\colon V \to V$ such that $e = (u, v) \in E_1$ if and only if $e' = (\pi(u), \pi(v)) \in E_2$. The graph isomorphism language (for graphs of size n) is the set of graphs (G_1, G_2) on n nodes that are isomorphic.

Construct a Σ-protocol for the graph isomorphism problem with soundness $1/2$ and where the verifier's challenge message is a single bit $b \xleftarrow{R} \{0,1\}$. Prove completeness, soundness, and honest-verifier zero-knowledge of your protocol. In this case, the prover and verifier both know the graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$. The prover additionally knows the permutation $\pi\colon V \to V$ between G_1 and G_2. **Hint:** The prover will begin by sampling a random permutation $\tau\colon V \to V$.

Problem 3. Private Contact Tracing [54 points]. Due to the ongoing COVID-19 pandemic, there have been many proposals for developing privacy-preserving contact tracing protocols. Such a protocol would allow infected individuals to privately notify individuals that they have come in contact with without needing to reveal who they are. We will consider a simple approach here. We will assume that all users are running the private contract tracing app on their phone.

Basic protocol. Once a day, the application samples a fresh token $t \xleftarrow{R} \{0,1\}^\lambda$. Every epoch (e.g., five minutes), the application broadcasts its token to other users in the immediate vicinity. Each device maintains two lists: a list of tokens it has broadcast and a list of tokens that it has received. If a user learns that they have been infected, they will publish the list of tokens that they have broadcast to a central database. Once a day, the application checks whether any of the tokens it has received appears in the central database. If so, then the user learns that they have been exposed to an infected individual. While this basic protocol enables contact tracing, it leaks the identity of the infected individual (e.g., whenever a user receives a token, they can annotate it with the contact information of the individual who sent it and then check if that token is the one that appears in the database).

Private contact tracing. To support *private* contact tracing, we will use a "mix-net" that will blind and shuffle user tokens. We will work in a group $\mathbb{G}$ of prime order p and generator g and let $H\colon \{0,1\}^\lambda \to \mathbb{G}$ be a hash function (that we will model as a random oracle). The mix-net has a secret key $k \xleftarrow{R} \mathbb{Z}_p$. Let $t_1, \ldots, t_n \in \{0,1\}^\lambda$ be the set of tokens a user Alice receives over the course of a day. The protocol now proceeds as follows:

- Alice samples $r \xleftarrow{R} \mathbb{Z}_p$ and computes $x_i \leftarrow H(t_i)^r \in \mathbb{G}$ for $i = 1, \ldots, n$. Alice sends $x_1, \ldots, x_n$ to the mix-net.
- The mix-net samples a random permutation $\pi\colon \{1, \ldots, n\} \to \{1, \ldots, n\}$ and computes $y_i \leftarrow x_{\pi(i)}^k \in \mathbb{G}$ for each $i \in \{1, \ldots, n\}$. The mix-net replies to Alice with the values $y_1, \ldots, y_n$. You may assume that the mix-net will first perform a simple de-duplication check (if Alice submits multiple copies of an element x_i, the mix-net will drop all duplicate copies of x_i).
- Alice computes the set of blinded tokens $y_1^{r^{-1}}, \ldots, y_n^{r^{-1}} \in \mathbb{G}$.

At the end of the protocol, Alice learns $H(t_1)^k, \ldots, H(t_n)^k$ in a *permuted* order. For a token t, we refer to $H(t)^k$ as the "blinded" token (in fact, this is pseudorandom, though we will not show this here). Alice can use the same protocol as above to obtain the blinded version of her own tokens. When a user is infected, they now publish their *blinded* tokens to the central database, which can be queried and compared exactly as in the basic scheme.

(a) *Privacy for infected individuals.* In 1-2 sentences, explain why Alice can only learn the *number* of individuals in her list of contacts that are infected, but not their identities. You should assume that all parties follow the protocol.

(b) *Privacy against the mix-net.* Show that under the DDH assumption, the mix-net does not learn anything about Alice's tokens $H(t_1), \ldots, H(t_n)$. Namely, show that under DDH (and modeling H as a random oracle), Alice's message $(x_1, \ldots, x_n)$ is computationally indistinguishable from a vector of n uniformly random group elements. You may use the fact that under DDH, the following two distributions are computationally indistinguishable:

$$(g, g^x, g^{y_1}, \ldots, g^{y_n}, g^{xy_1}, \ldots, g^{xy_n}) \quad \text{and} \quad (g, g^x, g^{y_1}, \ldots, g^{y_n}, g^{z_1}, \ldots, g^{z_n}),$$

where all exponents are sampled independently and uniformly from $\mathbb{Z}_p$. In the "random oracle" model, you are free to choose the values of $H(t_1), \ldots, H(t_n)$, so long as their distribution is independent and uniform over $\mathbb{G}$.

Extra Credit [8 points]. Show that this assumption holds under DDH.

(c) *Handling a malicious mix-net.* A malicious mix-net could try to compromise security by incorrectly performing the mixing operation (e.g., replace elements by bogus elements). Design an honest-verifier zero-knowledge protocol with negligible soundness error that the mix-net can use to prove that it performed the blind-and-shuffle operation correctly. Prove correctness, soundness, and honest-verifier zero-knowledge of your protocol. Your construction should *not* rely on general NP-hardness reductions. **Hint:** Adapt the ideas from the graph isomorphism protocol and the Chaum-Pedersen protocol. Note that your construction will probably not use either protocol as a black box, so you will probably need to give an independent analysis of each of the properties.

(d) *Insecurity with a malicious client.* Unfortunately, this protocol is not secure against a malicious client. Suppose that Alice has a collection of tokens $(t_1, \ldots, t_n)$ and wants to learn $(H(t_1)^k, \ldots, H(t_n)^k)$ in this order. In other words, Alice can undo the permutation and associate blinded tokens with identities. Show that Alice can do this with high probability using a *single* execution of the above protocol and providing a vector of $2n$ (maliciously-chosen) messages to the mix-net.

Problem 4: Time Spent [3 extra credit 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. To receive the extra credit for this problem, you must submit your homework to Gradescope (with the provided template) and properly assign *all* problems to their respective pages.

Optional Feedback [0 points]. 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?