

SEGURIDAD DE LA INFORMACIÓN

INTRODUCCIÓN A PYCRYPTODOME

CIFRADO Y DESCIFRADO

PyCRYPTODOME

Cifrado en PyCryptodome

- El proceso es sencillo:

1. Se inicializa los parámetros

- `key = get_random_bytes(8)` # Clave aleatoria, p.ej. 64 bits DES → 8 bytes
- `IV = get_random_bytes(8)` # IV aleatorio, p.ej. 64 bits DES → 8 bytes

2. Se instancia un objeto de cifrado con **new()**

- `Crypto.Cipher.AES/DES.new()`
- `Crypto.Cipher.AES/DES.new(key, modo de operación, IV)`

3. Se cifra el dato con **encrypt()**

- `Crypto.Cipher.AES/DES.encrypt(plaintext) → ciphertext`

4. Se descifra el criptograma con **decrypt()**

- `Crypto.Cipher.AES/DES.decrypt(ciphertext) → plaintext`

Instanciar objetos de cifrado en PyCryptodome

- En el paquete **Crypto.Cipher**

– `Crypto.Cipher.DES/AES.new(key, mode, *args, **kwargs)`

– Parámetros obligatorios:

- **key** (*bytes/bytearray/memoryview*): la clave
- **Mode**: el modo de operación → CBC, ECB, OFB, EAX, ...


– Parámetros opcionales:

- **IV** (*byte string*): vector de inicialización para los modos de operación
- Otros...

Instanciar objetos de cifrado en PyCryptodome


```
cipher = AES.new(key, AES.MODE_EAX)
```

 **AESCipher** 

 args


 block_size

 blockalgo

 key

 key_size

 kwargs

 MODE_CBC

 MODE_CFB

 MODE_CTR

 MODE_ECB

 MODE_OFB

```
cipher = AES.new([key, AES.MODE_
```

 **MODE_CBC** 

 **MODE_CFB**

 **MODE_CTR**

 **MODE_ECB**

 **MODE_OFB**

 **MODE_OPENPGP**

 **MODE_PGP**

Cifrado en PyCryptodome

- Ejemplo 1:

```
from Crypto.Cipher import DES
from Crypto.Random import get_random_bytes

key = get_random_bytes(8)
plaintext = "Hola Mundo".encode("utf-8")
""" debemos trabajar con UTF-8 """

cipher = DES.new(key, DES.MODE_ECB)

msg = cipher.encrypt(plaintext)
```

Encode / Decode

- En Python, el formato interno de las *cadena* (*strings*) y los *arrays de bytes* es distinto
- Los parámetros de las funciones criptográficas en PyCryptodome utilizan *arrays de bytes*
 - ¿Qué pasa si le pasamos una *cadena* por parámetro?

```
TypeError: can only concatenate str (not "bytes") to str
```

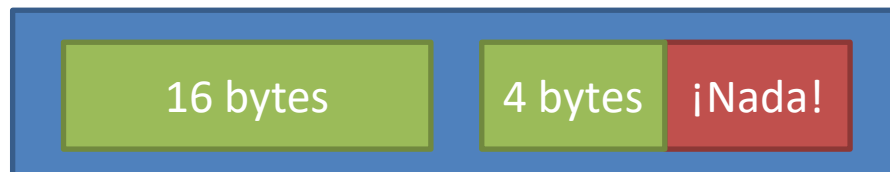
```
TypeError: Object type <class 'str'> cannot be passed to C code
```

```
...
```

- Una cadena **debe** convertirse en array de bytes antes de utilizarse
 - `datos = cadena.encode("utf-8")`
- Es posible realizar la operación contraria (array de bytes a cadena)
 - `cadena = datos.decode("utf-8", "ignore")`

Padding

- Los cifrados en bloque están diseñados para trabajar con mensajes compuestos de **bloques** de un tamaño específico
 - Ejemplo: AES-128 trabaja con bloques de 128 bits (16 bytes)
 - Problema: Supongamos que usamos AES-128 (16 bytes),
 - ¿Qué ocurre cuando queremos cifrar un mensaje que ocupa, por ejemplo, 20 bytes?
 - Tendremos un primer bloque de 16 bytes, y un segundo bloque de 4 bytes
 - El primer bloque lo podemos cifrar sin problemas
 - Al segundo bloque tenemos que añadirle algo al final (“**padding**”)



**Mensaje de
20 bytes**

Padding

- Esquemas de Padding:
 - ISO 10126: añadir bytes aleatorios, excepto el último, que indicará la longitud del padding
 - |12 63 12 65 E7 82 A7 C1|B7 02 9E 29 4E 8C 7B 05|
 - ISO/IEC 7816-4: añadir ceros, excepto el primero, que siempre tendrá el valor 80:
 - |12 63 12 65 e7 82 a7 c1|b7 02 9e 80 00 00 00 00|
 - Zero Padding: simplemente añadir ceros
 - |12 63 12 65 e7 82 a7 c1|b7 02 9e 00 00 00 00 00|
 - Problema: si el mensaje original acaba en alguna secuencia de ceros, no es posible determinar dónde empieza el padding
 - PKCS#5, PKCS#7: Si necesitamos N bytes de padding, usamos N veces el valor N
 - |12 63 12 65 e7 82 a7 c1|b7 02 9e 05 05 05 05 05|

Padding in PyCryptodome

- En el paquete: **Crypto.Util.Padding**

– `Crypto.Util.Padding.pad(data_to_pad, block_size, style='pkcs7')`

– Parámetros:

- **data_to_pad** (*byte string*): la cadena que necesita padding
- **block_size** (*integer*): el tamaño de bloque a usar con padding. La longitud de salida es múltiplo del tamaño de *block_size*
- **style** (*string*) – El algoritmo de padding
 - PKCS#7 es por defecto

Padding in PyCryptodome

Texto en claro



¡¡ENCODE!!



Padding



Cifrado



Criptograma



Texto en claro



¡¡DECODE!!



Un-padding



Descifrado



Criptograma

Cifrado con Padding en PyCryptodome

- Sin embargo, el cifrado puede requerir de padding:

1. **Se inicializa los parámetros**

- `key = get_random_bytes(8)` # Clave aleatoria de 64 bits
- `IV = get_random_bytes(8)` # IV aleatorio de 64 bits

2. **Se instancia un objeto de cifrado con `new()`**

- `Crypto.Cipher.AES/DES.new()`
- `Crypto.Cipher.AES/DES.new(key, modo de operación, IV)`

3. **Hacer padding antes del cifrado con `pad()`**

- `Crypto.Util.padding.pad(plaintext, BLOCK_SIZE)` → plaintext + padding

4. **Se cifra el dato con `encrypt()`**

- `Crypto.Cipher.AES/DES.encrypt(plaintext)` → ciphertext

5. **Se descifra el criptograma con `decrypt()`**

- `Crypto.Cipher.AES/DES.decrypt(ciphertext)` → plaintext

6. **Deshacer padding con el texto descifrado con `unpad()`**

- `Crypto.Util.padding.unpad(plaintext, BLOCK_SIZE)`

Modos de Operación

Ejemplo de modo de operación

• Modo CTR

```
Crypto.Cipher.<algorithm>.new(key, mode, *, nonce=None, initial_value=None, counter=None)
```

Create a new CTR object, using <algorithm> as the base block cipher.

Parameters:

- **key** (*bytes*) – the cryptographic key
- **mode** – the constant `Crypto.Cipher.<algorithm>.MODE_CTR`
- **nonce** (*bytes*) – the value of the fixed nonce. It must be unique for the combination message/key. Its length varies from 0 to the block size minus 1. If not present, the library creates a random nonce of length equal to block size/2.
- **initial_value** (*integer or bytes*) – the value of the counter for the first counter block. It can be either an integer or *bytes* (which is the same integer, just big endian encoded). If not specified, the counter starts at 0.
- **counter** – a custom counter object created with `Crypto.Util.Counter.new()`. This allows the definition of a more complex counter block.

Returns: a CTR cipher object

A *counter block* is exactly as long as the cipher block size (e.g. 16 bytes for AES). It consists of the concatenation of two pieces:

1. a fixed **nonce**, set at initialization.
2. a variable **counter**, which gets increased by 1 for any subsequent counter block. The counter is big endian encoded.

The `new()` function at the module level under `Crypto.Cipher` instantiates a new CTR cipher object for the relevant base algorithm. In the following definition, `<algorithm>` could be `AES`:

```
key= get_random_bytes(16)          # Clave aleatoria de 128 bits
nonce = get_random_bytes(8)        # contador de 64 bits empezando desde 0

aes_cifrado = AES.new(key, AES.MODE_CTR, nonce = nonce)
```

Ejemplo de modo de operación

- Modo CBC

```
key = get_random_bytes(8) # Clave aleatoria de 64 bits
IV = get_random_bytes(8)  # IV aleatorio de 64 bits para CBC
BLOCK_SIZE_DES = 8 # Bloque de 64 bits
data = "Hola amigos de la seguridad".encode("utf-8") # Datos a cifrar
print(data)
```

ORIGEN

```
cipher = DES.new(key, DES.MODE_CBC, IV)
```

UN OBJETO PARA CIFRAR

```
ciphertext = cipher.encrypt(pad(data, BLOCK_SIZE_DES))
print(ciphertext)
```

DESTINO

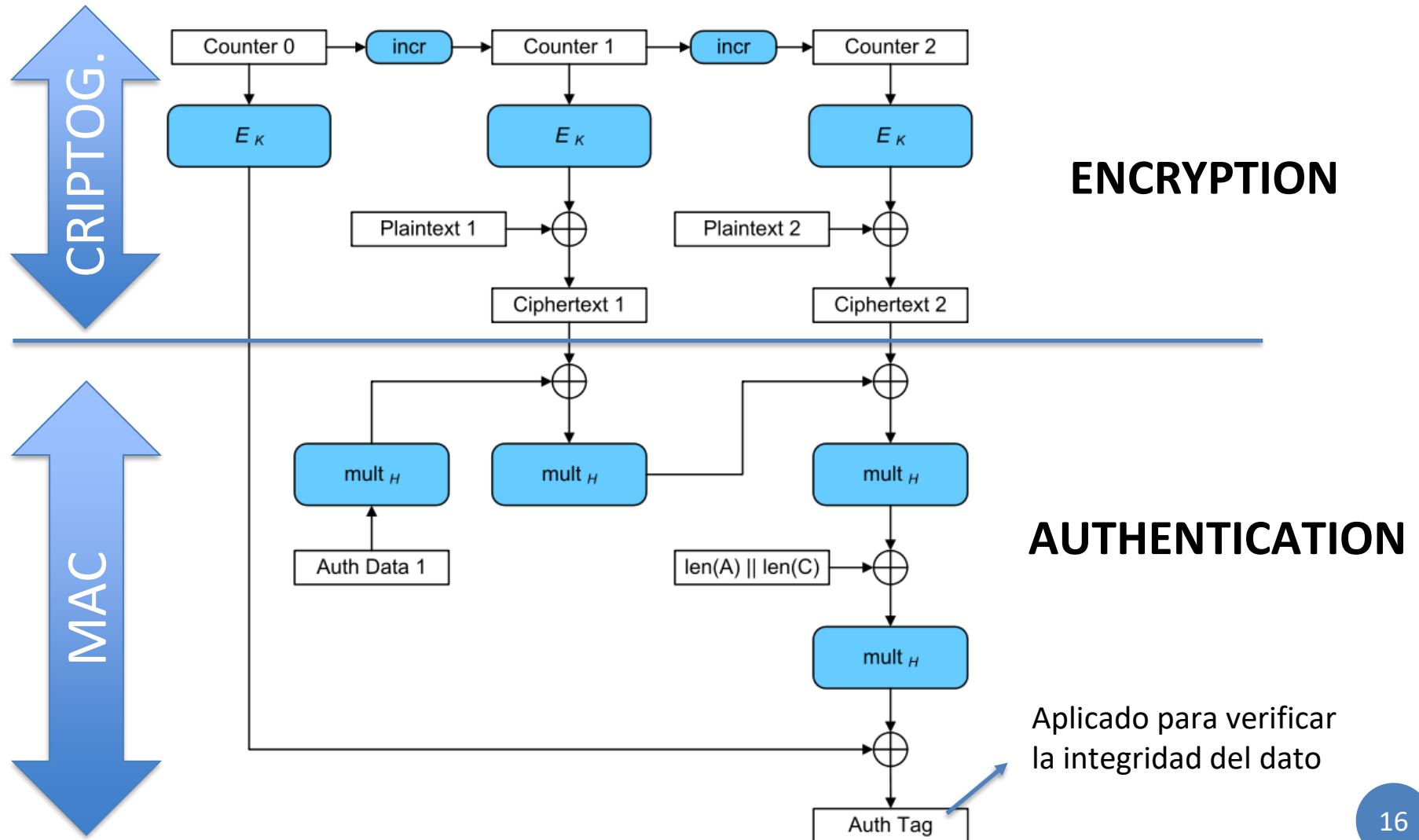
```
decipher_des = DES.new(key, DES.MODE_CBC, IV)
```

OTRO OBJETO PARA DESCIFRAR

```
new_data = unpad(decipher_des.decrypt(ciphertext), BLOCK_SIZE_DES)
               .decode("utf-8", "ignore")
print(new_data)
```

Extra: Modos de operación con integridad

- Galois-CTR (GCM)



- Modo GCM

ORIGEN

```
key = get_random_bytes(16) # Clave aleatoria de 128 bits
IV = get_random_bytes(16//2) # Nonce aleatorio de 64 bits para GCM
BLOCK_SIZE_AES = 16 # Bloque de 128 bits
data = "Hola amigos de la seguridad".encode("utf-8") # Datos a cifrar
print(data)
mac_size=16

aes_cipher = AES.new(key, AES.MODE_GCM, nonce=IV, mac_len=mac_size)
ciphertext,mac_cifrado = aes_cipher.encrypt_and_digest(pad(data,BLOCK_SIZE_AES))
print(ciphertext)
```

- Modo GCM

```
key = get_random_bytes(16) # Clave aleatoria de 128 bits
IV = get_random_bytes(16//2) # Nonce aleatorio de 64 bits para GCM
BLOCK_SIZE_AES = 16 # Bloque de 128 bits
data = "Hola amigos de la seguridad".encode("utf-8") # Datos a cifrar
print(data)
mac_size=16
```

```
aes_cipher = AES.new(key, AES.MODE_GCM, nonce=IV, mac_len=mac_size)
ciphertext, mac_cifrado = aes_cipher.encrypt_and_digest(pad(data,BLOCK_SIZE_AES))
print(ciphertext)
```



```
Try:
aes_decipher = AES.new(key, AES.MODE_GCM, nonce=IV)
text=unpad(aes_decipher.decrypt_and_verify(ciphertext, mac_cifrado),
BLOCK_SIZE_AES).decode("utf-8", ignore))
print(text)
```

```
Except (ValueError,KeyError) as e:
    print("ERROR")
```



DESTINO

Hay que descifrar y verificar

- Modo GCM

```
key = get_random_bytes(16) # Clave aleatoria de 128 bits
IV = get_random_bytes(16//2) # Nonce aleatorio de 64 bits para GCM
BLOCK_SIZE_AES = 16 # Bloque de 128 bits
data = "Hola amigos de la seguridad".encode("utf-8") # Datos a cifrar
print(data)
mac_size=16
```

ORIGEN

```
aes_cipher = AES.new(key, AES.MODE_GCM, nonce=IV, mac_len=mac_size) UN OBJETO PARA CIFRAR
ciphertext, mac_cifrado = aes_cipher.encrypt_and_digest(pad(data,BLOCK_SIZE_AES))
print(ciphertext)
```

Try:

DESTINO

```
aes_decipher = AES.new(key, AES.MODE_GCM, nonce=IV) OTRO OBJETO PARA DESCIFRAR
text=unpad(aes_decipher.decrypt_and_verify(ciphertext, mac_cifrado),
BLOCK_SIZE_AES).decode("utf-8", ignore))

print(text)

Except (ValueError,KeyError) as e:
    print("ERROR")
```

Referencias bibliográficas

Bibliografía básica

- “Python 3 documentation”

<https://docs.python.org/3/tutorial/>

- PyCryptodome

<https://pycryptodome.readthedocs.io/en/latest/src/util/util.html>