Homework 3

1 Problem 1 (20 points)

In the definition of security for identity-based encryption, we allowed the adversary to make it's challenge query on id^*, m_0, m_1 at any point in the experiment. In many cases, we don't know how to directly prove this notion, which is sometimes called "adaptive" security. Instead, what gets proved is "selective" security, where we require the adversary to send id^* at the very beginning of the experiment, even before seeing the master public key (note that by constraining the adversary, this makes the definition weaker). The challenge query is then on a message pair m_0, m_1, and the message m_b is encrypted to the identity id^*.

(a) Show that there are selectively secure IBE schemes that are not adaptively secure. You may assume as a starting point any selectively secure IBE scheme, which may or may not be adaptively secure. Then compile it into a selectively secure IBE scheme which you can demonstrate is not adaptively secure. You will need to prove the selective security of the resulting IBE scheme, and also demonstrate an adaptive attack.

(b) Suppose you have a selectively secure IBE scheme for identity space $\{0,1\}^\lambda$. Suppose the scheme is guaranteed to be *sub-exponentially* secure. We will take sub-exponential security to mean that any polynomial time adversary in the selective security game must have advantage at most $2^{-\lambda^c}$ for some constant $c, 0 < c < 1$.

Construct from this scheme an *adaptively* secure IBE scheme for identity space $\{0,1\}^\lambda$, which is also sub-exponentially secure for the adaptive security experiment (though perhaps with a difference constant c). Hint: since you have sub-exponential security, your reduction can take a large sub-exponential loss of say $2^{-\lambda^{c'}}$ for any $c' < c$, and the resulting advantage will still be negligible.

Remark 1 *The best known attacks on essentially all cryptographic assumptions are at best sub-exponential, so sub-exponential security is usually considered reasonable. However, it is still considered better to have a reduction that does not require sub-exponential hardness. Similar tricks can also be used to boost selective versions of many cryptographic objects to adaptive security, such as attribute-based encryption and functional encryption.*

2 Problem 2 (30 points)

(a) Let $\mathcal{F} = \{f_1, \ldots, f_t\}$ be a polynomial-sized collection of arbitrary efficiently computable functions (that is, $t = \mathsf{poly}(\lambda)$). Show how to build an FE scheme supporting secret keys for functions in $\mathcal{F}$ from any general public key encryption scheme. Prove security under the definition seen in class, though you may assume the adversary commits to the challenge messages m_0, m_1 at the beginning of the experiment, *before* seeing the public key (in other words, you only need to prove selective security). Thus, the interesting case for functional encryption is when the set of functions supported is at least super-polynomial.

(b) Let f be an injective one-way function. Since $f(x)$ hides x, we would hope that a functional encryption scheme still hides the message, even if an adversary gets a secret key for f. To the contrary, devise a functional encryption scheme supporting a secret key for the function f, satisfying the definition of security seen in class, but for which the adversary with no secret keys can learn the message in its entirety. This shows that the definition given in class is not sufficient for certain applications.

(c) Let $F : \mathcal{K} \times [2^\lambda] \to \{0,1\}$ be a pseudorandom function, and consider an FE scheme supporting the class of functions $\mathcal{F} = \{F(\cdot, x) : x \in [2^\lambda]\}$ which have a fixed input x hard-coded, interpret the plaintext as a secret key k for F, and evaluate $F(k, x)$. Here, you will show that, no matter the FE scheme, the adversary can do something with encryptions of k that he cannot do just given $F(k, x)$ for several x. Consider a two-stage adversary $A = (A_0, A_1)$ in the following experiment:

- First A_0 gets the master public key and the secret keys for functions $F(\cdot, x)$ for $x = 1, \ldots, t$, for a parameter t to be chosen later.
- Then A_0 is given an encryption $c = \mathtt{Enc}(\mathsf{mpk}, k)$ for a randomly chosen key k
- A_0 then produces a string $v \in \{0,1\}^{t-1}$.
- A_1 then gets all the secret keys that A_0 received above, as well as v (but not c). It receives no other information (in particular, A_0 cannot pass state to A_1, except through v). It must output $y := (F(k, x))_{x \in [t]}$.

Note that A_0 is certainly able to compute $y \in \{0,1\}^t$. So in essence A_0 is compressing y into a slightly smaller string v.

(i) Show how, for any functional encryption scheme, there exists a sufficiently large polynomial t and adversaries A_0, A_1 that can succeed in the above experiment with probability 1.

(ii) On the other hand, consider adversaries A'_0, A'_1 (for the same scheme and t as above) for the same experiment, except that A'_0 does *not* get the ciphertext c, and instead only gets y and the secret keys. Show that, assuming F is a secure PRF, any efficient A'_0, A'_1 can only win with probability less than, say, 3/4.

In other words, the compression task is possible given an encryption of k, but impossible just given the outputs of the various functions on k.

3 Problem 3 (50 points)

In this problem, we will see how to build an identity-based encryption scheme from assumptions related to factoring. We will first need to define the *Legendre symbol* for integer a and prime p:

$$\left(\frac{a}{p}\right) = \begin{cases} 0 & \text{if } a \equiv 0 \bmod p \\ 1 & \text{if there is some } x \text{ s.t. } x \not\equiv 0 \bmod p \text{ and } x^2 \equiv a \bmod p \\ -1 & \text{otherwise} \end{cases}$$

In other words, the symbol is 1 for non-zero quadratic residues, -1 for non-residues, and 0 if $a \bmod p = 0$. Note that, mod primes, we can determine quadratic reciprocity efficiently, so we can efficiently compute the Legendre symbol.

For non-prime N, we can extend to the *Jacobi symbol* as follows. Let $N = \prod_i p_i^{t_i}$ for primes p_i and integers t_i. Then $\left(\frac{a}{N}\right) = \prod_i \left(\frac{a}{p_i}\right)^{t_i}$. Note that $\left(\frac{a}{N}\right) = 0$ if a, N share a common factor, and $\left(\frac{a}{N}\right) = \pm 1$ if a, N are relatively prime. Also note that, if a is a quadratic residue relatively prime to N, then $\left(\frac{a}{N}\right) = 1$.

(a) Show that the converse is not true. In particular, show that if $N = pq$ for distinct odd prime factors p, q, exactly half of the $a \in \mathbb{Z}_N$ such that $\left(\frac{a}{N}\right) = 1$ are quadratic residues, the other half being non-residues. You may use as given that mod a prime, exactly half of non-zero elements are quadratic residues.

Based on the definition above, computing the Jacobi symbol appears to require factoring N. It turns out, however, that the Jacobi symbol can be computed efficiently. We won't prove this fact, but will use it to construct our scheme:

- The master public key contains $N = pq$ where p, q are secret distinct primes. For each identity id, the master public key also contains a random quadratic residue a_{id}. The master secret key contains p, q.

- The secret key for user id is a square root r_{id} of a_{id}, which can be computed from p, q efficiently.
- To encrypt a message $m \in \{-1, 1\}$ to a user id, choose a random u such that $\left(\frac{u}{N}\right) = m$ (which can be done efficiently by sampling random u until the Jacobi symbol is m; here we use the fact that the Jacobi symbol can be efficiently computed). Output $c = t + a_{\mathsf{id}}/t \bmod N$
- To decrypt c using r_{id}, output

  $$\left(\frac{c + 2r_{\mathsf{id}}}{N}\right) = \left(\frac{t + 2r_{\mathsf{id}} + r_{\mathsf{id}}^2/t}{N}\right) = \left(\frac{t(1 + r_{\mathsf{id}}/t)^2}{N}\right) = \left(\frac{t}{N}\right)\left(\frac{1 + r_{\mathsf{id}}t}{N}\right)^2 = m$$

  Above we used the fact that the Jacobi symbol is multiplicative in the numerator: $\left(\frac{ab}{N}\right) = \left(\frac{a}{N}\right)\left(\frac{b}{N}\right)$.

The problem with the above scheme, of course, is that the master public key must contain all the a_{id}. This limits us to a polynomial number of identities. We will fix this later, but for now we will discuss security. The *quadratic residuosity assumption* is that it is infeasible to distinguish a random quadratic residue from a random *non-residue* a conditioned on having Jacobi symbol 1. This assumption requires that factoring integers be hard, but whether or not it is equivalent is unknown.

(b) Show that, under the quadratic residuosity assumption, encryptions of -1 or 1 to any identity id are indistinguishable. Hint: note that if $c = t + a_{\mathsf{id}}/t \bmod N$, then $c = t_0 + a_{id}/t_0 \bmod N$, where $t_0 = a_{\mathsf{id}}/t \bmod N$. There are two more values t_1, t_2 such that $c = t_1 + a_{id}/t_1 = t_2 + a_{id}/t_2 \bmod N$. What are they? What are the Jacobi symbols of t_0, t_1, t_2 if a_{id} is a quadratic residue? What if we switch to a_{id} being a non-residue with $\left(\frac{a_{\mathsf{id}}}{N}\right) = 1$; what are the Jacobi symbols of t_0, t_1, t_2 now?

We will now fix the fact that we need an exponential number of a_{id} to get an exponential identity space. A first attempt would be to derive $a_{\mathsf{id}} = H(\mathsf{id})$ where H is a well-designed hash function. If we treat the outputs of the has hash function as random independent values, this almost works. The problem is that a_{id} needs to be a quadratic residue, but $H(\mathsf{id})$ may not be. Worse, we cannot even tell whether $H(\mathsf{id})$ is a residue or not, under the quadratic residuosity assumption. We could set $a_{\mathsf{id}} = H(\mathsf{id})^2$ to guarantee a residue, but this will not be secure since anyone can compute a square root of a_{id} by simply computing $H(\mathsf{id})$.

(c) Suggest a way to fix the above problem. The first step for encrypting to a user id will be to run $h_{\mathsf{id}} = H(\mathsf{id})$. Then from h_{id}, you will derive *multiple* different a_{id} values, and encrypt the message relative to *each* of the a_{id}. Explain how the a_{id} should be chosen, and what the secret key should look like. Prove security under the assumption that each of the a_{id} generated are uniformly random elements.