Massachusetts Institute of Technology
6.857: Network and Computer Security
Professor Ronald L. Rivest

Handout 4
March 2, 2009
Due: March 16, 2009

Problem Set 3

This problem set is due *via email,* to `6857-hw@mit.edu` on *Monday, March 16* by the beginning of class.

You are to work on this problem set in groups of three or four people. You should have received an email with your group assignment for this problem set. If not, please email `6.857-tas@mit.edu`. Be sure that all group members can explain the solutions. See Handout 1 (*Course Information*) for our policy on collaboration.

Homework must be submitted electronically! Each problem answer must appear on a separate page. Mark the top of each page with your group member names, the course number (6.857), the problem set number and question, and the date. We have provided templates for LaTeX and Microsoft Word on the course website (see the *Resources* page).

Grading and Late Policy: Each problem is worth 3 points. Late homework will not be accepted without prior approval.

With the authors' permission, we will distribute our favorite solution to each problem as the "official" solution—this is your chance to become famous! If you do not wish for your homework to be used as an official solution, or if you wish that it only be used anonymously, please note this on your homework.

Problem 3-1. Side-Channel Attacks on AES

In lecture, we saw how an attacker can use the CPU cache to determine which S-boxes are used during an AES encryption operation. A more sophisticated attacker can use a cache-contention attack to determine the number of times each S-box was used in an encryption. The attacker can use these S-box counts to determine the AES secret key.

We have set up a server to simulate this type of side-channel attack. Our server encrypts random strings using AES with a secret key k. The server also "leaks" the number of times every S-box was used during each encryption. If you query `http://courses.csail.mit.edu/6.857/2009/aes.php` you will receive 100 (plaintext, ciphertext, S-box counts) triples. The plaintext and ciphertext are printed as space-separated byte values. The 256 S-box counts are also space-separated and arranged in ascending order (i.e., the first number is the number of times entry 0 in the S-box table was used). The plaintexts are chosen randomly, so you can query the server multiple times to retrieve additional data.

(a) Recover the secret key k. Describe the algorithm that you used.

(b) How many (plaintext, ciphertext, S-box counts) triples does your attack require? Can you think of some ways to reduce the number of triples required?

(c) What assumptions does your algorithm make about the plaintexts being encrypted? Are there situations where these assumptions are invalid?

Problem 3-2. Time/Memory Trade-offs and Generic Attacks

An adversary with 1024 computers would like to find preimages in a hash function with a 64-bit digest using a generic preimage attack. The adversary has 128 message digests to invert, and the adversary wants the online portion of the attack to run as quickly as possible while inverting each digest with high probability. The hash function is known not to be a single-cycle permutation. Each computer has 64 GB of available storage space. Assume that storage operations and hash function computations require 1 microsecond each.

(a) Explain how to structure the offline portion of the Hellman Time/Memory tradeoff attack. How many different tables should the adversary create? How many random messages should the adversary store in each table, and what should the chain length be? What are the time and space requirements of this step? You may choose to formulate the attack with or without distinguished points.

(b) How long does the online step of the algorithm take to compute the preimages of all (or at least most) of the 128 message digests?

Problem 3-3. Meet-in-the-Middle Attacks and Collision Finding

Recall the "double encryption" scheme from the second question of problem set 2. In this scheme, we defined $E(K, K', M) = \text{AES}(K', \text{AES}(K, M))$. In problem set 2, you (presumably) used a meet-in-the-middle attack to determine K and K'. In this problem, you will use a collision-finding algorithm to determine K and K'.

In order to convert this problem into a collision-finding problem, we must first define a function f in which to find a collision. A meet-in-the-middle attack effectively finds a "collision" in the intermediate value of the double encryption. We would like to define a function which captures this idea, i.e. some function f s.t. $f(K) = f(K')$ implies $AES_K(P) = AES^{-1}_{K'}(C)$, given a plaintext-ciphertext pair (P, C). Unfortunately we cannot define f to be both the encryption and decryption routines simultaneously, so we must introduce another parameter.

(a) Define a two-parameter function f with a domain of K, b and a range of K, b, where K is an AES key and b is a single bit. Define f such that $f(K, b) = f(K', b')$ and $b \neq b'$ implies $AES_K(P) = AES^{-1}_{K'}(C)$. Remember that to use the collision-finding algorithms discussed in lecture, f's output must be "sufficiently random".

(b) How can you use f to reduce key-recovery to collision finding with some extra conditions? What is the attack's success probability?

(c) What is the time and space complexity of this attack in terms of n, the bit length of K and K'? How does this compare to complexity of the meet-in-the-middle attack from problem set 2?