Homework 6

1 Problem 1 (20 points)

Let $N = pq$ be the product of two primes. In this problem, we will see that, in addition to p and q being large, it is important that $p-1$ and $q-1$ have large prime factors.

(a) Suppose you know an integer r that is a multiple of $p-1$, but not $q-1$. Explain how to factor N.

(b) Suppose $p-1$ is t-smooth (recall that this means all of the factors of $p-1$ are at most t). Explain how to compute an integer r that is a multiple of $p-1$. Your r should be no larger than about p^t (so its bit length is at most $t \log_2 p$), and should take time polynomial in t and $\log_2 p$ to compute.

(c) You are not quite done, as your multiple r might also be a multiple of $q-1$. Explain how to detect this case.

(d) If your r is a multiple of both $p-1$ and $q-1$, then show how to derive a different integer r' that is a multiple of $p-1$ but not $q-1$, or vice versa. Assume $p \neq q$ (if $p = q$, we can easily factor by taking square roots).

One option to avoid this attack is to choose p, q to be safe primes, which means that $(p-1)/2$ and $(q-1)/2$ are also prime. However, this is not actually necessary, as it turns out that a random large prime p will, with high probability, have $p-1$ not be smooth.

2 Problem 2 (20 points)

Let $N = pq$ be the product of two large unknown primes. In the RSA problem, you are given an integer e such that $GCD(e, \phi(N)) = 1$, and your goal is to compute eth roots mod N.

One potential way to compute eth roots is to compute an integer d such that ed mod $\phi(N) = 1$. Here, we will show that computing such a d from e is as hard as factoring.

(a) Show, given e and d, how to compute a value r that is a multiple of $\phi(N)$.

(b) If we were lucky and $r = \phi(N)$, explain how to recover p and q.

In general, r might be a large multiple of $\phi(N)$, so the above does not directly work. Instead, let $r = 2^k \times s$ for some odd s. Consider the following. Choose a random $g \in \mathbb{Z}_N^*$. Then compute $g_0 = g^s \bmod N, g_1 = g^{2s} \bmod N, g_2 = g^{4s} \bmod N, \ldots, g_k = g^{2^k s} = g^r \bmod N$.

(c) Explain how to compute s and k from r

(d) Prove that g_0 is not equal $\pm 1 \mod N$. However, note that $g_k = 1 \mod N$. Therefore, there is some i such that $g_i = 1 \mod N$ but $g_{i-1} \neq 1 \mod N$.

(e) Prove that with reasonable probability (over the choice of random g), $g_{i-1} \neq -1 \mod N$. For this problem, it will be useful to use Chinese Remaindering to think about the sequence $g_0, \ldots, g_k \bmod p$ and $\bmod q$. What does it mean that $g_{i-1} \neq -1 \mod N$ but $g_i = 1 \mod N$?

(f) If you were lucky enough go choose a g such that $g_{i-1} \neq -1 \mod N$, explain how to factor N.

(g) At a high level, explain why this *does not* mean computing eth roots is as hard as factoring.

3 Problem 3 (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.

4 Problem 4 (10 points)

Consider the following variant of ElGamal. There is a group $\mathbb{G}$ of prime order p, as well as $\ell+1$ generators $g, g_1, \ldots, g_\ell$ that were sampled randomly, and which everyone knows.

The secret key is a string $v \in \{0,1\}^\ell$. Write $v = (v_1, \ldots, v_\ell)$ for bits $v_i \in \{0,1\}$. The public key is $g, g_1, \ldots, g_\ell$, together with $h = g_1^{v_1} \times g_2^{v_2} \times \cdots \times g_\ell^{v_\ell}$. To encrypt a message $b \in 0, 1$, choose a random $r \in \mathbb{Z}_p$, and compute $(g_1^r, g_2^r, \ldots, g_\ell^r, h^r \times g^b)$. To encrypt multiple bit messages, simply encrypt each bit independently.

(a) Explain how to decrypt, given the secret key v.

(b) Given nothing but the public key, show how to devise for each i a "ciphertext" $(c_1, ..., c_\ell, d)$, such that when fed into the decryption procedure from part (a), outputs v_i. Thus, it is possible to devise a "ciphertext" that decrypts to the secret key v, without knowing v

(c) Explain why the ciphertext constructed in (b) is not a ciphertext that would have been produced by the encryption procedure applied to v_i.

Thus, in this scheme, the secret key holder can devise a ciphertext that encrypts the secret key itself, and give it out. This will not hurt security, since the encryption of the secret key could have been generated by anyone. With some more effort, it is possible to have the secret key holder actually encrypt the secret key as specified by the encryption algorithm, and still have security. While it is in general unsafe to encrypt your own secret key, this situation does arise in a few natural settings such as disk encryption. A "circularly secure" encryption scheme like above can be useful in such settings.