CES571S - L2 - Crypto Encryption

467261 - Yifu Wang

2018 - 09 - 26

I'm using SM.MS, a free and clean image host, to store my screenshots online, it's pretty safy. If that cause you any trouble, I will include my screenshots in pdf in future labs.

1 Substitution Cipher

I didn't reinvent the crypto algorithm for substitution cipher. I did some research in this field and use the substitution solver to solve this online. The following steps is the path I will take to implement a solver by myself.

1. Scan the ciphertext, record the frequency of each letter, record the frequency of each 1-letter, 2-letter, 3-letter words, record the pattern of two consequent same letter.
2. The frequency of what we recorded in step one in plaintext of English should be

   letter: {ETAOIN/RSHDL}>0.04, {CU/FMPGWY}>0.02, {BV/KXJQZ}<0.02

   1-letter word: a, i

   2-letter word: of, to, in, it, is, be, as, at, so, we, he, by, or, on, no...

   3-letter word: the, and...

   consequent pattern: ss, ee, tt, ff, ll, mm, oo...
3. Sort the a-z letters by the frequency we recorded and use this as an initial key for our algorithm. This algorithm could be Hill Climbing, Simulated Annealing or Genetic Algorithm. In order to find the best solution which make our translated text fit the frequency table listed in step 2 best. We need a algorithm to measure the fitness of current solution. This algorithm can be as simple as measure how close a 'out-of-order' sequence to a 'well-ordered' sequence.
4. After we get the solution from step 3. The key we obtained might be still not right. Then we exchange the letter in the same group in the **letter** listed in step 2 by exhaustive method. Quit the algorithm once the grading of current solution exceed the pre-set threshold.

Here is the solution I obtained and the solver I used.

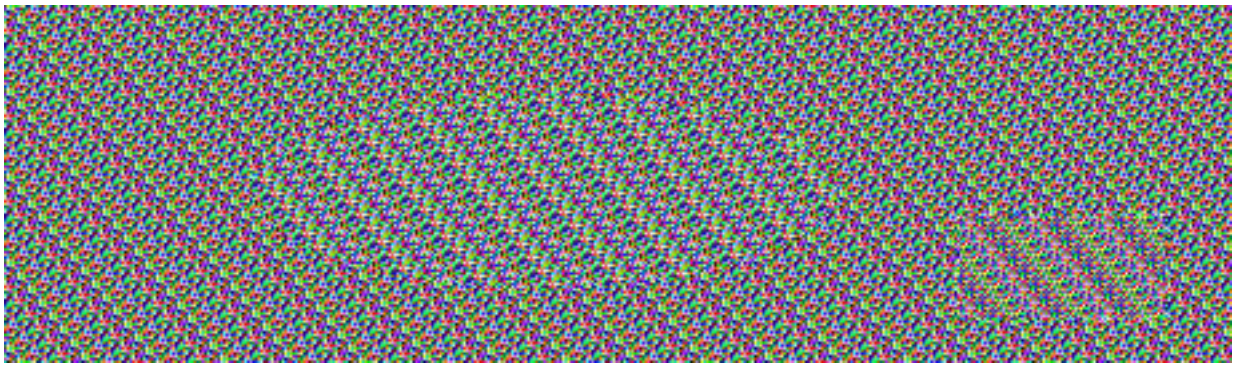
2 Encryption Mode - ECB vs. CBC

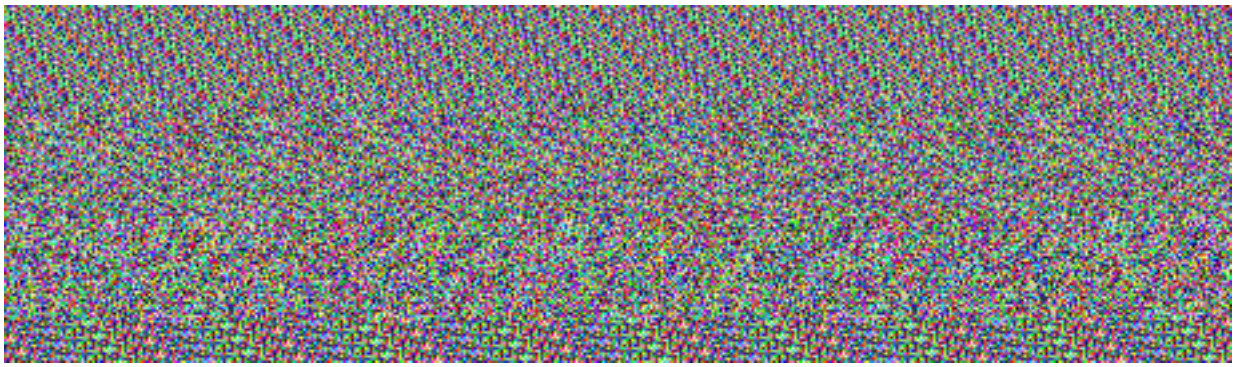
This task is somehow related to another course I take, CSE554T, where we learn to process 2D or 3D img data using Mathematica language. It's a interactive programming language. I choose use it for this task and I upload the source code to github. The following is a simple explaination of the code.

1. `KeyGen` generate a L-byte-long random key.
2. The block cipher used in `ECB` and `CBC` are `BitXor` for simple.

3. The IV used in `CBC` is randomly generated, since we didn't really need to decrypt the image.

The following 3 images are origin, ECB and CBC.





You can see that the encrypted img produced by ECB still preserve the shape of original img. That is because when using an ECB same plaintext will be encrypted into same ciphertext. By applying CBC you can eliminate such disadvantage.
Following 3 images reproduce this phenomenon using another original image. original ECB CBC

3 Error Propagation - Corrupted Cipher Text

To answer the question first.

- **ECB,OFB** will only affect the block where the corruption happend.
- **CBC,CFB** will affect the block where the corruption happend and the block after it.

In order to observe the corruption easily. The plaintext I choose will be character `'1'` repeated 1008 times. Here is the plaintext.
I'm using this online AES tools for this task. The following table is a summary of the results.

	Encrypted	Corrupted	Decrypted	Results
ECB	ecb0.png	ecb1.png	ecb2.png	Only 4th block changed
CBC	cbc0.png	cbc1.png	cbc2.png	4th and 5th blocks changed
CFB	cfb0.png	cfb1.png	cfb2.png	4th and 5th blocks changed
OFB	ofb0.png	ofb1.png	ofb2.png	Only 4th block changed

Though there is nothing unexpected. But let take a look at result result of CBC and OFB. In 5th block only 6th byte changed(we changed 6th byte in 4th block), that is because we actually calculated the xor of original ciphertext with the edited ciphertext and the only difference is the 6th byte. The similar reason also apply to the result of OFB where only the 6th byte of the 4th block changed.

4 Initial Vector

4.1

The Javascript source code is upload to github. The plaintext is

```
'The quick brown fox jumps over the laze dog.    '
```

`key`, `iv1` and `iv2` are randomly generated. `e1` and `e2` using `iv1` as IV, `e3` using `iv2`. `e1` and `e2` are excatly same. Here is the screen shot. Thus since using same IV will cause the same plaintext encrypted into the same ciphertext. It's vulnerable under strong attacking.

4.2

Yes we can. Since we have

$$\begin{aligned} C_i &= P_i \oplus O_i \\ P_i &= C_i \oplus O_i \\ O_i &= E_k(O_{i-1}) \\ O_0 &= IV \end{aligned}$$

thus $O_i = C_i \oplus P_i$. Furthermore if IV remains same, all O_i will remain same. By running this algorithm. We have

```
P2 = 'Order: Lauch a missile!'
```

If we replace OFB with CFB. We will at least reveal the first bolck of `P2`. Since by the definition

$$\begin{aligned} C_i &= E_K(C_{i-1}) \oplus P_i \\ P_i &= E_K(C_{i-1}) \oplus C_i \\ C_0 &= IV \end{aligned}$$

only C_0 will remain unchanged between two cipher processes.

4.3

The definition of CBC is (only one block)

$$\begin{aligned} C1 &= E_K(IV1 \oplus P1) \\ C2 &= E_K(IV2 \oplus P2) \end{aligned}$$

Thus if we construct

$$P2 = IV2 \oplus IV1 \oplus P1$$

we shall get $C1 = C2$. The Javascript source code is uploaded to github. And from the debug view , we can see `P1 = Yes`.

When conduct this attacking. I encountered a truble. That is when the plaintext is not a multiple of 16 (key size in byte). aes-js package will throw a error and refuse to continue any calculation. And I don't know what has been append to the P1 by default. So I have to using the key (known only to Bob) to decrypt the C1 and found out that `'\r'`s have been added to make a 16 length P1. And the code of `''␍`s 13. That's the reason of line 11.

5 Programming using the Crypto Library

The Javascript source code is uploaded to github. From the debug view , we can see

```
key = 'oracle##########'
```

The main truble I encountered by using Javascript is I don't know what character should be append to the plaintext to perform the encryption. I tried using `'\r'` as it was used in task 4.3, but that doesn't work. So I just try in the opposite direction, I decrypt the ciphertext and check if it start with plaintext. This works prerry fine, only cost 225ms in V8 engine.