Sam Holmes
14307915
Fundamental Concepts of Cryptography 200
Assignment 1

Affine Cipher Questions

There are 12 numbers that are co-prime with 26 (1,3,5,7,9,11,15,17,19,21,23,25) and a potential shift of 26 for each of these numbers. Therefore there are 12 * 26 potential possible keys that can be selected. This is considered as highly insecure and susceptible to attack.

$$\mathrm{E}(x) = (ax + b) \mod m,$$

The decryption output of any character can be found through :

$$\mathrm{D}(x) = a^{-1}(x - b) \mod m,$$

This decryption algorithm can be found mathematically by finding the inverse of the encryption algorithm :

My program correctly implements the affine cipher for all valid test cases. It loops over the user input for the key to ensure that what is provided to the algorithm is correct (i.e. : a is co-prime of 26).

The algorithm to determine the output character for any input character is found mathematically through the following algorithm :

$$\begin{aligned}\mathrm{D}(\mathrm{E}(x)) &= a^{-1}(\mathrm{E}(x) - b) \mod m \\ &= a^{-1}(((ax + b) \mod m) - b) \mod m \\ &= a^{-1}(ax + b - b) \mod m \\ &= a^{-1}ax \mod m \\ &= x \mod m.\end{aligned}$$

(Source : http://en.wikipedia.org/wiki/Affine_cipher)

The letter distribution can be found by executing the the frequency function from the main menu of my program. An example for a = 5 & b =5 is below.

Encrypted version:
A : 0 B : 3 C : 5 D : 0 E : 6 F : 14 G : 0 H : 0 I : 5 J : 5 K : 3 L : 1 M : 15 N : 4 O : 6 P : 6 Q : 1 R : 14 S : 12 T : 15 U : 2 V : 1 W : 17 X : 17 Y : 0 Z : 18

Decrypted version:
A : 14 B : 3 C : 6 D : 2 E : 18 F : 6 G : 5 H : 6 I : 15 J : 0 K : 0 L : 5 M : 4 N : 12 O : 17 P : 5 Q : 0 R : 15 S : 14 T : 17 U : 3 V : 0 W : 1 X : 1 Y : 1 Z : 0

One thing to note about the encrypted version of the code, is that the frequency of popular letters will remain. This means that common use letters such as E will have a higher frequency, even after the algorithm encrypts it, just at a different letter. This can be seen in the encryption version where Z has a frequency of 18 and obviously corresponds to E. This won't be true in all cases, however it can give an indication of the contents of the encrypted message, creating another reason why this cipher isn't very secure. It is impossible to determine the key without the frequency of the original decrypted version, however with access to the frequency of the input & output, you could map the outputs to a simultaneous equation that would provide you with the key.

Data Encryption Standard (DES) Questions

My program operates correctly for all inputs. It will correctly encode and decode the messages passed to it in a file and then output the decrypted message. One issue that my program has is that the formatting of the input file string is not taken into account when outputting the final decrypted message. What this means is that the output file does not have the same format of the original. This is due to my program not handling white space in anyway and is only concerned with the data in the file provided.

I convert the file data into a set of strings that I then convert into binary representation that are 64 bits long. Originally I was intending to simply store these numbers as an unsigned long which would be 64 bits long and handle all the values I require, however, given that I designed my algorithm to run in Java, I do not have access to the unsigned keyword, therefore I couldn't store values 2^64 , only up to 2^63 in a primitive variable. There were a couple of workarounds to this problem, such as using the BigInteger objects to store the variables, however I decided to use an array of numbers to store the bit string instead. Each element would equate to one bit. This is extremely wasteful of memory, however the DES algorithm requires operations on individual bits at a number of times, making these operations extremely easy as the numbers are already divided into individual elements, ready to be used.

Converting the starting key (56 bit binary key) to all 1's and running the encryption algorithm still manages to perform the encryption and decryption correctly. To run this version of the algorithm, un-comment the lines 460 & 465, which will cause it to override the initial key with all 1's.