**=============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';e\xc6\xd3\xb5\xed\xcaz\xd82\x97{`\x02\xd0\xee\xdf%\x18\xeaf\xaa/,\'3\xael\x83\xd9\xf2u\xda\'\xf5\xb0\xad"q\xfa\xf1\n\xecRZ?rh\x92{\x07\xaf@J4Y\xd2\x9a\xad9\xf0\xf4\x90\xf1'* |
|  |

*The message comes in four 16-symbol blocks, and the answer is hidden in the second and third blocks of the message. The message originated from the Cryogenic-Bezier-Curve and not the primitive version, Ectoplasmic-Cuckoo-Boogaloo. Fortunately, there is an alien Oracle that lives in the planet, but it only speaks in alien tongue. It only will say "****ree****" if your alien message submitted has proper Planetary-Kebab-Cosmic-Solomon-7 padding (like one used on Earth) and "****yon****" if it does not. A means of communicating with the alien Oracle has been provided in client-challenge.py. You realise you are missing essential information for the first block, but the Oracle says it is irrelevant.*

*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…

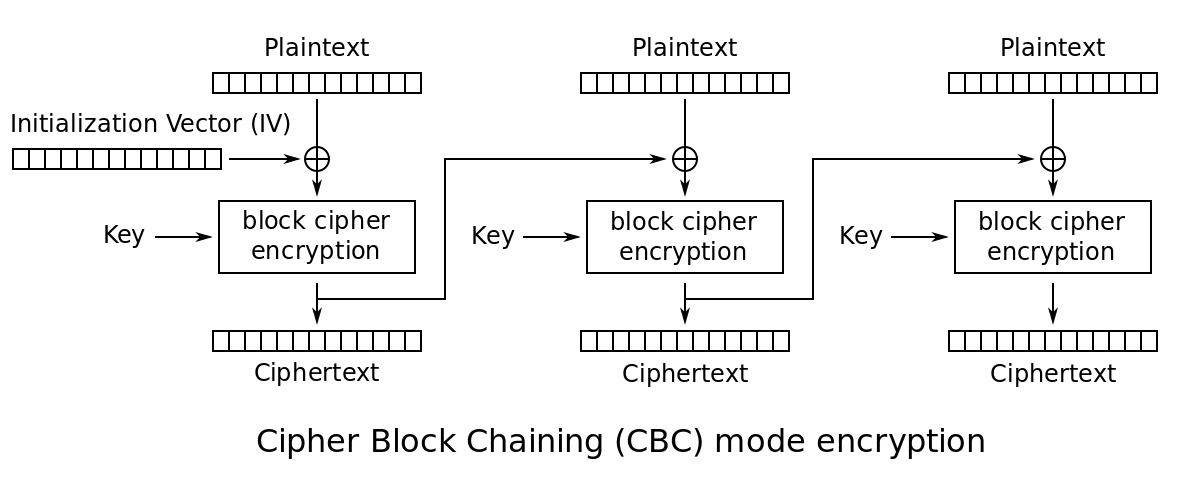
The objective of the challenge is to have attackers understand how CBC works and how the leakage of information (whether the padding of an encrypted message is correct) can lead to the confidentiality of the ciphertext being compromised.

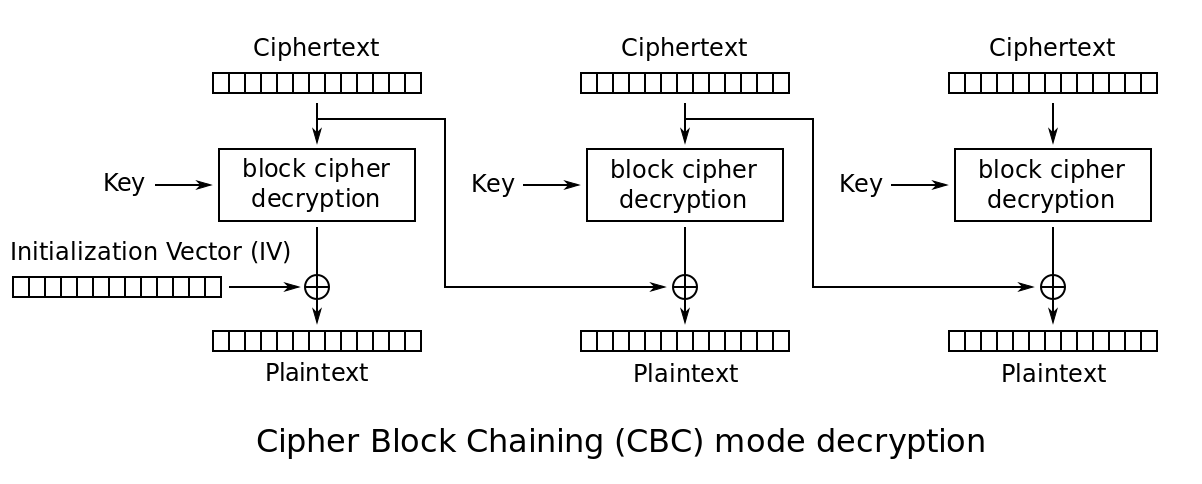
Users will perform a padding oracle attack on our system to obtain the flag. Details of an attacker’s approach and likely pseudocode are below.

## 

**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:**

We can pass any ciphertext to the oracle, and the oracle will reply whether the decrypted ciphertext is a plaintext with a valid padding or not. We can use the padding information to exploit CBC.

We first craft a message C1’ + C2, where C1’ is a cipher block specially crafted by us, and C2 is the ciphertext block we are trying to decrypt.

To find the last byte of P2, we first need to craft C1’.

We keep C1[0]…C[15] the same, and C1’[16] to be C1[16] ^ guess ^ 0x01, where ‘guess’ can be any byte value from 0-255

If our ‘guess’ is correct, I2[16] ^ C1[16] ^ guess ^ 0x01 = 0x01, as I2[16] ^ C1[16] ^ ‘guess’ will be cancelled out.

This will result in a plaintext with a valid padding ending with 0x01. When C1’ + C2 is sent to the oracle, he will reply that the message has a correct padding. This is where we know the last byte of P2 is in fact our guess, let’s say *g[16].*

However, if the oracle replies with a wrong padding, we know that we have made a wrong guess. We will then vary our ‘guess’ until we receive a positive response from the oracle. The guess which gives a positive response in the loop will be the last byte of the plaintext.

To guess the second last byte of the plaintext, we need to make the plaintext padding 0x02 0x02.

Again, we craft a message C1’ + C2, where C1’ is,

C1’ = C1[0]…C[14] + C1[15] ^ guess ^ 0x02 + C1[16] ^ *g[16]* ^ 0x02

This message C1’ + C2 is passed to the oracle to be checked for the correct padding.

The last byte of the plaintext will always be decrypted to 0x02, as C1[16] ^ correctly guessed byte will cancel out.

We now need to change the second last byte of the plaintext to 0x02.

Similarly, we loop our guess from a range of 0 to 255,

If our guess is correct, I2[15] ^ C1[15] ^ guess ^ 0x02 = 0x02, as I2[15] ^ C1[15] ^ guess will be cancelled out.

This will result in the oracle replying with a positive response. This way we will know the correct 15th byte *g[15].*

This is repeated until all the bytes of the plaintext is found.

**Note:** The first block of the plaintext cannot be decrypted as we did not provide the Initialisation Vector to the participants, however it is irrelevant.

**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