YALE UNIVERSITY
DEPARTMENT OF COMPUTER SCIENCE

CPSC 467: Cryptography and Computer Security *Handout #6*
Professor M. J. Fischer *September 23, 2013*

Problem Set 3

Due: Wednesday, October 2, 2013

1 Goal

The goal of this problem is to explore the method of brute force attack on a cryptosystem to get a better feeling for what it can and cannot do.

2 SnakeOil

This problem set refers to Happy Hacker's SnakeOil cryptosystem described in Problem Set 2 ((.pdf).

Happy boasted that he only had to remember the pair of key indices since everything else was stored on his hard disk. Unfortunately, this was too much for Happy. He forgot the correct pair of key indices and so was unable to decrypt a message he received from Alice.

Clever Charlie told him not to worry. His system was easily compromised by anyone with access to the key share file. A program could be written to brute force the possible key pairs, and the computer could be decrypting Alice's message while they went out for pizza.

Sure enough, when they returned, Alice's message was displayed on the screen.

3 A Brute Force Attack on SnakeOil

For Happy's problem, a brute force attack decrypts Alice's ciphertext c for every possible key index pair (k_i, k_j) $(0 \leq i < j < 100)$ in Happy's key file until the correct pair is found. The principal difficulty is recognizing the correct decryption once it has been found. How can one distinguish the correct decryption from the wrong one?

In general one cannot, but if some messages are more likely than others, then some decryptions of c will "look more like a valid message" than others. The attack chooses the decryption m' of c that looks most like a valid message and guesses that it is the real message m. It also guesses that the key index pair $k' = (k_i, k_j)$ that produced it is the real pair.

The guess m' will not always be correct. If it is, we say the attack *succeeds in breaking the cryptosystem*. If in addition k' is correct, we say that the attack *totally breaks* the cryptosystem. Otherwise, the attack *fails*. We are interested in exploring how well this attack can be made to work in practice.

3.1 Letter Frequencies

Clever's method for choosing m' relies on the letter frequencies of English text. Assume you have a large corpus of representative English text, so that Alice's message is likely to have a similar letter frequency distribution. Assume further that a decryption of c using a randomly chosen incorrect index pair k' yields a random-looking string m' with more or less uniform distribution of the letters. If c is sufficiently long, then very likely the correct decryption m is the only one whose letter frequency distribution closely resembles that of the corpus.

3.2 Measuring Similarity

For each octet (8-bit byte) b, let $f(b)$ be the frequency of occurrence of b in a corpus of English text, and let $r = \sum_b f(b)$ be the total size of the corpus. Then the normalized frequency $p(b) = f(b)/r$ is a close estimate of the probability that an arbitrary byte in a random piece of text is equal to b. Similarly, let $q(b)$ be the normalized frequency of b in the message.

We measure the (dis)similarity of p and q by the sum of the squares of the difference at each byte of their normalized frequencies. That is, we define

$$\text{divergence}(p, q) = \sum_b (p(b) - q(b))^2.$$

We apply this measure by computing the divergence between each possible decryption of c and the corpus and choosing the decryption (and corresponding key) that minimizes the divergence.

4 Assignment

Please solve the two problems below.

You will find a C++ implementation of SnakeOil on the Zoo in directory `/c/cs467/assignments/ps3/src/`. You can compile a `snakeoil` executable by copying the files to your own directory and typing `make snakeoil`.

You will also find a partially-written brute force key analyzer, complete with routines for reading and normalizing frequency tables. However, it is missing two key functions: `guessKey()` and `divergence()`, so it will give compiler errors if you try to build it.

Problem 1: Coding

You should write the two missing functions in the brute force key analyzer. Compile your program and test it on the file `sample.enc`.

This should not take more than around 50 lines of C++ code, but of course it has to work in the context of the rest of the program, so you will need to spend some time reading my code in order to see how one invokes the AES primitives in order to decrypt a file with a particular master key.

I realize that some of you might not be very familiar with C++, so please feel free to ask the TA or myself for help.

Problem 2: Experiments

You will find several files in the Zoo directory `/c/cs467/assignments/ps3/data/`. Files ending in `.dat` are frequency tables. Files ending in `.enc` are ciphertexts. The file `keyshares` is the key share file that was used in all of the encryptions. All ciphertexts are valid encryptions of English-language text files, but not all are easy to decipher.

You should run your brute force key finder with every pair of frequency table and ciphertext file. For each pair, report the key indices that the program found, the decryption produced, and whether or not the decryption appears to be meaningful text.

Next, you should analyze your results and try to draw conclusions about the effects of the frequency table and the length of the ciphertext on the effectiveness of the attack.

Based on the insights gained, construct a 40 character "message" that cannot be cracked using any of the furnished frequency tables. (Your message does not have to be real English text, but it must consist of printable ASCII characters.)

5 Submission

All work should be submitted in electronic form as described in Problem Set 0 (handout 3) using the script `/c/cs467/bin/submit` on the Zoo. The written solutions should preferably be in the form of a PDF file. The code and data files should be archived as a .tar.gz or a .zip file and submitted. Please be aware that the submit script can only handle files, not whole directory trees.