Assignment 1—version 0.0

Date of Submission - **17th February, Sunday**

1 Introduction

This assignment is about cryptography. In particular, it is about an encoding technique called *monoalphabetic substitution*. However, the more interesting part is the decoding of an encoded message, which is what you have to implement in this assignment.

The idea behind monoalphabetic substitutions is easy. We are given a message in the English language (which we shall call *plaintext* from now) to be encoded. The basis of encryption is called a *key*, and the key is a secret that is shared between the sender of the message and the receiver. In the case of monoalphabetic substitution, the key is a fixed permutation of the alphabet. However, remembering a permutation of 26 letters without writing it down is difficult, and anything written may fall into the wrong hands. Therefore we use an agreed upon word, called a *secret word*, to generate a permutation. As an example, the secret word could be the word "WISDOM". For the purposes of this assignment, the secret word will not have any repeated letter. Based on the secret word, the permutation is generated as follows: The first six letters of the alphabet (A-F) map to w, i, s, d, o and m respectively. The letter after F, namely G, maps to n, the letter after m. H maps to p, since m is already used up, and so on.

To avoid cluttering this writeup, I shall present all the examples in the file `example.txt`. The key is shown in this file. However, since we know the order of letters in the plaintext alphabet, we can simply enumerate the ciphertext latters, and the key can be recreated. In this case we shall call this representation of the key as *compact key*. Note that plaintext will be written in capital letters and the encoded message, also called *ciphertext* will be denoted by small letters. The plaintext in this example, and, given the key, the ciphertext for this example are shown in `example.txt`.

Given a ciphertext, the problem in this assignment is to decode it using a set of strategies (more appropriately, heuristics[1]) that we shall describe below. The top level function of our method, denoted as `crack-cipher`, can be represented by Figure 1. At the end of each strategy, we either discover the secret word from which we can recreate the complete key. In this case our method is considered to be successful. Else we try the next strategy with the key returned by the previous strategy. If we exhaust all the strategies without producing the secret key, we report failure.

[1] A heuristic is a technique designed for solving a problem more quickly when classic methods are too slow, or for finding an approximate solution when classic methods fail to find any exact solution.

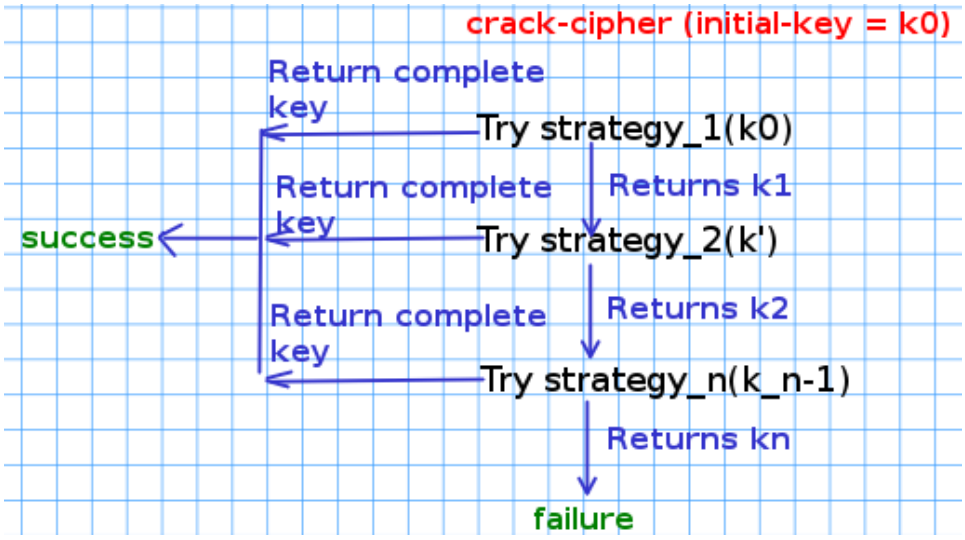

Figure 1: The top level workflow.

2 Strategies

We shall consider several strategies, and each strategy will be described by a function that takes as input: the partially completed key returned by the previous strategy and produces as output either the complete key or a (possibly) modified key. To do this, the strategy first generates a list of *substitutions*. Each substitution is a partial map from plaintext alphabet to ciphertext alphabet. These are the mappings discovered by this strategy. An example of a substitution is `((E . e) (T . o) (A . q) (I . w))`.

To get a sense of how the substitutions are discovered, examine the inputs and outputs of the strategy called `etai` which tries to discover the mappings of letters E, T, A and I. The output contains several possible mappings for this group of letters. We generate several mappings since the reasoning is based on probabilities and we are not certain about the actual mapping. `example.txt` shows the result of applying the strategy `etai`. The result is a list of substitutions. We then pick each substitution in turn and perform an operation called DC*+SWE (dictionary closure and secret word enumeration or dictionary closure for short) that we describe below. There are three possible outcomes of performing DC*+SWE for a substitution.

1. It might result in the secret word being discovered, in which case the whole process ends with the complete key.

2. Or it may result in the modification of the key. *It is important to note that when we modify the key, we ensure that if the modified key is used to partially decode the ciphertext, each word in the ciphertext has at least one possible completion.* In this case, we do not try the rest of the substitutions, and move over to the next strategy.

3. However, if DC*+SWE on the current substitution does not result in a modification of the

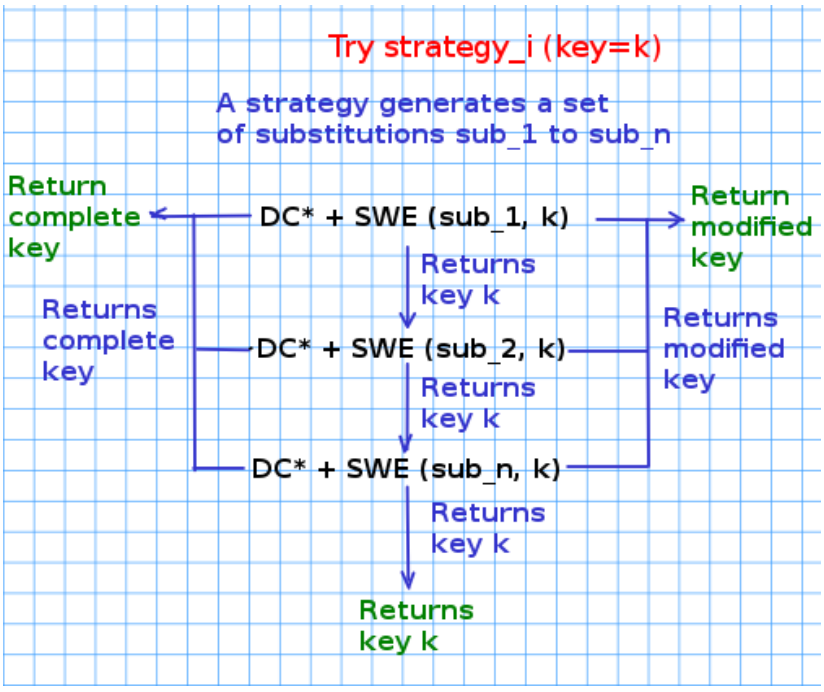

Figure 2: Internals of a strategy.

current key, we simply move over to the next substitution.

2.1 Strategy for discovering E, T, A and I

As an example, let us discuss the strategy for discovering the letters E, T, A and I. We call this strategy `etai` describe how to implement it..

In the English language, the most frequently appearing letters are, in order, E, T, A, O, I, N, S, H, R, D, L, U, ... However, this is the frequency order *on the average*, and this order holds with high probability for texts of large size. For shorter texts consisting of a few paragraphs, we assume (note: this is an assumption) that E, T and A and I will be amongst the top five most frequently occurring letters in a text. This is true of the example that we are currently working with in which the five most frequent letters and their mapping are (T, e), (E, o), (I, q), (O, y) and (A, w). A possible sequence of steps of the strategy are:

1. Find the top five frequently occurring letters in the ciphertext. In the current example, they are e, o, q, y, w.
2. *Identifying A and I:* Separate the single lettered words in the text. These are w and q. We can assume that either A maps to w and I to q or vice versa.
3. *Identifying T:* A way of separating the consonant T from the vowels E and A. Note that each vowel can have a large number of alphabets as their neighbours. The table in `example1.txt`

shows what are the letters that each of the cipher-letters e, o, q, y, w has as its neighbours. From the table, we can conclude that the (relatively) unfriendly alphabet e is the consonant T.

4. *Identifying E:* Thus E is one of o and y. Consulting the frequency of cipher-monograms in `example.txt`, we can see that o has a higher frequency than y. Instead of rushing to the conclusion that E maps to o, we can leave both possibilities open.

2.2 Other strategies

Just like `etai`, here are some other strategies that you could possibly use,

1. *common-words-double:* This is a step that identifies two lettered words based on their frequency of occurrence. The common two lettered words in order of frequency are: OF, TO, IN, IT, IS, BE, AS, AT, SO, WE, HE, BY, OR, ON, DO, IF, ME, MY, UP, AN, GO, NO, US, AM ....

2. *common-words-triple:* Similar. The list of three lettered words in frequency order are: THE, AND, FOR, ARE, BUT, NOT, YOU, ALL, ANY, CAN, HAD, HER, WAS, ONE, OUR, OUT, DAY, GET, HAS, HIM, HIS, HOW, MAN, NEW, NOW, OLD, SEE, TWO, WAY, WHO, BOY, DID, ITS, LET, PUT, SAY, SHE, TOO, USE ...

3. *bigrams:* An *n-gram* is a contiguous sequence of n characters which can occur somewhere within a word. We shall provide tables of the most frequently occurring n-grams for n= 1,2,3 and 4. Thus `etai` uses a *1-gram* also called a *monogram*.

   We can also identify letters through the use of *2-grams* or *bigrams*. The most frequently occurring bigram in the ciphertext can be taken to be the encoding of the most frequent bigrams in the bigram list. The bigram list starts with: TH, ER, ON, AN, RE, HE, IN, ED, ND, HA, AT, EN, ES, OF, OR, NT, EA, TI, TO, IT, ST, IO, LE, IS, OU, AR, AS, DE, RT, VE ...

4. *trigrams:* Similar. The trigram list is: THE, AND, THA, ENT, ION, TIO, FOR, NDE, HAS, NCE, EDT, TIS, OFT, STH, MEN ...

5. *common-initial-letters:* Not all letters are equally likely to be the first letter of a word. This is the order of frequency with which a word is likely to begin a word: T, O, A, W, B, C, D, S, M, R, H, I, Y, E, G, L, N, P, U, J, K...

6. *common-final-letters:* The list is: E, S, T, D, N, R, Y, F, L, O, G, H, A, K, M, P, U, W ...

7. *common-double-letters:* Finally, only some letters can appear together in words. The frequency order for such letters are: SS, EE, TT, FF, LL, MM, OO...

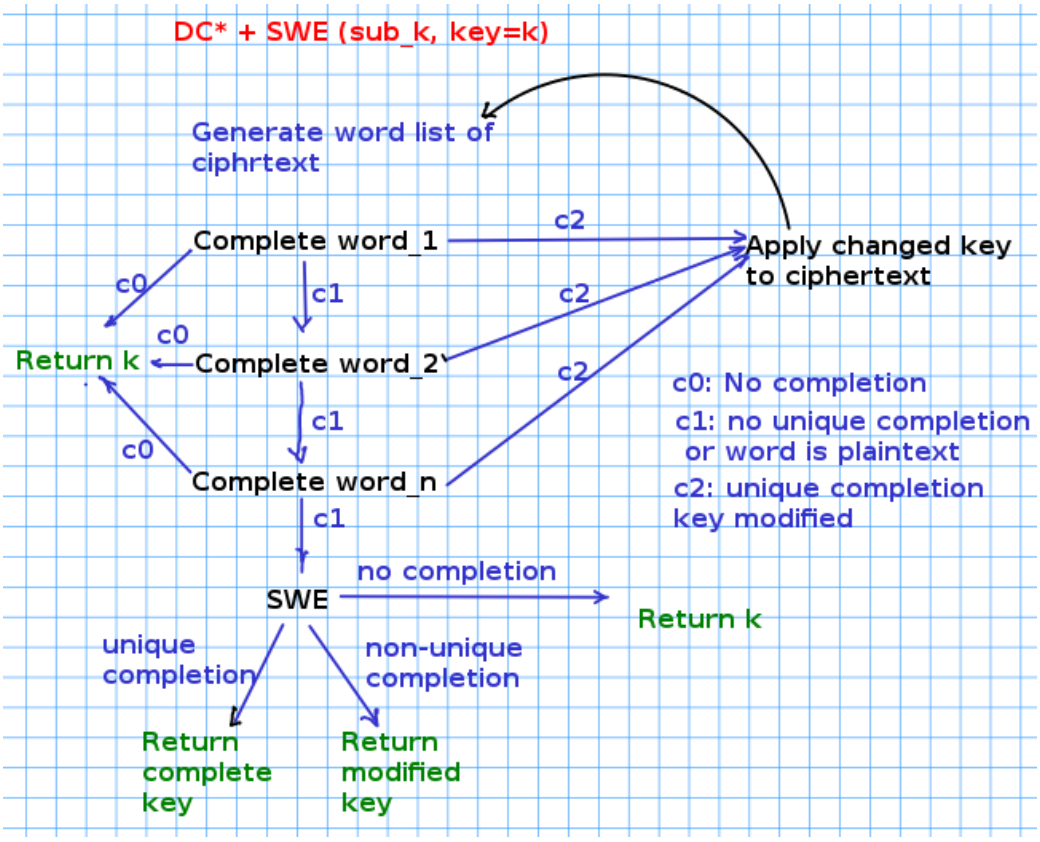

Figure 3: Internals of a strategy.

3 Dictionary closure (DC) and secret word estimation(SWE)

Dictionary closure is an amazingly powerful operation. The idea behind this is as follows. Suppose we currently have a key the current strategy *strat* has resulted in a set of substitutions *subs* of which we are examining the substitution sub_k. Using sub_k we modify the current-key to obtain a new key. The new key can be applied on the ciphertext to obtain a partially decoded ciphertext. Now we can examine the first word of the partially decoded ciphertext, and using a English dictionary attempt to complete it. There are three possibilities.

1. *The word can be completed in a unique way:* The completion results in a further substitution. We change the ciphertext on the basis of of this substitution and start once again from the first word.

2. *The word can be completed, but completion is not unique:* We simply step over to the next word with the current key.

3. *The word cannot be completed:* The substituion sub_k was wrong in the first place. Exit from this substitution with the original key that we started with. that

If we have finished the complete list of words without coming out in step 3, then we examine whether the secret word can be uniquely completed (the SWE step). Unique completion signifies success. If the secret word can be completed in more than one ways, we accept the modifications made in this substitution and return the modified key. Finally, if the secret word cannot be completed at all, we undo the modifications of this substitution and return the original key that we came with.

Continuing with the run of the example described in `example.txt`, The first substitution to be tried is `((E . e) (T . o) (A . q) (I . w))`. After incorporating this substitution in the empty key we get the modified key. Withthis modified key we start the dictionary closure. The dictionary closure fails after a few multiple matches, and we try the next substitution. The first success comes with the substitution ((E . o) (T . e) (A . w) (I . q)) and you can see that after incorporating this into the empty key we get some letters of the secret word right. This in turn completes the word DISGUSTS uniquely and more substitutions follow. The key now fills rapidly, which enables more unique completions. When DC* is over, the key is near complete: `(w i s d o m n p q _ t u v x y z a b c e f g h _ k _)`. SWE comletes the secret word uniquely and consistently with the rest of the key, thereby completing the key.

4 What do you have to do

In our experience, the `etai` strategy followed by DC*+SWE can decode most of the ciphertexts. Therefore in this assignment you are required only to implement `etai` followed by DC*+SWE. However,those of you who would like to go *beyond the assignment* can attempt the following more difficult problem. What makes the current encoding scheme to crack is that the key is based on a secret word. If we remove this restriction and let the key be *any* permutation of the plaintext alphabet, then the problem becomes harder to solve and you have to do use the other strategies mentioned in writeup.

Apart from this writeup, we shall provide the following support materials.

1. A few photographed pages from the book by Simon Singh which gave me the idea of the assignment.
2. The web page `https://simonsingh.net/The_Black_Chamber/chamberguide.html` contains interesting information about codes.
3. The skeleton of the implementation iprepared by Ananya Bahadur. This is a set of files with holes. You have to fill the holes with your code.
4. An executable of the model implementation by Ananya.
5. The file `example.txt` which contains a trace of what happens when some the example `samples/text-1-feynman.txt` is run on the model implementation.