Homework Assignment 6

Due on Tuesday, October 13, 2020

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

Both parts use the computer. The first part involves some C++ programming. The second part involves running some experiments on your code. You should start now in order to have sufficient time to finish the assignment by the deadline. Remember to cite your sources.

1 SnakeOil

This problem set refers to Happy Hacker's SnakeOil cryptosystem described in Homework Assignment 4 ((.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.

2 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 automatically 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 attacker 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 key pair.

The guessed m' and k' will not always be correct. If m' is correct, 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.

2.1 Letter Frequencies

Clever's method for choosing m' relies on the letter frequencies of English text. Assume Alice's message is in English and you have a large corpus of representative English text. Then the corpus and Alice's message are likely to have a similar letter frequency distributions. 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.

2.2 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. 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 English text is equal to b. Similarly, let $q_m(b)$ be the normalized frequency of b in message m.

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

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

We compute the divergence between p and q_m for each possible decryption m of the ciphertext c. We then choose the decryption m (and corresponding key) that minimizes the divergence.

3 Assignment

3.1 SnakeOil Implementation on the Zoo **(4 points)**

You will find a C++ implementation of SnakeOil on the Zoo in directory `/c/cs467/assignments/hw6/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, two key functions are incomplete: `guessKey()` and `divergence()`, so it will give compiler errors and/or not work correctly if you try to build it.

SnakeOil is written to use the botan-2 crypto library, which is already installed on the Zoo. The supplied `Makefile` contains the compiler switches needed to access the botan-2 includes and library.

You should modify `analyze.cpp` by filling in the missing code for the two incomplete functions. Type `make bruteforce` to compile the modified brute force analyzer, and test it on the file `sample.enc`. It was encrypted using the furnished `keyshares` file and key indices 4 and 31.

No more than 50 lines of C++ code should be needed to complete the brute force analyzer. Of course, you will need to read and understand my code in order to see how your code fits in.

I realize that some of you might not be very familiar with C++, so **start early on this part**, and please feel free to ask the TAs or me for help.

3.2 Experiments

You will find several files in the Zoo directory `/c/cs467/assignments/hw6/data/`. Files ending in `.dat` are corpora of English text from which frequency tables are derived. 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.

3.2.1 Gather Data (4 points)

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, whether or not the decryption appears to be correct, and if it does, the decrypted plaintext itself. You might find it convenient to write a shell script or a python program to automate this part of the assignment.

3.2.2 Analyze Results (2 points)

Analyze your results and draw some conclusions about the effects of the frequency table and the length of the ciphertext on the effectiveness of the attack. Explain.

4 Submission

All work should be submitted as usual via Canvas. The written solutions should be in the form of a PDF file. The code and data files should be put in a `.zip` or `tar.gz` compressed archive file and submitted. **All files necessary to compile and run your code should be included in the archive, including those that were furnished in the assignment directory and not modified.** The idea is that the grader should be able to unpack your archive on the Zoo, type `make`, and your code should compile and link without errors and produce executable files `snakeoil` and `bruteforce`. Please do *not* submit generated `.o`, executable files, or hidden files that your IDE might generate. The grader will recompile them from your source files.