CMPUT 333: Assignment 1

By: Justin Barclay, Mackenzie Bligh, and Vuk Radovich

Group 7

Workload: Justin completed parts 1 and 2 of the assignment with some help from Mackenzie and Vuk. Justin did parts 1 and 2 because he had previous experience with Viginere square. Mackenzie worked on part 3. Vuk is working on part 4.

Part 1:

1) Decrypted Text

Tonight I can write the saddest lines.

Write, for example,'The night is shattered

and the blue stars shiver in the distance.'

The night wind revolves in the sky and sings.

Tonight I can write the saddest lines.

I loved her, and sometimes she loved me too.

Through nights like this one I held her in my arms

I kissed her again and again under the endless sky.

She loved me sometimes, and I loved her too.

How could one not have loved her great still eyes.

Tonight I can write the saddest lines.

To think that I do not have her. To feel that I have lost her.

To hear the immense night, still more immense without her.

And the verse falls to the soul like dew to the pasture.

What does it matter that my love could not keep her.

The night is shattered and she is not with me.

This is all. In the distance someone is singing. In the distance.

My soul is not satisfied that it has lost her.

My sight searches for her as though to go to her.

My heart looks for her, and she is not with me.

The same night whitening the same trees.

We, of that time, are no longer the same.

I no longer love her, that's certain, but how I loved her.

My voice tried to find the wind to touch her hearing.

Another's. She will be another's. Like my kisses before.

Her voide. Her bright body. Her infinite eyes.

I no longer love her, that's certain, but maybe I love her.

Love is so short, forgetting is so long.

Because through nights like this one I held her in my arms

my sould is not satisfied that it has lost her.

Though this be the last pain that she makes me suffer

and these the last verses that I write for her.

- This file can also be found at decrypted1.txt
- The key was 0x50 0x2f 0x08 0x7c 0x5f 0x30 0x00

2) I used the Friedman method for cracking Viginere Squares to determine the most likely length of the key. The Friedman method relies on the idea that the plaintext we're decrypting has some sort of known structure. In this case, because it is plaintext, we can assume it's written language and because the class is being taught in English, I also assumed the plaintext was in English.

With these assumptions we can narrow down the search field and know that the english language has an index of coincidence of ~0.067. This means that I needed to split up the cipher text into chunks of varying sizes and calculate the IoC(index of coincidence) for each "chunk" of text. Then, I averaged out the IoC for all chunks of a given size. This then led me to the fact that the key length was 7. Now within each of these 7 chunks we know that the English language follows another common structure, such that E,T,A,O,I,N,S,R are the most common letters. So, I then used each of these letters as guesses at the plaintext and counted up how many times a hex key came up. For each section the most frequent hex key that was revealed from the guessing I set as key.

3) As explained in part 2, I automated the guessing of the key by counting up the frequencies of each key each chunk of the split cipher text. I then took the most frequent keys in each section and used those to reverse the Viginere. I do admt that I got a bit lucky, in that there were no ties in key frequency otherwise I would have had to do a bit more work by hand.

4) If the plaintext file had been zipped before decrypting, I would then generate keys using various zip file signatures as plain text. After I got the keys I would decrypt the file using the keys and hand check to see which file can me a valid file with readable plaintext inside.

Reference:

https://en.wikipedia.org/wiki/Vigenère_cipher - Friedman_test

Part 2:

1) The plaintext associated with the given cyphertext can be found in this folder at plaintext.jpg.

- The key used to encrypt the plaintext was "53.503563N,-113.528894W"

2) The first thing I did to determine the key was to create two databases of common file signatures. I used both https://en.wikipedia.org/wiki/List_of_file_signatures and http://www.garykessler.net/software/index.html#filesigs. With Wikipedia, I used an HTML parser to get the required information from the table and then I cleaned it up by hand, until it was in an easy to use format. As with Gary Kessler's site I downloaded a zip file of headers, and again cleaned it up until it was in a suitable state.

After I acquired the necessary data, I ran getKeys.py which took in one of my files with a set of file signatures and applied a transformation that where given a ciphertext and a guessed plaintext a key was produced. I then threw out any keys that were not printable 32 < n <127.This gave me a partial key, that looked like a GPS coordinate and the file signature being that of a jpg/exif. With some guessing I put in an initial set of GPS coordinates and tried to decipher the encrypted information from there. Now that I knew the structure of the file I could look for sections of partially deciphered text that matched some known pattern in jpg/exif and decrypt the rest of the key with a little bit of guessing and checking with the plaintext.

3) I did not use an automated method to determine if I had the right key or not because I was not decrypting a plaintext file and was unsure of the structure of the file. Which means that I could not use knowledge of things like IoC or letter distribution to help get the structure and contents of the key. That meant after I got the partial key, I had to look at the partially decrypted plaintext and find a place with a readable partially decrypted plaintext and use that plaintext to finish getting the key. So, I could not automate this process because the key was too long given and I did not initially have enough information about the file structure to use patterns to help with my decryption.

References:

https://en.wikipedia.org/wiki/List_of_file_signatures
http://www.garykessler.net/software/index.html#filesigs

Part 3:

file size of plaintext: 80 bytes

password used: "01234567"

Comparison for ecb mode

file size of cipherecb.enc: 88 bytes

i) The size of cipherecb.enc increases by 8 bytes as the key is use to encrypt a partial block, and that information is put into the resulting file (cipherecb.enc)

ii) The pattern repeats in the cipher text as it does in the plaintext because ecb encrypts independently encrypts each bit block of the plaintext. Therefore, there will be 1 block that is incompletely filled by the plaintext that is padded out to entirely fill the block. This results in the ciphertext becoming longer than the plaintext by 8 bytes

Comparison for cbc mode

file size of ciphercbc.enc: 88 bytes

i) The size of ciphercbc.enc increases by 8 bytes as a partial block must be padded out to the length of the full block for the XOR operation and encryption to function according to the algorithm

ii) No discernable pattern occurs as the initialization vector (password) is first XORed with the cipher text which will help to obfuscate any pattern in the text, as a cleartext block is continually XORed with different ciphertexts. No cipher text XORed with a plaintext block will ever be the same, so no pattern can occur

Comparison for cfb mode

file size of ciphercfb.enc: 80 bytes

i.) The size of ciphercfb.enc does not change because no padding of the plaintext needs to occur in the final partial block. The ciphertext bits that do not have a corresponding bit in the plaintext to be XORed are simply not used

ii.) No pattern occurs as ciphertext used in the XOR operation performed at each stage is unique. Therefore it will remove any pattern that is present in the plaintext

Comparison for ofb mode

file size of cipherofb.enc: 80 bytes

i.) The size of cipherecb.enc does not change, as no padding of the plaintext needs to occur in the final (partial) block. The ciphertext bits that do not have a corresponding plaintext bit to be XORed with are simply discarded

ii.) No patter occurs as the ciphertext used in the XOR operation is always unique. Therefore, no patter can be carried forward over more than 1 XOR operation, rendering it impossible for there to be a pattern

Effect of modification & decryption

On ECB

The file remained mostly intact, with the 3rd set of "01234567" (3rd 64 bit block) being damaged (meaning decrypts into gibberish). The impact of the error was that the 64 bit block in which the 19th byte was modified becomes undecryptable, but the other blocks are uncompromised because they are encrypted separately.

On CBC

The file remains mostly intact with the 3rd block, and 1st byte of the 4th 64bit block being damaged (meaning decrypts into gibberish). The impact of the error was that the other blocks are uncompromised as if a block of plaintext is corrupt but subsequent plaintext blocks are correct each subsequent block is XORed with the ciphertext of the previous block, and not the plaintext. The previous block does not need to be decrypted before using it as the IV for the decryption of the subsequent blocks

On CFB

The file is completely compromised, meaning it is not decrypted at all. This is because when XORing the ciphertext, with the result of block cipher, it will return altogether different value. This value is then fed into the next block decryption, and causes a cascading effect of incorrect decryption.

On OCB

The file is completely compromised, meaning it is not decrypted at all. This is because when XORing the ciphertext with the result of the block cipher (initialization vector) in the first round of decryption, produces an incorrect result that is fed into the next round of decryption, and causes the error to propogate through the rest of the data.

Distribution of Workload:

The workload was divided evenly, to the best of our knowledge at time of assignment. Justin handled part 1 and 2, Mackenzie handled part 3 and collaborated with Vuk to work on creating rainbow tables for use in John the ripper. Vuk handled setting up, testing, and running John the Ripper. While individual group members were in charge of handling a specific area of the assignment, all group members worked collaboratively with lesser roles on all parts of the assignment.

Sources:

Part3:

https://adayinthelifeof.nl/2010/12/08/encryption-operating-modes-ecb-vs-cbc/

http://stackoverflow.com/questions/3283787/size-of-data-after-aes-cbc-and-aes-ecb-encryption

https://en.wikipedia.org/wiki/Data_Encryption_Standard