PROJECT 1: VIGENERE & IMAGES
CECS 564-01

AARON A. HILL

1. Encryption

The Encryption function, 'EncryptImage' was written in Matlab, it takes a file name (*string*) of an image to be encrypted, a key (*string*), the file name in which to save the input (*string*), it returns an array of encrypted bytes (*double*, in the format of unsigned 8-bit integers).

```
function enc = EncryptImage(x,k,f)
fd=fopen(string(x), 'r');
fr=fread(fd);
fclose(fd);
fr = fr';
k1=double(k);
n=size(fr,2);
m=size(k,2);
for i=1 : n
xp = double(fr(i));
ii = mod(i-1,m)+1;
enc(i)=mod(xp+k1(ii),256);
end;
fd=fopen(string(f),'w+');
fwrite(fd,enc);
fclose(fd);
```

The file I submitted to my lab partner was a .jpg encrypted with the 32 character pangram "packmyboxwithfivedozenliquorjugs".

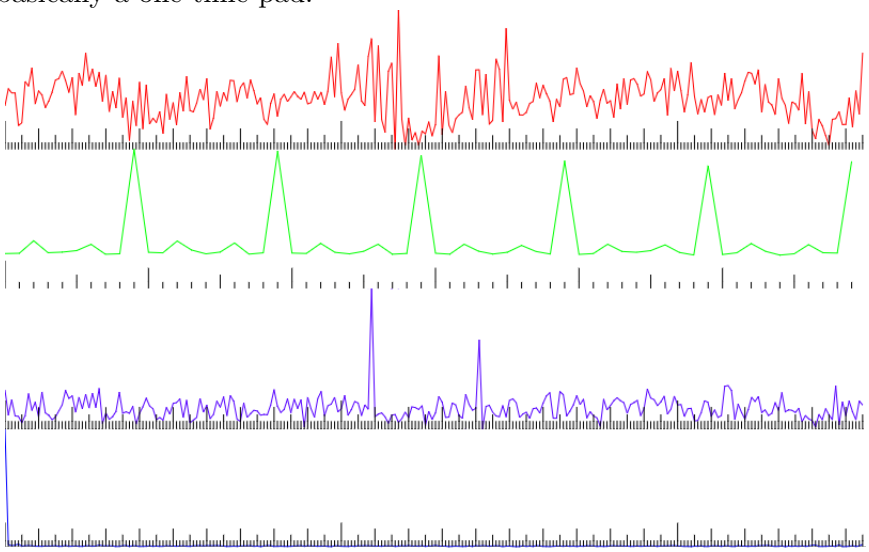
2. Decryption

I handled decryption and attack with html/javascript. The decryption function itself simply reversed the encryption function. That is, the ASCII code for each character of the key was subtracted from the corresponding data character's ASCII code (mod key length) ASCII code, and that was taken as the positive mod 256 value. A new string was then created by concatenation of this decrypted array, encrypted to base64, and displayed as a jpg. The code can be viewed in the "decrypt" function of the attached "attack_image.html".

3. Attack

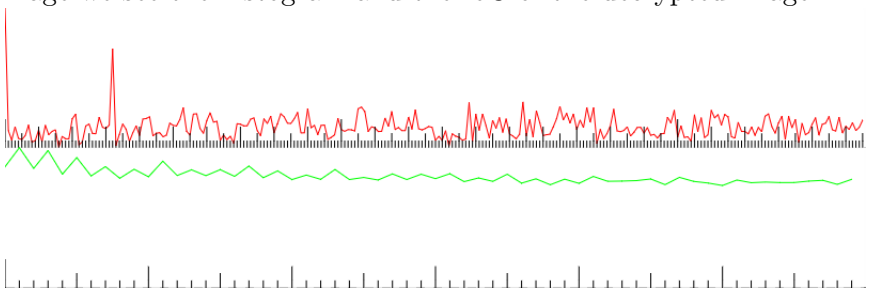
3.1. **Plaintext Attack.**

In general, the shortest plaintext ASCII message that will present the secret keyword is one of the same length as the keyword. If the keyword length is unknown, it would be reasonable to assume that a message of length long enough that the first three characters of the keyword are found to be repeated would be sufficient. In this example, I encrypted my image with a 32-character encryption. Had I chosen a less well known string, or better yet, a random string or a string of random words, it is unlikely anything less than 35 characters would yield the key word. However, even this could be defeated if one had the foresight to intentionally repeat the characters at seemingly random intervals. Of course, at some point we approach what is basically a one-time pad.



3.2. **Ciphertext Attack.**

In the first image we see, from top to bottom, the histogram of the encrypted image (sent to me by my lab partner), a graph of the Index of Coincidence, a histogram of the first letter given a keyword length of 10 (the period of the IoC graph), and a reference histogram formed from 305 jpgs. In the second image we see the histogram and the IoC of the decrypted image.



3.2.1. *Histograms.*
The histogram of the plaintext is clearly similar to that of the reference histogram, whereas the histogram of the ciphertext is a great deal more entropic.

3.2.2. *Index of Coincidence.*
We see clearly from the graph, that the Index of Coincidence has a periodicity of 10- indicating that 10 is the most likely keyword length.

3.2.3. *Keyword.*
The full program is included with this report submission. The keyword for the file provided to me was "miraculous". I will attempt to summarize the method used for the attack, neglecting the particulars only relevant to the platform chosen.

- load file
- for s from 0 to maxKeyLength do
  for i from 0 to dataLength do
  if the i'th data element is equal to the (i+s+1)'th data element increment the s'th histogram bin.
- display histogram
- physically determine periodicity of IoC, n
- create n arrays
- for i from 0 to dataLength do
  push the i'th data element onto the (i mod n)'th array.
- compare the mode of each array to the mode of the reference histogram (or 0) to determine most likely shift character
- concatenation of shift characters in order is likely keyword.
  repeat process with combination of 1st and 2nd (etc) modes if file does not decrypt or, if meaningful string is suspected, guess.