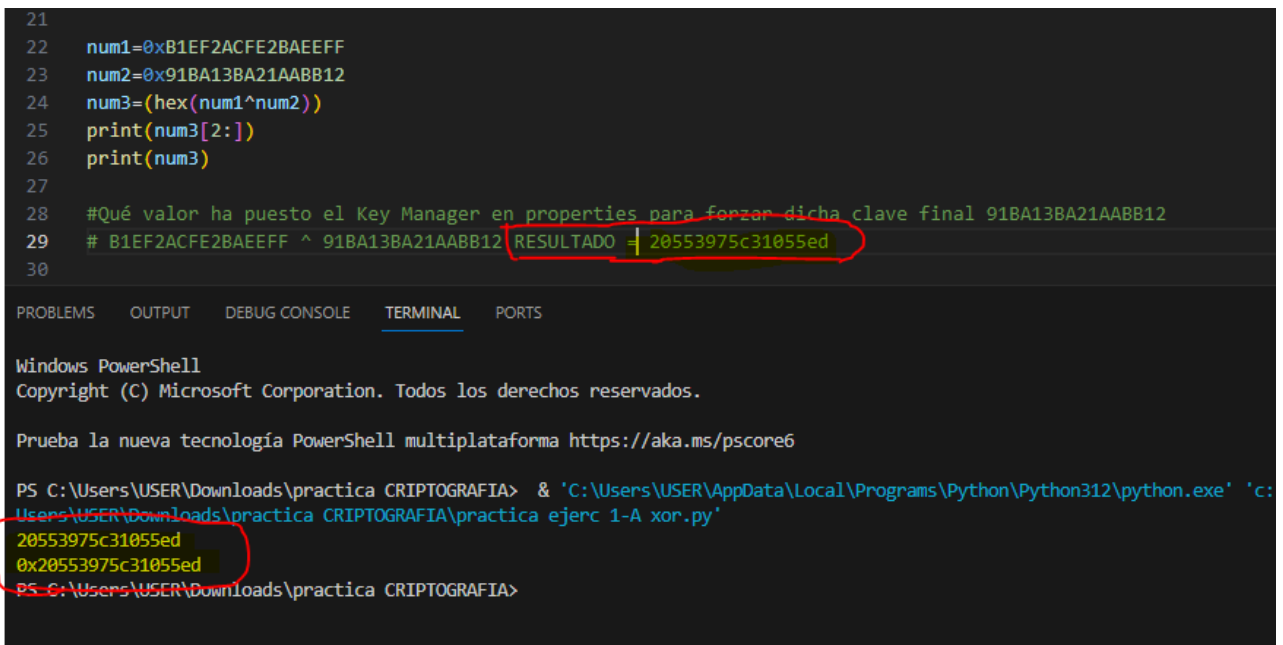


SOLUCIONES PRÁCTICA CRIPTOGRAFÍA

1 Tenemos un sistema que usa claves de 16 bytes. Por razones de seguridad vamos a proteger la clave de tal forma que ninguna persona tenga acceso directamente a la clave. Por ello, vamos a realizar un proceso de disociación de la misma, en el cuál tendremos, una clave fija en código, la cual, sólo el desarrollador tendrá acceso, y otra parte en un fichero de propiedades que rellenará el Key Manager. La clave final se generará por código, realizando un XOR entre la que se encuentra en el properties y en el código.

RESPUESTA:

20553975c31055ed



```
21
22 num1=0xB1EF2ACFE2BAEEFF
23 num2=0x91BA13BA21AABB12
24 num3=(hex(num1^num2))
25 print(num3[2:])
26 print(num3)
27
28 #Qué valor ha puesto el Key Manager en properties para forzar dicha clave final 91BA13BA21AABB12
29 # B1EF2ACFE2BAEEFF ^ 91BA13BA21AABB12 RESULTADO = 20553975c31055ed
30
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma <https://aka.ms/pscore6>

PS C:\Users\USER\Downloads\practica CRIPTOGRAFIA> & 'C:\Users\USER\AppData\Local\Programs\Python\Python312\python.exe' 'c:\Users\USER\Downloads\practica CRIPTOGRAFIA\practica ejerc 1-A xor.py'

20553975c31055ed
0x20553975c31055ed

PS C:\Users\USER\Downloads\practica CRIPTOGRAFIA>

La clave fija en código es B1EF2ACFE2BAEEFF, mientras que en desarrollo sabemos que la clave final (en memoria) es 91BA13BA21AABB12. ¿Qué valor ha puesto el Key Manager en properties para forzar dicha clave final?

RESPUESTA:

8653f75d31455c0

```
22 num1=0xB1EF2ACFE2BAEEFF
23 num2=0xB98A15BA31AEBB3F
24 num3=(hex(num1^num2))
25 print(num3[2:])
26 print(num3)
27
28 # B1EF2ACFE2BAEEFF ^ B98A15BA31AEBB3F RESULTADO = 8653f75d31455c0
29
```

PROBLEMS OUTPUT DEBUG CONSOLE **TERMINAL** PORTS

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos los derechos reservados.

Prueba la nueva tecnología PowerShell multiplataforma <https://aka.ms/pscore6>

PS C:\Users\USER\Downloads\practica CRIPTOGRAFIA> & 'C:\Users\USER\AppData\Local\Programs\Python\Python3\Users\USER\Downloads\practica CRIPTOGRAFIA\practica ejerc 1-A xor.py'

8653f75d31455c0
0x8653f75d31455c0

PS C:\Users\USER\Downloads\practica CRIPTOGRAFIA>

2 Dada la clave con etiqueta “cifrado-sim-aes-256” que contiene el keystore. El iv estará compuesto por el hexadecimal correspondiente a ceros binarios (“00”). Se requiere obtener el dato en claro correspondiente al siguiente dato cifrado: TQ9SOMKc6aFS9SlxhfK9wT18UXpPCd505Xf5J/5nLI7Of/o0QKIWXg3nu1RRz4QWElezdrLA D5LO4US t3aB/i50nvvJbBiG+le1ZhpR84oI=

Para este caso, se ha usado un AES/CBC/PKCS7. Si lo desciframos, ¿qué obtenemos?



¿Qué ocurre si decidimos cambiar el padding a x923 en el descifrado?

RESPUESTA: es igual al PKCS7 porque tiene padding de 1 byte, NO OCURRE NADA. Fallaría en

una situación normal porque no tiene el PKCS7

¿Cuánto padding se ha añadido en el cifrado?

se ha añadido el último dígito porque, al poner NO PADDING en Cyberchef, no quitará el padding.

The screenshot shows the CyberChef interface with an AES Decrypt recipe. The Key is set to A2CFF885901A5449E... and the IV is 0000000000000000... The Mode is CBC/NoPadding. The Input is a long hexadecimal string. The Output is a long hexadecimal string.

Recipe	Input
AES Decrypt Key: A2CFF885901A5449E... IV: 0000000000000000... Mode: CBC/NoPadding Input: Hex Output: Hex	4d0f5238c29ce9a152f5297185f2bdc13d7c517a4f09de74e577f927fe672c8ece7ffa3440a2165e0de7bb5451cf84161257b376b2c00f92cee144a ddda07f8b9d27bef25b0621be95ed5986947ce28z

Output: 4573746f20657320756e206369667261646f20656e20626c6f7175652074c3ad7069636f2e2052656375657264612c2076617320706f7220656c206
275656e2063616d696e6f2e20c3816e696d6f2e01

4 Tenemos el siguiente jwt, cuya clave es “Con KeepCoding aprendemos”.
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmVlIjoibG9uIFBlcGl0byBkZSB
sb3MgcGFsb3RlcylsInJvbCI6ImIzTm9ybWFsIiwiaWF0IjoxNjY3OTMzMzQzLnR5bW0
dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE

¿Qué algoritmo de firma hemos realizado?

RESPUESTA : HMACSHA256

Algorithm HS256

Encoded PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmVlIjoibG9uIFB1cG10byBkZSBsb3MgcGFsb3RlcyIsInJvbCI6Im1zTm9ybWFsIiwiaWF0IjoxNjY3OTMzMzQ.gfhw0dDxp6oixMLXXRP97W4TDTrv0y7B5YjD0U8ixrE
```

Decoded EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{
  "typ": "JWT",
  "alg": "HS256"
}
```

PAYLOAD: DATA

```
{
  "usuario": "Don Pepito de los palotes",
  "rol": "isNormal",
  "iat": 1667933533
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

¿Cuál es el body del jwt?

RESPUESTA:

```
{
  "usuario": "Don Pepito de los palotes",
  "rol": "isNormal",
  "iat": 1667933533
}
```

Un hacker está enviando a nuestro sistema el siguiente jwt:

Está intentando escalar privilegios sin una firma válida, dentro del cuerpo se muestra el rol: isAdmin, es decir quiere tener privilegios de administrador

Algorithm

HS256

Encoded

PASTE A TOKEN HERE

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJ1c3VhcmIvIjoirG9uIFB1cG10byBkZSBsb3MgcGFsb3R1cyIsInJvbCI6Im1zQWRtaW4iLCJpYXQiOiE2Njc5MzM1MzN9.krgBkzCBQ5WZ8JnZHuRvmnAZdg4ZMeRNv2CIAOD1HRI
```

Decoded

EDIT THE PAYLOAD AND SECRET

HEADER: ALGORITHM & TOKEN TYPE

```
{  "typ": "JWT",  "alg": "HS256"}
```

PAYLOAD: DATA

```
{  "usuario": "Don Pepito de los palotes",  "rol": "isAdmin",  "iat": 1667933533}
```

VERIFY SIGNATURE

```
HMACSHA256(  base64UrlEncode(header) + "." +  base64UrlEncode(payload),  your-256-bit-secret)
```

☐ secret base64 encoded

⊗ Invalid Signature

SHARE JWT

¿Qué ocurre si intentamos validarlo con pyjwt?

No valida

5 El siguiente hash se corresponde con un SHA3 Keccak del texto “En KeepCoding aprendemos cómo protegernos con criptografía”.
¿Qué tipo de SHA3 hemos generado?

SHA3-256

This SHA3-256 online tool helps you calculate hash from string or binary. You can input UTF-8, UTF-16, Hex to SHA3-256.

Input Type UTF-8

En KeepCoding aprendemos cómo protegernos con criptografía

☐ Remember Input

Hash ☒ Auto Update

bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d56fe

SHA3

[SHA3-224](#)

[SHA3-224 File](#)

[SHA3-256](#)

[SHA3-256 File](#)

[SHA3-384](#)

[SHA3-384 File](#)

[SHA3-512](#)

[SHA3-512 File](#)

RESPUESTA:

`sha3_256`

por la longitud de salida que tiene

Y si hacemos un SHA2, y obtenemos el siguiente resultado:
4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f
6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833 ¿Qué hash hemos realizado?

SHA2-512

Por la longitud que tiene

SHA-2 Hash Generator

[Add to Fav](#)[New](#)[Save & Share](#)

Enter the plain or Cipher Text:

[Sample](#)

En KeepCoding aprendemos cómo protegernos con criptografía

Size : 58 B, 58 Characters

☒ Auto[Generate](#)[File..](#)[Load URL](#)

SHA2-224

a4544beb16e1dfb9b578d518bf19e2a8109ffe27cab9172911e5!

56
length

SHA2-256

13067f558aed141a490bf95775e0c6fc583a09178ae7a0fefe93a
8336be8123764
length

SHA2-384

dca9a06f36b492b374216e60dc7668bea8119ec35ca259aa797
ec8125654f4dc088144b00f16d5155bcb3c1e295784f496
length

SHA2-512

4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f
4f2908aa5d63739506f6468833d77c07cfd69c488823b8d85828
3f1d05877120e8c5351c833128
length

Genera ahora un SHA3-256 bits con el siguiente texto: “En KeepCoding aprendemos cómo protegernos con criptografía.” ¿Qué propiedad destacarías del hash, atendiendo a los resultados anteriores?

La difusión y efecto avalancha: un cambio pequeño, provoca un enorme cambio en el hash

SHA3-256: 302be507113222694d8c63f9813727a85fef61a152176ca90edf1cfb952b19bf

Código fuente python utilizado:

```
C:\> Users > USER > Documents > bootcamp CIBERSEGURIDAD > ejercicios resueltos pract cripto > practica ejerc 5 keccak.py > ...
1  import hashlib
2
3
4  # initiating the "s" object to use the sha512 del ejercicio
5  s = hashlib.sha512()
6  s.update(bytes("En KeepCoding aprendemos cómo protegernos con criptografía", "UTF-8"))
7  print("sha_512 "+s.digest().hex())
8  #sha3-256 keccak sin punto
9  s = hashlib.sha3_256()
10 s.update(bytes("En KeepCoding aprendemos cómo protegernos con criptografía", "UTF-8"))
11 print("sha3_256 "+s.digest().hex())
12
13 s= hashlib.sha3_256()
14 s.update(bytes("En KeepCoding aprendemos cómo protegernos con criptografía.", "utf8"))
15 print("SHA3-256: " + s.digest().hex())
16
17
```

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS

[Running] set PYTHONIOENCODING=utf8 && python -u "c:\Users\USER\Documents\bootcamp CIBERSEGURIDAD\ejercicios resueltos pract cripto\practica ejerc 5 keccak.py"

sha_512 4cec5a9f85dcc5c4c6ccb603d124cf1cdc6dfe836459551a1044f4f2908aa5d63739506f6468833d77c07cfd69c488823b8d858283f1d05877120e8c5351c833

sha3_256 bced1be95fbd85d2ffcce9c85434d79aa26f24ce82fbd4439517ea3f072d56fe

SHA3-256: 302be507113222694d8c63f9813727a85fef61a152176ca90edf1cfb952b19bf

[Done] exited with code=0 in 0.088 seconds

7 Trabajamos en una empresa de desarrollo que tiene una aplicación web, la cual requiere un login y trabajar con passwords. Nos preguntan qué mecanismo de almacenamiento de las mismas proponemos. Tras realizar un análisis, el analista de seguridad propone un hash SHA-1. Su responsable, le indica que es una mala opción. ¿Por qué crees que es una mala opción?

RESPUESTA: El SHA-1 está deprecado (obsoleto) está roto, dado un hash podemos obtener el texto en claro

Después de meditarlo, propone almacenarlo con un SHA-256, y su responsable le pregunta si no lo va a fortalecer de alguna forma. ¿Qué se te ocurre?

RESPUESTA: se puede mejorar con un SALT y mejor aún con un PEPPER

Parece que el responsable se ha quedado conforme, tras mejorar la propuesta del SHA-256, no obstante, hay margen de mejora. ¿Qué propondrías?

RESPUESTA: usar un algoritmo p ej. ARGON2ID

8 ¿Qué algoritmos usarías?

RESPUESTA:

Utilizaría un sistema basado en cifrado más un mac, por ejemplo AES y HMAC, o un AES y CBC-MAC.

Podría ser también AES-GCM, que es el más potente.

Si consideramos el rendimiento habría que modificar el API, teniendo una parte encriptada (ENCRYPTED) dentro de la información (INFO), y al mismo nivel que el INFO, el MAC.

10 El responsable de Raúl, Pedro, ha enviado este mensaje a RRHH:

Se debe ascender inmediatamente a Raúl. Es necesario mejorarle sus condiciones económicas un 20% para que se quede con nosotros.

Lo verificamos de esta forma:

```
gpg --verify MensajeRespoDeRaulARRHH.txt.sig MensajeRespoDeRaulARRHH.txt
Generamos la firma:
```

```
gpg -u F2B1D0E8958DF2D3BDB6A1053869803C684D287B --output dpc.sig --detach-sign
MensajeRRHHARespoDeRaul.txt
```

También podemos lanzar este script:

```
gpg --local-user F2B1D0E8958DF2D3BDB6A1053869803C684D287B --armor --output dpc.sig
--detach-sign RRHHRespuesta.txt
```

Ciframos el mensaje con el siguiente comando:

```
gpg --output FicheroCifrado.gpg --encrypt --recipient
1BDE635E4EAE6E68DFAD2F7CD730BE196E466101 --recipient
F2B1D0E8958DF2D3BDB6A1053869803C684D287B FicheroParaCifrar.txt.
```

El resultado es el fichero FicheroCifrado.gpg

11 Nuestra compañía tiene un contrato con una empresa que nos da un servicio de almacenamiento de información de videollamadas. Para lo cual, la misma nos envía la clave simétrica de cada videollamada cifrada usando un RSA-OAEP. El hash que usa el algoritmo interno es un SHA-256.

Si has recuperado la clave, vuelve a cifrarla con el mismo algoritmo. ¿Por qué son diferentes los textos cifrados?

RESPUESTA: Descifrado: e2cfff885901a5449e9c448ba5b948a8c4ee377152b3f1acfa0148fb3a426db72

son diferentes porque al generar el padding se usa un random que cambia el texto cifrado

13 Se desea calcular una firma con el algoritmo PKCS#1 v1.5 usando las claves contenidas en los ficheros clave-rsa-oaep-priv y clave-rsa-oaep-publ.pem del mensaje siguiente: El equipo está preparado para seguir con el proceso, necesitaremos más recursos. ¿Cuál es el valor de la firma en hexadecimal?

RESPUESTA:

Firma
a4606c518e0e2b443255e3626f3f23b77b9d5e1e4d6b3dcf90f7e118d6063950a23885c6dece92aa3d6
eff2a72886b2552be969e11a4b7441bdeadc596c1b94e67a8f941ea998ef08b2cb3a925c959bcaae2ca
9e6e60f95b989c709b9a0b90a0c69d9eaccd863bc924e70450ebbbb87369d721a9ec798fe66308e0454
17d0a56b86d84b305c555a0e766190d1ad0934a1befbbe031853277569f8383846d971d0daf05d02354
5d274f1bdd4b00e8954ba39dacc4a0875208f36d3c9207af096ea0f0d3baa752b48545a5d79cce0c2eb
b6fff601d92978a33c1a8a707c1ae1470a09663acb6b9519391b61891bf5e06699aa0a0dbae21f0aaaa6

f9b9d59f41928d

Calcula la firma (en hexadecimal) con la curva elíptica ed25519, usando las claves ed25519-priv y ed25519-publ.

```
(kali@kali)-[~/Documents]  
$ python pyt.py
```

Firma Generada (64 bytes):
b'470434f69bb45c7772bc64bc164081bbce669e5c24b98d5b3f77b903ecb321d9a12af9aa09f9ffae6e4732612e7e09e03772be03fc4025f6b485c0d7c9f8f80d'

14 Necesitamos generar una nueva clave AES, usando para ello una HKDF (HMAC-based Extractand-Expand key derivation function) con un hash SHA-512. La clave maestra requerida se encuentra en el keystore con la etiqueta “cifrado-sim-aes-256”. La clave obtenida dependerá de un identificador de dispositivo, en este caso tendrá el valor en hexadecimal:

La clave es:

Clave key1: e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a

```
C: > Users > USER > Documents > bootcamp CIBERSEGURIDAD > ejercicios resueltos pract cripto > practica ejerc 14 HKDF.py > ...  
1  from Crypto.Protocol.KDF import HKDF  
2  from Crypto.Hash import SHA512  
3  import secrets  
4  from Crypto.Random import get_random_bytes  
5  
6  #elemento_diversificacion = secrets.token_bytes(16)  
7  #clave obtenida de un identificador de dispositivo  
8  elemento_diversificacion = bytes.fromhex("e43bb4067cbc fab3bec54437b84bef4623e345682d89de9948fbb0afedc461a3"  
9  
10 #salt = get_random_bytes(16)  
11 #master_secret = secrets.token_bytes(64)  
12 #clave maestra requerida que se encuentra en el keystore con la etiqueta "cifrado-sim-aes-256".  
13 master_secret = bytes.fromhex("A2CFF885901A5449E9C448BA5B948A8C4EE377152B3F1ACFA0148FB3A426DB72")  
14  
15  
16  
17 key1, key2 = HKDF(master_secret, 32, elemento_diversificacion, SHA512, 2)  
18  
19 print("Clave key1: ", key1.hex())  
20  
  
PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS  
[Running] set PYTHONIOENCODING=utf8 && python -u "c:\Users\USER\Documents\bootcamp CIBERSEGURIDAD\ejercicios resu  
Clave key1: e716754c67614c53bd9bab176022c952a08e56f07744d6c9edb8c934f52e448a  
  
[Done] exited with code=0 in 0.157 seconds
```

15 Nos envían un bloque TR31:

D0144D0AB00S000042766B9265B2DF93AE6E29B58135B77A2F616C8D515ACDB
E6A5626F79FA7B4071E9EE1423C6D7970FA2B965D18B23922B5B2E5657495E0
3CD857FD37018E111B

Algoritmo

AES

Modo de uso:

Para cifrar y descifrar

Exportabilidad:

SI

Uso de la clave:

Para cifrar datos

Valor clave:

c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1c1

código fuente python:

```
C:\Users > USER > Documents > bootcamp CIBERSEGURIDAD > ejercicios resueltos pract cripto > practica ejerc 15 cabeceraTR31.py > ...
1 from psec import tr31
2
3 #Documentado en este fichero
4 #https://github.com/knovichikhin/psec/blob/master/psec/tr31.py
5
6 header, key = tr31.unwrap( (b'pk-bytes.fromhex("A1A1010101010101010101010102")',
7                             key_block="D8144D8AB00500042766B9265B2DF93AE6E29B58135B77A2F616C8D515ACDBE6A5626F79FA7B4071E9EE1423C6D7970FA2B965D18E23922B5B2E5657495E03C8D57FD37018E111B")')
8 print(key.hex())
9
10 print("Key Version ID: " + header.version_id )
11 print("Algoritmo: " + header.algorithm)
12 print("Modo de uso: " + header.mode_of_use)
13 print("Uso de la clave: " + header.key_usage)
14 print("Exportabilidad: " + header.exportability)
```