
BHack
SEGURANÇA · TI · CONHECIMENTO

Attacking the Oracle

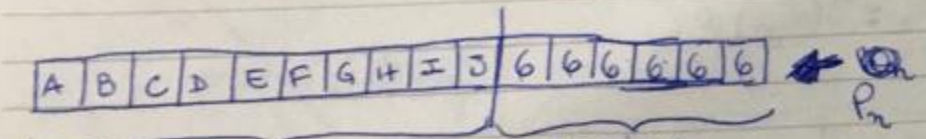
Debugging the Padding

PADDING ORACLE ATTACK

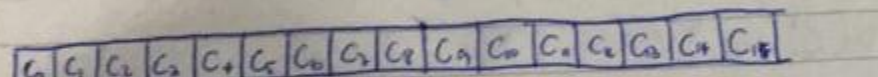
2nd turn

$P_1 = E(C_1 \oplus IV, K_1) \rightarrow E(P_2 \oplus IV, K_2)$

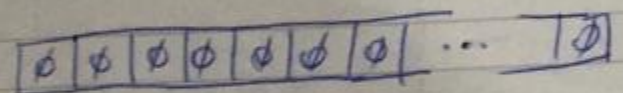
$E(E(C_1 \oplus IV, K_1), K_2)$



ENCRYPT



$C_n = ??$

$C' \Rightarrow$ 

stream = $C' \| C_n$, where $C_n = E(P_n \oplus C_{n-1})$

$\begin{cases} P'_1 = D(C') \oplus IV \rightarrow \text{garbage} \\ P'_2 = D(C_n) \oplus C' \end{cases}$

$P_n = (C_{n-1} \oplus C') \oplus P'_2$

$\rightarrow P'_2 = D(E(P_n \oplus C_{n-1})) \oplus C'$

$P_n \oplus C_{n-1}$

Last char

$P_2[K] = (P_n[K] \oplus C_{n-1}[K]) \oplus C'[K]$ } K^{th} char

? ? OK OK

$\{1 \dots FF\}$

$0x01$ ← PADDING OK? → (NO ERROR) $0xNN$ ← BRUTE FORCE

REWRITING

$P_n[K] = P_2[K] \oplus C_{n-1}[K] \oplus C'[K]$

$P_n[K] = 0x01 \oplus C_{n-1}[K] \oplus C'[K]$ ✓

OK OK OK

$P_n[K-1] = ???$

$P_n[K-1] = P_2[K-1] \oplus C_{n-1}[K-1] \oplus C'[K-1]$

$0x02$

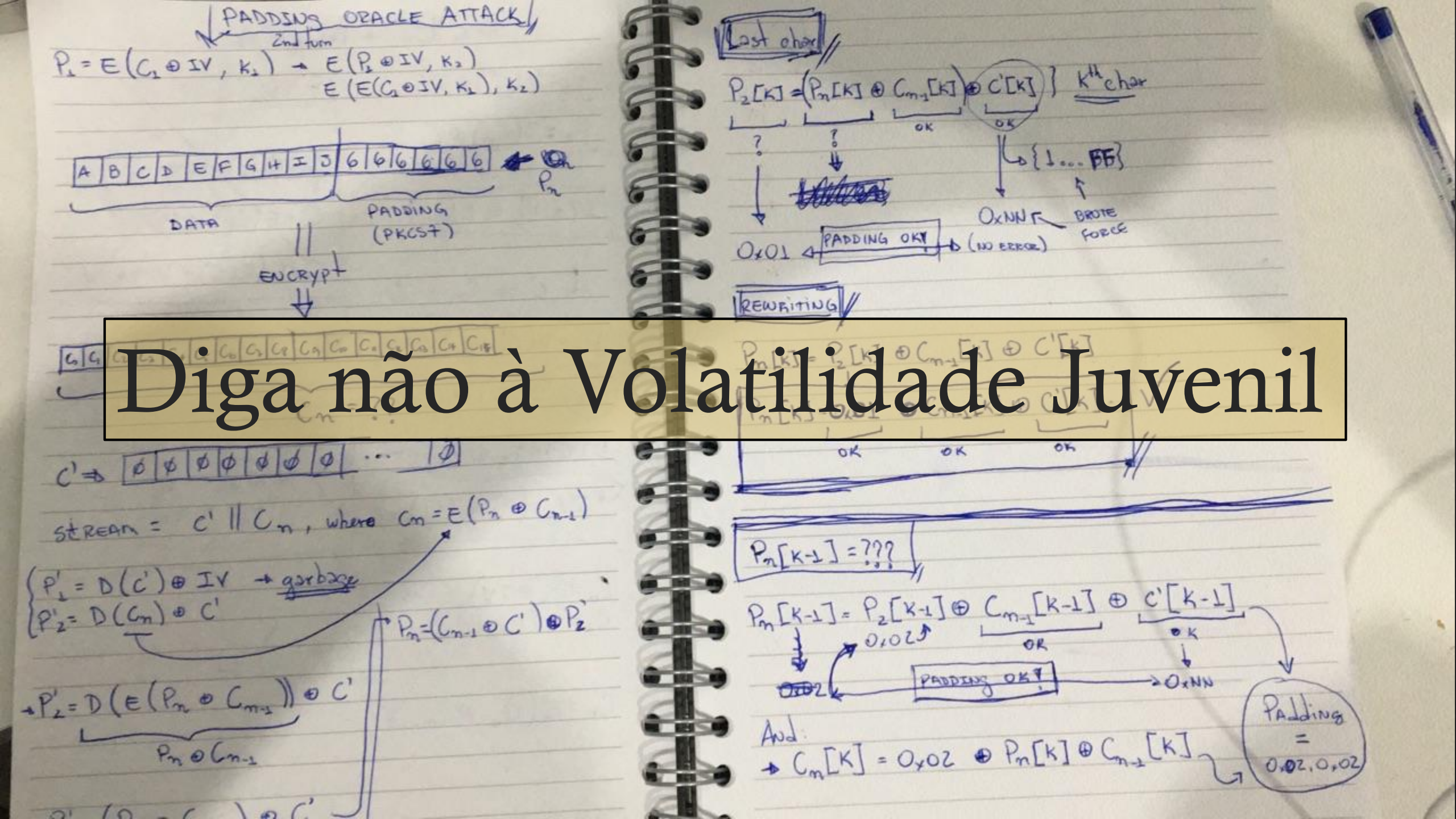
OK OK

PADDING OK? → $0xNN$

And

$\rightarrow C_n[K] = 0x02 \oplus P_n[K] \oplus C_{n-1}[K]$

Padding = 0x02, 0x02



Diga não à Volatilidade Juvenil

Agenda

- What is an Oracle?
- Exclusive OR (XOR)
- Cipher Block Chaining (CBC)
- Padding with Public Key Cryptography Standards (PKCS #7)
- The Padding Oracle Attack
- Demo

$ whoami

- BSc & MSc in Computer Science
- Senior Security Consultant
- Security Researcher @ Hack N' Roll
- Main duties:
  - Web Application Pentest
  - Network Pentest
  - Mobile Application Pentest
  - ATM Pentest
  - Embedded Devices Pentest

What is an Oracle?

What is an Oracle?

General Idea

- Nothing to do with Oracle Database nor Company!
- More related to oracle of Delphi (Pythia)
- … or The Oracle in The Matrix movie.

What is an Oracle?

Applied in cryptographic systems

”An **oracle** is an individual who knows the personal cell phone number of a god. This enables him (or her) to obtain some information which is usually considered as out of reach of mere mortals, such as glimpses of the future. In cryptography, that's the same, except that no deity is involved: **an oracle is any system which can give some extra information on a system, which otherwise would not be available**.”

https://security.stackexchange.com/questions/10617/what-is-a-cryptographic-oracle

What is an Oracle?

Using (my) easy words!

”An Oracle is a system that **knows the secret key** and the algorithm used to decrypt some data that **you have but don’t know what means**. Generally these data are provided by the application and, as you don't know the secret key, the application **expects** the guarantee of the **integrity and confidentiality** of the data.”

My poor definition (-:

Exclusive OR (XOR)

Bitwise Operation

Exclusive OR (XOR)

Bitwise operation

- Logical Operator as ⊕
- Used in programming languages as ^
- Used in Digital Systems as XOR Gate:

Exclusive OR (XOR)

Some properties

A	B	$A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Exclusive OR (XOR)

Some properties

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

A	0	1	0	0	1	1	0	1	77_{10}
B	0	0	1	1	1	0	0	1	57_{10}
A ⊕ B	0	1	1	1	0	1	0	0	116_{10}

Exclusive OR (XOR)

Some properties

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

A	0	1	0	0	1	1	0	1	77_{10}
B	0	0	1	1	1	0	0	1	57_{10}
A ⊕ B	0	1	1	1	0	1	0	0	116_{10}

Exclusive OR (XOR)

Some properties

A	B	A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

A	0	1	0	0	1	1	0	1	77_{10}
B	0	0	1	1	1	0	0	1	57_{10}
A ⊕ B	0	1	1	1	0	1	0	0	116_{10}

A ⊕ A	0
A ⊕ 1	~A
A ⊕ 0	A
A ⊕ B	B ⊕ A
(A ⊕ B) ⊕ C	A ⊕ (B ⊕ C)
(A ⊕ B) ⊕ B	A ⊕ (B ⊕ B) = A ⊕ 0 = A
(A ⊕ B) = C	A ⊕ C = B B ⊕ C = A

Cipher Block Chaining (CBC)

Cipher Block Chaining (CBC)

Basic Notation

- `P` → Decrypted plaintext
- `C` → Encrypted ciphertext
- P_n / C_n → plaintext/ciphertext of the block *n* (n^{th} block)
- `N` → Number of blocks
- `IV` → Initialization Vector (a random string)
- `E()` → Single block encryption function
- `D()` → Single block decryption function

Cipher Block Chaining (CBC)

CBC Encryption

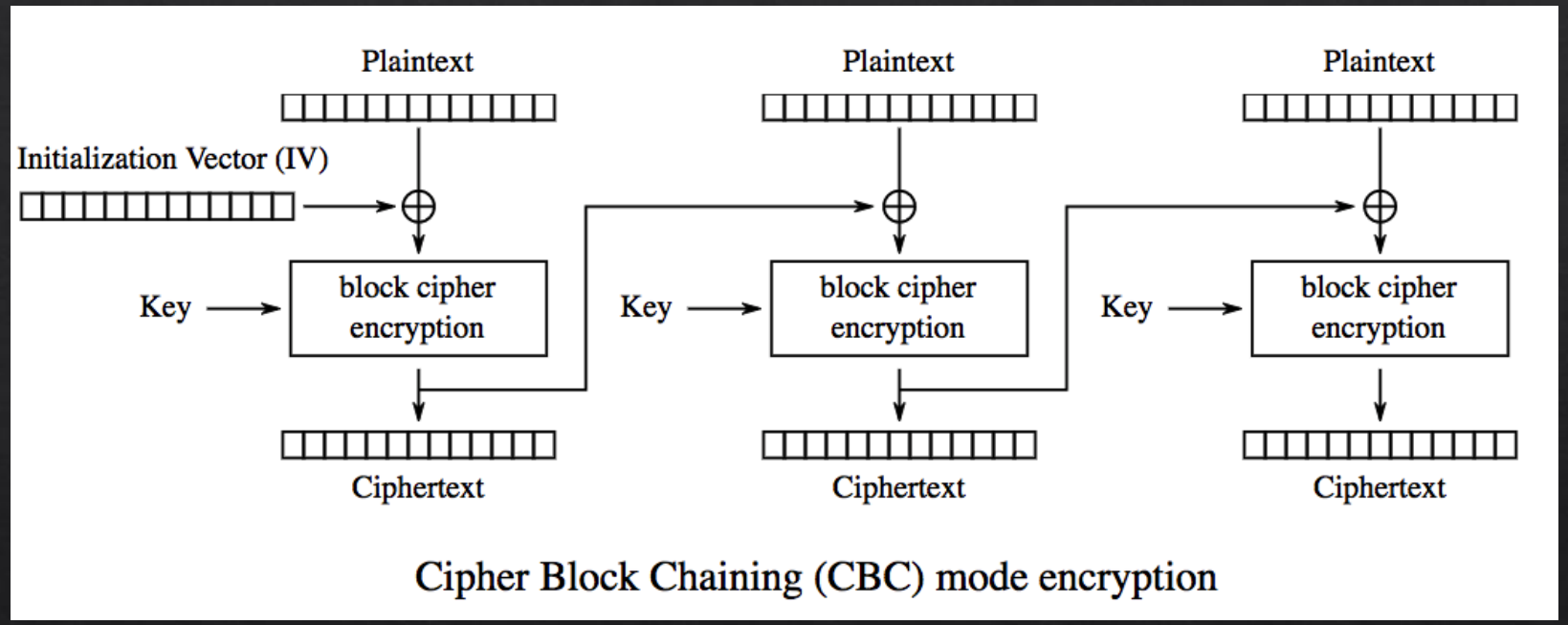
$$C_1 = E(P_1 \oplus IV)$$
$$C_2 = E(P_2 \oplus C_1)$$
$$C_3 = E(P_3 \oplus C_2)$$
$$\ldots$$
$$C_n = E(P_n \oplus C_{n-1})$$

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Block_Chaining_.28CBC.29

Cipher Block Chaining (CBC)

CBC Encryption

$$C_1 = E(P_1 \oplus IV)$$
$$C_2 = E(P_2 \oplus C_1)$$
$$C_3 = E(P_3 \oplus C_2)$$
$$\ldots$$
$$C_n = E(P_n \oplus C_{n-1})$$



Cipher Block Chaining (CBC) mode encryption

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Block_Chaining_.28CBC.29

Cipher Block Chaining (CBC)

CBC Decryption

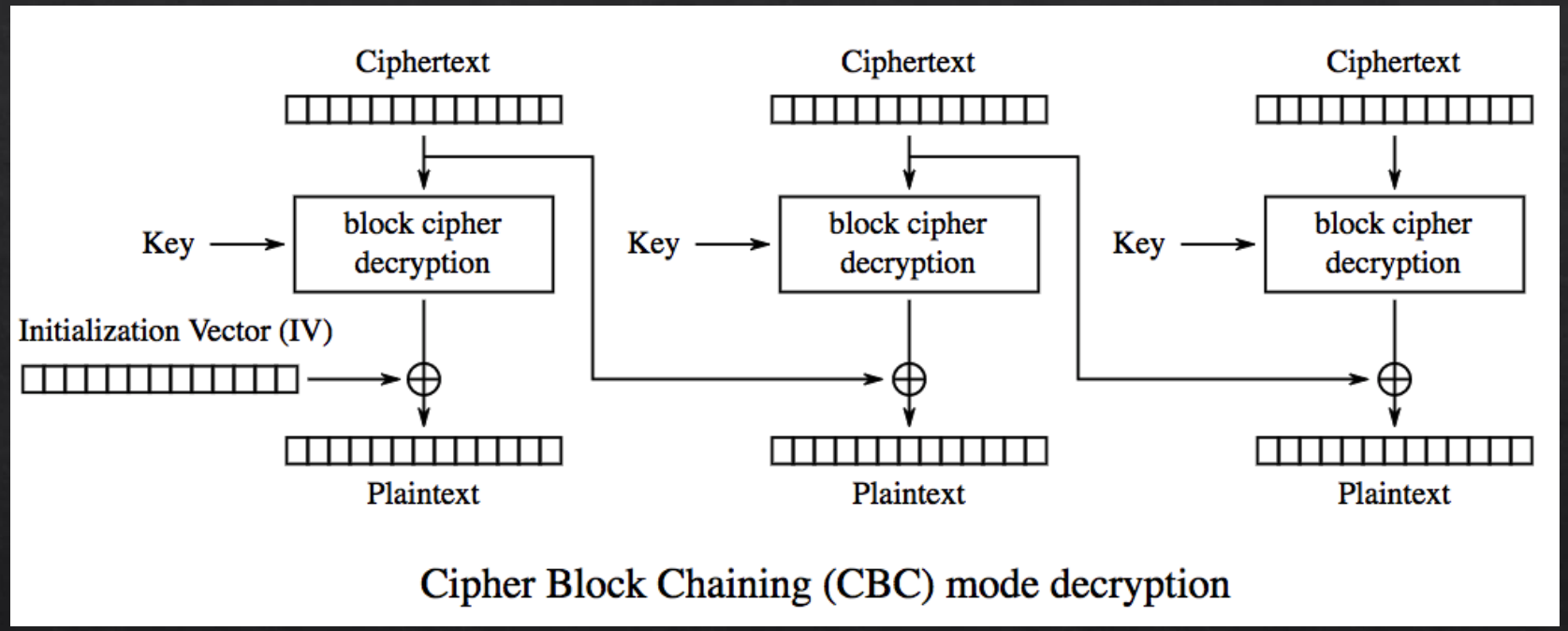
$$P_1 = D(C_1) \oplus IV$$
$$P_2 = D(C_2) \oplus C_1$$
$$P_3 = D(C_3) \oplus C_2$$
$$\ldots$$
$$P_n = D(C_n) \oplus C_{n-1}$$

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Block_Chaining_.28CBC.29

Cipher Block Chaining (CBC)

CBC Decryption

$$P_1 = D(C_1) \oplus IV$$
$$P_2 = D(C_2) \oplus C_1$$
$$P_3 = D(C_3) \oplus C_2$$
$$\ldots$$
$$P_n = D(C_n) \oplus C_{n-1}$$



Cipher Block Chaining (CBC) mode decryption

https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation#Cipher_Block_Chaining_.28CBC.29

Padding with Public Key Cryptography Standards (PKCS #7)

PKCS #7

General Information

- Defined by RFC 2315
- Padding described on the Section 10.3

> Some content-encryption algorithms assume the input length is a multiple of k octets, where k > 1, and let the application define a method for handling inputs whose lengths are not a multiple of k octets. For such algorithms, the method shall be to pad the input at the trailing end with k - (l mod k) octets all having value k - (l mod k), where l is the length of the input. In other words, the input is padded at the trailing end with one of the following strings:
>
> ```
>          01 -- if l mod k = k-1
>       02 02 -- if l mod k = k-2
>                    .
>                    .
>                    .
>    k k ... k k -- if l mod k = 0
> ```
>
> The padding can be removed unambiguously since all input is padded and no padding string is a suffix of another. This padding method is well-defined if and only if k < 256; methods for larger k are an open issue for further study.

https://tools.ietf.org/html/rfc2315

PKCS #7

General Information

- Defined by RFC 2315
- Padding described on the Section 10.3

> Some content-encryption algorithms assume the input length is a multiple of k octets, where k > 1, and let the application define a method for handling inputs whose lengths are not a multiple of k octets. For such algorithms, the method shall be to pad the input at the trailing end with k - (l mod k) octets all having value k - (l mod k), where l is the length of the input. In other words, the input is padded at the trailing end with one of the following strings:
>
> ```
>          01 -- if l mod k = k-1
>       02 02 -- if l mod k = k-2
>                    .
>                    .
>                    .
>    k k ... k k -- if l mod k = 0
> ```
>
> The padding can be removed unambiguously since all input is padded and no padding string is a suffix of another. This padding method is well-defined if and only if k < 256; methods for larger k are an open issue for further study.

https://tools.ietf.org/html/rfc2315

PKCS #7

General Information

- Defined by RFC 2315
- Padding described on the Section 10.3

”ABCDEFGHIJ”

Some content-encryption algorithms assume the input length is a multiple of k octets, where k > 1, and let the application define a method for handling inputs whose lengths are not a multiple of k octets. For such algorithms, the method shall be to pad the input at the trailing end with k - (l mod k) octets all having value k - (l mod k), where l is the length of the input. In other words, the input is padded at the trailing end with one of the following strings:

```
         01 -- if l mod k = k-1
      02 02 -- if l mod k = k-2
                  .
                  .
                  .
   k k ... k k -- if l mod k = 0
```

The padding can be removed unambiguously since all input is padded and no padding string is a suffix of another. This padding method is well-defined if and only if k < 256; methods for larger k are an open issue for further study.

https://tools.ietf.org/html/rfc2315

PKCS #7

General Information

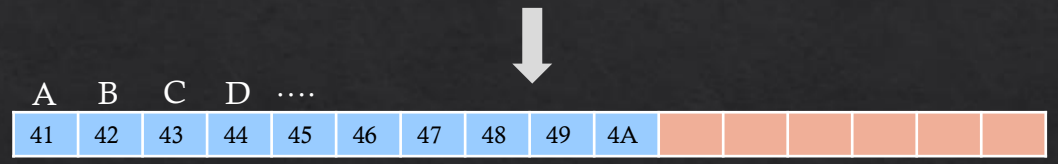
- Defined by RFC 2315
- Padding described on the Section 10.3

Some content-encryption algorithms assume the input length is a multiple of k octets, where k > 1, and let the application define a method for handling inputs whose lengths are not a multiple of k octets. For such algorithms, the method shall be to pad the input at the trailing end with k - (l mod k) octets all having value k - (l mod k), where l is the length of the input. In other words, the input is padded at the trailing end with one of the following strings:

```
         01 -- if l mod k = k-1
      02 02 -- if l mod k = k-2
                  .
                  .
                  .
   k k ... k k -- if l mod k = 0
```

The padding can be removed unambiguously since all input is padded and no padding string is a suffix of another. This padding method is well-defined if and only if k < 256; methods for larger k are an open issue for further study.

”ABCDEFGHIJ”



https://tools.ietf.org/html/rfc2315

PKCS #7

General Information

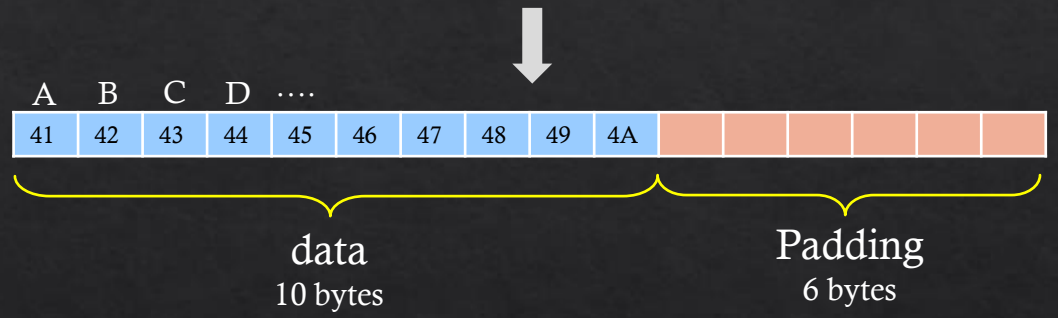
- Defined by RFC 2315
- Padding described on the Section 10.3

Some content-encryption algorithms assume the input length is a multiple of k octets, where k > 1, and let the application define a method for handling inputs whose lengths are not a multiple of k octets. For such algorithms, the method shall be to pad the input at the trailing end with k - (l mod k) octets all having value k - (l mod k), where l is the length of the input. In other words, the input is padded at the trailing end with one of the following strings:

```
01 -- if l mod k = k-1
02 02 -- if l mod k = k-2
.
.
.
k k ... k k -- if l mod k = 0
```

The padding can be removed unambiguously since all input is padded and no padding string is a suffix of another. This padding method is well-defined if and only if k < 256; methods for larger k are an open issue for further study.

”ABCDEFGHIJ”



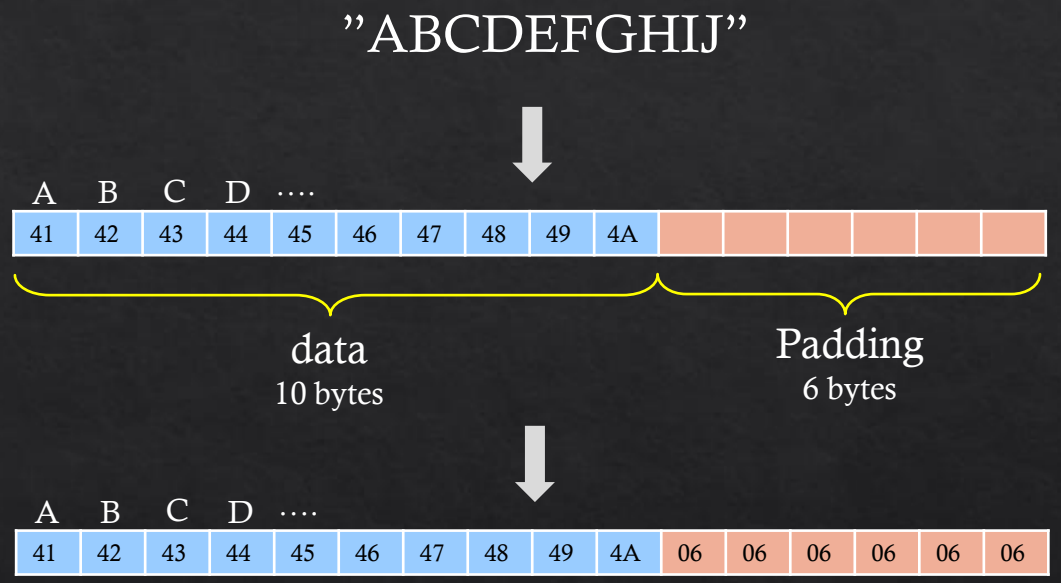
https://tools.ietf.org/html/rfc2315

PKCS #7

General Information

- Defined by RFC 2315
- Padding described on the Section 10.3

> Some content-encryption algorithms assume the input length is a multiple of k octets, where k > 1, and let the application define a method for handling inputs whose lengths are not a multiple of k octets. For such algorithms, the method shall be to pad the input at the trailing end with k - (l mod k) octets all having value k - (l mod k), where l is the length of the input. In other words, the input is padded at the trailing end with one of the following strings:
>
> ```
> 01 -- if l mod k = k-1
> 02 02 -- if l mod k = k-2
>                .
>                .
>                .
> k k ... k k -- if l mod k = 0
> ```
>
> The padding can be removed unambiguously since all input is padded and no padding string is a suffix of another. This padding method is well-defined if and only if k < 256; methods for larger k are an open issue for further study.



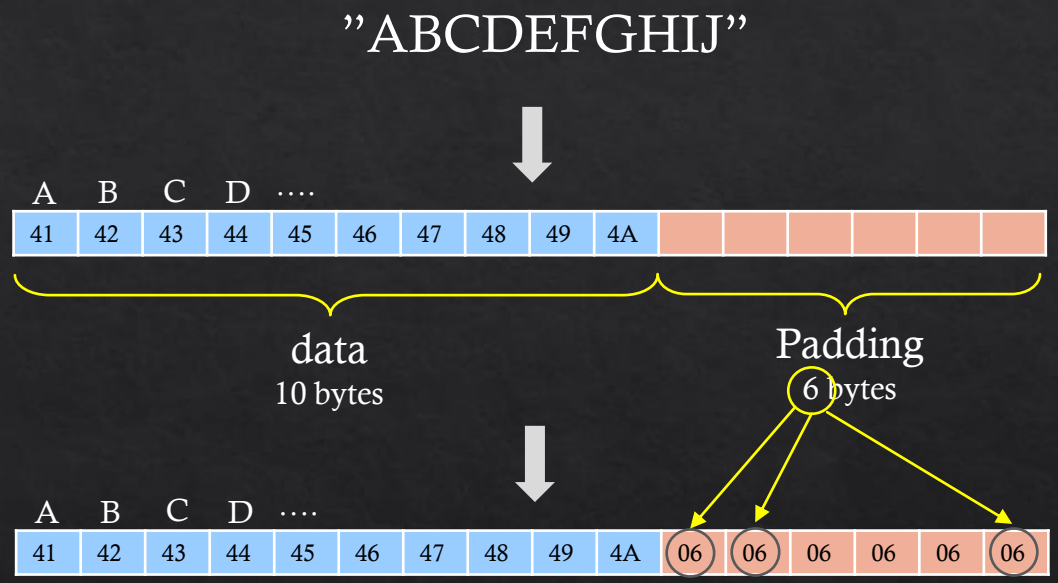
https://tools.ietf.org/html/rfc2315

PKCS #7

General Information

- Defined by RFC 2315
- Padding described on the Section 10.3

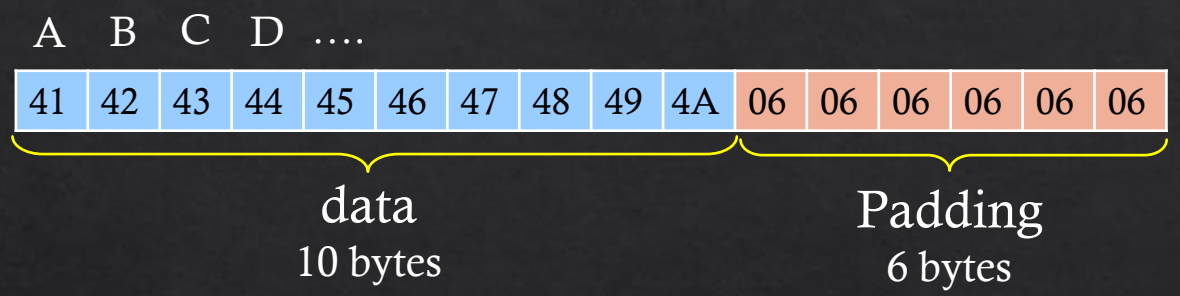
> Some content-encryption algorithms assume the input length is a multiple of k octets, where k > 1, and let the application define a method for handling inputs whose lengths are not a multiple of k octets. For such algorithms, the method shall be to pad the input at the trailing end with k - (l mod k) octets all having value k - (l mod k), where l is the length of the input. In other words, the input is padded at the trailing end with one of the following strings:
>
> ```
> 01 -- if l mod k = k-1
> 02 02 -- if l mod k = k-2
>                .
>                .
>                .
> k k ... k k -- if l mod k = 0
> ```
>
> The padding can be removed unambiguously since all input is padded and no padding string is a suffix of another. This padding method is well-defined if and only if k < 256; methods for larger k are an open issue for further study.



https://tools.ietf.org/html/rfc2315

PKCS #7

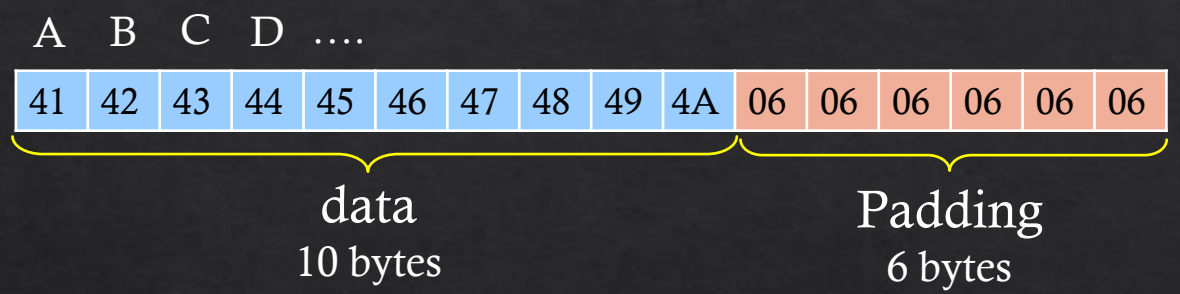
More examples

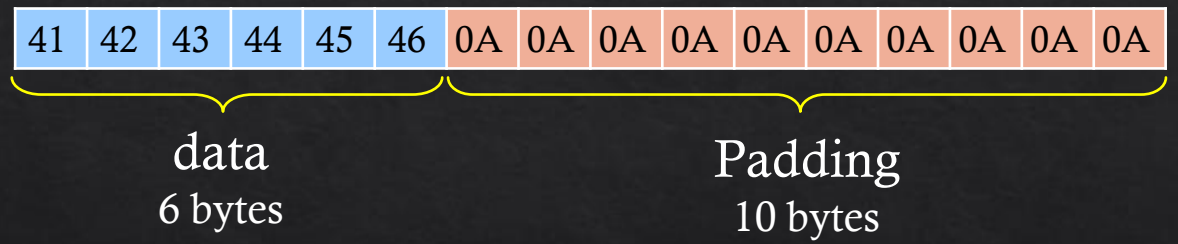


https://tools.ietf.org/html/rfc2315

PKCS #7

More examples

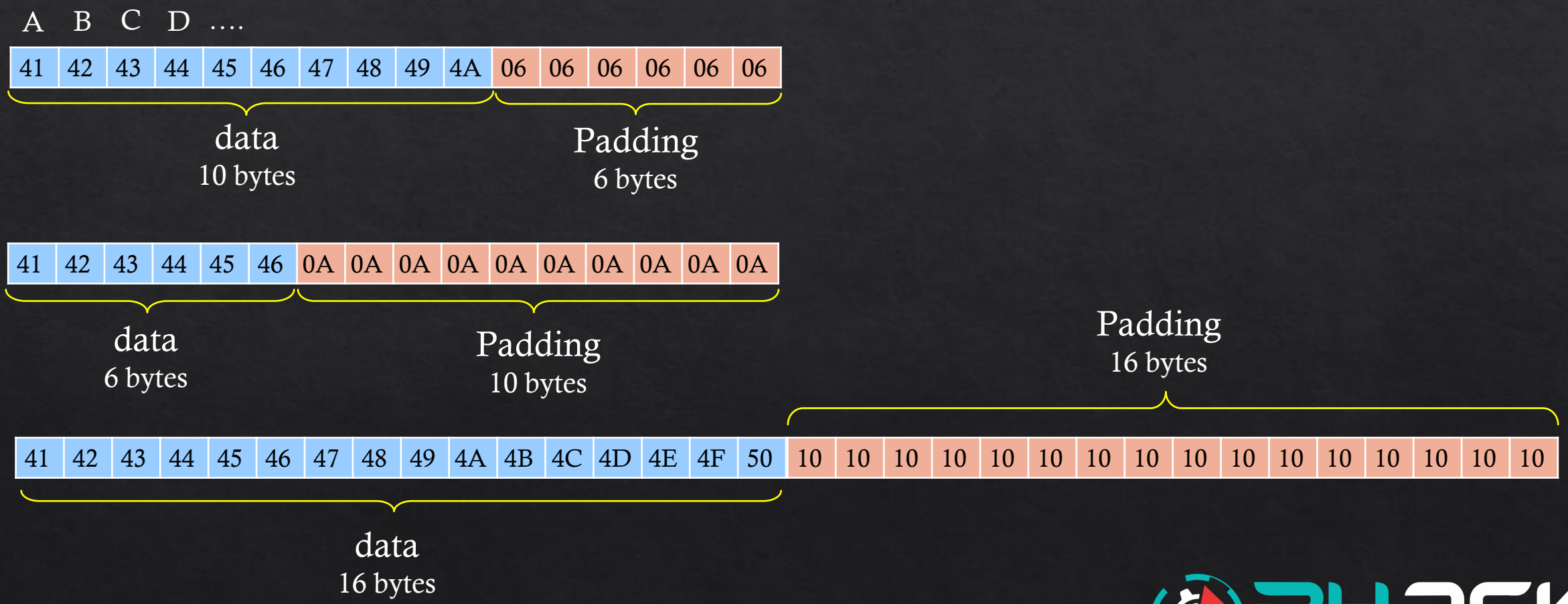




https://tools.ietf.org/html/rfc2315

PKCS #7

More examples



https://tools.ietf.org/html/rfc2315

The Padding Oracle Attack

The Padding Oracle Attack

Prerequisites

- An application that implements an Oracle cryptography system.
  - The application trust in a data sent encrypted to the client side (a.k.a. the attacker), and try to decrypt it when get it back.

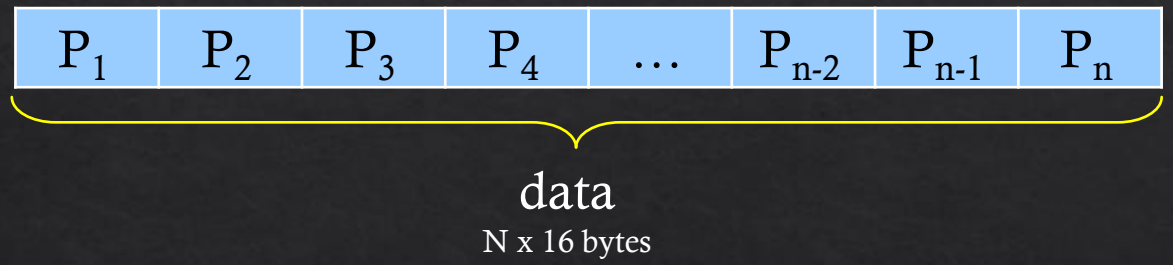
- If something goes wrong (e.g. padding error), the application should inform the client.

The Padding Oracle Attack

- Let's assume the following padded plaintext P_n with N blocks:

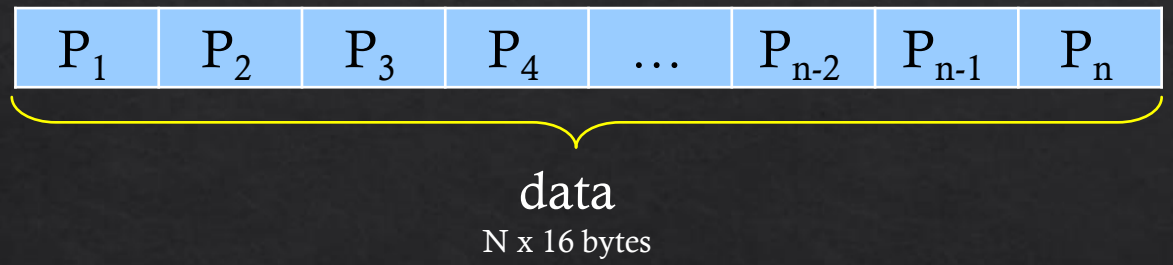
The Padding Oracle Attack

- Let's assume the following padded plaintext P_n with N blocks:

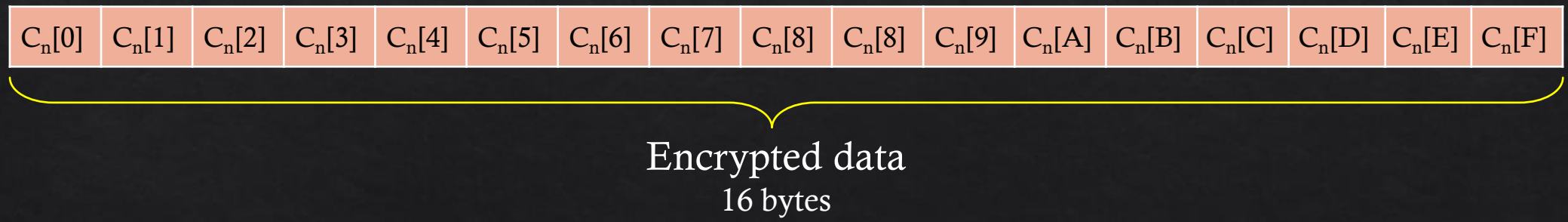


The Padding Oracle Attack

- Let's assume the following padded plaintext P_n with N blocks:



- After the encryption, we will have the following ciphertext C_n for the last block:



- Where $C_n[x]$ is the x^{th} byte in the last ciphertext block.

The Padding Oracle Attack

Get from the server

- To perform the attack, we need only the ciphertext (spitting on blocks):

The Padding Oracle Attack

Get from the server

- To perform the attack, we need only the ciphertext (spitting on blocks):

- Create a new ciphertext block (C') with random data (usually zero filled):

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

The Padding Oracle Attack

Get from the server

- To perform the attack, we need only the ciphertext (spitting on blocks):

$C_n[0]$	$C_n[1]$	$C_n[2]$	$C_n[3]$	$C_n[4]$	$C_n[5]$	$C_n[6]$	$C_n[7]$	$C_n[8]$	$C_n[8]$	$C_n[9]$	$C_n[A]$	$C_n[B]$	$C_n[C]$	$C_n[D]$	$C_n[E]$	$C_n[F]$

- Create a new ciphertext block (C') with random data (usually zero filled):

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

- We create a new ciphertext making the concatenation (||) operation between the original ciphertext block and the ciphertext block we just created:
  - $C = C' \ || \ C_n$

The Padding Oracle Attack

Back to the server

- What happened when we send back to the application server the C ciphertext?

The Padding Oracle Attack

Back to the server

- What happened when we send back to the application server the C ciphertext?

- The application server decrypt it:
  - $P_1 = D(C') \oplus IV$
  - $P_2 = D(C_n) \oplus C'$

The Padding Oracle Attack

Back to the server

- What happened when we send back to the application server the C ciphertext?

- The application server decrypt it:
  - $P_1 = D(C') \oplus IV$ ⟵ Garbage!
  - $P_2 = D(C_n) \oplus C'$

The Padding Oracle Attack

Back to the server

- What happened when we send back to the application server the C ciphertext?

- The application server decrypt it:
  - $P_1 = D(C') \oplus IV$ ⟵ Garbage!
  - $P_2 = D(C_n) \oplus C'$

- We already know that:
  - $C_n = E(P_n \oplus C_{n-1})$

The Padding Oracle Attack

Back to the server

- What happened when we send back to the application server the C ciphertext?

- The application server decrypt it:
  - $P_1 = D(C') \oplus IV$ ⟵ Garbage!
  - $P_2 = D(C_n) \oplus C'$
- We already know that:
  - $C_n = E(P_n \oplus C_{n-1})$
- So…
  - $P_2 = D(E(P_n \oplus C_{n-1})) \oplus C'$
  - $P_2 = P_n \oplus C_{n-1} \oplus C'$

The Padding Oracle Attack

Back to the server

- What happened when we send back to the application server the C ciphertext?

- The application server decrypt it:
  - $P_1 = D(C') \oplus IV$ ⟵ Garbage!
  - $P_2 = D(C_n) \oplus C'$

After rewriting

$$P_n = P_2 \oplus C_{n-1} \oplus C'$$

UNKNOWN **KNOWN**

- We already know that:
  - $C_n = E(P_n \oplus C_{n-1})$

- So…
  - $P_2 = D(E(P_n \oplus C_{n-1})) \oplus C'$
  - $P_2 = P_n \oplus C_{n-1} \oplus C'$

The Padding Oracle Attack

Simplifying (and solving) the problem

- We have:

$$P_n = P_2 \oplus C_{n-1} \oplus C'$$

The Padding Oracle Attack

Simplifying (and solving) the problem

- We have:

$$P_n = P_2 \oplus C_{n-1} \oplus C'$$

- As the XOR operation is bitwise, we can simplify the formula applying it only on the last k^{th} char:

$$P_n[k] = P_2[k] \oplus C_{n-1}[k] \oplus C'[k]$$

The Padding Oracle Attack

Simplifying (and solving) the problem

- We have:

$$P_n = P_2 \oplus C_{n-1} \oplus C'$$

- As the XOR operation is bitwise, we can simplify the formula applying it only on the last k^{th} char:

$$\underbrace{P_n[k]}_{???} = \underbrace{P_2[k]}_{???} \oplus \underbrace{C_{n-1}[k]}_{Ok} \oplus \underbrace{C'[k]}_{Ok}$$

The Padding Oracle Attack

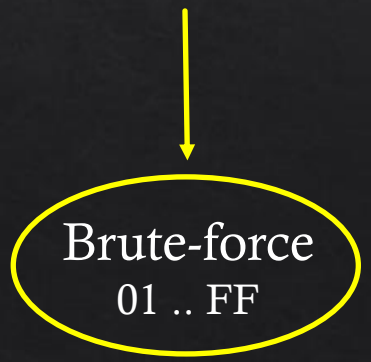
Simplifying (and solving) the problem

- We have:

$$P_n = P_2 \oplus C_{n-1} \oplus C'$$

- As the XOR operation is bitwise, we can simplify the formula applying it only on the last k^{th} char:

$$\underbrace{P_n[k]}_{???} = \underbrace{P_2[k]}_{???} \oplus \underbrace{C_{n-1}[k]}_{Ok} \oplus \underbrace{C'[k]}_{Ok}$$



The Padding Oracle Attack

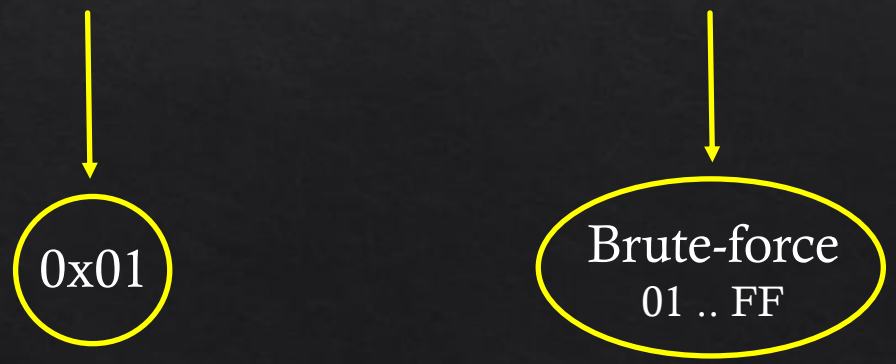
Simplifying (and solving) the problem

- We have:

$$P_n = P_2 \oplus C_{n-1} \oplus C'$$

- As the XOR operation is bitwise, we can simplify the formula applying it only on the last k^{th} char:

$$\underbrace{P_n[k]}_{???} = \underbrace{P_2[k]}_{???} \oplus \underbrace{C_{n-1}[k]}_{Ok} \oplus \underbrace{C'[k]}_{Ok}$$



The Padding Oracle Attack

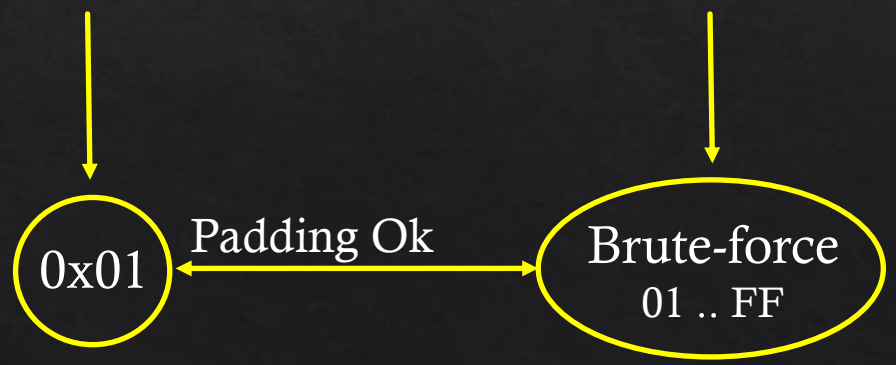
Simplifying (and solving) the problem

- We have:

$$P_n = P_2 \oplus C_{n-1} \oplus C'$$

- As the XOR operation is bitwise, we can simplify the formula applying it only on the last k^{th} char:

$$\underbrace{P_n[k]}_{???} = \underbrace{P_2[k]}_{???} \oplus \underbrace{C_{n-1}[k]}_{Ok} \oplus \underbrace{C'[k]}_{Ok}$$



The Padding Oracle Attack

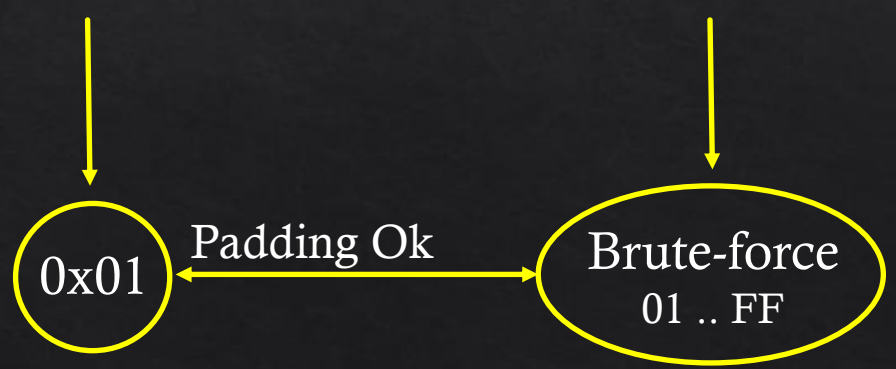
Simplifying (and solving) the problem

- We have:

$$P_n = P_2 \oplus C_{n-1} \oplus C'$$

- As the XOR operation is bitwise, we can simplify the formula applying it only on the last k^{th} char:

$$\underbrace{P_n[k]}_{???} = \underbrace{P_2[k]}_{???} \oplus \underbrace{C_{n-1}[k]}_{Ok} \oplus \underbrace{C'[k]}_{Ok}$$



$$P_n[k] = 0x01 \oplus C_{n-1}[k] \oplus C'[k]$$

The Padding Oracle Attack

What about the $P_n[k-1]$?

- We already know:

$$P_n[k] = 0x01 \oplus C_{n-1}[k] \oplus C'[k]$$

The Padding Oracle Attack

What about the $P_n[k-1]$?

- We already know:

$$P_n[k] = 0x01 \oplus C_{n-1}[k] \oplus C'[k]$$

- We can fixe the last char in 0x02 with:

$$C'[k] = 0x02 \oplus P_n[k] \oplus C_{n-1}[k]$$

The Padding Oracle Attack

What about the $P_n[k-1]$?

- We already know:

$$P_n[k] = 0x01 \oplus C_{n-1}[k] \oplus C'[k]$$

- We can fixe the last char in 0x02 with:

$$C'[k] = 0x02 \oplus P_n[k] \oplus C_{n-1}[k]$$

- We following the same approach to look for the padding "\x02\x02"

$$P_n[k-1] = P_2[k-1] \oplus C_{n-1}[k-1] \oplus C'[k-1]$$

The Padding Oracle Attack

What about the $P_n[k-1]$?

- We already know:

$$P_n[k] = 0x01 \oplus C_{n-1}[k] \oplus C'[k]$$

- We can fixe the last char in 0x02 with:

$$C'[k] = 0x02 \oplus P_n[k] \oplus C_{n-1}[k]$$

- We following the same approach to look for the padding "\x02\x02"

$$P_n[k-1] = P_2[k-1] \oplus C_{n-1}[k-1] \oplus C'[k-1]$$

Brute-force
01 .. FF

The Padding Oracle Attack

What about the $P_n[k-1]$?

- We already know:

$$P_n[k] = 0x01 \oplus C_{n-1}[k] \oplus C'[k]$$

- We can fixe the last char in 0x02 with:

$$C'[k] = 0x02 \oplus P_n[k] \oplus C_{n-1}[k]$$

- We following the same approach to look for the padding "\x02\x02"

$$P_n[k-1] = P_2[k-1] \oplus C_{n-1}[k-1] \oplus C'[k-1]$$

The Padding Oracle Attack

What about the $P_n[k-1]$?

- We already know:

$$P_n[k] = 0x01 \oplus C_{n-1}[k] \oplus C'[k]$$

- We can fixe the last char in 0x02 with:

$$C'[k] = 0x02 \oplus P_n[k] \oplus C_{n-1}[k]$$

- We following the same approach to look for the padding "\x02\x02"

$$P_n[k-1] = P_2[k-1] \oplus C_{n-1}[k-1] \oplus C'[k-1]$$

Let's Write a Beautiful Python Code

Installing Python Padding Oracle Package

```
$ pip install --user paddingoracle
Downloading/unpacking paddingoracle
  Downloading paddingoracle-0.2.2-py2-none-any.whl
Installing collected packages: paddingoracle
Successfully installed paddingoracle
Cleaning up...
$
```

```python
from paddingoracle import BadPaddingException, PaddingOracle

def check_paddind_error(data):
    ...


class PadBuster(PaddingOracle):

    def __init__(self, **kwargs):
        super(PadBuster, self).__init__(**kwargs)

    def oracle(self, data, **kwargs):
        if check_paddind_error(data):
            raise BadPaddingException


if __name__ == '__main__':

    padbuster = PadBuster()
    cleartext = padbuster.decrypt(encrypted_data, block_size=8, iv=bytearray(8))

    print "Cleartext: {}".format(cleantext)
```

```python
from paddingoracle import BadPaddingException, PaddingOracle

def check_paddind_error(data):
    ...


class PadBuster(PaddingOracle):

    def __init__(self, **kwargs):
        super(PadBuster, self).__init__(**kwargs)

    def oracle(self, data, **kwargs):
        if check_paddind_error(data):
            raise BadPaddingException


if __name__ == '__main__':

    padbuster = PadBuster()
    cleartext = padbuster.decrypt(encrypted_data, block_size=8, iv=bytearray(8))

    print "Cleartext: {}".format(cleantext)
```

```python
from paddingoracle import BadPaddingException, PaddingOracle

def check_paddind_error(data):
    ...


class PadBuster(PaddingOracle):

    def __init__(self, **kwargs):
        super(PadBuster, self).__init__(**kwargs)

    def oracle(self, data, **kwargs):
        if check_paddind_error(data):
            raise BadPaddingException


if __name__ == '__main__':

    padbuster = PadBuster()
    cleartext = padbuster.decrypt(encrypted_data, block_size=8, iv=bytearray(8))

    print "Cleartext: {}".format(cleantext)
```

```python
from paddingoracle import BadPaddingException, PaddingOracle

def check_paddind_error(data):
    ...


class PadBuster(PaddingOracle):

    def __init__(self, **kwargs):
        super(PadBuster, self).__init__(**kwargs)

    def oracle(self, data, **kwargs):
        if check_paddind_error(data):
            raise BadPaddingException


if __name__ == '__main__':

    padbuster = PadBuster()
    cleartext = padbuster.decrypt(encrypted_data, block_size=8, iv=bytearray(8))

    print "Cleartext: {}".format(cleantext)
```

```python
from paddingoracle import BadPaddingException, PaddingOracle

def check_paddind_error(data):
    ...


class PadBuster(PaddingOracle):

    def __init__(self, **kwargs):
        super(PadBuster, self).__init__(**kwargs)

    def oracle(self, data, **kwargs):
        if check_paddind_error(data):
            raise BadPaddingException


if __name__ == '__main__':

    padbuster = PadBuster()
    cleartext = padbuster.decrypt(encrypted_data, block_size=8, iv=bytearray(8))

    print "Cleartext: {}".format(cleantext)
```

```python
from paddingoracle import BadPaddingException, PaddingOracle

def check_paddind_error(data):
    ...


class PadBuster(PaddingOracle):

    def __init__(self, **kwargs):
        super(PadBuster, self).__init__(**kwargs)

    def oracle(self, data, **kwargs):
        if check_paddind_error(data):
            raise BadPaddingException


if __name__ == '__main__':

    padbuster = PadBuster()
    cleartext = padbuster.decrypt(encrypted_data, block_size=8, iv=bytearray(8))

    print "Cleartext: {}".format(cleantext)
```

```python
from paddingoracle import BadPaddingException, PaddingOracle

def check_paddind_error(data):
    ...


class PadBuster(PaddingOracle):

    def __init__(self, **kwargs):
        super(PadBuster, self).__init__(**kwargs)

    def oracle(self, data, **kwargs):
        if check_paddind_error(data):
            raise BadPaddingException


if __name__ == '__main__':

    padbuster = PadBuster()
    cleartext = padbuster.decrypt(encrypted_data, block_size=8, iv=bytearray(8))

    print "Cleartext: {}".format(cleantext)
```

```python
from paddingoracle import BadPaddingException, PaddingOracle

def check_paddind_error(data):
    ...


class PadBuster(PaddingOracle):

    def __init__(self, **kwargs):
        super(PadBuster, self).__init__(**kwargs)

    def oracle(self, data, **kwargs):
        if check_paddind_error(data):
            raise BadPaddingException


if __name__ == '__main__':

    padbuster = PadBuster()
    cleartext = padbuster.decrypt(encrypted_data, block_size=8, iv=bytearray(8))

    print "Cleartext: {}".format(cleantext)
```

```python
from paddingoracle import BadPaddingException, PaddingOracle

def check_paddind_error(data):
    ...


class PadBuster(PaddingOracle):

    def __init__(self, **kwargs):
        super(PadBuster, self).__init__(**kwargs)

    def oracle(self, data, **kwargs):
        if check_paddind_error(data):
            raise BadPaddingException


if __name__ == '__main__':

    padbuster = PadBuster()
    cleartext = padbuster.decrypt(encrypted_data, block_size=8, iv=bytearray(8))

    print "Cleartext: {}".format(cleantext)
```

```python
from paddingoracle import BadPaddingException, PaddingOracle

def check_paddind_error(data):
    ...


class PadBuster(PaddingOracle):

    def __init__(self, **kwargs):
        super(PadBuster, self).__init__(**kwargs)

    def oracle(self, data, **kwargs):
        if check_paddind_error(data):
            raise BadPaddingException


if __name__ == '__main__':

    padbuster = PadBuster()
    cleartext = padbuster.decrypt(encrypted_data, block_size=8, iv=bytearray(8))

    print "Cleartext: {}".format(cleantext)
```

Demonstration

```
maycon@DayOfEvil:~/BHack18$ hexdump -C bhack.bin
00000000  cb 3f 73 dd 38 76 1b 17  5c 9b 38 07 ef 00 46 9b  |.?s.8v..\.8...F.|
00000010  06 18 49 70 21 0b e8 34  eb 84 f2 64 cf f4 39 ba  |..Ip!..4...d..9.|
00000020  48 35 a0 be d5 1c c5 c2  ca 38 c4 0e 8a 78 00 19  |H5.......8...x..|
00000030  9c 99 d1 e8 29 b5 0a d2  62 9b 52 b2 ff b3 31 aa  |....)...b.R...1.|
00000040  91 91 cb b6 68 99 d6 3f  70 63 7e 2f 9d 7b 33 32  |....h..?pc~/.{32|
00000050
```

- Remote oracle running on 127.0.0.1:1043

Maycon Maia Vitali
Maycon@hacknroll.com

Thank you