CS255: Cryptography and Computer Security **Winter 2007**

Final Exam

Instructions
– Answer **four** of the following five problems. Do not answer more than four.
– All questions are weighted equally.
– The exam is open book and open notes. A calculator is fine, but a laptop is not.
– You have two hours.

Problem 1. General questions.

a. Briefly describe two differences between CBC mode encryption and counter mode encryption.

b. Why use a random IV in CBC mode encryption? What goes wrong if one uses a fixed IV?

c. When implementing CBC-MAC, should one use a fixed IV or a random IV? Briefly explain why.

d. Suppose that when implementing encrypt-then-MAC using CBC mode encryption, the implementor forgot to include the IV in the MAC calculation on the ciphertext. Briefly explain why the resulting system does not provide integrity (in particular, show that the system does not provide ciphertext integrity).

e. Explain how given an RSA public key (N, e) and the factorization of N, one can derive the secret key d.

Problem 2. Hash functions. Let $X = \{0,1\}^m$ and $Y = \{0,1\}^n$ where m is much larger than n.

a. Explain what it means for a hash function $H : X \to Y$ to be one-way. Give an example application where one-way hash functions are used. How will your example application break if the function used in not one-way?

b. Explain what it means for a hash function $H : X \to Y$ to be collision resistant. Give an example application where collision resistant hash functions are used. How will your example application break if the function used in not collision resistant?

c. Suppose $H : X \to Y$ is collision resistant. Is H also one-way? If so, explain why. If not, give an example of a (presumed) collision resistant function that is not one-way.

d. Suppose $H : X \to Y$ is one-way. Is H also collision resistant? If so, explain why. If not, give an example of a (presumed) one-way function that is not collision resistant.

Problem 3. Distributing watermarked content. Suppose a movie is divided into n scenes (e.g. $n = 1024$) and there are two versions of each scene, say shot from slightly different angles. These scenes are denoted by $(L_1, R_1), \ldots, (L_n, R_n)$ where L_i is scene i shot from the left and R_i is scene i shot from the right, for $i = 1, \ldots, n$.

A movie distributor wishes to enforce the following policy: when a customer u buys the movie she should obtain a version of the movie where her name and address are embedded in

the movie. For example, if the customer's name u is represented as the n-bit binary string $u = 1011010\ldots \in \{0,1\}^n$ then the customer should obtain the movie $(R_1, L_2, R_3, R_4, L_5, R_6, L_7, \ldots)$. We denote this watermarked movie by M_u. If a pirated version of the movie appears on the black market, the distributor can extract a name from the pirated copy.

The distributor wishes to distribute one large bulk data B to all customers (by stamping a million identical DVDs or by placing the bulk data on BitTorrent). Once a customer u obtains the bulk data – which is distributed for free – she contacts the distributor and buys a decryption key k_u. The key k_u enables the customer u to decrypt B and obtain M_u and nothing else. We write $D(k_u, B) = M_u$. Another customer, v, will receive a different key k_v so that $D(k_v, B) = M_v$.

a. Describe an encryption mechanism supporting these requirements that enables the distributor to publish bulk data B that is only twice as large at the movie. A customer's decryption key k_u in your system will consist of n AES keys.

b. Show that one can reduce the size of the customer's key to $n/4$ AES keys at the cost of adding additional $4n$ short ciphertexts to the bulk data B.

c. Show that two users who collude can create a version of the movie that contains the name of neither of them.

Problem 4. Questions about RSA

a. Let p be a prime with $p = 2 \bmod 3$. Show that given $\alpha \in \mathbb{Z}_p^*$ one can efficiently find the cube root of α modulo p. That is, one can solve the equation $x^3 - \alpha = 0 \bmod p$.
Hint: recall how RSA decryption works.

b. Is your algorithm from part A able to compute cube roots modulo a composite $N = pq$ when the factorization of N is unknown? If so, prove it. If not, describe an alternate algorithm or explain why you believe no such efficient algorithm exists.

c. Some proposals suggest making the RSA modulus a product of three primes $N = pqr$ of equal size. Describe the RSA system in this case. That is, explain how e and d are chosen.

d. Explain why using products of three primes may be a good idea. Think of the running time of the decryption process when the Chinese Remainder Theorem (CRT) is used. Compare the decryption time when $N = pq$ is used and when $N' = p'q'r'$ is used. Assume both N and N' are 1024 bits long.
Hint: recall that to compute $M^d \bmod p$ it suffices to compute $M^{d \bmod p-1} (\bmod p)$ resulting in a potentially much smaller exponent.

Problem 5. One time password authentication.

a. Explain how the S/key one time password system works using a one-way hash function H. You may assume that the user sets things up so that he can login n times using the hash chain (say $n = 1024$).

b. Suppose the user only stores the base of the hash chain (namely, the first element in the chain). After n logins, how many times did the user have to evaluate the hash function H? How many times did the server evaluate the hash function H?

c. Suppose that in addition to the base of the hash chain h_0, the user also stores the midpoint, namely $h_{n/2} = H^{n/2}(h_0)$ where $H^{n/2}(h_0)$ refers to $n/2$ repeated applications of H. Explain why this reduces the user's total number of hash evaluations after n logins by about a factor of 2.

d1. Generalize part (c) — show that by storing $\log_2 n$ points along the chain the user can reduce the total number of hashes after n logins to $O(n)$. Hence, the user only does a constant number of hashes on average per login.

d2. Alternatively, show that by storing the base point plus one more point (i.e. the total storage is as in part (c)) the user can, in fact, reduce the total number of hashes after n logins to $O(n^{3/2})$. Hence, the user does $O(\sqrt{n})$ hashes on average per login by storing only two values.

note: Please either answer part (d1) or (d2), but not both.