Assignment 4 – Due Wednesday, April 2

1. A "key recovery attack" takes a matching plaintext/ciphertext pair (P, C) and finds a key K such that $E_K(P) = C$. We can formalize this by saying that there is a probabilistic polynomial time algorithm $\mathsf{findkey}(P, C)$ that computes K. Show that if such a function exists, then the encryption scheme is not IND-CPA secure (to do this, define the adversary functions $A_1^{\mathcal{E}}$ and $A_2^{\mathcal{E}}$ for the chosen-plaintext game, where the adversary functions can call both the encryption oracle and the findkey function). In addition to giving the algorithms, analyze the advantage of your adversary algorithm under the assumption that findkey always succeeds.

2. In the handout on security models, it was shown that no stateless, deterministic encryption can be IND-CPA secure. For this problem, consider CBC mode in which the IV is picked in a way that can be predicted by the adversary (but may be stateful and/or non-deterministic).

   (a) Describe CBC mode using algorithm specifications (both encryption and decryption), where the IV is selected during encryption using a function named $\mathsf{getIV}(\lambda)$. You need to give names to both full CBC as well as individual block cipher encryption/decryption functions.

   (b) Assume the adversary has a $\mathsf{predictIV}()$ function that always predicts the next IV that will be returned by $\mathsf{getIV}(\lambda)$ (and hence the next IV that will be used in an encryption). Define adversary algorithms $A_1^{\mathcal{E}}$ and $A_2^{\mathcal{E}}$ that can win the CPA game. (Hint: Think about what is fed into the block cipher as plaintext. Can you arrange it so that you can do multiple encryptions in which the same values are fed into the block cipher?) Do a quick analysis of your algorithms to find the advantage of your adversary.

   (c) What if the predictIV algorithm isn't perfect? In other words, what if predictIV only predicts the correct IV with probability p? What is the advantage now?

   (d) CBC-Chain mode is a variant of CBC mode in which the encryption function keeps a record of the last ciphertext block produced, and uses that as the IV for the next encryption (so a series of CBC-Chain encryptions is the same as a single long CBC encryption). Show that CBC-Chain mode is not IND-CPA secure. (Note: This attack appears in the real world as part of the basis for the BEAST attack on SSL, discovered in 2011.)

3. It's clear that repeating a deterministic block cipher, as in ECB mode, is not secure. But what about repeating a secure (non-deterministic) cipher? In other words, let $E_K(P) \rightarrow C$ be an IND-CCA encryption scheme, and then define a two-block version of this by $E2_K(P_1, P_2) \rightarrow (C_1, C_2) = (E_K(P_1), E_K(P_2))$. It turns out that this construction is IND-CPA secure, but *not* IND-CCA secure. Prove the second part of that statement (in other words, give an adversary that wins the CCA game against the $E2$ two-block encryption scheme — like almost all CCA attacks, the trick is to disguise your decryption oracle requests so they don't exactly repeat the challenge ciphertext). Make sure you analyze the advantage of your adversary!

4. (*Note: Taking large modular powers is tricky, but modern programming languages have good support for this — for example, in Java you can use the* `modPow` *function in the* `BigInteger` *class, and in Python you can use the built-in* `pow` *function, where* `pow(a,x,n)` *computes* $a^x \bmod n$.) Consider the value $n = 8911$ (this is a composite number with factors 7, 19, and 67).

   (a) Select three different random a values in the range $2, \ldots, 8909$ that are relatively prime to n, and calculate $a^{n-1} \bmod n$ for each of these three a values. Does it seem that n behaves like a prime number as far as Fermat's Little Theorem is concerned?

   (b) Use your random a values from part (a) to run the Miller-Rabin primality-testing algorithm on n, showing each step. If your first a value returns "composite" you can stop with just that one simulation — otherwise, try the other a values until you get "composite."

5. Use Table 8.1 on page 235 to pick two random primes (p and q) in the range $1500, \ldots, 2000$. Compute $n = pq$ and $\phi(n)$. Pick a random a in the range $2, \ldots, n-1$ such that it is relatively prime to n. Compute $b = a^{\phi(n)-1} \bmod n$. Finally, compute the product $a \cdot b \bmod n$. What does this tell you about the relation between a and b? Will this relationship always hold for any values that you pick according to these directions? Justify your answer.

6. What are the primitive roots of 31? (A very simple program can quickly solve this problem.)

7. Understanding the density of prime numbers is important for reasoning about lots of things in cryptography. The basis for this question is the "Prime Number Theorem," which states that the number of prime numbers less than or equal to n (written $\pi(n)$) is approximately equal to $\frac{x}{\ln x}$.

   (a) Estimate the number of 512-bit prime numbers (here, "512-bit" means with no leading zeroes, so a number x is 512 bits if $2^{511} \leq x < 2^{512}$).

   (b) The RSA cryptosystem uses a modulus that is the product of two large prime numbers, and if you could factor the modulus you could break the system. Sometimes when people hear this, they think that if the modulus is restricted to be the product of 512-bit prime numbers then they can make a table of all 512-bit prime numbers and use that table to factor the modulus. Is that reasonable? How big would such a table be? What is the size of the largest hard drive that you can buy today? See if you can estimate the total amount of disk storage that has ever been manufactured in the history of mankind. How does that compare to the size required for such a table?

   (c) When picking a random prime number, a typical algorithm will pick a random odd integer, use the Miller-Rabin primality testing algorithm to see if it's prime, and if it's not prime start over by picking a different random odd integer. How many iterations would you expect this to take if you wanted to pick a random 512-bit prime number? *Hint*: This is the scenario that is sketched at the end of Section 8.3, but with a different size (200 bits rather than 512 bits). For this part, adjust the reasoning to 512 bits, and be a little more formal and careful in the analysis, using your result from part (a).

   (d) Picking a random *odd* number to start is really just a way of saying we are excluding multiples of 2. We can quickly test and adjust a random number so that it is not a multiple of other small primes as well. Redo part (c) where when we pick a random number we ensure that it is not a multiple of 2, 3, 5, 7, 11, or 13 before running Miller-Rabin. How many iterations (in other words, how many Miller-Rabin tests) are expected now?