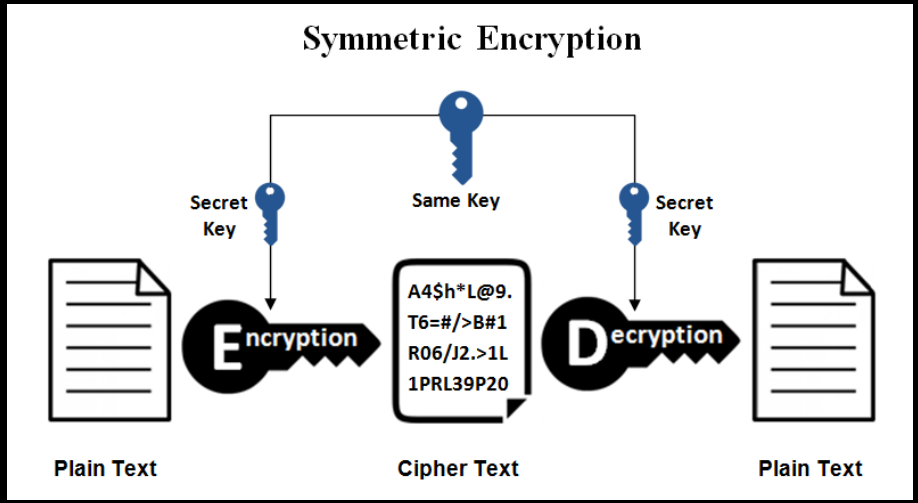
Cifrados simétricos

Usan una sola **clave** para cifrar y descifrar.

Esta clave se comparte con quien quiera descifrarlo.

Sin la clave, es muy **difícil** recuperar los datos cifrados.



HACKMADRID
%27

Existen dos tipos

de cifrados simétricos

Cifrados de bloque:

agrupan información en bloques y la cifran/descifran.

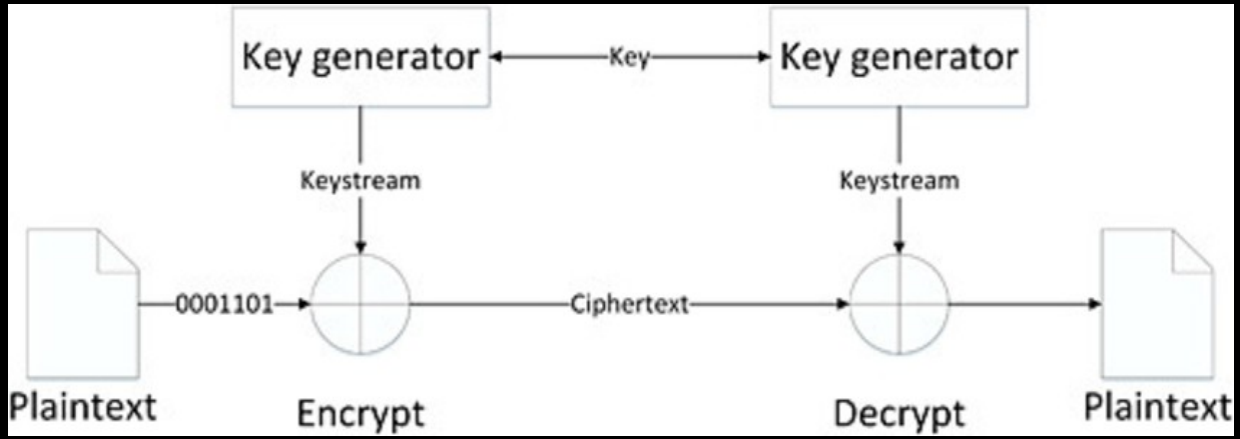
Cifrados de flujo:

reciben info dígito a dígito, y cifran/descifran cada uno.

Cifrados de flujo

Van dígito a dígito sincronizando texto y clave.

Mezclan un dígito del texto con uno de la clave.



HACKMADRID
%27

Cifrados de flujo comunes

A5 se utilizó para encriptar telefonía GSM.

Algoritmo se mantuvo en secreto y tiene vulns.

RC4 fue el estándar en WEP y SSL, pero muy débil:

roto fácilmente. Los primeros bytes son muy poco aleatorios.

ChaCha es muy utilizado ahora.

TLS, Google, etc.

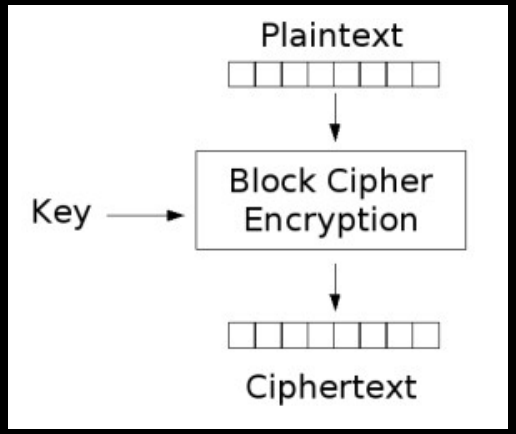
Cifrados de bloques

Parten el mensaje en bloques de X bytes.

Cifran cada bloque con el algoritmo en cuestión, y una clave elegida.

Hay “modos”, maneras distintas de usar el algoritmo.

Para descifrar se aplica la inversa del algoritmo.



HACKMADRID %27

Son más populares

Con diferencia son los más usados en cualquier implementación de la criptografía.

Algunos de los algoritmos más conocidos son:

AES (“Rijndael”), Serpent, IDEA, DES/3DES.

Algunos aceptan bloques de **8 bytes** (DES, IDEA…)

Otros utilizan bloques de **16 bytes** (AES).

AES: suficiente para la NSA

Ganó entre varios finalistas: “Advanced Encryption Standard”.

Claves de **128**, 196 y **256** bits.

Gobierno EE.UU.: SECRET y TOP SECRET autorizados para ser cifrados con AES.

HACKMADRID
%27

Cómo usa los bloques AES

Se parte el texto plano en bloques de **16 bytes**.

Antes de empezar, **siempre** se aplica un **relleno** (*padding*) al final del texto plano. Asegura bloques de 16.

PKCS#7 es el padding más común para criptografía.

Cada bloque se hace pasar por unas operaciones matemáticas, mezclándolo con la clave que proveíste.

Relleno PKCS criptográfico

El padding PKCS#7 para AES funciona así:

Si en el texto plano **quedan por rellenar N bytes** para completar un bloque de 16, **se llena ese espacio con bytes de valor N**.

Si el bloque está completo, se añade al final un **bloque entero** (16 bytes), con el valor de “16”, que es 0x10.

Ahora vemos ejemplos de esto...

Por ejemplo...

Si queda un byte, se llena con “unos” (0x01)

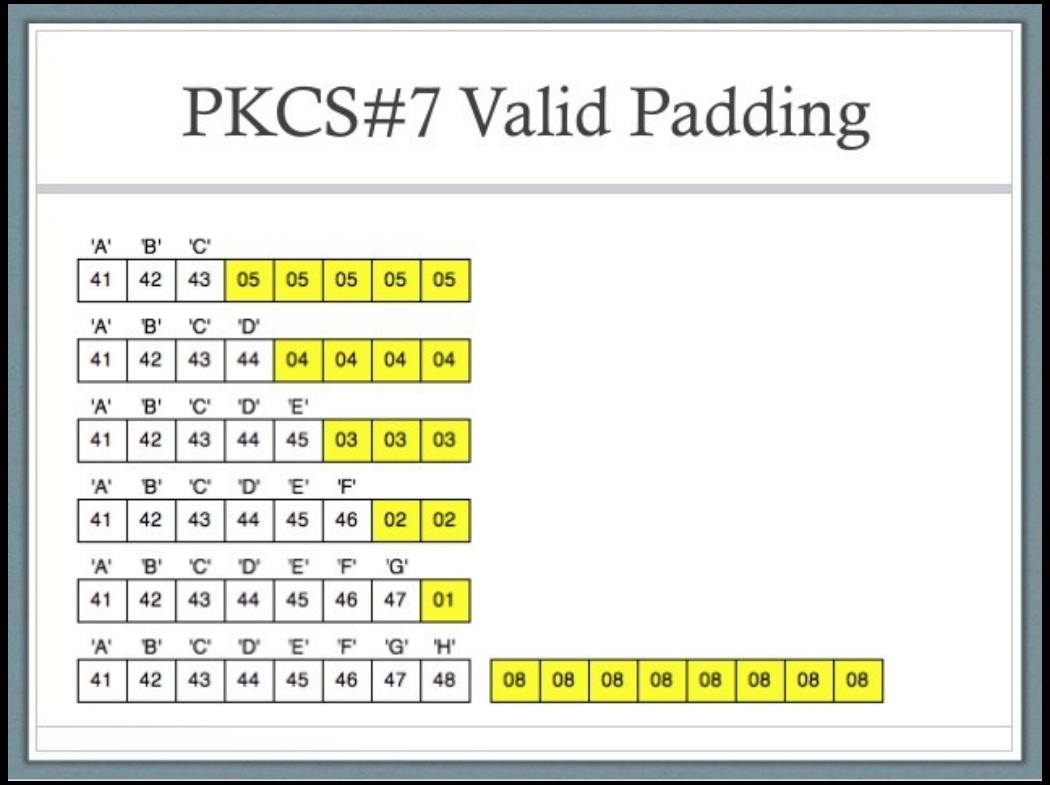
Si quedan 3 bytes, con “treses” (0x03 0x03 0x03).

Si quedan 0, añadimos 16 veces “16” (0x10 … 0x10).

El número son los bytes que hemos añadido.

Ayuda al ordenador a distinguir entre info y relleno.

PKCS rellenando 8 bytes en vez de 16

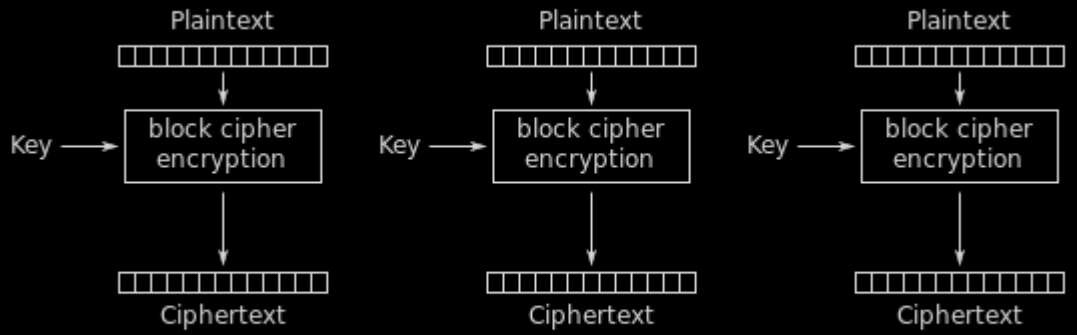


HACKMADRID
%27

La aplicación simple: ECB

Electronic Code Book es un "modo" de AES, una manera de aplicarlo.

Cogemos cada bloque de texto y lo ciframos tal cual.



Electronic Codebook (ECB) mode encryption

HACKMADRID
%27

Anda, ECB funciona guay.

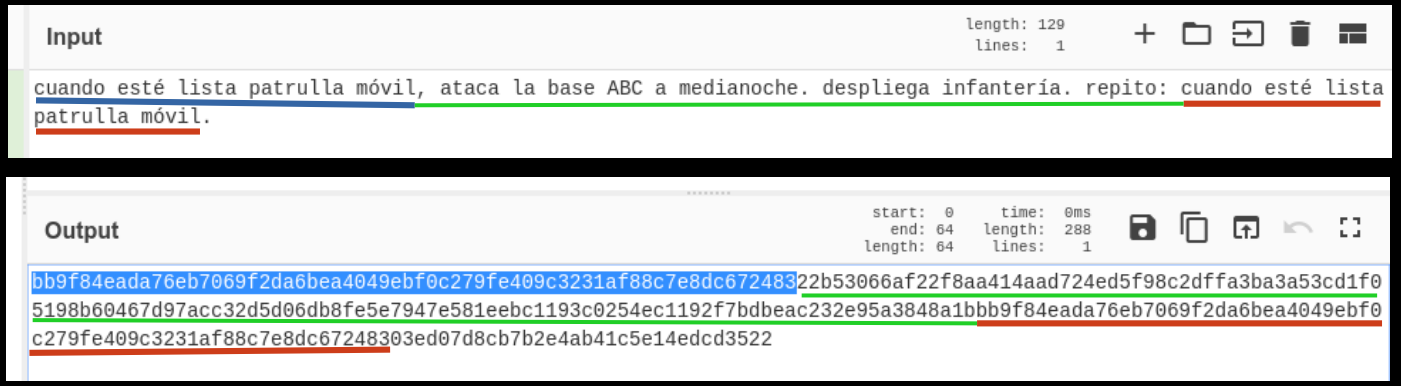
Input length: 129 lines: 1

```
cuando esté lista patrulla móvil, ataca la base ABC a medianoche. despliega infantería. repito: cuando esté lista patrulla móvil.
```

Output start: 0 end: 64 length: 64 time: 0ms length: 288 lines: 1

```
bb9f84eada76eb7069f2da6bea4049ebf0c279fe409c3231af88c7e8dc67248322b53066af22f8aa414aad724ed5f98c2dffa3ba3a53cd1f0
5198b60467d97acc32d5d06db8fe5e7947e581eebc1193c0254ec1192f7bdbeac232e95a3848a1bbb9f84eada76eb7069f2da6bea4049ebf0
c279fe409c3231af88c7e8dc67248303ed07d8cb7b2e4ab41c5e14edcd3522
```

Whoops.



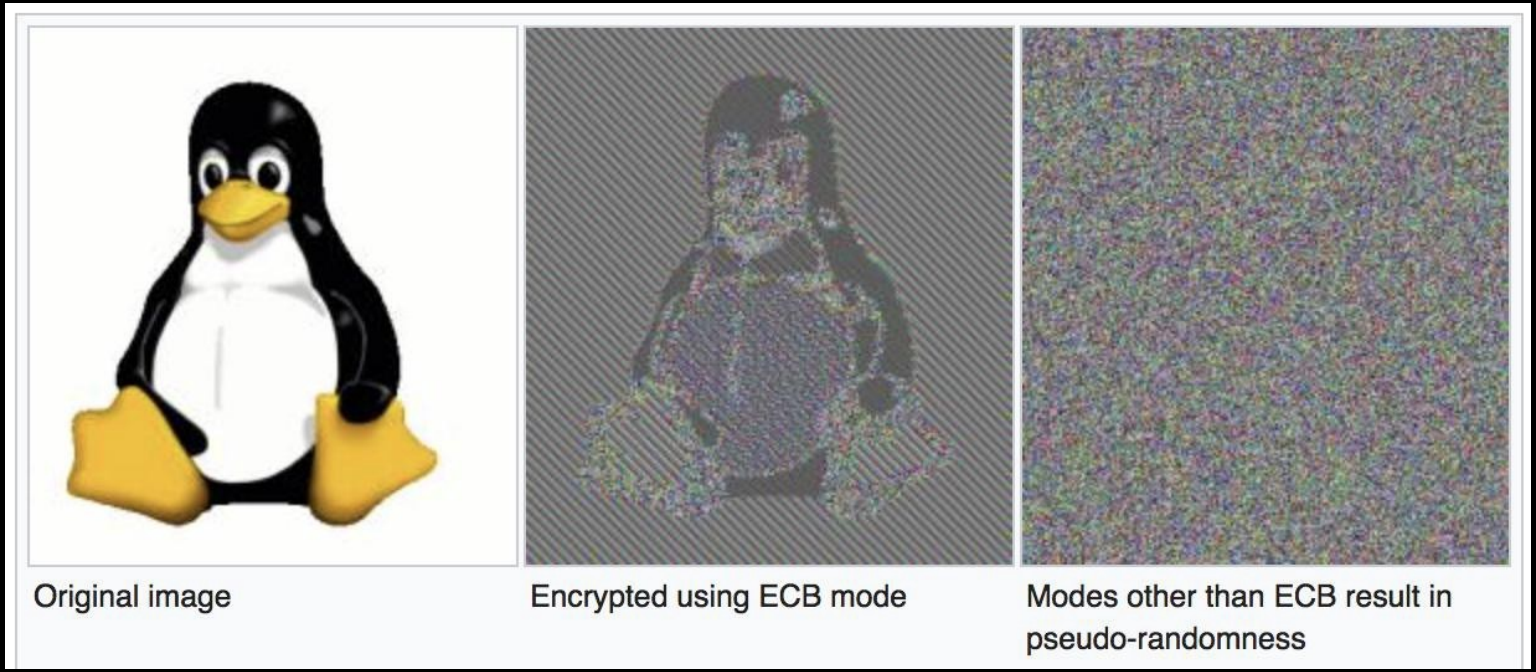
16 bytes del mismo texto pueden cifrarse de la misma forma. Sólo hace falta que se alineen bien.

En *azul* 2 x 16B. En *verde* 4 x 16B de mensaje. En *rojo* 2 x 16B.

“cuando esté lista patrulla móvil” -> “bb9f...2483”

HACKMADRID %27

“Ya no me parece tan bonito ese algoritmo tan seguro...”



Original image

Encrypted using ECB mode

Modes other than ECB result in pseudo-randomness

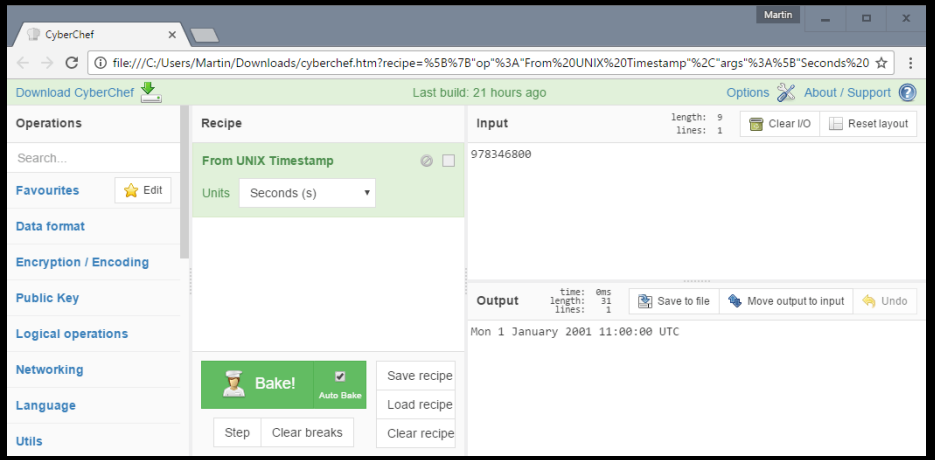
HACKMADRID %27

Demo Time (ECB).

Se puede usar la página de Cyberchef.

https://gchq.github.io/CyberChef/

También (avanzado) existe la lib *cryptography* en Python. Las funciones que componen AES se encuentran en "hazmat".
https://cryptography.io/




```
>>> import os
>>> from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
>>> from cryptography.hazmat.backends import default_backend
>>> backend = default_backend()
>>> key = os.urandom(32)
>>> iv = os.urandom(16)
>>> cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=backend)
>>> encryptor = cipher.encryptor()
>>> ct = encryptor.update(b"a secret message") + encryptor.finalize()
>>> decryptor = cipher.decryptor()
>>> decryptor.update(ct) + decryptor.finalize()
b'a secret message'
```

HACKMADRID
%27

Entra en el modo CBC

Cada bloque de texto cifrado depende del anterior.

Se considera como el modo más confiable.

Requiere un “vector de inicialización” (IV) además de la clave.

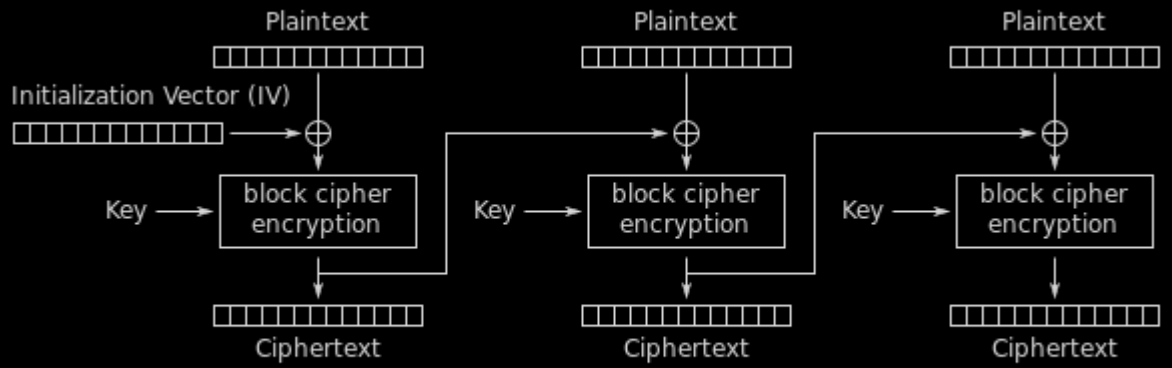
- El IV es de 16 bytes, igual que un bloque.

Cómo funciona CBC

Cada bloque cifrado se calcula como:

AESCifrar(texto plano XOR texto cifrado anterior).

El IV se necesita para hacer la primera suma, ya que no hay texto cifrado anterior.



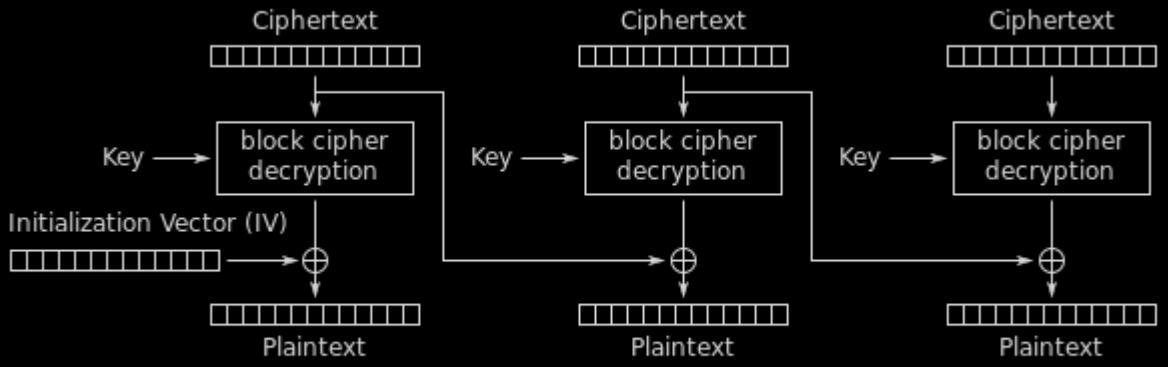
Cipher Block Chaining (CBC) mode encryption

HACKMADRID
%27

Descifrando CBC

La operación inversa sigue siendo posible.

Necesitamos **la clave y también el IV** para obtener el texto plano.



Cipher Block Chaining (CBC) mode decryption

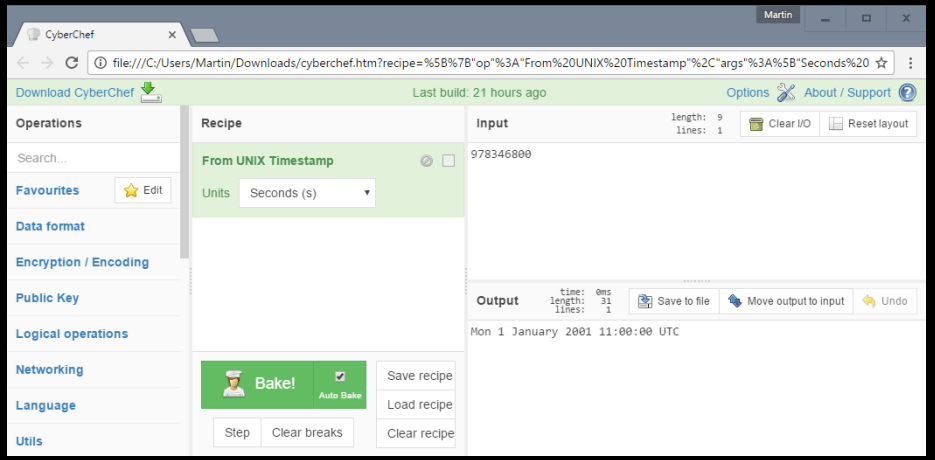
HACKMADRID
%27

Demo Time (CBC).

Podéis usar de nuevo CyberChef o *cryptography* en Python.
Comprueba que ahora no existen los problemas de ECB.

https://gchq.github.io/CyberChef/

https://cryptography.io/




```
>>> import os
>>> from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
>>> from cryptography.hazmat.backends import default_backend
>>> backend = default_backend()
>>> key = os.urandom(32)
>>> iv = os.urandom(16)
>>> cipher = Cipher(algorithms.AES(key), modes.CBC(iv), backend=backend)
>>> encryptor = cipher.encryptor()
>>> ct = encryptor.update(b"a secret message") + encryptor.finalize()
>>> decryptor = cipher.decryptor()
>>> decryptor.update(ct) + decryptor.finalize()
b'a secret message'
```