Padding Oracle Attack

Threat Model:

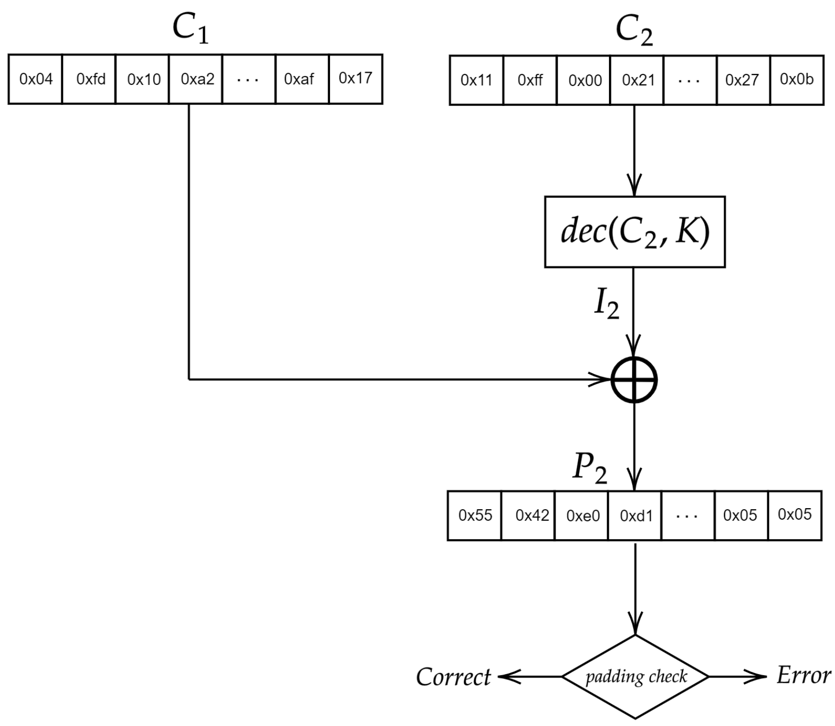
- Say we captured 2 ciphertexts C_1, C_2 from server S to client A
- We don't know the server key K thus we cannot decrypt and recover P_1, P_2
- We can however send C_1, C_2 to the server for decryption
- Note that sending C_1, C_2 is not dangerous: the server will decrypt them and reply with message: <decryption_ok>

Attack Idea:

- We will keep block C_2 constant
- We will modify block C_1 in order to to cause a padding error in the server!
- The goal is to recover P_1, P_2

Padding Oracle Attack

- Server workflow with the original ciphertext blocks C_1, C_2



- C_2 gets decrypted producing I_2 and $P_2 = I_2 \oplus C_1$
- The padding is checked returning potential error messages

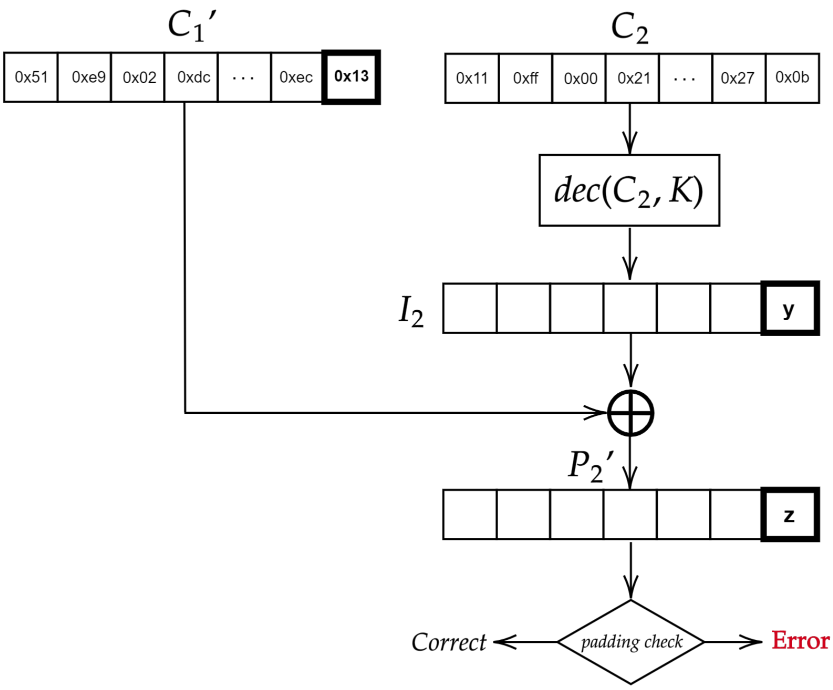
Padding Oracle Attack

- This attack scenario offers oracle access to the server
- The oracle offers a single bit of information about the decryption
  e.g. padding correct `<decryption_ok>` or paddding error `<incorrect_padding>`

$$oracle(IV, C_1, C_2) = \begin{cases} 0, & \text{if there is a padding error} \\ 1, & \text{if there is no padding error} \end{cases}$$

Padding Oracle Attack

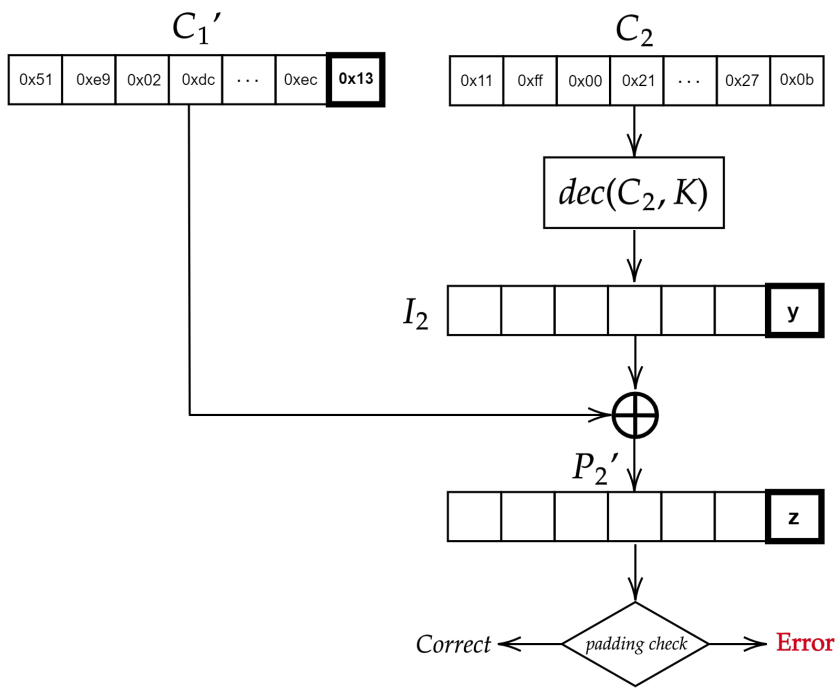
- We now replace C_1 with a random value C_1'



- I_2 remains the constant because C_2 is constant
- P_2 is replaced by P_2' because $P_2' = I_2 \oplus C_1'$

Padding Oracle Attack

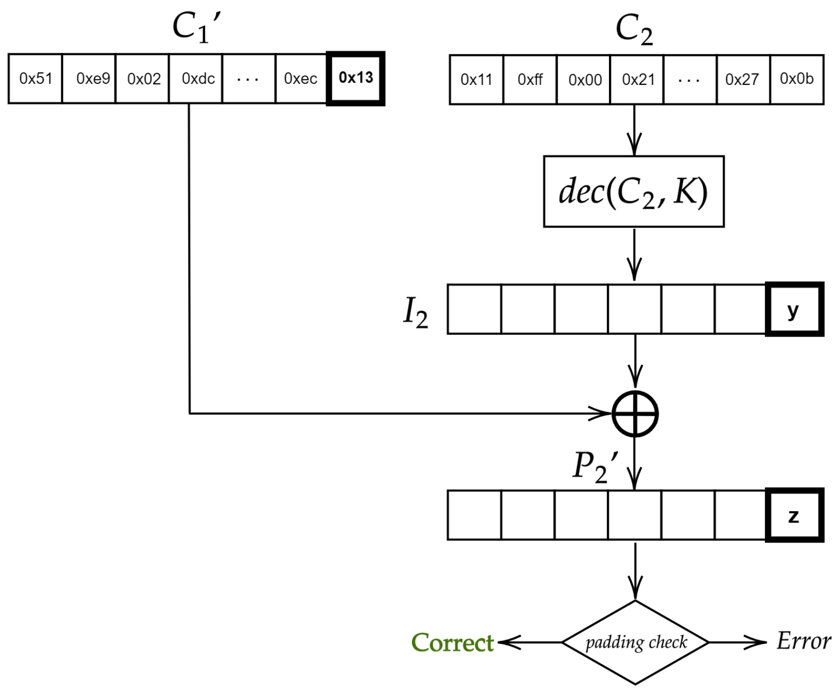
- Can we learn the value of byte z when the oracle returns 0 (error)?



- No! The value z cannot be substantially narrowed down

Padding Oracle Attack

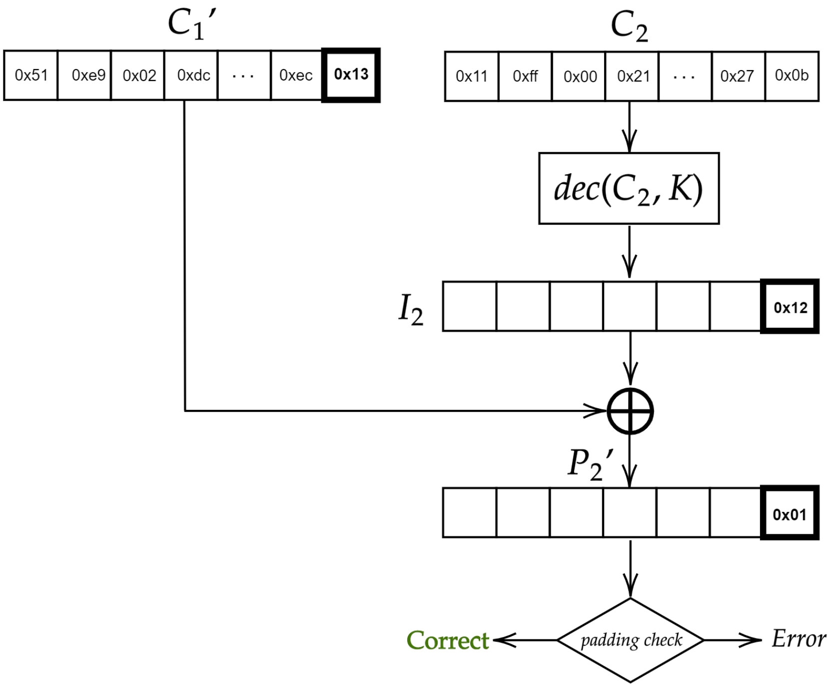
- Can we learn the value of byte z when the oracle returns 1 (correct)?



- Yes! The value z must be part of one of the valid paddings:
  i.e. $z =$0x01 or $z =$0x02 or ... or $z =$0x0f

Padding Oracle Attack

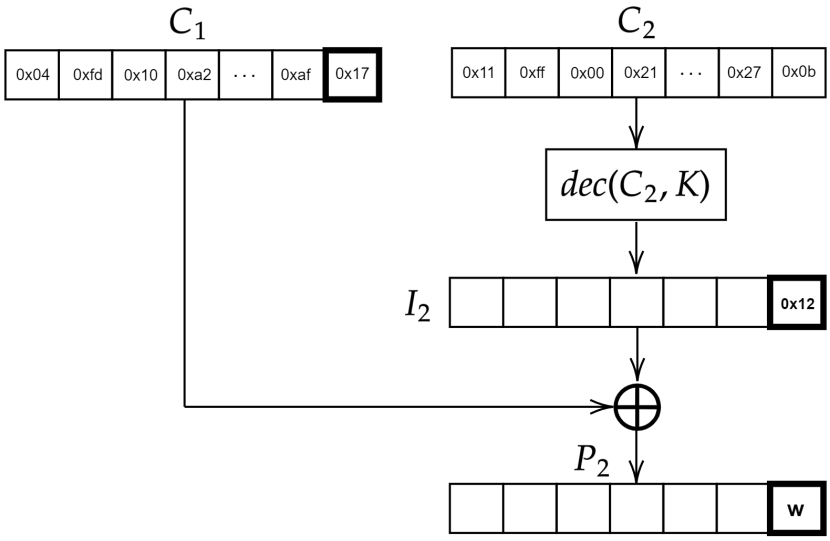
- Let's assume that $z =$0x01 (single-byte padding)
  This is one of the valid paddings (we will cover the other cases later)



- We can now easily compute the last byte of I_2
  $y = 0x13 \oplus 0x01 = 0x12$

Padding Oracle Attack

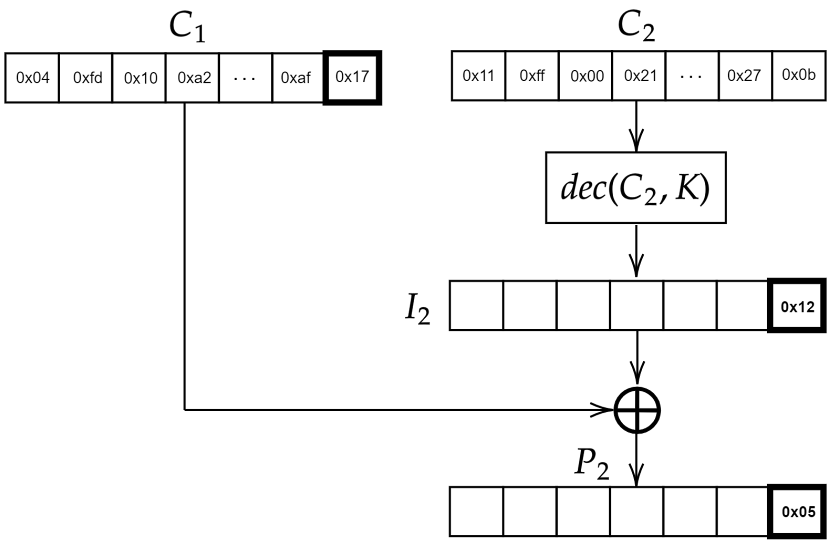
- We now return to the original ciphertext blocks C_1, C_2
  i.e. we no longer use the modified block C_1'



- Notice that we have already found the last byte of I_2
- Notice that I_2 does not depend on C_1 or C_1'
- We can now compute the last byte of P_2
  $w = \texttt{0x17} \oplus \texttt{0x12} = \texttt{0x05}$

Padding Oracle Attack

- We have now recovered the last byte of P_2



Padding Oracle Attack

Last-Byte Oracle:

Input: Ciphertext blocks C_1, C_2 and IV, cipher block size b (in bytes)
We denote with $X(i)$ the ith byte of block X, $i = 1, 2, \ldots, b$
Output: The last byte of plaintext block P_2 i.e. $P_2(b)$

C_1' = generate_random_bytes(b) `// create a random b-byte block` C_1'
for $x = 0$ *until* 255 **do**
    $C_1'(b) = x$ `// set the last byte of` C_1' `to` x
    $reply = oracle([IV, C_1', C_2])$ `// call the oracle`
    if $reply == 1$ **then**
        $x_{correct} = x$ `// no padding error`
        break
    end
end
$I_2(b) = x_{correct} \oplus 0x01$ `// recover the last byte of` I_2
$P_2(b) = I_2(b) \oplus C_1(b)$ `// use the original` $C_1(b)$ `to recover` $P_2(b)$

Padding Oracle Attack

Last-Byte Oracle (extra check)

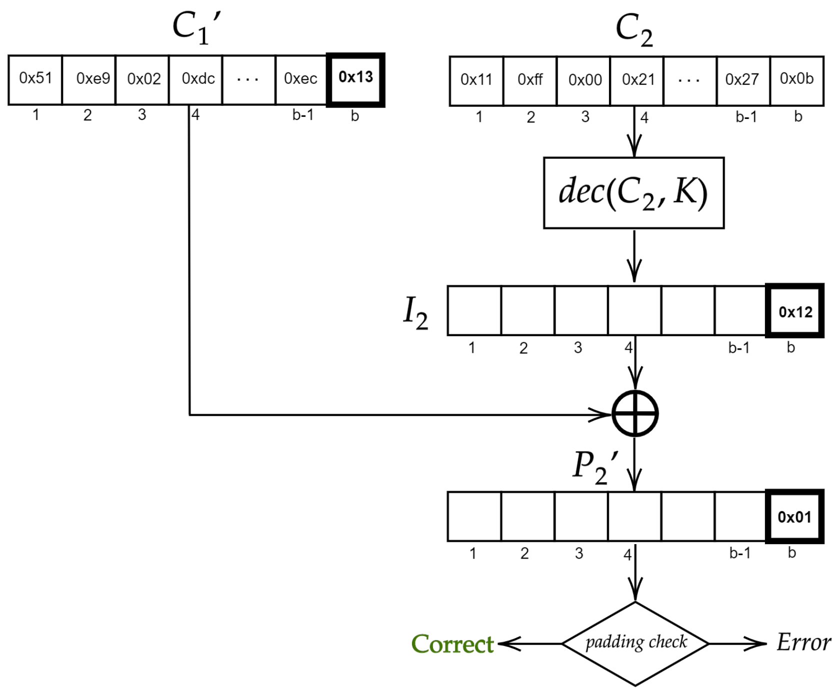
- When the oracle replies with 1, the padding is correct and in line 10 we assumed that the correct padding is `0x01`
- Although `0x01` is the most likely one (due to randomly selecting C_1'), we can also check for case `[0x02 0x02]` or case `[0x03 0x03 0x03]` etc.

- We have found that the padding is correct when $C_1' = [C_1'(1), C_1'(2), \ldots, C_1'(b-1), x_{correct}]$
- How do we check that the correct padding is e.g. `[0x02 0x02]`?
- We flip a bit in $C_1'(b-1)$ and send $[IV, C_1', C_2]$ to the oracle
  - If the oracle replies with 1, then altering $C_1'(b-1)$ doesn't affect the padding. Thus the padding was `0x01` after all.
  - If the oracle replies with 0, then $C_1'(b-1)$ affects the padding. Thus the padding can be `[0x02 0x02]` or `[0x03 0x03 0x03]` etc.
- To exclude the padding `[0x03 0x03 0x03]`, we flip a bit in $C_1'(b-2)$ and check the oracle again
- The process is repeated until the oracle replies with 1 or we reach and check $C_1'(1)$

Padding Oracle Attack

- The Last-Byte Oracle chooses the last byte of C_1' such that the `0x01` padding appears in the last byte of P_2' and then recovers the last byte of I_2

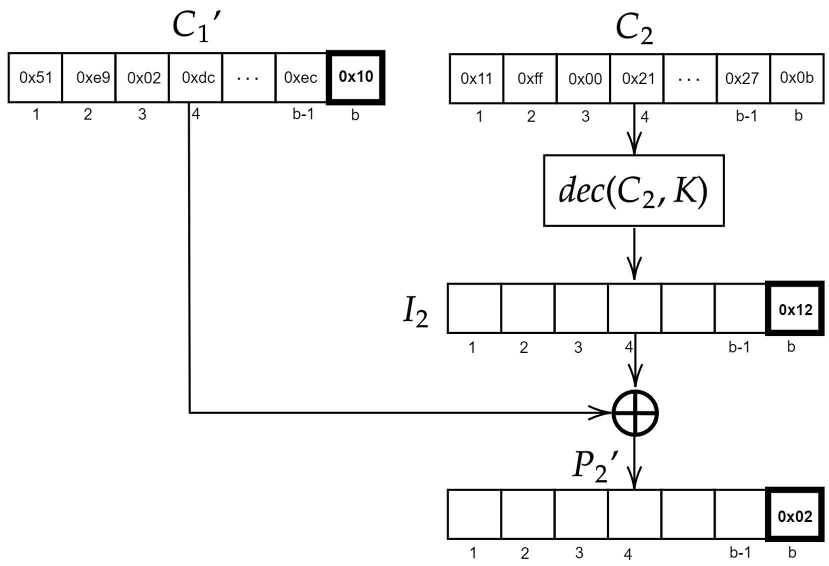
  i.e. it chooses $C_1'(b)$ such
  that $P_2'(b) = \texttt{0x01}$ and recovers $I_2(b)$ where b is the input size of the block cipher



Padding Oracle Attack

- Since we now know $I_2(b)$, we can choose $C_1'(b)$ such that $P_2'(b) = \texttt{0x02}$

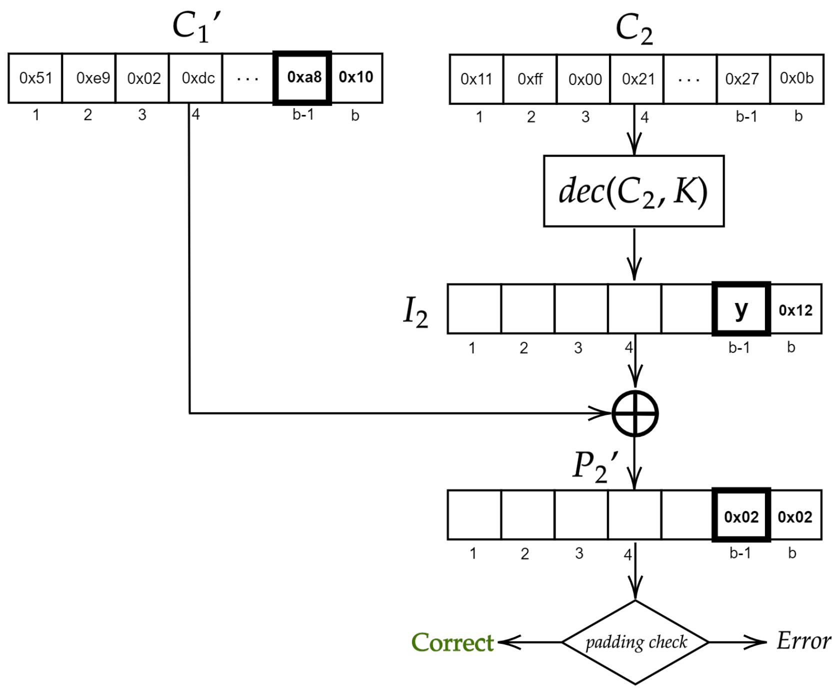
  $P_2'(b) = \texttt{0x02} \iff I_2(b) \oplus C_1'(b) = \texttt{0x02} \iff C_1'(b) = \texttt{0x02} \oplus \texttt{0x12} \iff C_1'(b) = \texttt{0x10}$



Padding Oracle Attack

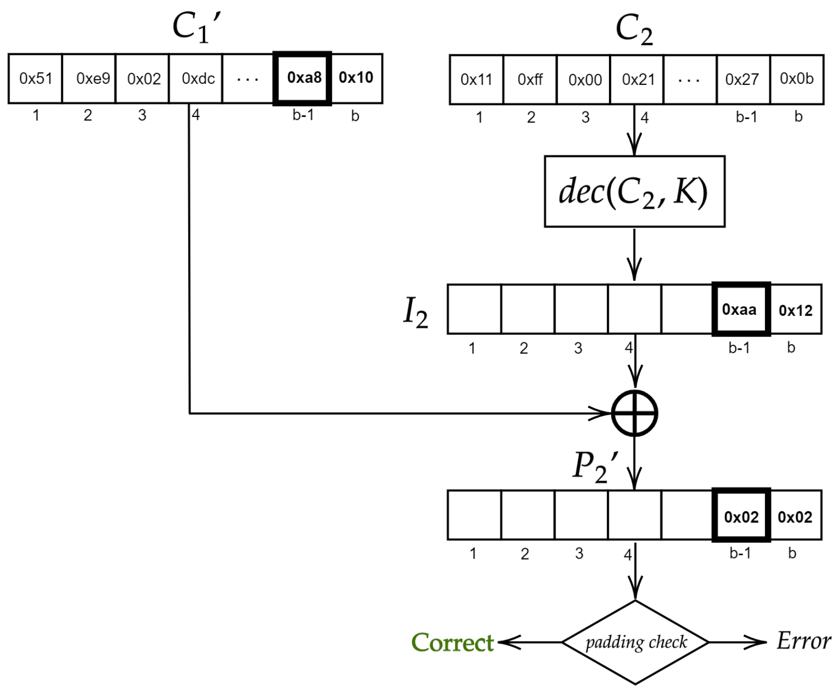
- While $P'_2(b) = \texttt{0x02}$, we will vary $C'_1(b-1)$ until $P'_2(b-1) = \texttt{0x02}$ as well
- The goal is to create the padding [0x02, 0x02] in bytes $[P'_2(b-1),\ P'_2(b)]$

  e.g. the valid padding [0x02, 0x02] is detected for $C'_1(b-1) = \texttt{0xa8}$



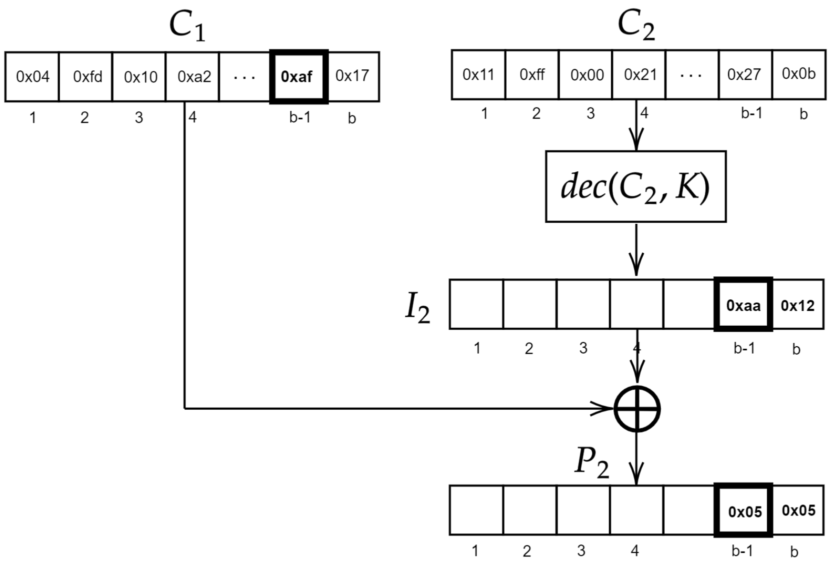
Padding Oracle Attack

- We can now compute $I_2(b-1) = C_1'(b-1) \oplus \texttt{0x02} = \texttt{0xa8} \oplus \texttt{0x02} = \texttt{0xaa}$



Padding Oracle Attack

- Having recovered $I_2(b-1)$, we go back to the original ciphertext block C_1
- We can compute $P_2(b-1) = I_2(b-1) \oplus C_1(b-1) = \texttt{0xaa} \oplus \texttt{0xaf} = \texttt{0x05}$



- We can continue this process by choosing $[C_1'(b-2), C_1'(b-1), C_1'(b)]$ such that $[P_2'(b-2), P_2'(b-1), P_2'(b))] = [\texttt{0x03 0x03 0x03}]$. Then we can compute $I_2(b-2)$ and use $C_1(b-2)$ to recover $P_2(b-2)$.
- Repeating the process yields gradually all bytes of P_2 and we refer to it as the **Last-Block Oracle**.

Padding Oracle Attack

Final notes on the padding oracle attack:

- It is a chosen ciphertext attack that uses the padding check mechanism (and not the padding of the original ciphertext)
- It is possible in symmetric and public key cryptography
- It shows why computer security has a strange reputation
- It can be (partially) prevented by being careful with the error messages
- It can be prevented by stopping unauthorized decryption requests to the server