CS 6222: Introduction to Cryptography **Spring 2021**

Homework 4: Public-Key Cryptography

Due: April 16, 2021 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/sp21/static/homework.tex

You must submit your problem set via Gradescope. Please use course code **D55GP5** 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: DDH in Composite-Order Groups [7 points]. Let $\mathbb{G}$ be a cyclic group of order $2q$ where q is odd. Let g be a generator of $\mathbb{G}$. Show that the DDH assumption does not hold in $\mathbb{G}$. (Specifically, for this setting, the DDH assumption asserts that the distributions of (g, g^a, g^b, g^{ab}) and (g, g^a, g^b, g^r) where $a, b, r \xleftarrow{R} \mathbb{Z}_{2q}$) are indistinguishable to any efficient adversary.)

Remark: This shows that the DDH assumption does not hold over $\mathbb{Z}_p^*$ whenever $p = 2q+1$ for some odd q. In fact, the DDH assumption does not hold in $\mathbb{Z}_p^*$ for any prime p (there is an efficient distinguisher based on Legendre symbols). However, assumptions such as CDH or discrete log still plausibly hold over $\mathbb{Z}_p^*$.

Problem 2: Encrypted Group Chat [18 points]. Suppose a group of n people (denoted $P_1, \ldots, P_n$) want to set up a shared key for an encrypted group chat. At the end of the group key-exchange protocol, everyone within the group should know the key, but an eavesdropper on the network should not. We will use the following variant of Diffie-Hellman over a group $\mathbb{G}$ of prime order p and generator g:

- At the beginning of the protocol, P_1 chooses $s \xleftarrow{R} \mathbb{Z}_p$. We will view P_1 as the group administrator that all of the other parties know.
- Each of the other parties P_i ($2 \le i \le n$) samples $r_i \xleftarrow{R} \mathbb{Z}_p$ and sends $x_i \leftarrow g^{r_i}$ to the group administrator P_1. The administrator P_1 replies to P_i with x_i^s.
- The group key is then defined to be $k \leftarrow H(g^s)$, where $H \colon \mathbb{G} \to \{0,1\}^\lambda$ is a hash function.

Both the group description $(\mathbb{G}, p, g)$ and the hash function H are public and known to everyone (both the protocol participants and the eavesdropper).

(a) Show that both the group administrator P_1 and each of the parties P_i ($2 \le i \le n$) are able to efficiently compute the group key.

(b) We say that the group key-exchange protocol is secure against eavesdroppers if no efficient adversary who sees the transcript of messages sent by the honest parties $P_1, \ldots, P_n$ is able to distinguish the group key k from a uniform random string over $\{0,1\}^\lambda$, except perhaps with negligible probability.

If we model H as an "ideal hash function" (i.e., random oracle), it suffices to argue that the shared Diffie-Hellman secret g^s is *unguessable*: namely, for all efficient adversaries $\mathcal{A}$,

$$\Pr[\mathcal{A}(x_2, x_2^s, \ldots, x_n, x_n^s) = g^s] = \mathrm{negl}(\lambda), \tag{1}$$

where $x_i = g^{r_i}$ and $r_2, \ldots, r_n, s \xleftarrow{R} \mathbb{Z}_p$. This means that an eavesdropper who only observes the messages sent by the honest parties cannot guess g^s, and correspondingly, the shared key $H(g^s)$ is uniformly random and unknown to the adversary.

Show that under the CDH assumption in $\mathbb{G}$, the shared Diffie-Hellman secret g^s in the group key-exchange protocol above is unguessable (i.e., Eq. (1) holds for all efficient adversaries $\mathcal{A}$). As usual, you should consider the contrapositive: show that if there exists an efficient adversary $\mathcal{A}$ that can predict g^s from the above challenge tuple $(x_2, x_2^s, \ldots, x_n, x_n^s)$, then there exists an efficient algorithm $\mathcal{B}$ that breaks CDH in $\mathbb{G}$. **Hint:** Your algorithm $\mathcal{B}$ may need to invoke $\mathcal{A}$ *more than once*. Remember to compute the advantage of the adversary you construct.

Problem 3: Computing on Encrypted Data [20 points]. Let $N = pq$ be an RSA modulus and suppose that $\gcd(N, \varphi(N)) = 1$. Consider the following public-key encryption scheme with message space $\mathbb{Z}_N$. The public key $\mathsf{pk} = N$ is the RSA modulus $N = pq$ and the secret key sk is the factorization $\mathsf{sk} = (p, q)$. Let $g = 1 + N \in \mathbb{Z}_{N^2}^*$. To encrypt a message $m \in \mathbb{Z}_N$, sample $h \xleftarrow{R} \mathbb{Z}_{N^2}^*$ and compute $c \leftarrow g^m h^N \in \mathbb{Z}_{N^2}^*$.

(a) Show that the discrete logarithm assumption base g in $\mathbb{Z}_{N^2}$ is easy. Namely, give an efficient algorithm that takes as input (g, h) where $h = g^x$ for some $x \in \mathbb{Z}_N$, and outputs x. **Hint:** Use the binomial theorem: $(a+b)^k = \sum_{i=0}^{k} \binom{k}{i} a^i b^{k-i}$.

(b) Show how to *efficiently* implement the decryption algorithm $\mathsf{Decrypt}(\mathsf{sk}, c)$. Namely, describe an efficient algorithm that given the secret key $\mathsf{sk} = (p, q)$ and a ciphertext $c = g^m h^N$, outputs the message $m \in \mathbb{Z}_N$. You may use the fact that $\varphi(N^2) = N\varphi(N)$. **Hint:** Remember that the decrypter can compute $\varphi(N) = (p-1)(q-1)$ from the secret key $\mathsf{sk} = (p, q)$.

(c) Show that this public-key encryption scheme is semantically secure assuming that no efficient adversary is able to distinguish the following two distributions:

$$(N, u) \quad \text{and} \quad (N, v),$$

where $N = pq$ is an RSA modulus, $u \xleftarrow{R} \mathbb{Z}_{N^2}^*$ and $v \xleftarrow{R} \{h \in \mathbb{Z}_{N^2}^* : h^N\}$. Namely, show that the above encryption scheme is semantically secure assuming that it is hard to distinguish random values in $\mathbb{Z}_{N^2}^*$ from random N^{th} powers in $\mathbb{Z}_{N^2}^*$. **Hint:** You can follow the same structure as the analysis of ElGamal encryption from lecture.

(d) Show that given the public key pk and two ciphertexts $c_1 \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_1)$, $c_2 \leftarrow \mathsf{Encrypt}(\mathsf{pk}, m_2)$, there is an efficient algorithm that outputs a new ciphertext c where $\mathsf{Decrypt}(\mathsf{sk}, c) = m_1 + m_2 \in \mathbb{Z}_N$. Your algorithm should only depend on *public* parameters and *not* the value of the messages m_1, m_2.

Remark: This is an example of an encryption scheme that supports computation on *encrypted* values.

Problem 4: RSA Signatures with Same Modulus [14 points]. A company is issuing a signing/verification key for the RSA-FDH signature scheme to each of its employees. To simplify key management, the company decides to use the *same* RSA modulus N for everyone. Each employee has a different (and independently sampled) public and private exponent. Let d_i denote the secret signing exponent and e_i denote the public verification exponent for employee i. Recall that in the RSA-FDH signature scheme (with hash function H), e_i, d_i are chosen to satisfy $e_i d_i = 1 \bmod \varphi(N)$, a signature on a message m under the i^{th} signing key is $\sigma_i \leftarrow H(m)^{d_i} \bmod N$, and to verify the signature σ on message m (with respect to the public key e_i), one checks that $\sigma_i^{e_i} = H(m) \bmod N$.

In this problem, we will show that reusing the modulus N is completely broken. Namely, any employee i can use their secret signing key d_i together with knowledge of another employee's public verification key e_j to forge signatures on arbitrary messages on behalf of employee j. You should assume here that employee i knows N, e_i, d_i, but *not* necessarily $\varphi(N)$.

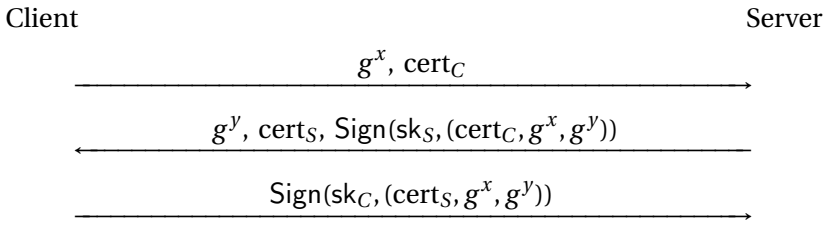
(a) Show that employee i can use their public/private key-pair (e_i, d_i) to efficiently compute *some* multiple of $\varphi(N)$. Denote this value N'.

(b) Suppose employee i is trying to forge a signature on behalf of employee j (with verification key e_j), and suppose moreover that $\gcd(N', e_j) = 1$. Show that using the value N' from the previous part, employee i can efficiently compute a signature σ on an *arbitrary* message m that verifies under e_j (i.e., compute σ such that $\sigma^{e_j} = H(m) \bmod N$).

(c) Show how to generalize your algorithm from the previous part to work even if $\gcd(N', e_j) \neq 1$.

Remember to prove the correctness and efficiency of each algorithm you develop.

Problem 5: Authenticated Key Exchange [16 points]. Consider the following protocol for authenticated key exchange (AKE) with mutual (i.e., two-sided) authentication. Both the client and the server have a public/private key-pair $(\mathsf{vk}_C, \mathsf{sk}_C)$ and $(\mathsf{vk}_S, \mathsf{sk}_S)$ for a digital signature scheme, respectively. They also have certificates cert_C and cert_S that authenticate vk_C and vk_S, respectively. The AKE protocol operates over a group $\mathbb{G}$ of prime order p and generator g. The client samples a fresh $x \xleftarrow{R} \mathbb{Z}_p$ and the server samples a fresh $y \xleftarrow{R} \mathbb{Z}_p$ in each invocation of the protocol:



In the second step, the client validates the signature with respect to the verification key contained in cert_S before computing its third message. At the end of the protocol, if all of the signatures verify (with respect to the verification keys identified by the certificates), the client and server computes the shared key as $k \leftarrow H(g, g^x, g^y, g^{xy})$. Moreover, the client outputs the party identified by cert_S as its peer in the connection and the server outputs the party identified by cert_C as its peer. Throughout this problem, you should consider an *active* network adversary that is allowed to register a certificate of its own (i.e., the adversary has a certificate cert_A for its identity A, which is *different* from both the client's identity C associated with cert_C and the server's identity S associated with cert_S).

(a) Suppose the server does not sign cert_C in its reply to the client. Namely, the server computes $\mathsf{Sign}(\mathsf{sk}_S, (g^x, g^y))$ instead of $\mathsf{Sign}(\mathsf{sk}_S, (\text{cert}_C, g^x, g^y))$. Show that there is an identity misbindinng attack on this protocol.

(b) Suppose the client only signed the server's certificate and not the Diffie-Hellman shares in the final message. Namely, the client computes $\mathsf{Sign}(\mathsf{sk}_C, \text{cert}_S)$ instead of $\mathsf{Sign}(\mathsf{sk}_C, (\text{cert}_S, g^x, g^y))$. Show that an adversary is able to establish a session with the server such that the adversary knows the shared key k, but the server thinks it is communicating with the party identified by cert_C (i.e., the client). **Hint:** Remember that an active network adversary is allowed to observe (and tamper with) *multiple* interactions between the client and the server.

(c) Suppose that the client signed its Diffie-Hellman share in its first message, and dropped the third message entirely. Namely, the client's first message is now $(g^x, \text{cert}_C, \mathsf{Sign}(\mathsf{sk}_C, g^x))$ and the overall protocol now completes in two rounds. Show that there is an identity misbinding attack on this protocol.

(d) Suppose that instead of signing the pair (g^x, g^y), the client and server instead signed g^{xy}. Explain why this is a bad idea.

This exercise illustrates that designing AKE protocols is very delicate, and simple modifications will lead to broken designs.

Problem 6: 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.

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?