CS4600 Homework 2

1. Simple substitution cipher text can quickly be test via frequency analysis and anagramming because in spoken languages, certain letters are frequently used more than others. Combining letter frequencies with grammatical structure and semantics through anagramming, cipher text can be decoded much more efficiently than brute force.
2. A 2GHz processor can run 2\*109 cycles per second. It takes 100 cycles to check 1 possible key so the 2GHz processor can check 2\*109 / 100 = 2\*107 keys per second. So, the time it takes to crack a 56-bit DES encryption by testing 256 all possible keys is 256 / 2\*107 = 3.6\*109 seconds or about 114.2 years.

Similarly, to crack 128-bit AES encryption, it will take 2128 / 2\*107 = 1.70141183460469224e+31 seconds or about 5.4\*1020 millenniums, significantly longer than the estimated lifetime of the universe.

1. The code output is as shown below.

A screenshot of a computer

Description automatically generated with medium confidence

1. For task 1, I used cryptool.org to find out the distribution of letters and replace the top three letters with E, T, and A. By further analysis and looking at grammatical structure, after a total of 12 iterations, I was able to decode the message.

A sample of the command I use is tr ‘nyv’ ‘ETA’ < ciphertext.txt > text1.txt. This command replaces n, y, and v with E, T, and A, respectively. It takes the ciphertext.txt file as input and output a new file with the substituted letters.

Please see attached for the decoded text file and the screenshot shows you the iterations.

Graphical user interface, application

Description automatically generated

1. Task 2 – Note that a salt is also use for added security and option -iter 100000 & -pbkdf2 have to be included in the new version of OpenSSL 1.1.1

All these other options make brute force attack harder to accomplish

Text

Description automatically generated

Above is encryption using AES-256-CBC with the decrypted file from Task 1. Next, we can decrypt and check if the file have been decrypted.

Text

Description automatically generated

As shown, we can confirm the decryption is the original file

------------------------------------------------------------------------------------------------------------

Encryption for DES-ECB as shown below

Text

Description automatically generated

Decryption and confirmation following.

A screenshot of a computer

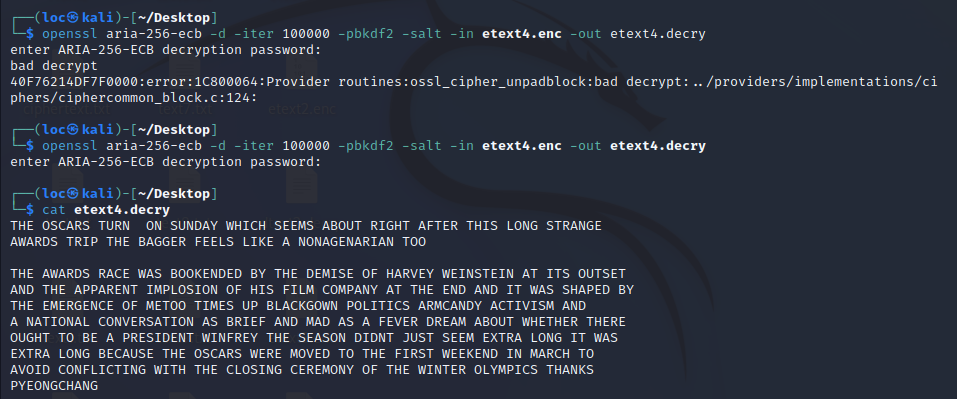
Description automatically generated

Encryption using ARIA-256-ECB

Text

Description automatically generated

Confirmation of decryption



Note – I try to enter a different password for decryption, and it obviously did not work.

1. Task 3

First, I encrypted the file using AES-256-CBC encryption. Then, I extract the 54-bit header from pic\_original.bmp and save it to a file named header. Next, extract the body encrypted picture’s header by tail offset 55 and discard the 54-bit encrypted header. Lastly, concatenate unencrypted header with the encrypted body and output in a new file.

Text

Description automatically generated

Shape, circle

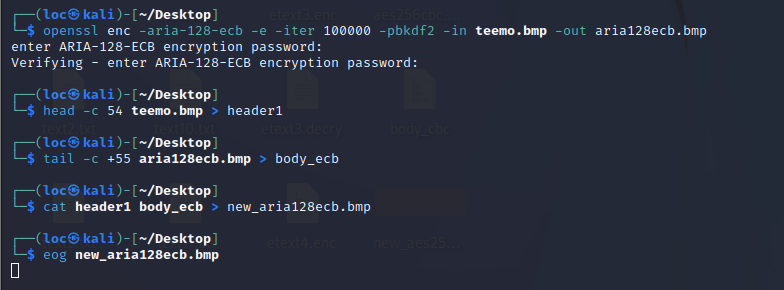
Description automatically generated

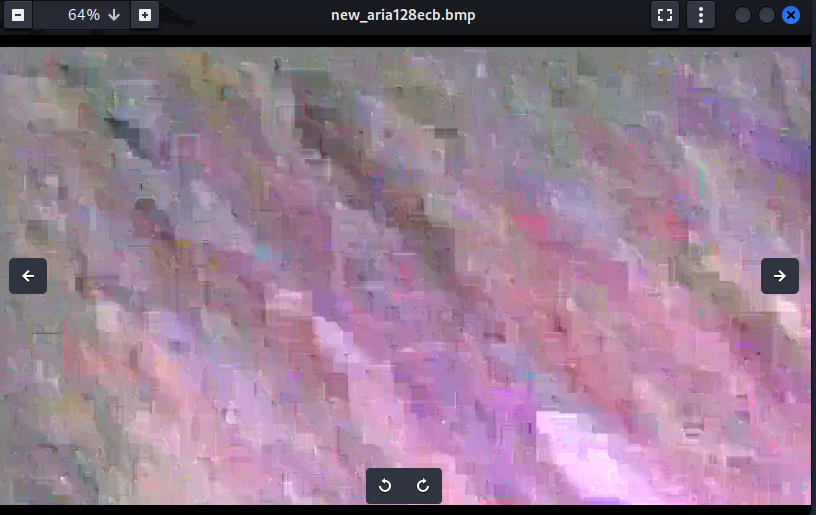
Background pattern

Description automatically generated

The encrypted picture doesn’t resemble the original at all and useful information can’t be derived from it.

Using the same method as above, this time I will use ARIA-128-ECB and picture of choice below.





The encrypted picture doesn’t resemble the original picture at all.

1. Task 5

A screenshot of a computer screen

Description automatically generated with medium confidence

Use file *text12.txt* from task 1 which is at least 1000 bytes long.

**Hypothesis:**

ECB – only corrupted block is affected. Everything else OK

CBC – the original corrupted block and immediately following block is affected. Everything else OK

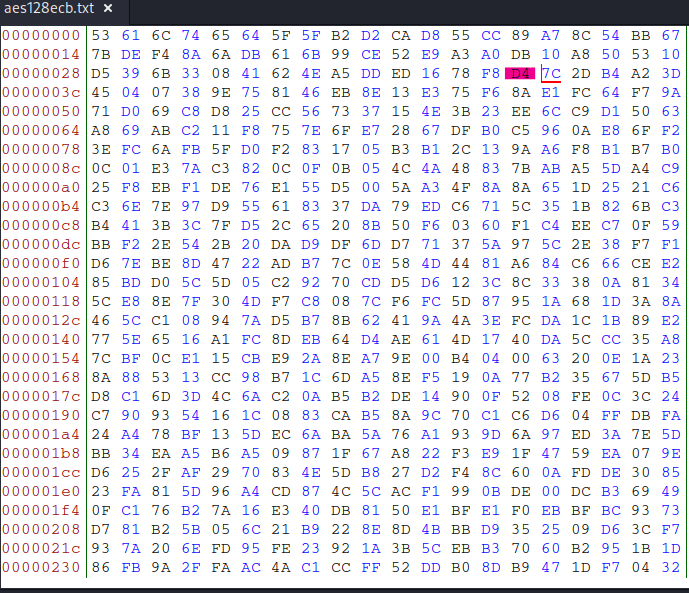
CFB – the original corrupted block and immediately following block is affected. Everything else OK

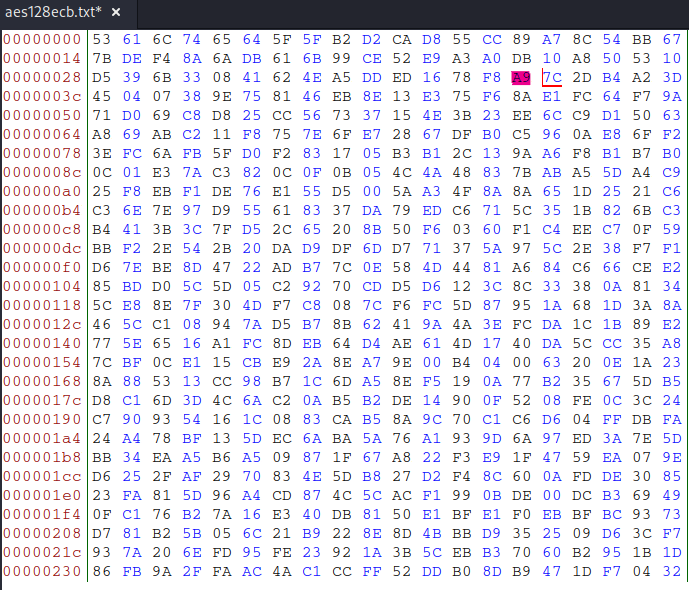
OFB – only corrupted block is affected. Everything else OK

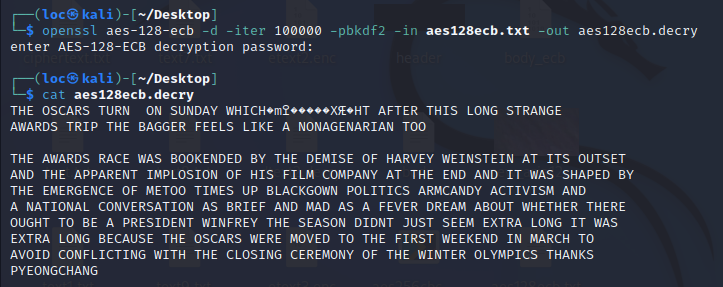
**Testing:**

AES-128-ECB

Original vs edited hex-code to follow







As observed, my prediction was correct, only the corrupted block is affected. Everything else is OK.

AES-128-CBC

Text

Description automatically generated

As observed, my prediction is off because the length of corrupted text is the same as ECB even though it supposed to affect more than 1 block of text though everything else is OK.

AES-128-CFB

Text

Description automatically generated

As observed, my prediction is right because the length of corrupted text isn’t the same as ECB and a different area of the text was affected.

AES-128-OFB

Text

Description automatically generated

Using this method, only a space is corrupted, and my prediction didn’t expect this.

1. Task 7

First, update the IV and ciphertext.

Next, add padding by nesting two *for* loop, one traversing the password index, the other to add # as padding when the \n newline character is detected. Finally, append null terminator to that word.

The next step is to encrypt the modified password using the aesecnrypt() method. This method will encrypt the password and then compare it to the expected password for a match. If matched, it will return 1. Else, it will return 0.

A *while* loop will be active and encompass the password modification, encryption, and checking until a match is found.

When found, the *while* loop will terminate and the correct key will be printed out.

Output

Text

Description automatically generated