CMPUT333 LAB 1 GROUP 08 PROJECT REPORT

Group Members

Shuming Zhang
Michael Xi
Michael Rijlaarsdam

Part 1

The frequently used words are used to decide which key is right. We test the " th" and " and " (including the empty spaces) and get the following key results.

```
KEYLIST    OCCURRENCE TIMES      START INDEX
" th":
[68, 69, 27] 5 [586, 740, 901, 1006, 1419]
[95, 117, 16] 5 [221, 319, 557, 1439, 1502]
[0, 80, 47] 5 [111, 279, 797, 1042, 1588]***
[47, 8, 124] 6 [540, 645, 666, 1100, 1408, 1485]***
[80, 47, 8] 5 [154, 700, 833, 924, 1428]***
[95, 48, 0] 9 [480, 508, 571, 872, 1019, 1320, 1523, 1537, 1572]***
[127, 69, 96] 5 [481, 572, 1020, 1524, 1573]
[48, 0, 80] 5 [19, 194, 502, 1160, 1566]***
" and ":
[99, 64, 107, 28, 113] 2 [4, 179]
[8, 124, 95, 48, 0] 2 [226, 387]***
[19, 35, 92, 95, 48] 3 [414, 715, 1464]
[80, 47, 8, 124, 95] 2 [329, 756]***
```

The result shows the key lists with stars is continuous and occurs more than others. So We guessed that the right key list might be [..., 0, 80, 47, 8, 124, 95, 48, 0, ...]. But deciding which one the first key element is still needs calculation on index. We selected the [48, 0, 80] on the character with index=19 of the whole text and got the first key of character:

Index	0	1	2	3	4	5	6	7	8	9
Key	80	47	8	124	95	48	0	80	47	8
Index	10	11	12	13	14	15	16	17	18	19
Key	124	95	48	0	80	47	8	124	95	48

With keylist = [80, 47, 8, 124, 95, 48, 0], the decrypted text is meaningful. We found the right key.

The strategy would have been changed due to the zipped text contains non-printable ASCII characters if the plaintext was compressed.

Part 2

We first tried to figure out the file header, so we built a program that would take in a couple of bytes as a guess at the plaintext, and decipher them as if they were correct and check if the resultant key is comprised of printable ascii characters. We then went through Gary Kessler's list for common signatures and got matches with an obscure variation called DOC, and JPG so we went with JPG. Then we started guessing what the rest of the key was based on JPEG Exif and TIFF file layouts, until we reached a ,- that corresponds with the trailer for a JPEG EXIF file ",-"
We later realized that we don't necessarily end on ",-" which are the eleventh and twelfth bytes of our key but that they would be the remainder of the file length divided by the key length. 3,055,616%|k| = 12, so the key length is 763901, 1527802, or 3055604 bytes long. At which point it is highly unlikely that we do not have another ,- pair of key bytes. Check the readme for how the plaintext guessing is going right before submission.

For the anatomy of a JPEG file we relied heavily upon these three sources
http://itbrigadeinc.com/post/2012/03/06/Anatomy-of-a-JPG-image.aspx
http://partners.adobe.com/public/developer/en/tiff/TIFF6.pdf
http://www.awaresystems.be/imaging/tiff/tifftags/baseline.html

Part 3

"innocentciphertext3" has not been modified at all. Nope. No-siree-bob. Just an innocent, unchanged file that has been safely encrypted to ensure safety. You can trust your own encryption can't you? Have more faith in yourself.

All files were encrypted with the key "nikolaidis"
Looking at the contents with "xxd -ps cipherXYZ.enc > cipherXYZ.txt"

Size comparisons:
cfb and ofb are the same length as p.txt, which is 81 bytes. While ecb and cbc are 88 bytes because they get padded to multiples of the block size. I can also conclude that the block size is 8 bytes.

Observations on the Electronic Codebook Encryption:
The encrypted file is seven bytes longer than the original file. I can see it added a footer of 36 3b 72 2a 8c f4 97 91. The rest is ten repetitions of 4da7705abe6767be as you would expect from a file with ten repetitions encrypted with ecb, which isn't used due to its inability to hide data patterns.

Looking closer to why p.txt is 81 bytes, I realized that I had a linebreak at the end of it so I am going to put these files in a folder called "lbFail" for posterity and doing it again.

p.txt now has 80 bytes but this has not changed the amount of padding in ecb.txt; though the footer has changed. The padding is there because ECB and CBC need padding because the decryption algorithm expects it. It does not have an efge case for this situation because 8 bytes is negligible space overhead and is deemed preferable to losing data or possibly corrupting the file. Testing it with 79bytes in p.txt, the result, cipherecb79bytes.enc does indeed have 80 bytes, which supports the needs padding and 8 byte block size conclusions.

Observations on the Cipher Block Chaining Encryption:
As expected for cbc, no repetitions no pattern, because the ciphertext is randomized

Observations on the Cipher Feedback Encryption:
As expected for cfb, no repetitions no pattern.

Observations on the Output Feedback Encryption:
As expected for ofb, no repetitions no pattern. I did however notice that the first 8 bytes of ofb and cfb are the same which tells me that they use the same IV and block cipher encryption algorithm as well as the same plaintext.

Observations on the Electronic Codebook Decryption:
I Switched "p" to "o" to cause the error. This caused the third block of the decrypted file to be completely garbled to ç£1æƒø³Ê, while all the other blocks are unchanged. This is expected in ecb because all bytes affect all other bytes in a block but each block is completely independent of each other.

Observations on the Cipher Block Chaining Decryption:
I Switched "Á" to "A". This lead to the third block becoming gibberish except the fourth byte and the third byte of the fourth block. I believe it is coincidence that the fourth byte of the third block is unchanged. The third byte of the fourth block being changed makes sense because it is XORed with the byte I changed after being run through block decryption; I believe it coincidence that it ended up being a superscript 2 while it should have been a regular 2. All other bytes are unchanged.

Observations on the Cipher Feedback Decryption:

I Switched "A" to "B". This lead to the fourth block being garbled and the byte I changed being a "1" instead of a "2". This makes sense because the only effect a block of ciphertext on its corresponding block of plaintext is that it gets XORed with the result of the preceding block of ciphertext being fed through the block encryption algorithm; leading to the nineteenth block being off by 'B'-'A'. This also explains why it was the fourth block that got garbled, because the third block got decrypted so its error spread throughout the value integral to the fourth block.

Observations on the Output Feedback Decryption:
I Switched "í" to "i". This lead to the nineteenth byte being very different and no other changes. this nineteenth was not off by one because "í" and "i" have very different bit value despite their superficial resemblance. This modes corruption resistance is due to the fact that neither the plaintext or ciphertext is ever run through the block cipher encryption algorithm. Instead, to decrypt, the ciphertext gets XORed with the result of the block cipher encryption.

Workload Breakdown

Part 1: Shuming Zhang
Part 2: Michael Xi & Michael Rijlaarsdam
Part 3: Michael Rijlaarsdam
Part 4: Michael Xi