Padding Oracle Exercises:

Questions

1. What does CBC stand for?

2. Below is diagram depicting CBC, figure out what each colour means with these options:

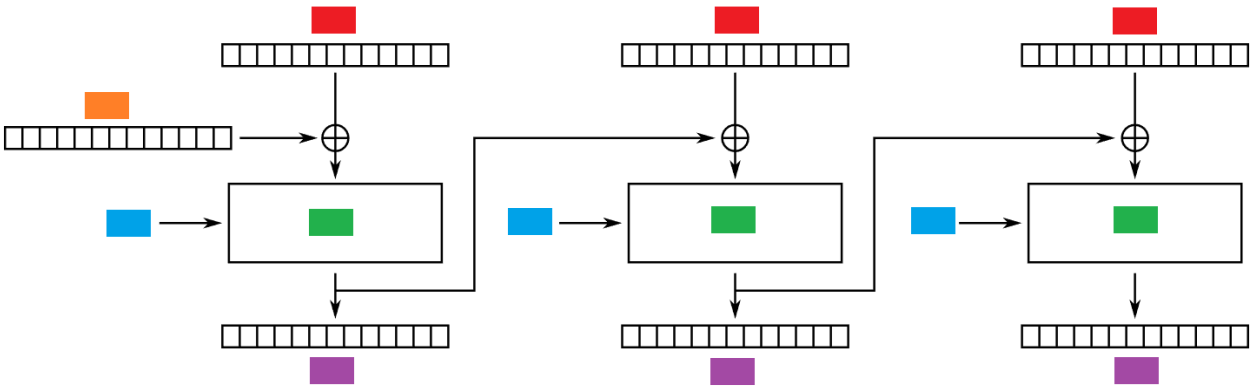
    1. Block cipher encryption

    2. Ciphertext

    3. Initialisation Vector (IV)

    4. Key

    5. Plaintext



3. Briefly explain CBC.

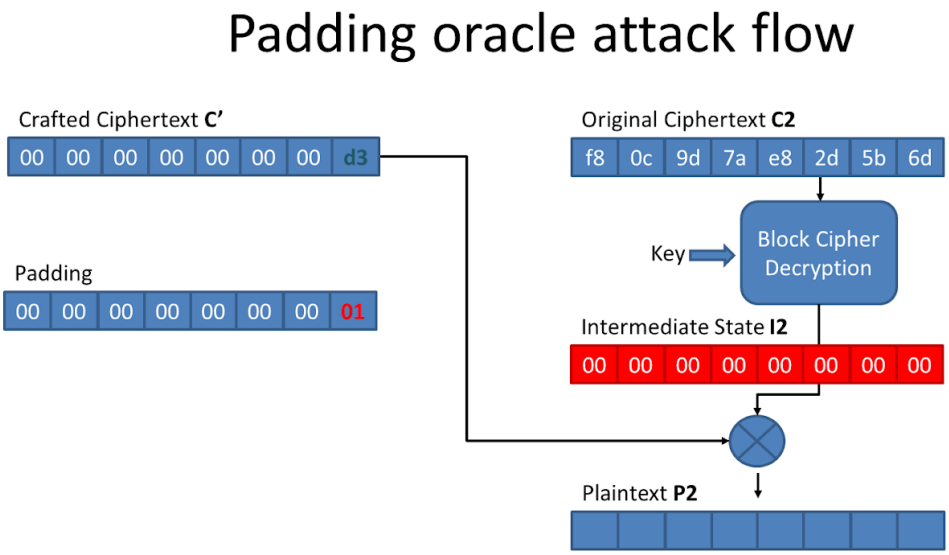
4. What is the Initialisation Vector(IV)?

5. Should the IV be a random value?

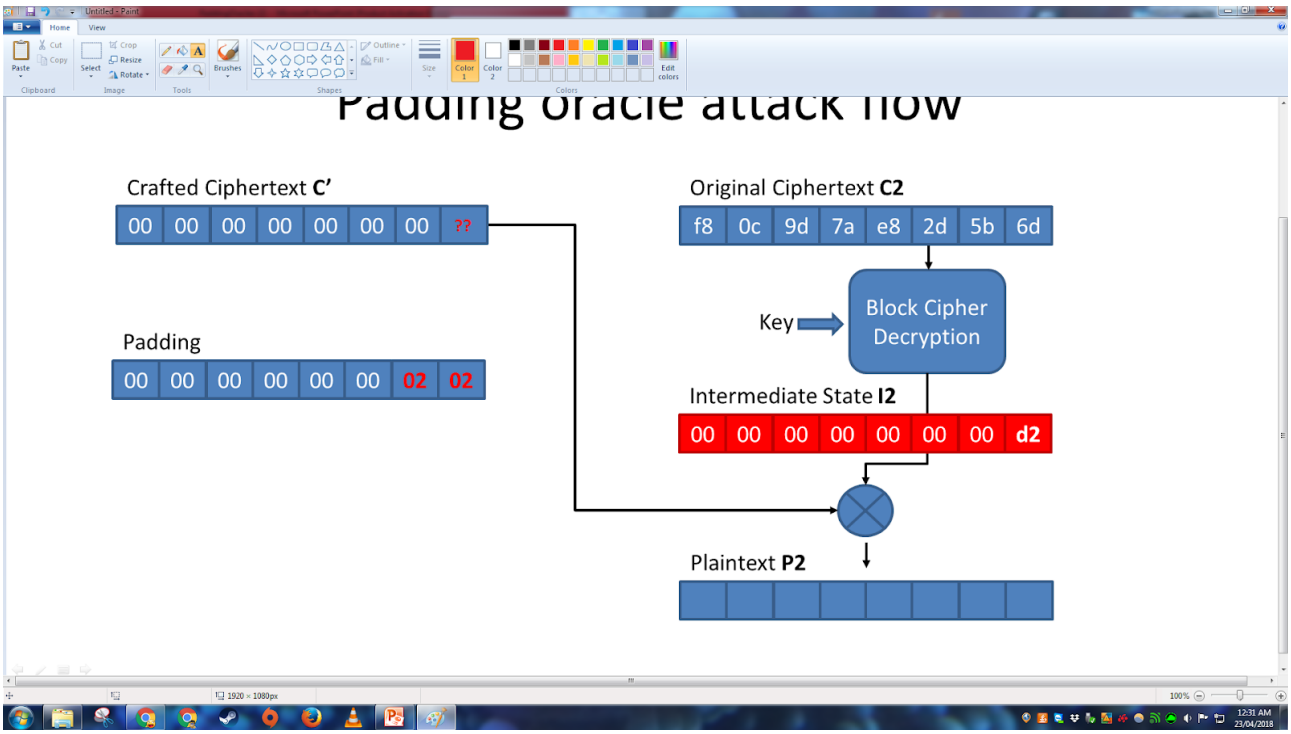
6. Explain the rules when different bytes of padding is needed (PCKS#7). What if no bytes padding are needed?

7. What is the difference of these two errors(pad error, MAC error) in padding oracle?

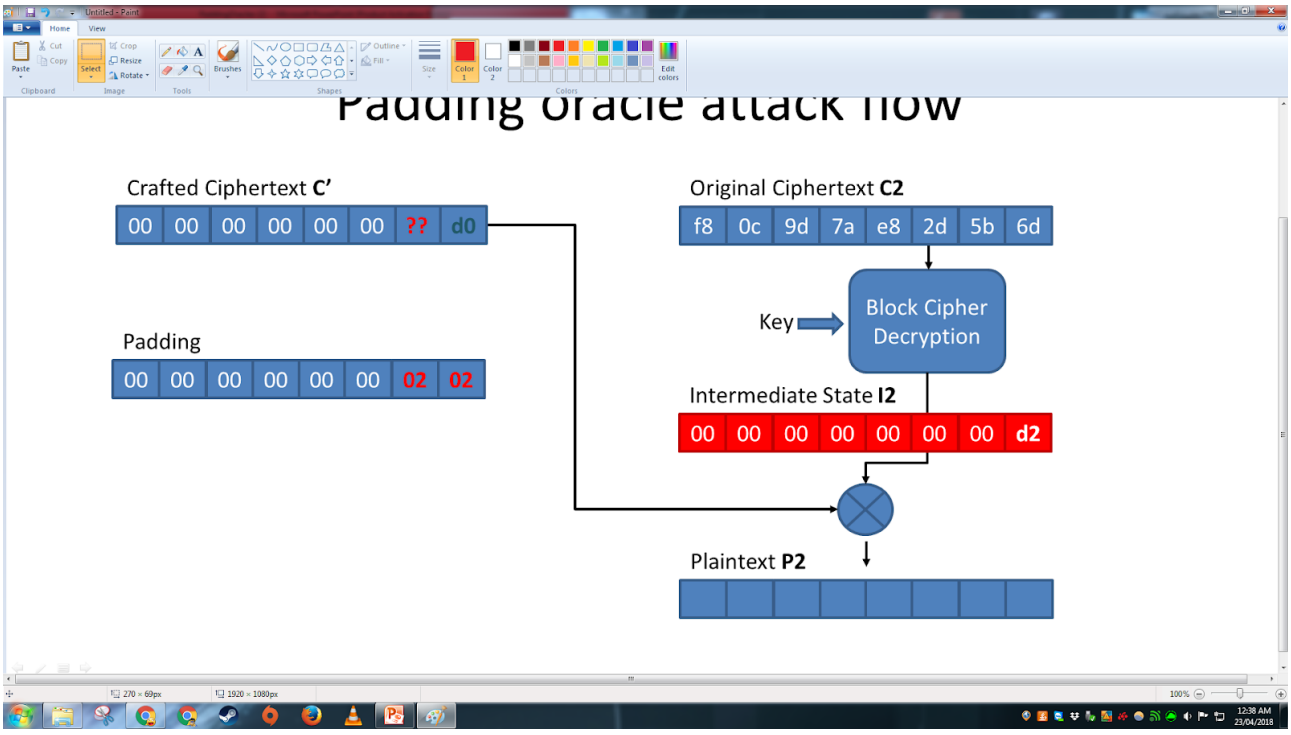
8. Below are the states for when we begin the decryption attack. We control the first block **C'** in order to find from the oracle that a successful padding of 0x01 occurred. It turns out 0xd3 in **C'** gives us a successful padding. What is the first intermediate state value in **I2**?



9. Now that we worked out 0xd2 is the first value in **I2**, we can target our next padding of two 0x02. What value in **C'** marked as ?? is required to be XORed with **I2**'s 0xd2 to provide a 0x02 in the padding?

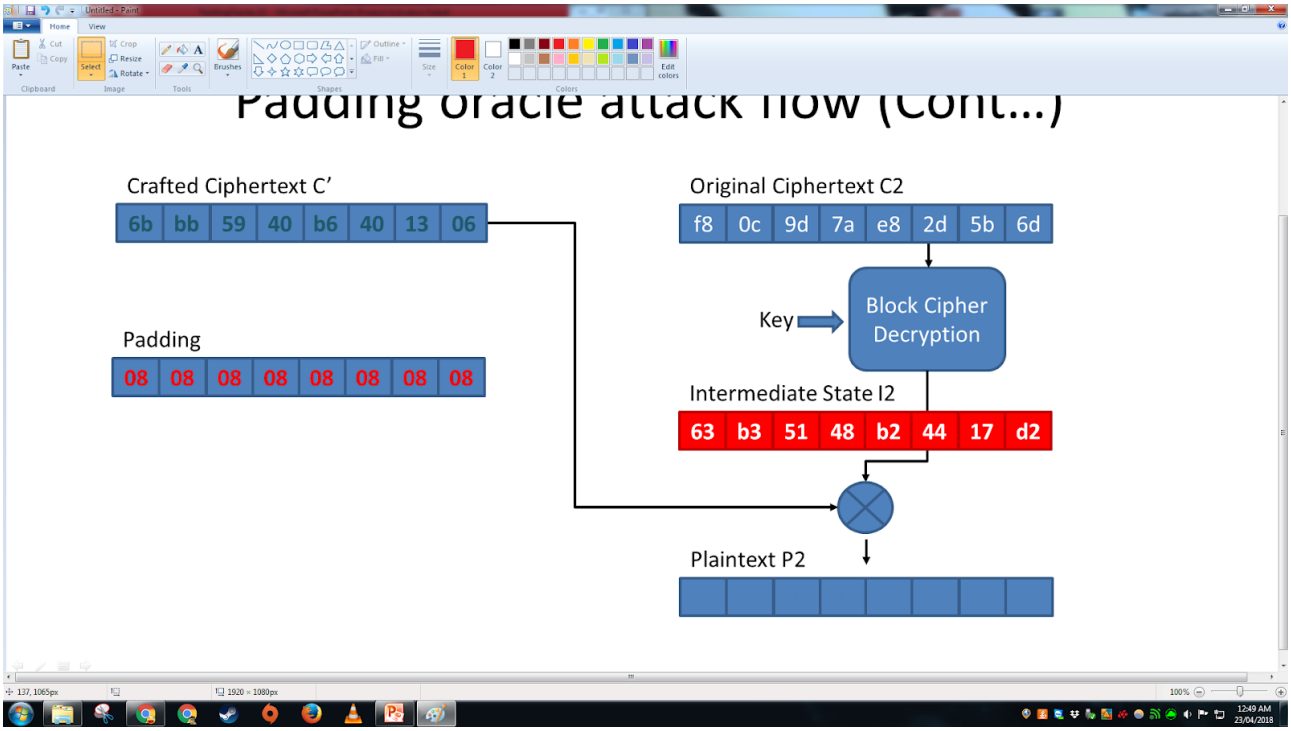


10. This is a very similar case to Question 8. What's the most amount of tries required for ?? in **C'** before we get a sign from the oracle that a successful padding of 0x02 has occurred? If this ?? value turns out to be 0x15, what is the 2nd entry in **I2**?



11. This particular example has an expected padding of eight 0x08. Continuing the previous steps, we constructed C' to target each particular padding, and worked out each byte in I2, one by one.

What do we need to decrypt plaintext P2, in the place of C'?



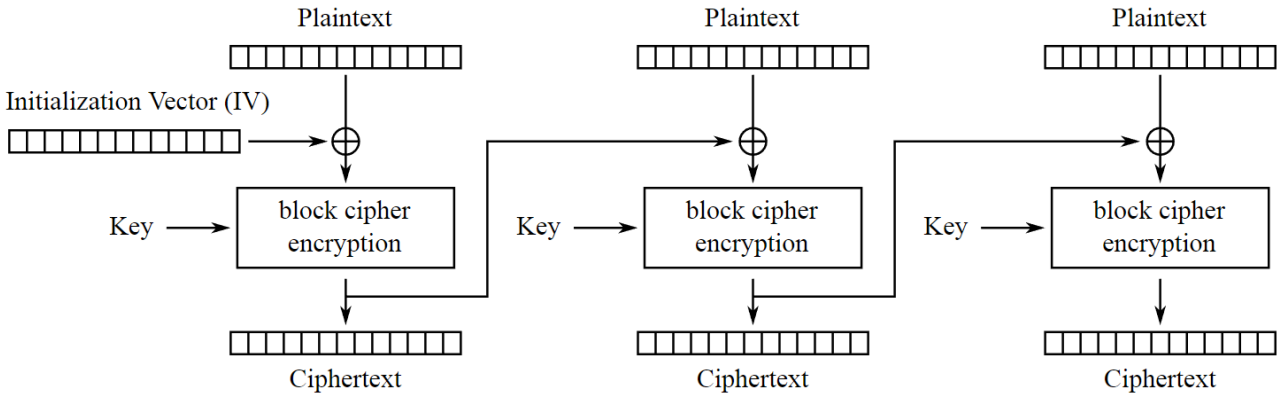
12. Can padding oracle attacks be completed in less than 256 number of encrypted bytes attempts? Why is that?

13. Can padding oracle attacks decodes the cleartext without knowing the key?

Answers

1. **Can you briefly explain what is CBC?** Cipher block chaining (CBC) is a mode of operation for a block cipher. This is when a sequence of bits are organised into blocks and is encrypted with a cipher key. Cipher block chaining starts off with an initialisation vector (IV) of a certain length and is mixed in with the input (XORed). Each block is encrypted, and the output is mixed (XORed) in with the next block's encryption.

2. **Below is diagram depicting CBC, figure out what each colour means with these options: Block cipher encryption, Ciphertext, Initialisation Vector (IV), Key, Plaintext.** Completed diagram:



3. **What is the initialisation vector(IV)?** An initialisation vector (IV) is an arbitrary number that can be used along with a secret key for data encryption. This number, also called a nonce, is employed only one time in any session. The use of an IV prevents repetition in data encryption, making it more difficult for a hacker using a dictionary attack to find patterns and break a cipher. For example, a sequence might appear twice or more within the body of a message. If there are repeated sequences in encrypted data, an attacker could assume that the corresponding sequences in the message were also identical. The IV prevents the appearance of corresponding

duplicate character sequences in the ciphertext.

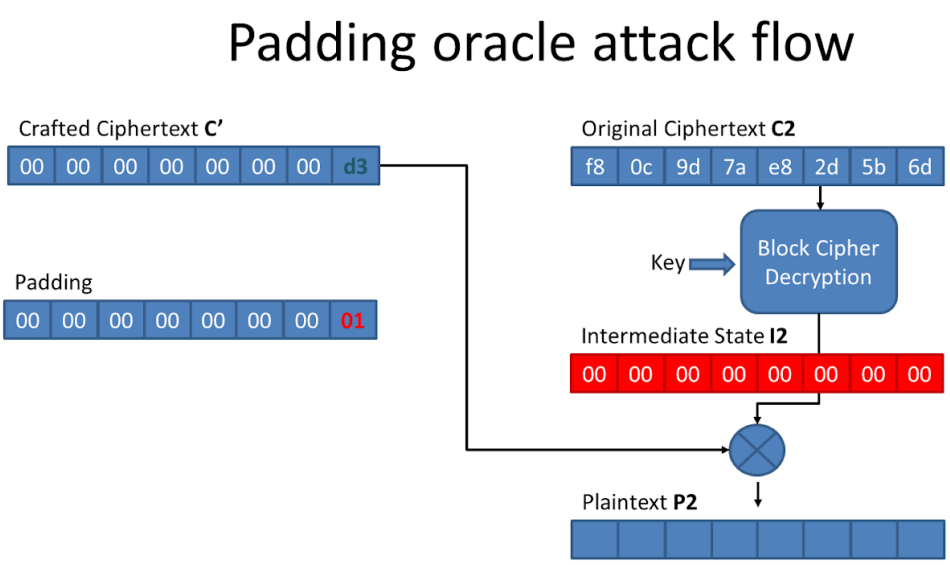
4. **Should the IV be a random value?** The ideal IV is a random number that is made known to the destination computer to facilitate decryption of the data when it is received. The IV can be agreed on in advance, transmitted independently or included as part of the session setup prior to exchange of the message data. The length of the IV (the number of bits or bytes it contains) depends on the method of encryption. The IV length is usually comparable to the length of the encryption key or block of the cipher in use.

5. **Explain the rules when different bytes of padding is needed (PCKS#7). What if no bytes padding are needed?**

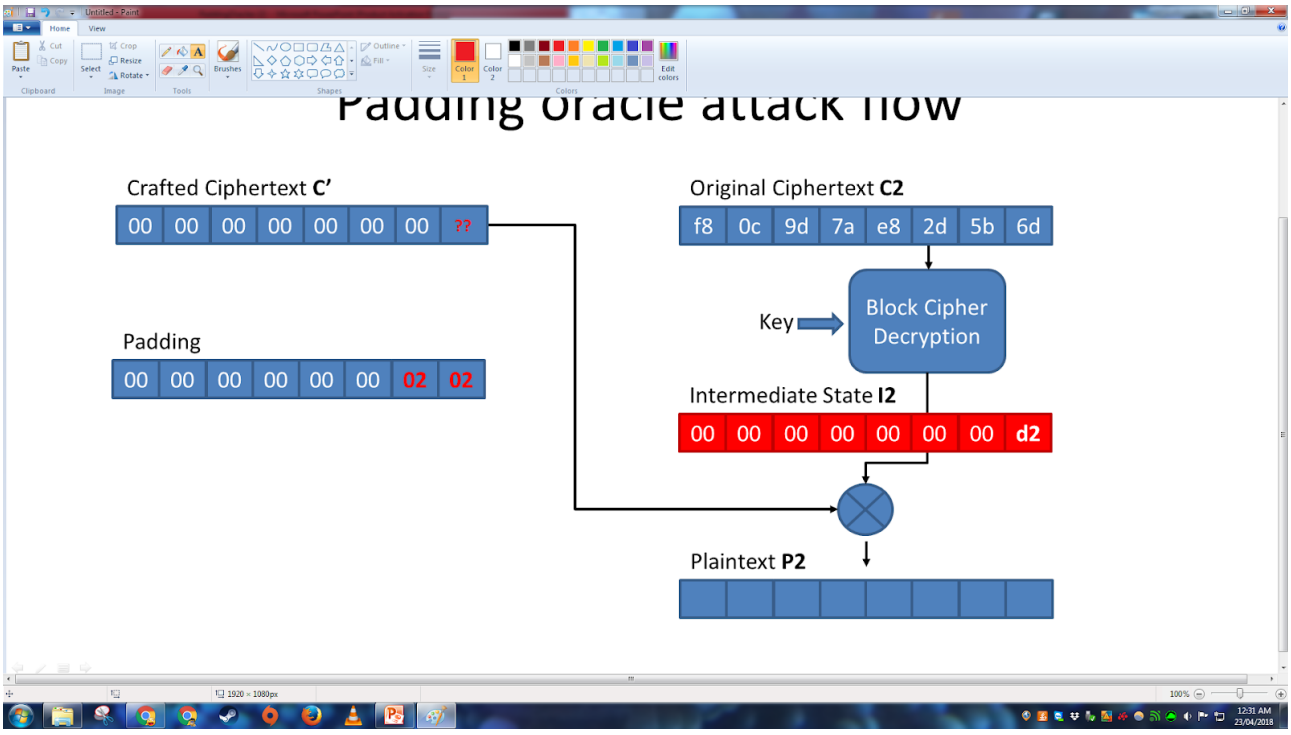
6. **What is the difference of these two errors(pad error, MAC error) in padding oracle?**

7. **What is the difference of these two errors(pad error, MAC error) in padding oracle?**

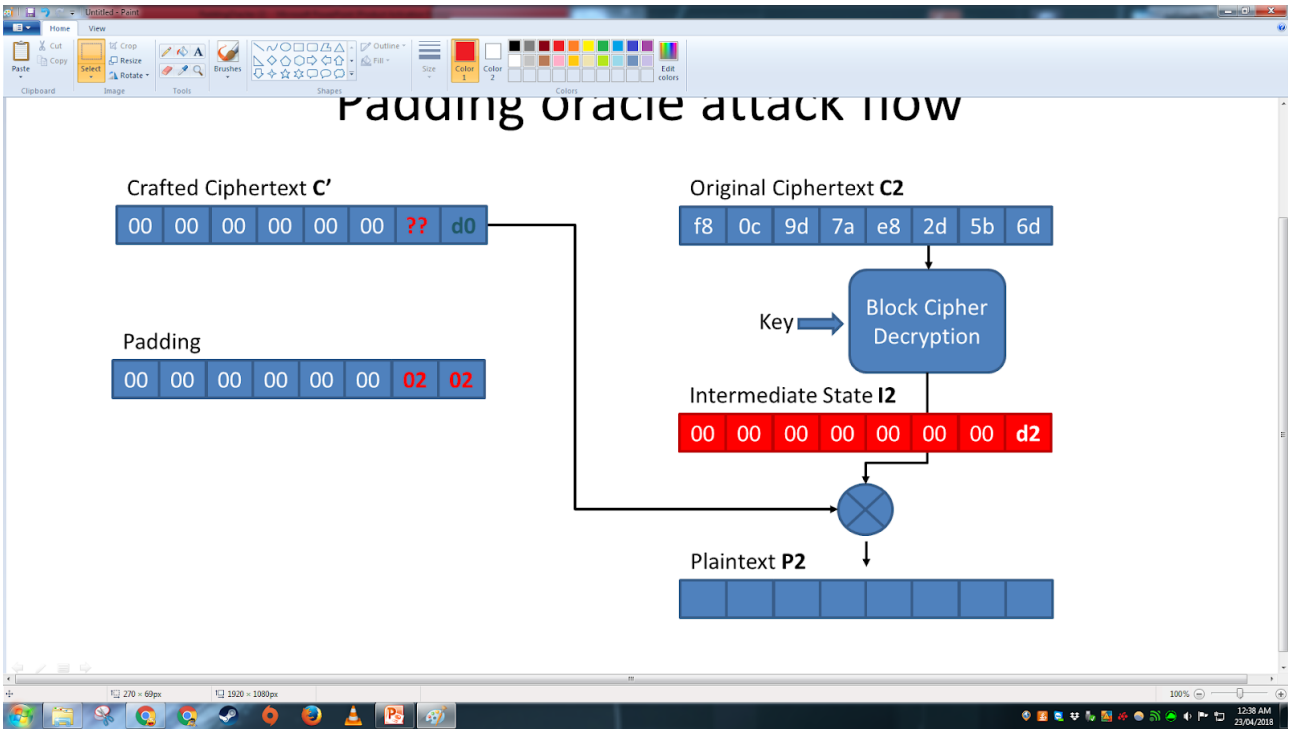
8. **Below are the states for when we begin the decryption attack. We control the first block C' in order to find from the oracle that a successful padding of 0x01 occurred. It turns out 0xd3 in C' gives us a successful padding. What is the first intermediate state value in I2?** Here we could have tried up to 256 values in C' before getting a Valid Padding, i.e. Q XOR R == 0x01. The mathematical form would be Q $\oplus$ R == 0x01. We know Q == R $\oplus$ 0x01 or in other words, when we XOR the expected padding to the random guess, we get the cleartext value of the corresponding byte. So to get the first value in I2, it is 0xd3 $\oplus$ 0x01 = 0xd2.



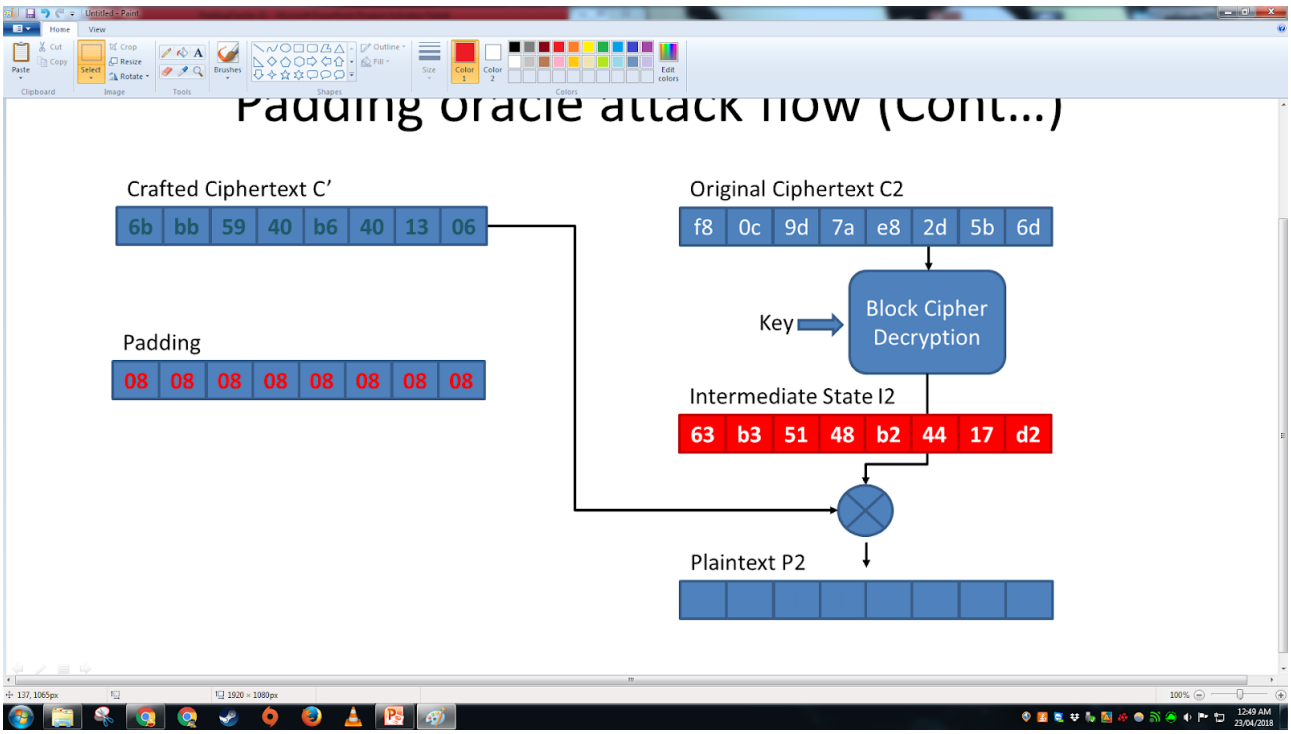
9. **Now that we worked out 0xd2 is the first value in I2, we can target our next padding of two 0x02. What value in C' marked as ?? is required to be XORed with I2's 0xd2 to provide a 0x02 in the padding?** 0x?? ⊕ 0xd2 = 0x02. Hence 0x?? == 0x02 ⊕ 0xd2 == 0xd0.



10. **This is a very similar case to Question 8. What's the most amount of tries required for ?? in C' before we get a sign from the oracle that a successful padding of 0x02 has occurred? If this ?? value turns out to be 0x15, what is the 2nd entry in I2?** Each byte could be tested in C' in order to get the oracle to respond in different ways. This means only up to 256 attempts is required at most per byte. If our second slot value (??) in C' is 0x15, the second slot entry in I2 is 0x15 ⊕ 0x02 = 0x17.



11. **This particular example has an expected padding of eight 0x08. Continuing the previous steps, we constructed C' to target each particular padding, and worked out each byte in I2, one by one. What do we need to decrypt plaintext P2, in the place of C'?** C' was crafted to get the particular padding responses that lead to I2 being worked out. Our last value found in I2 is



12. **Can padding oracle attacks be completed in less than 256 number of encrypted bytes attempts? Why is that?**

13. **Can padding oracle attacks decodes the cleartext without knowing the key?**

A: Let's assume the byte we are modifying has the value Q (for Question Mark). And we are trying all 256 values of R (for Random) , so that the result is Valid Padding, i.e.
Q XOR R == 0x01
and if you want the mathematical form, that would be
Q ⊕ R == 0x01)
Now's it's time for a *little bit* of Math. Not too much, though. We can XOR the same value to both sides of the equation and the equation will still be true. If we XOR the same number to itself, it becomes all zeros, and Q ⊕ 0x00 == Q. Therefore we now know the following:
Q == R ⊕ 0x01)
Or in other words, when we XOR the expected padding to the random guess, we get the cleartext value of the corresponding byte.

6. Once you crack the last byte, how do you crack the previous byte?

A: Once we know the value for the last byte (Byte 16) of the block N, we XOR the desired padding for a 2-byte pad into block N-1. The 2-byte padding is 0x02 0x02). In reality, we need to XOR three values together:

- The original 16-byte block that precedes the block we are attacking
- The padding (the number of bytes depends on which pad we are using)
- The guessed cleartext (initially all nulls)

If, for example, we learn that byte 16 is 'A' 0x41, then to guess byte 15, we modify the previous block by XORing the bytes

OriginalBlock XOR (0x02 0x02) XOR hex (0x00 0x41)

But I should make it clear that we are working with 16-byte blocks

- OriginalBlock
- hex(00 00 00 00 00 00 00 00 00 00 00 00 00 00 **02 02**)
- hex (00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 **41**)

and we XOR all possible 256 values into the byte we are randomly guessing, which is byte 15 in this case:

- hex (00 00 00 00 00 00 00 00 00 00 00 00 00 XX 00)

When one of the values returns valid padding, we know that the cleartext for the 15th byte is R ⊕ 0x02

And for Byte 14, we XOR the newly learned value for byte 15 into the guessed cleartext, use (0x03 0x03 0x03) as the padding we XOR, and then we try all values for byte 14. That is, we shift one byte to the left, and change the padding string to be the proper length and replicated value, shown in Table 1 above.

If we continue this, we can guess all 16 bytes of the last block.