Problem Set 1

Due by midnight on Wednesday, September 24, 2008.

1 Goal

The goal of this problem is to show the feasibility of fully automating a brute-force attack on the Caeser cipher. A brute force attack on a cryptosystem means trying all possible keys to see which one "unlocks" the encrypted message.

The difficulty in automating such an attack, aside from the time it takes to explore a potentially large key space, is knowing when you have succeeded. How can one distinguish the correct decryption from the wrong one? In general one cannot, but if some messages are more likely than others to be the decryption of a given ciphertext string c, then it is sensible to guess the message m' that is the most likely among the possibilities. The guess m' might or might not be equal to the original message m. If it is, we say the attack *succeeds*; otherwise it *fails*.

2 Assignment

The assignment consists of two parts:

1. To write a C or C++ program to crack a Caeser cipher.

2. To experimentally evaluate the efficacy of your program, i.e., how likely is it to succeed?

3 Automatic cryptanalysis of the Caeser cipher

For this problem, we assume a fixed known probability distribution P on single letters, so for example, $P(\text{'A'})$ is the probability of choosing letter 'A'. Messages are of a fixed length r and are composed of letters chosen independently at random according to P. Thus, if $r = 3$,

$$\text{prob}[m = \text{'CAT'}] = P(\text{'C'}) \cdot P(\text{'A'}) \cdot P(\text{'T'}).$$

Given a ciphertext $c = E_k(m) = (k + m) \bmod 26$, the possible decryptions are $D_0(m)$, ..., $D_{25}(m)$ corresponding to each of the 26 possible keys $0, \ldots, 25$. Each of these messages has a certain a priori probability as defined above. Your program should guess the most likely message (and corresponding key), i.e., the one with the highest probability. Ties are broken by choosing the message corresponding to the smaller key.

An *experiment* takes two inputs: A probability distribution P on single letters and a length r. The steps of conducting an experiment are:

1. Choose a random message m of length r according to the distribution on length-r strings induced by P.

2. Choose a key k uniformly at random from $\{0, \ldots, 25\}$.

3. Compute $c = E_k(m)$, where E is the Caeser encryption function on length-r strings.
4. Run the procedure `cryptanalyze`(P, r, c), where "cryptanalyze" is the analysis procedure described above to guess a message m' and corresponding key k'.
5. If $m' = m$, output "success" and the pair (m', k'). Otherwise, output "failure".

Finally, you should gather data by running a bunch of experiments. For each $r = 1, 2, \ldots$ up to some reasonable number (say 100), run a large number of experiments (say 100) and compute the fraction of successes. Repeat this several times to get a feeling for the variance in your results. Plot the results and find numbers r_1 and r_2 (if possible) such that the observed success rate is less than 10% for $r < r_1$, between 10% and 90% for $r_1 \leq r \leq r_2$, and greater than 90% for $r > r_2$.

4 Programming and submission details

You will find some files on the Zoo in /c/cs467/assignments/ps1 that describe probability distributions on which to test your code. Each such file contains 26 whitespace-delimited integers, where the i^{th} integer gives the frequency of occurrence of the i^{th} letter in the alphabet. To convert these frequencies to probabilities, you will need to normalize by dividing each frequency by the sum of the frequencies. See the Readme file for detailed descriptions of how these distributions were obtained.

Beyond this description, you are free to organize your code any way you see fit. Keep in mind that your code has multiple purposes. Beside showing me (and the TA) that you know how to program in C/C++, the purpose is to get an understanding through experimental means of how well a simple cryptanalysis tool can do under controlled circumstances and to see how its efficacy depends on the assumed underlying probability distribution on the message space and on the length of the message. Designing experiments and presenting experimental results in a compelling way is a useful skill to acquire, quite independent of the particular topics of cryptography and security on which this course focuses.

One caveat: Experimental results are worthless if the experimental code is flawed, so you also need to make a convincing case that your code is correct, i.e., correctly implements the scheme described by the problem assignment. Programs that generate random numbers can often be hard to debug because runs may not be repeatable. To make them repeatable, you should explicitly seed your random number generator during debugging. Of course, for production runs where you are performing 100 trials, each trial will need its own random seed. The result of `time(NULL)` can be used for this purpose since it will necessarily be different on each run. There are many random number generators available. For this assignment, you should use the standard `rand()` and `srand()` functions.

If you write your program in the straightforward way, you may run into floating point underflow problems when computing the probability of long messages. The easiest solution to this problem is to compute the logarithm of the probability rather than the probability itself. This works because you are not really interested in the actual value of the probability but only in determining which of the probabilities is the largest. Since the logarithm function is strictly monotonic increasing, the largest probability will also have the largest logarithm. To compute the logarithm of a product pq, use the formula $\log(pq) = \log(p) + \log(q)$. The base of the logarithms you use does not matter as long as you use the same base for all of your calculations.

When your program is working, you should gather data by running the experiments described above on the designated probability distributions. The results of your experiments should be presented in both tabular and graphical form.

You should submit your work using the submission script in the `/c/cs467/bin` course directory on the Zoo. Your submission should include the program or programs you have written, test runs showing the correctness of the various pieces, and the data tables and graphs resulting from your experiments. You should also submit written documentation describing in some detail what you have done.