**=============THE FOLLOWING GIVEN TO THE PARTICIPANTS=============**

## **The Alien Oracle**

## **Challenge Category:** Crypto

## **Challenge Description:**

*Once upon a time, you are stuck in an alien planet,*

*with your only means to return home hidden within the alien message:*

|  |
| --- |
|  |
| *b'P\xe2s\x14&o\xda\x04\x124\xb3\xf2\x8d\x97\xeaG\xa4H\n5}m\t.\xe1\xaf\xa7\x0f\xc3\x8d8\x04!\x06AP\x13\xa8[\x01#\xcao\xb9\xc6\xbf^\x97\*A\x92j)MfmEL\xa5\xef\xbc.(\xfe\xc2\xf1\xb3e@Z\x91=\x19\xba\xdc\xbb\x1a\x01Eu'* |
|  |

*You have only three clues:*

* *The message comes in 16-symbol blocks, and the answer is hidden in the centre block of the message.*
* *The message originated from the Cryogenic-Bezier-Curve and not the primitive version, Ectoplasmic-Cuckoo-Boogaloo.*
* *The message is padded with Planetary-Kebab-Cosmic-Solomon-7 and obviously protected from barbaric-mass-times-acceleration attacks.*

*Fortunately, there is an alien oracle that lives in the planet, but it only speaks in alien tongue.*

*It only will say "ree" if the padding of your alien message is valid, and "yon" if the padding is not.*

*A means of communicating with the alien oracle has been provided in client-challenge.py.*

*You have only 75 hours before nightfall in this alien planet, where you will freeze over in its subzero temperatures.*

*Will you be able to return home?*

**Challenge Source Files for participants:**

client-challenge.py (skeleton code for attack)

**Challenge Server running in LEET Lab:**

l2server.py (Code for running the server, **not given** to participants)

**=============THE FOLLOWING IS FOR PROF NILS EYES ONLY=============**

## **Team Reyon**

Members:

* Kenneth ,1001092
* Weihan ,1001003
* Wei Sheng ,1000978
* Lim Zhi Han Ryan ,1000985

Challenge Name: The Alien Oracle

Category: Crypto

Topics: Chain Block Cipher (CBC) & Padding Oracle Attack

## **Summary of Challenge:**

In this CTF, we will provide a ciphertext that attackers need to decrypt to obtain the flag, and a server that acts as an oracle which will take in a ciphertext, and outputs a positive result if the padding of the encrypted message is correct, and a negative result otherwise. The padding we are using will be PKCS7 and in the form of the following:

|  |  |  |  |
| --- | --- | --- | --- |
| … | … | … | 0x01 |
| … | … | 0x02 | 0x02 |
| … | 0x03 | 0x03 | 0x03 |

And so on…

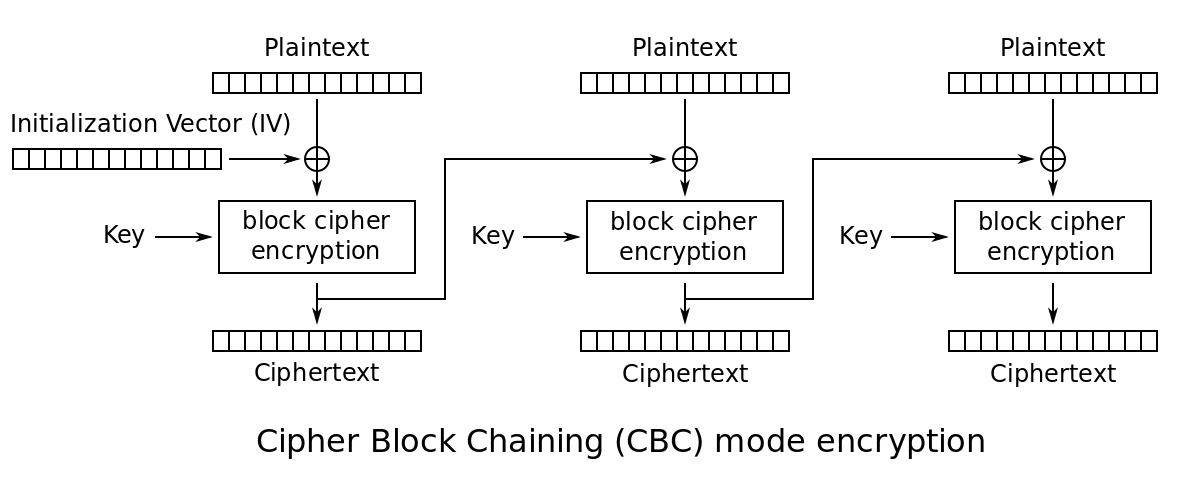
The objective of the challenge is to have attackers understand how CBC works and how the leakage of information (whether the padding of an encrypted message is correct) can lead to the confidentiality of the ciphertext being compromised.

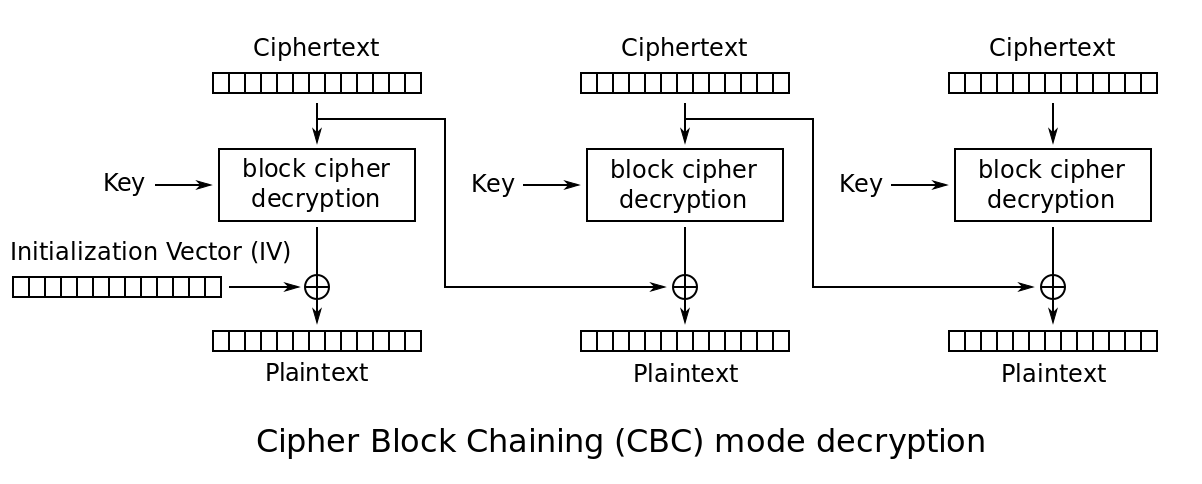
Users will perform a padding oracle attack on our system to obtain the flag. Details of an attacker’s approach and likely pseudocode are below.

## 

**Summary of Chain Block Cipher (CBC):**

In CBC mode, each block of plaintext is XORed with the previous ciphertext block before being encrypted. This way, each ciphertext block depends on all the plaintext blocks processed up to that point. To make each message unique, an initialization vector must be used in the first block.





## 

## **Summary of Attack:**

In the normal operation, the last byte is decrypted by the

XOR operation: C1[0] ^ I2[0] = P2[0]

The attacker will

XOR C1[0] with his guess, G[0], and a padding, say 0x01, to yield C1’[0].

C1’[0] = C1[0] ^ G[0] ^ 0x01.

If G[0] == P2[0]: C1[0] ^ G[0] ^ I2[0] = (C1[0] ^ I2[0]) ^ G[0] = P2[0] ^ G[0] = 0x0

Therefore the result of C1’[0] ^ I2[0] will be 0x01, which is a valid padding.

This will give the attacker the correct plaintext byte P2[0].

For the second byte, the padding byte to be XORed needs to be updated accordingly to yield a correct padding result from the oracle:

C1’[0] = C1[0] ^ G[0] ^ 0x02

C1’[1] = C1[1] ^ G[1] ^ 0x02

This process is repeated until we obtain the list of correct guesses G which will correspond to the decrypted plaintext P2, and the flag will be contained in this list in ascii.printable format.

**Pseudocode:**

(Using variables named in diagram above)

G = []

n = 0x0

while n < block\_size:

for byte in byte\_space:

for idx in range(n-1):

C1’[idx] = C1[idx] ^ G[idx] ^ n

C1’[n] = C1[n] ^ byte ^ n

submit C1’ to padding oracle

if oracle\_pass:

G[n] = byte

n++

break

## **Complexity:**

Since there are 256 possibilities per byte of ciphertext, enumerating all possibilities to find the intermediate and subsequently plaintext will take on average 128 iterations per byte, and given 16 bytes per block is will take 2048 iterations per block, which should not take very long. This attack essentially negates the need to find out what the key is or attacking the encryption algorithm(AES).

**Working Solution**:

* solution-client.py