

IT430 Project: Mode of Operation

Name: Caleb Walker

Alpha: 256642

Submission  
  
Part 2 submission: \_\_ On time \_\_ Late (-5 pts)

Full submission: \_\_ Early (+5 pts) \_\_ On time \_\_ Late (-10 pts)

Overall penalty/reward: \_\_\_\_\_\_\_\_ pts

Honor

I didn’t discuss the project with any other student.

I didn’t help any other student.

I didn’t show my code to any other student.

I didn’t copy another other student’s code.

I didn’t look at the online resources directly related to the project solutions.

Initial: CMW

External source   
The only external site I referred to during the project was the documentation for the Cryptodome package.

Challenges

(E.g., it was difficult to figure out how to xxx)

What I learned

(E.g., I learned that xxx.)

Feedback to the instructor

Sometimes, the examples and instructions were a little unclear, and caused some confusion as to what the expectation was for the program.

Part 1, 2, and 3

These parts were purely coding parts included in project.py.

Part 4

The phrase: "known plaintext atk CTR"

I was able to decrypt this message by just using XOR. Looking at how CTR encryption works, I knew that if I XOR'd the plaintext with the ciphertext, I would get the ctr **after** the block encryption. Then, all I needed to do was XOR that with the target, and I would get the original message back. This all worked because the XOR negates itself when performed twice, so extracting things was as easy as simply doing an additional XOR to negate the XOR done by the encryption scheme.

Part 5

For Part 5, after looking at the CBC encryption scheme for a while, I came upon the realization that if I could feed the exact same block as one of the potential messages into the block encryption, then I could determine whether or not that given message was encrypted or not (based on whether the ciphertext matched). So first, I had to figure out what was actually fed into the block encryption, which is just the XOR of the IV and the first (and only in this case) block. Once I had that, I just needed to figure out how to get that same exact thing using the new IV given by the chosen plaintext attack. The inverse of XOR is just XOR, so by doing the XOR of the my previous XOR and the new IV, I now had the plaintext that I would use in order to essentially encrypt a message using the original IV. Then, it was just a matter of seeing whether the ciphertext matched (which it did in the example - I highlighted the matching portion since CBC pads a full length message with an extra entire 16 byte block. I also forgot to give the 'open' command to show which message was actually chosen, but hopefully my explanation suffices. Because I used 'Yes' as my working message, that was my guess, and I know that the program chose 'Yes').



Part 6

How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, or CTR, respectively?

Originally, my answer was the following: using ECB or CTR, you would be able to recover almost the entire message, but using CBC there would be a cascading effect from the corruption that would leave a large portion unrecoverable. However, to my surprise, all three of the decrypted images of the corrupted file looked almost the exact same (just a few random pixels near the bottom left of the image). Then, upon taking a closer look at how CBC decryption works, I realized that a corrupted ciphertext on corrupts two blocks, since the decrypted plaintext is not used at all in the chaining.

Decrypted ECB bmp file:



Decrypted CBC bmp file:



Decrypted CTR bmp file:



Error propagation is not an issue for ECB and CTR because there are no dependencies - every block decryption is independent of other blocks. That is an easy and intuitive answer, but CBC is not. The reason CBC has almost no error propagation is because the corrupted ciphertext block only affects two blocks: itself and the next block. It is different than encryption because the chaining XOR comes **after** the block decryption, so there really isn't any propagation (excluding next block).

Part 7

To guess the secret message, I simply provided the decryption black-box with the same IV and a slight change to the first byte of the ciphertext (since I could not simply decrypt the actual ciphertext). Because CBC does **not** have error propagation (from part 6), I knew that only the first parts of the message would be corrupted. Sure enough, the decrypted message was some garbage followed by exactly the same letters as one of the messages, and the guess was correct.



(the first hex character in the ciphertext was changed from 'a' to 'b')