# 

# IT430 Project: Mode of Operation

Name: Owen Pitchford

Alpha: 255076

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: \_\_\_TOP\_\_\_\_

External source   
(E.g., I referred to this site for xxx)

[Error Propagation](https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Error_propagation)

Class Notes

[AES](https://www.pycryptodome.org/src/cipher/aes)

Stack overflow for basic python syntax

Challenges

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

Part 3 was challenging to figure out how to check if bit errors occurred in the padding.

Part 4 was challenging to understand what to xor together to get the plaintext message.

Part 5 was challenging for me to try and figure out the correct plaintext message to put into the part5 program.

What I learned

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

I learned how vulnerable CTR becomes when the CTR is reused. It was actually very simple to decrypt another message that was encrypted with the same CTR. Additionally, I learned about error propagation in different AES modes. I also learned about the vulnerabilities of CBC when the IV is known and how to execute a known plaintext attack.

Feedback to the instructor

(E.g., This was cool. This was too much. It would have been better if …)

I think this project was useful and helped me gain a better understanding of different AES modes. Some of the more conceptual parts were confusing and I wish the class notes did a better job at explaining how to execute certain attacks in different AES modes but I do not think it was too much work compared to other projects in the CS major.

# Part 1

Code on submit server

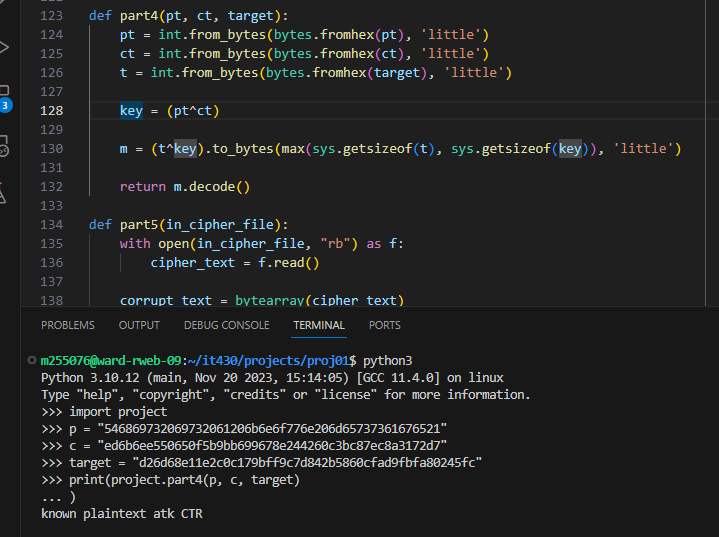
# Part 2

Code on submit server

# Part 3

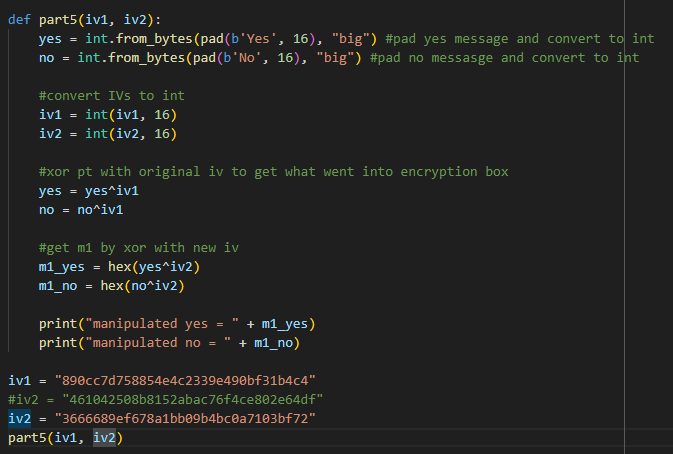
Code on submit server

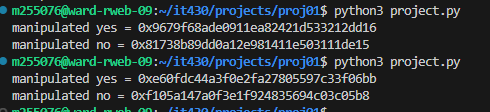
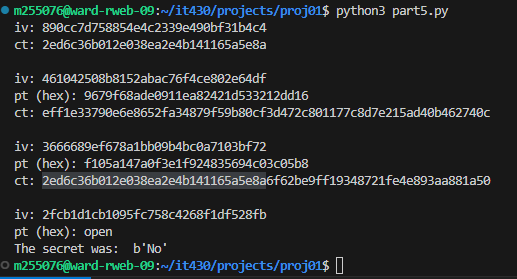
# Part 4



The phrase is “known plaintext atk CTR”

# Part 5



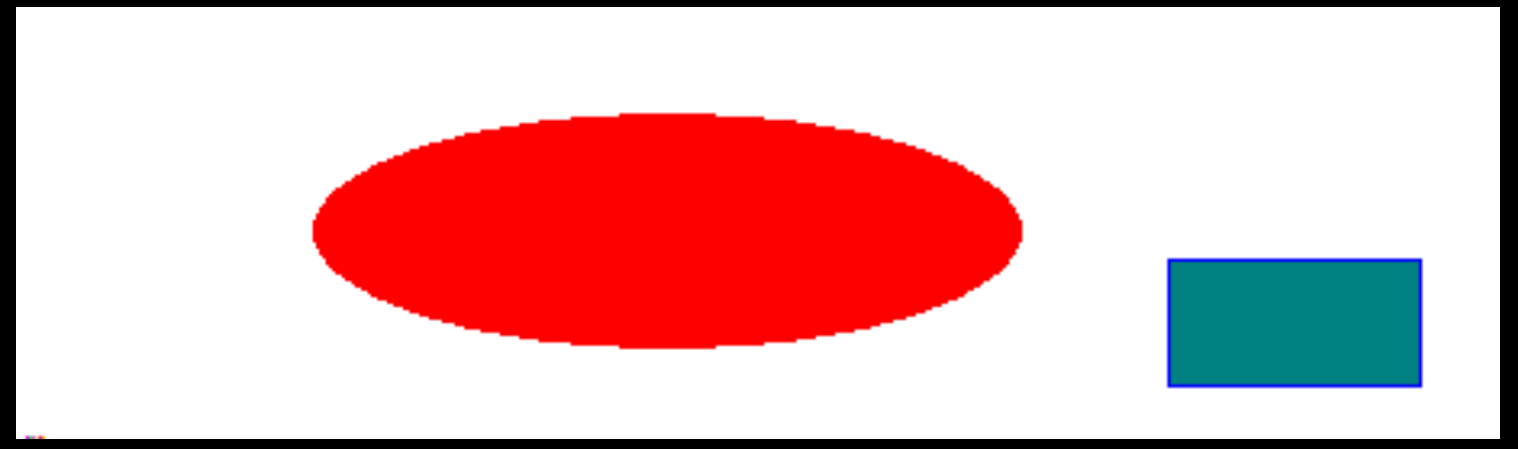


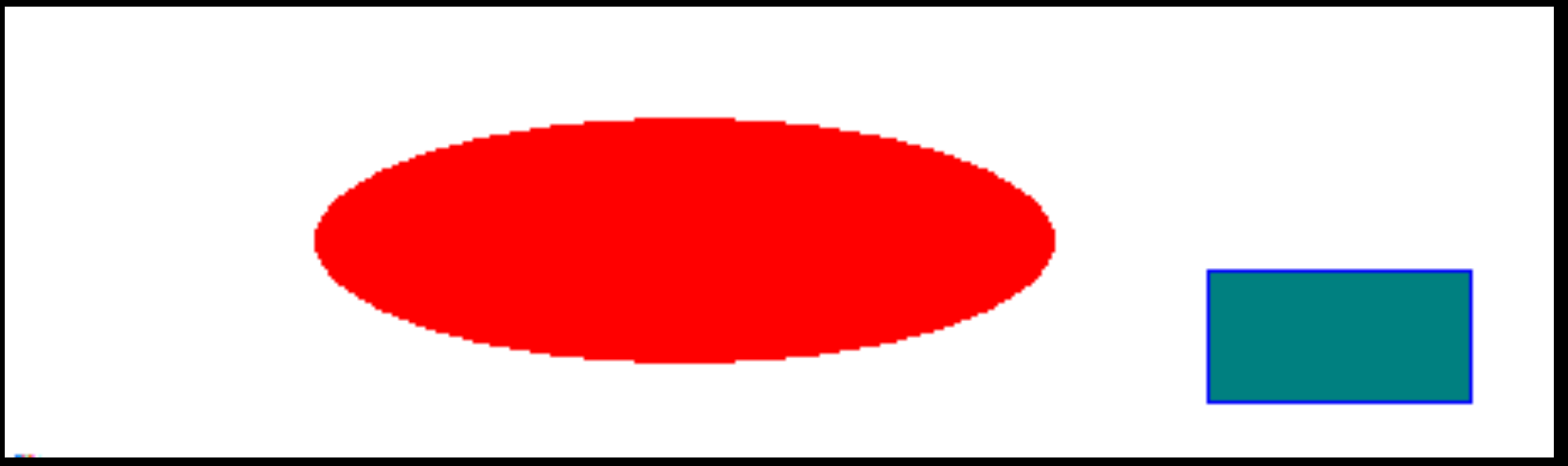
To figure out the message, I first stored the original iv (iv1) and the new iv (iv2). I used those in my function to create two hex strings to feed into the program. I inputted the manipulated yes string and if the first 16 bytes did not match up with the given ciphertext I knew it was No. To double check I updated iv2 and ran my function again to output a new No hexstring. I then plugged that in to the part5 program and the first 16 bytes of that ciphertext matched up with the given ciphertext and I knew it was No that had been encrypted.

My code is commented and explains the process I used to get the hexstrings to input into the part5 program.

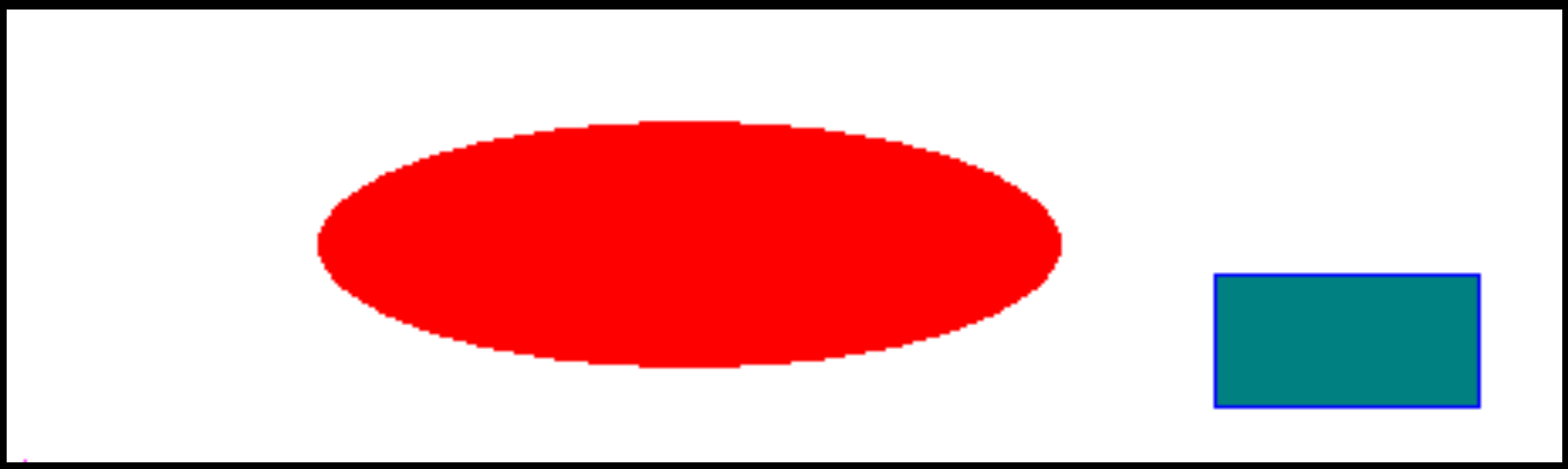
# Part 6

Q1: How much information can you recover by decrypting the corrupted file, if the encryption mode is ECB, CBC, or CTR, respectively?

ECB: Since ECB encrypts and decrypts each block separately, most of the information would be recovered except for the block that contains the corrupted byte. 

CBC: The one byte change of the ciphertext would corrupt the corresponding block of plaintext and since the ciphertext is used to decrypt the following the plaintext block, those bits would be flipped. Since we changed one byte, there will two specific byte errors in two plaintext blocks. Most of the information will be recoverable. S

CTR: The byte change of the ciphertext would lead to a specific byte error in the corresponding plaintext. Since we only changed one byte, most of the information will be still recovered because error propagation does not carry throughout CTR.



Discussion: My original answers were pretty much correct. You can see how the CBC encrypted image has more corruption than CTR and ECB because it corrupts two blocks of plaintext compared to one. Since CTR and ECB decryptions do not depend on previous blocks, bit errors in the ciphertext are in the same position in the plaintext. For CBC, bit errors will effect the corresponding block and the one following it.

# Part 7

For my attack, I made another cipher text (ct’) that is the exact same as ct except the first bit was different. I then decrypted it using the same iv and was able to determine which message was originally encrypted. I changed the first bit because that should leave the second block of the message intact.

