Homework 7

1 Problem 1 (10 points)

Recall that in the ElGamal cryptosystem, the public key is a pair (g, h) where g is a generator for a group $\mathbb{G}$ of prime order, and $h = g^a$ where $a \in \mathbb{Z}_p$ is the secret key. To encrypt a message $m \in \mathbb{G}$, choose a random r and output $(g^r, h^r \times m)$.

(a) Suppose you have two ElGamal ciphertexts c_0, c_1 encrypting m_0 and m_1, respectively, where m_0, m_1 are unknown. Show how to devise a new ElGamal ciphertext c_2 which encrypts $m_0 \times m_1$. You only know the public key and the ciphertexts; you do not know m_0, m_1, the secret decryption key a, or the encryption randomness.

Thus, ElGamal is multiplicatively homomorphic: given two ciphertexts, it is possible to devise a new ciphertext that encrypts the product of the two plaintexts knowing just the public key.

(b) Let c be an ElGamal ciphertext encrypting an unknown message m. Show how to devise another ElGamal ciphertext c' encrypting m. c' should look like a fresh random ciphertext: its distribution should be the same as if you encrypted m from scratch, and should be independent of c (except that it encrypts the same message). As before, you know only the public key and the ciphertext; you do not know m, a, or the encryption randomness.

Thus, ElGamal is re-randomizeable, meaning you can take a ciphertext, and produce a fresh looking ciphertext that encrypts the same message.

2 Problem 2 (15 points)

Here, you will show that computing discrete logs mod a composite integer $N = pq$ is as hard as factoring N. In other words, you are given an algorithm A such that given $g, h \in \mathbb{Z}_N^*$, A efficiently computes an integer x such that $g^x \bmod N = h$. (Note that in general $\mathbb{Z}_N^*$ is not cyclic, so the discrete log is not guaranteed to exist. The algorithm for discrete logs is only guaranteed to work when the discrete log exists). You may assume A finds a discrete log with probability 1 when it exists; there is no guarantee that the x outputted by A will lie in any particular range. Show that given A, you can factor N.

To help you, here are some hints:

- Consider running $A(g, g^y)$ for a random $g \in \mathbb{Z}_N^*$, and where y is uniform in $[0, 2N]$. Let x be the output of A. Show that $y \neq x$ with noticeable probability, no matter what A does.
- When $x \neq y$, what relationship must x and y satisfy?
- Can you extend the above to compute the order of g, for any $g \in \mathbb{Z}_N*$. Consider running A several times on the same g but different h's.
- Finally, if you could compute the order for any $g \in \mathbb{Z}_N^*$, how does this let you factor N?

3 Problem 3 (5 points)

Let $H : \mathcal{K}_\lambda \times \{0,1\}^{m(\lambda)} \to \{0,1\}^{n(\lambda)}$ be a (keyed) hash function. Assume H is collision resistant. Suppose $m(\lambda) \geq n(\lambda) + \lambda$. Show that H is one-way: for any polynomial time adversary A, there is a negligible ϵ such that

$$\Pr\left[H(k, x') = y : \begin{smallmatrix} k \leftarrow \mathcal{K}_\lambda \\ x \leftarrow \{0,1\}^{m(\lambda)} \\ y \leftarrow H(k,x) \\ x' \leftarrow A(k,y) \end{smallmatrix}\right] < \epsilon(\lambda)$$

4 Bonus Problem 4 (5 points)

Bonus problem: Suppose $m(\lambda) = n(\lambda) + 1$ in Problem 3. Show that H is not necessarily one-way. That is, construct a hash function H such that H is collision resistant, but H is not one-way. You may assume as a building block any collision resistant hash function H'.

5 Problem 5 (20 points)

Let $(\mathsf{Gen}, \mathsf{Sign}, \mathsf{Ver})$ be a signature scheme with message space $\{0,1\}^*$. Suppose the scheme is only *one-time* secure. You will show how to use it to build a *stateful* scheme that is secure for an arbitrary number of messages. The basic idea is that, every time you want to sign a message m_0, you will actually generate a new secret key/public key pair $(\mathsf{sk}_1, \mathsf{pk}_1) \leftarrow \mathsf{Gen}(\lambda)$, and sign the pair (m_0, pk_1). The overall signature will be pk_1 together with the signature on (m_0, pk_1). You will then update your secret key to sk_1, and the receiver will update the public key to pk_1. When you want to sign the next

message m_1, you will generate a new secret key/public key pair $(\mathsf{sk}_2, \mathsf{pk}_2) \leftarrow \mathsf{Gen}(\lambda)$, and sign the pair (m_1, pk_2). This process continues for every message sign. The result is that any secret key is only used to sign a single message.

(a) Turn the above idea into a functioning signature scheme. That is, formally describe how signing and verifying work. Prove that the attacker, even after seeing an arbitrary number of signed messages of their choice, will be unable to produce a forgery.

(b) One problem with the above scheme is that an adversary intercepting communication may drop a message, which will result in the sender and receiver becoming out of sync. To resolve this issue, explain how to modify the signature scheme so that the recipient does not need to keep any state. *Hint: the size of your signatures is allowed to grow linearly as more messages are signed*

(c) your answer to Part (b) had a signature size that was linear in the number of users. Show that, if you have an upper bound of L on the number of messages that will ever be signed, you can actually have signatures of length $O(\log L)$. Your secret key, the sender's state, and the time it takes to sign and verify should all be at most $O(\log L)$.

Note that by setting $L = 2^\lambda$, you can handle any polynomial number of messages while keeping everything efficient

Hint: Consider signing two public keys for every message, instead of just 1

(d) It turns out that with a bit more effort, it is actually possible to build a full many-time secure stateless signature scheme from *secret key* tools, such as substitution-permutation networks. However, in practice, no one uses such signatures. Explain why this might be the case.