CS641 - Modern CryptoGraphy
Assignment 3

Shashvat Singham
(200922)

On entering the sequence of commands: Enter and give we found a mouse trapped inside a hole. The mouse gave a password which when entered in the home screen of the level opened the invisible message.

The cipher-text consisted of alphabets of the English language. We did a frequency analysis and found that the frequency of the letters in the cipher-text is quite similar to the frequency of letters in the English language. Since substitution cipher was already done as part of level one, we tried to break the cipher assuming it to be substitution-permutation cipher as this cipher also preserves the frequency of the alphabets. We briefly describe the method of breaking substitution cipher we used in level 1. We extended this algorithm to break the substitution-permutation cipher of this level.

1 Substitution Cipher

We start with an initial mapping of characters. The initial mapping is generated using frequency analysis of the code. We find the most frequent 3-letter word and map it to 'the'. For example, if 'xyz' is the most frequent word 'x', 'y' and 'z' are mapped to 't', 'h' and 'e' respectively. We then use any partial information we have about the plain-text (described later) to guess some more mappings. The rest of the mapping is generated randomly.

We define quadgram score of a string as the sum of score of each quadgrams in the string. Example, "abcde" has two quadgrams "abcd" and "bcde". The score of a quadgram is proportional to its frequency in the language. We took the quadgram score file from here. The file was generated by considering large volume of English text replacing all non-alphabetic characters including whitespace and counting the number of times, each quadgram occur. For each quadgram we used to file from above link to calculate quadgram score as:

$$Score\ of\ a\ quadgram\ =\ log(\frac{Number\ of\ times\ this\ quadgram\ occurs}{Total\ Number\ of\ times\ that\ all\ quadgram\ occurs})$$

We took log to avoid numerical overflows as the numbers were quite large.

Using the quadgram score above, the initial mapping is improved. At each step, we take two elements from the mapping and swap them. Note that while selecting elements to swap randomly, we leave the mappings that were generated during frequency analysis. If the decryption from the new mapping has better score we keep it, else we revert to the previous mapping. Our algorithm may be stuck and loop indefinitely. To prevent this we introduce timeouts. Whenever the algorithm times out it aborts the current search, generates new initial mapping as described above and restarts. To determine if a correct decryption has been reached, we use dictionary matching.

Our algorithm is an instance of Hill-Climbing, a heuristic search algorithm for optimizations, where we used quadgram score as heuristic for improvement and dictionary matching as terminating condition.

2 Permutation Cipher

The cipher-text after removing all non-alphabetic characters was 280 characters long. We decided to not permute non-alphabetic characters as they appeared to be in correct positions. Thus, the possible block sizes for the permutation is 1, 2, 4, 5, 7 ... We started with smaller block sizes and eventually increased block size. For block size 5, we found that a permutation followed by substitution resulted in proper decryption. For a block size of n, there can be n! different permutation. We used the partial information we had (described below) to decrease the space of all possibilities.

3 Partial Information

Based on previous levels, we guessed that the plaintext must have the word "password". Looking at the structure of ciphertext it was clear that the word just before colon was the encrypted version of password. For a permutation block size of 5 we used this information as follows:

There were 10 characters after semicolon. Thus, all the characters of "password" must be in 10 characters just before colon. These 10 characters were: "cd bsldorah" (non-alphabets are not counted). Now if the 8-character word after space is "password" then after permutation its third and fourth characters must be the same. In these 10 characters only the letter 'd' is repeated twice. These 10 characters are made of the following 5 character blocks: "cd bsl" and "dorah". Now if both the 'd's have to come together then the second character of first block i.e. 'd' must be moved to 5th position and the 1st character of 2nd block must remain at the same position. Thus, we have determined two elements of permutation i.e. 2nd element of each 5-character block is moved to 5th position and 1st character of each block remains at its place. Thus, the number of possible permutations reduced from 5! (=120) to just 6.

Using the above fact we can also guess a good chunk of substitution key. Consider one of the 6 possible permutations: [1, 5, 2, 3, 4] i.e. 1st, 2nd, 3rd, 4th and 5th elements of a block are moved to 1st, 5th, 2nd, 3rd and 4th positions. Under this permutation key, the permutation of "cd bsldorah" is "cb slddhora". Now if this is correct permutation then "slddhora" must be substitution encryption of "password". Using this we can guess 7 substitution mapping initially. Then we can use the improvement algorithm to try decrypting this substitution cipher. If a proper decryption is found, this means that the guessed permutation was right other we try the remaining of the 6 permutations.

4 Concluding Notes

- We provide the code implementation of the described algorithm in the Submission. One of the file quadgrams.txt, required by our code could not be provided with the package due to Moodle's size constraint. Please download the file from here, rename it to quadgrams.txt and place it in same folder to run the code.
- We found that the letters 'x', 'z' and 'q' were present only once or twice in plaintext. Further 'x' and 'z' appeared only inside password which was not even proper English word. Thus, we had to try replacing 'x' and 'z' to get to actual password.

5 Code

The submission package has the code we wrote.

decrypt.py : Main Code. Usage: python3 decrypt.py sub_perm_decrypt.py: Has the code to solve substitution cipher. Used by decrypt.py Result: Sample Output of the Code

All the codes are well commented and implements the algorithms described above.