Lab 3 Report

Random J. Protocol-Designer has been told to design a scheme to prevent messages from being modified by an intruder. Random J. decides to append to each message a hash of that message. Why doesn't this solve the problem? (We know of a protocol that uses this technique in an attempt to gain security.)

* Anybody can generate and add a hash to any message. An attacker can modify the message and add the recomputed hash value which won’t be detected at the end.

Suppose Alice, Bob, and Carol want to use secret key technology to authenticate each other. If they all used the same secret key K, then Bob could impersonate Carol to Alice (actually any of the three can impersonate the other to the third). Suppose instead that each had their own secret key, so Alice uses KA, Bob uses KB, and Carol uses KC. This means that each one, to prove his or her identity, responds to a challenge with a function of his or her secret key and the challenge. Is this more secure than having them all use the same secret key K? (Hint: what does Alice need to know in order to verify Carol's answer to Alice's challenge?)

* It is not more secure. Alice needs to know Carol’s secret key to verify Carol’s answer to a challenge from Alice. She also needs Bob’s secret key to verify Bob’s answer. They all need each other’s secret keys to verify each other. Therefore, Bob could still impersonate Carol to Alice because he would have Carol’s secret key to answer Carol’s challenges. Even if this is more complex, it is not more secure.

As described in §[2.6.4](file:///Volumes/Hackintosh%20HDD%201/CSCE%20465/Network-Security-Textbook-CSCE-465/ch02lev1sec6.html#ch02lev2sec20) Downline Load Security, it is common, for performance reasons, to sign a message digest of a message rather than the message itself. Why is it so important that it be difficult to find two messages with the same message digest?

* If there are 2 messages that have the same digest, then it’s possible to forge the messages without any repercussions. This is because the fraudulent activities are hard to detect if there is more than one message with the same digest.

Token cards display a number that changes periodically, perhaps every minute. Each such device has a unique secret key. A human can prove possession of a particular such device by entering the displayed number into a computer system. The computer system knows the secret keys of each authorized device. How would you design such a device?

* This kind of authentication scheme is part of "two-factor authentication". In many popular 2FA solutions the user owns a small calculator-size device with a preconfigured PIN key. Upon entering the PIN, a one-time password is generated. By entering the generated password, associated with his username, the user "proves" he has the device and knows the PIN code.

How many DES keys, on the average, encrypt a particular plaintext block to a particular ciphertext block?

* Only 1 DES is needed to encrypt plaintext to cipher text, however the key is used 16 times. Once for each round that the plaintext goes through.

Suppose the DES mangler function mapped every 32-bit value to zero, regardless of the value of its input. What function would DES then compute?

* In this scheme, each DES round just swaps L and R. So, after 16 rounds, the initial 64-bit word would be unchanged. All that would happen is the initial permutation, left and right swap and the final permutation. If the swap was not there, DES would do nothing at all.

The pseudo-random stream of blocks generated by 64-bit OFB must eventually repeat (since at most 264 different blocks can be generated). Will K{IV} necessarily be the first block to be repeated?

* IV will be the first block to repeat.

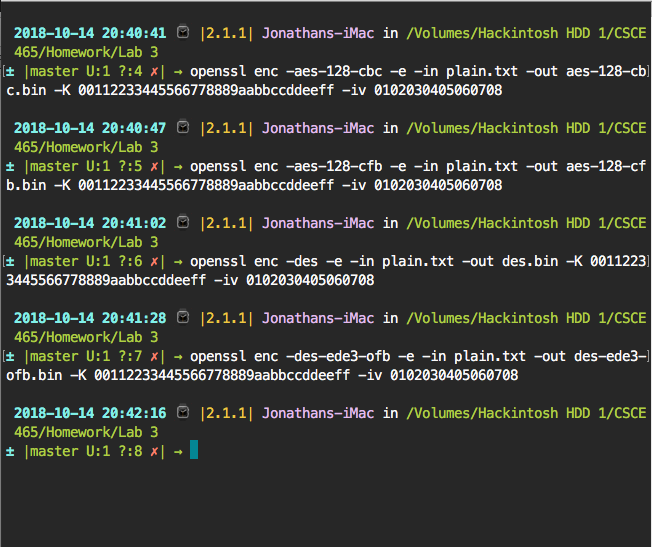
What is a practical method for finding a triple of keys that maps a given plaintext to a given ciphertext using EDE? Hint: It is like the meet-in-the-middle attack of §[4.4.1.2](file:///Volumes/Hackintosh%20HDD%201/CSCE%20465/Network-Security-Textbook-CSCE-465/ch04lev1sec4.html#ch04lev3sec4) Encrypting Twice with Two Keys.

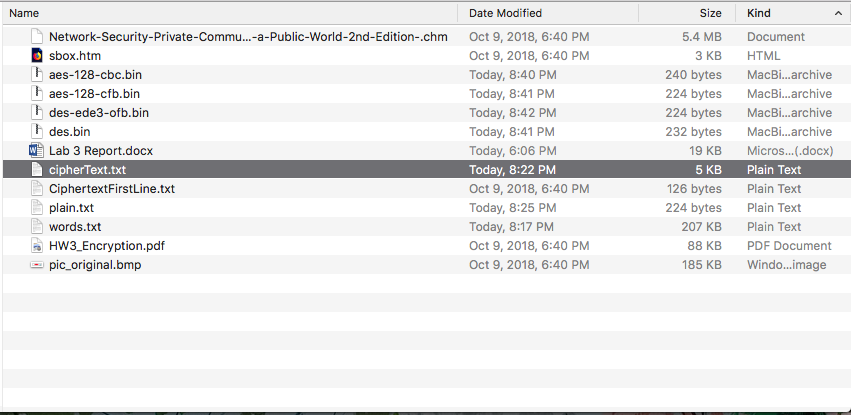
* To find a triple of keys, select a pair of keys at random and decrypt with the second key. Randomly select a pair of keys, the decrypt ‘c’ with the second key. Now perform encryption on the result with the result key and check whether the result is in the created table. If in the table, then you have the triple, otherwise try again.

Consider the following alternative method of encrypting a message. To encrypt a message, use the algorithm for doing a CBC decrypt. To decrypt a message, use the algorithm for doing a CBC encrypt. Would this work? What are the security implications of this, if any, as contrasted with the "normal" CBC?

* Yes, it would work. There is no complex encryption other than the XOR. The concern is if the plaintext blocks are all the same, then the ciphertext blocks will all be the same, instead of the first block being differentiated by the IV.

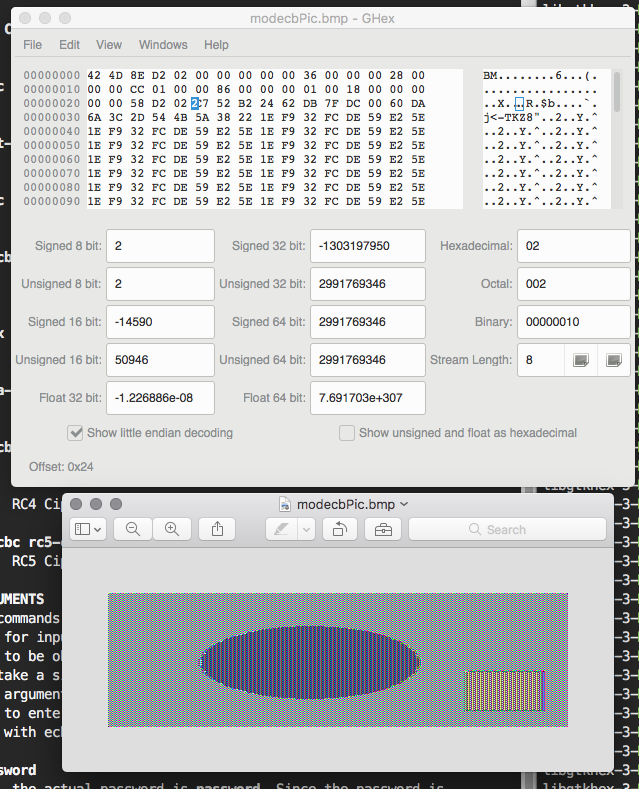
Task 1:





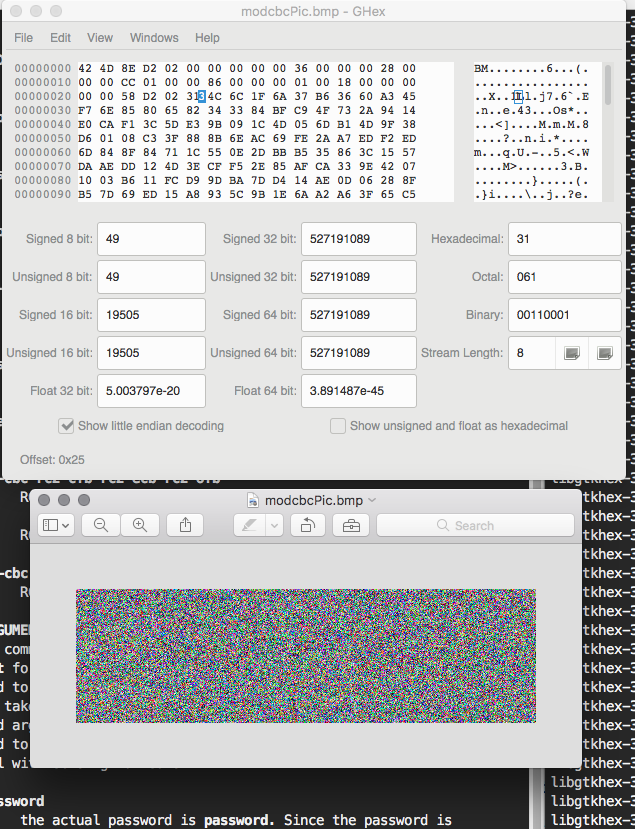
Task 2:

Encrypting with Electronic Code Book (ECB)



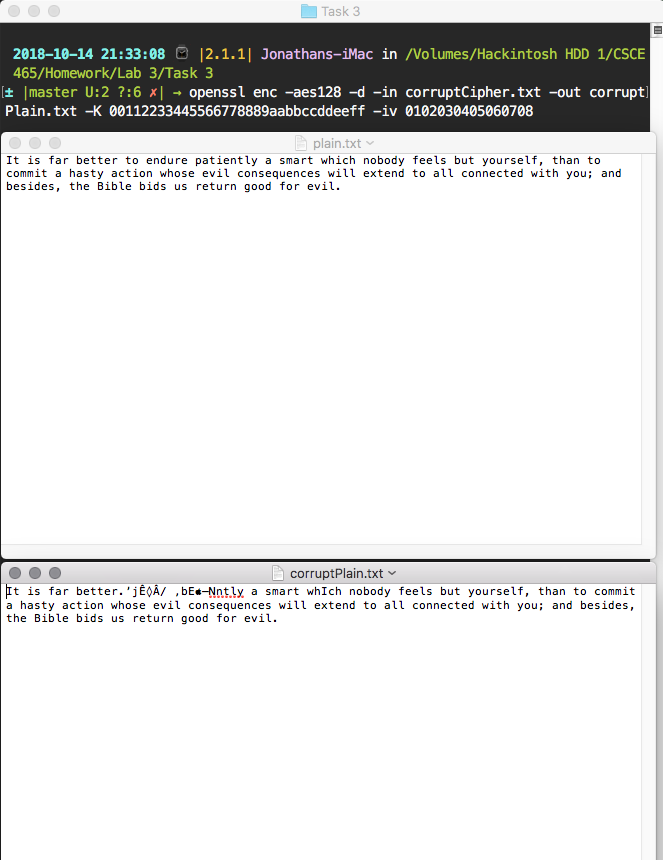
Because ECB is not very secure, it is still possible to make out the general shape of the picture. This means that ECB does not obscure the data very well.

Encrypting with Cipher Block Chain (CBC)



With cipher block chaining, the shape of the picture is completely scrambled and obscured.

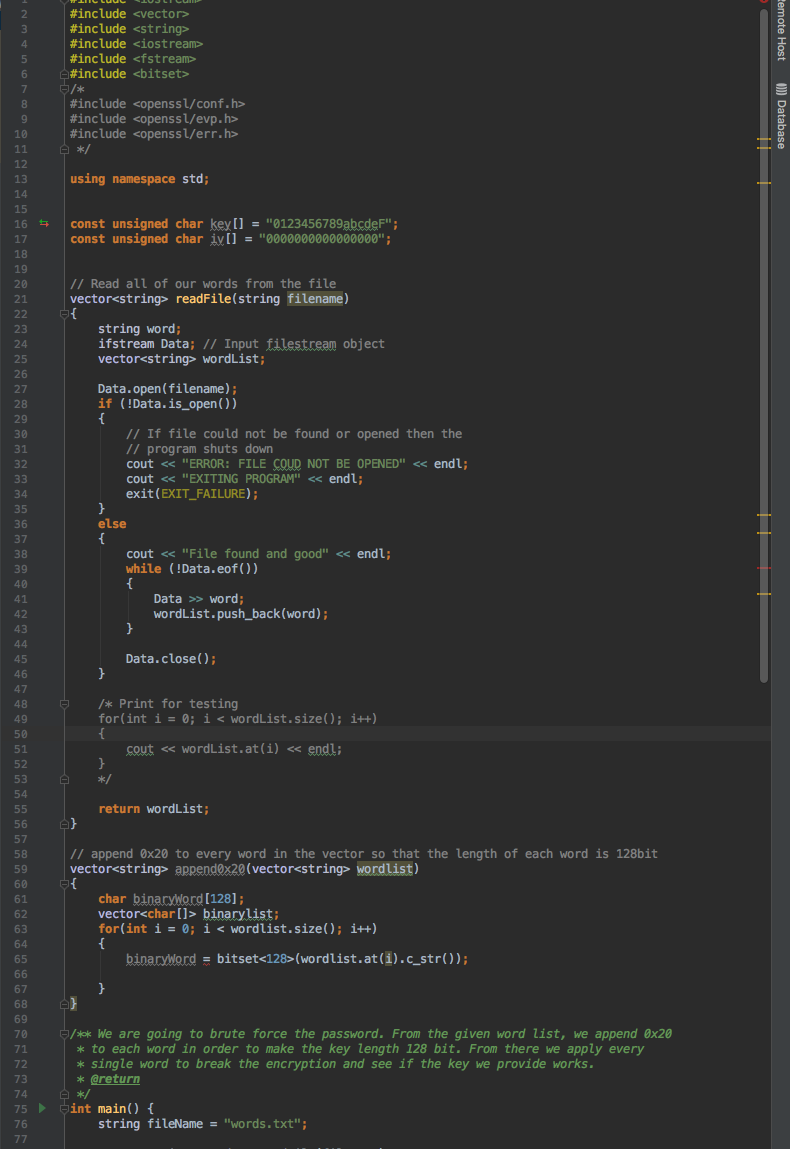
Task 3:

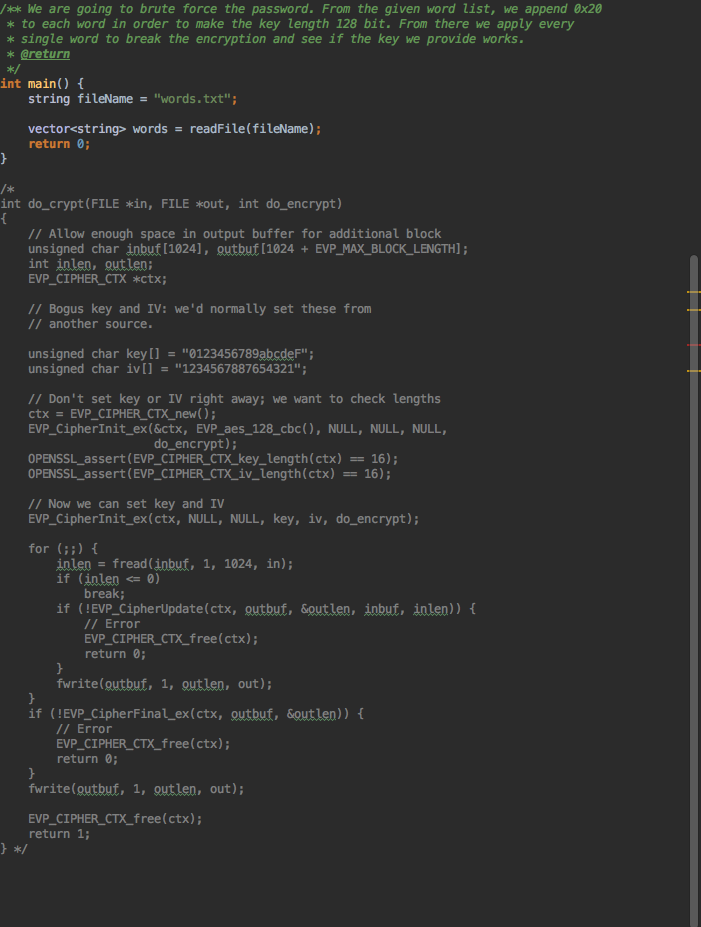


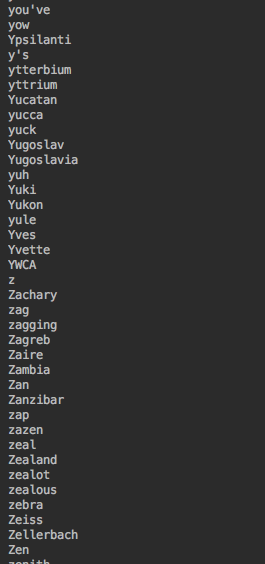
In ECB mode, only one block is affected when any problem in a cipher text happens; moreover, each block is decrypted independently. However, the corrupted bit of the 30th byte in cipher text block 8 bytes might spread to all n bits in plaintext block 8 bytes since we do the decryption one block at a time. In CBC mode, there was affect in two blocks. In CFB mode, there is problem in n / r number of blocks. Therefore, the error propagation criterion is poorer. In OFB mode, the feedback is only in the key-generation system. If the single digit of the 30th byte corrupted, then in plain text that only that byte or character is corrupted.

Task 4:

I spent all of my time on task 5 and was not able to complete task 4. In addition, I was not able to the openSSL EVP library to work with my code. I couldn’t compile it when it came to using it. My plan was to store the entire list of words into a vector. Since we know that the password is one of these words and is 128 bits long, we would append 0x20 (the space character) to the end of each word so that every word matched the correct length. From there, we apply these correct length passwords to decrypt the message. This is a brute force attempt using every single word in our word list to see if we can get a decrypted plaintext. Once we find a match (the decrypted cipher text and plaintext match up) then we declare we found the correct key.







Task 5:

DES Round Output



CODE:



