George, Levi

CS44500 – Computer Security

04/07/2023

Lab 10 – Secret-Key Encryption Lab

Env Setup.

I created my desired directory and curled the setup files from the seed website

Graphical user interface, text

Description automatically generated

I unzip the curled file and check the contents

Text

Description automatically generated

After unzipping, I ran dcbuild to have the containers ready to go for Task 6.

Text

Description automatically generated

Task 1:

Text

Description automatically generated



I ran the frequency mapping command

Text

Description automatically generated

Text

Description automatically generated

Text

Description automatically generated

Given that “n” is the most common letter and the 3-gram “ytn” is the most common. I assume that “n” = “e” and “ytn” = “the”



Using the second most common trigram in the English language I swapped, vup with AND

Then, I noticed that ixT could be LOT, Tx could be TO, and lHx could be WHO.



So, I swapped ix with LO, since this worked, I’ll also swap l with W

I found the phrase AT LEAqT, so I decided to swap q with S



My current list of cipher symbols to deciphered symbols





This phrase in the translated ciphertext makes me think of ARE FEW AND FAR BETWEEN

So, I’ll swap out h to R, b to F, and g to B

It seems that this says A NATIONAL CONVERSATION



So, I’ll swap out m to I, a to C, and f to V

These three snips make me think that I should switch r to G, c to M, z to U, and e to P

My most recent tr command







The above snippets, I am sure the d is Y, j must be Q, and s is K

Based on this, o is J, k is X, and by elimination w must be Z

This was the final command I ran to get the plaintext



Below is the final output

Text

Description automatically generated

A black screen with white text

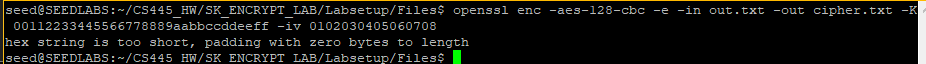
Description automatically generated with low confidence

Task 2:

We see from the MAN page on ENC that there are numerous encryption modes we can use. Not all of them need be AES, hence bf-cbc which uses blowfish

I ran my first OPENSSL encryption using aes-128-cbc

We know that AES is the specific algorithm of encryption, 128 is the total number of bits used, and CBC is the specific encryption mode (in this case Cipher Block Chaining)



I print my cipher.txt contents

Text

Description automatically generated

I encrypted out.txt using blow fish, then printed the cipher text



One thing of note is that the blowfish encrypted out.txt is slightly smaller than the AES-128 encrypted file

Graphical user interface, text

Description automatically generated

Then I used AES-128-CFB



Printed the resulting file

A picture containing text

Description automatically generated

To decrypt my AES-128-CFB file I ran this command, the -d command instead of e will reverse your encryption.

A computer screen capture

Description automatically generated with medium confidence

Task 6: Initial Vectors and Common Mistakes

Part 1. IV experiment

I declared my secret message



I encrypted it three times, twice with the same IV, once with a different IV

Graphical user interface, text

Description automatically generated with medium confidence

I cat the files, and the first two are the same, the third is different.

Text

Description automatically generated

This shows us that if we continue to use the same IV, our entire encryption scheme can be broken.

A screenshot of a computer

Description automatically generated with medium confidence

By running this plaintext and ciphertext (P1 and C1) through XOR, I can get back the plaintext. However, if I try to do this with the C2 (which has the unique IV)

I get back nonsense

A screenshot of a computer

Description automatically generated with medium confidence

Part 2 – USING THE SAME IV



Graphical user interface

Description automatically generated

I converted my plaintext message into hex, as well as my two ciphertext messages

Text

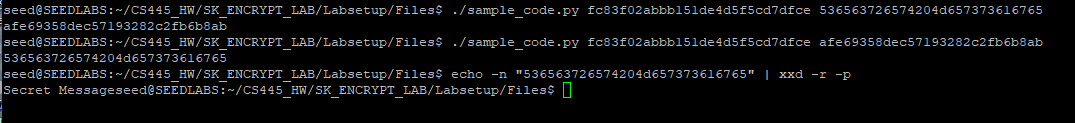
Description automatically generated

I made some modifications to the sample\_code.py

Text

Description automatically generated

The first thing I did was convert to hex then XOR to try and get my original message, which I do get back, so I can use my first hex value “afe…” to try and decrypt my second encrypted message.



Text

Description automatically generated

I was able to decrypt part of the message, but part of it wouldn’t fully print out. Despite me using the full hex encoded message.

This works with OFB but not CFB, this is because OFB just applies the key and IV then gives you your ciphertext. But CFB works with an extra piece of information that we need and that is the previous ciphertext, which acts as the IV for the subsequent encryptions.

I think that we could get partial decryption, but not total. Since our initialization vector is passed on to the next through each ciphertext we can possibly get some of the information back, but the use of the ciphertext as part of the subsequent encryption provides improved message security.

Task 6 Part 3

I got into the Encryption Oracle

Text

Description automatically generated

I guess that Bob’s response is Yes.

I convert my guess to hex and pad it with 00s, then XOR it twice (using Bob’s IV and then the Next IV)

Graphical user interface, text

Description automatically generated with medium confidence

Text

Description automatically generated

Then I go to encrypt my value into the oracle, while I do get Bob’s ciphertext back, I also get an extended portion for which I am not sure why it is generated. However, I can confirm that the Bob’s response is “Yes”

Text

Description automatically generated

Task 7:

For task 7, I started with the basic algorithm

-open a file

-read the file for each line

-remove the EOL char, format it with the “#” padding

-convert given info to proper format for use

- encrypt the information

-compare it the output (I let the program do this for me)

Below is the actual implementation

Text

Description automatically generated



Below is an example of it running and finding the appropriate key.

Text

Description automatically generated