P1 Seguridad

Tu turno 1: Probar el padding en el algoritmo DES en modo ECB. Probar el AES con el modo de operación CFB. Probar el AES con tamaños de claves superiores a 128 en modo CFB. Probar el DES3 con el modo de operación CFB.

Algoritmo DES en modo ECB

```
from Crypto.Cipher import DES
from Crypto import Random
import base64

key = b'11111111'

cipher = DES.new(key, DES.MODE_ECB)
msg = b'22222222'
print('Mensaje plano -->', msg)

msgc = cipher.encrypt(msg)
print('Mensaje cifrado -->', msgc)

msgd = cipher.decrypt(msgc)
print('Mensaje descifrado -->', msgd)
```

```
('Mensaje plano -->', '22222222')
('Mensaje cifrado -->', "\xf9'\x14\xe8\x0e\x83.\xdc")
('Mensaje descifrado -->', '22222222')
```

Probar padding en el algoritmo DES en modo ECB

```
from Crypto.Cipher import DES
from Crypto import Random
import base64

HEXA= lambda xx: ':'.join(hex(ord(x))[2:] for x in xx)

key = b'11111111'

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

msg = b'2222222222222222333'
print('Mensaje plano -->', msg)
print('\n')

msg=rellena_datos(msg, DES.block_size)
print('Mensaje plano con padding-->', msg)

msgc = cipher.encrypt(msg)
print('Mensaje cifrado -->', msgc)

msgd = cipher.decrypt(msgc)
print('Mensaje descifrado sin quitar el padding-->', msgd)
print('\n')

msgd=quita_relleno_datos(msgd,DES.block_size)
print('Mensaje descifrado quitando el padding-->', msgd)
```

```
('Mensaje plano -->', '2222222222222222333')


('Mensaje plano con padding-->', '22222222222222223333\x80\x00\x00\x00\x00')
('Mensaje cifrado -->', "\xf9'\x14\xe8\x0e\x83.\xdc\xf9'\x14\xe8\x0e\x83.\xdc\xdaD\xce\xfa\xe0\xe3\x92\xaf")
('Mensaje descifrado sin quitar el padding-->', '22222222222222223333\x80\x00\x00\x00\x00')


('Mensaje descifrado quitando el padding-->', '2222222222222222333')
```

Probar el AES en modo CFB

```
from Crypto.Cipher import AES
from Crypto import Random

key = b'1111111111111111'

iv = Random.new().read(AES.block_size)
print('IV -->', iv)
cipher1 = AES.new(key, AES.MODE_CFB, iv)
msg = b'2222222222222222'
print('Mensaje plano -->', msg)
print('\n')

msgc = iv + cipher1.encrypt(msg)
print('IV + Mensaje cifrado -->', msgc)

print('\n')
print('Envio de mensaje cifrado por canal inseguro')

iv_enviado = msgc[:AES.block_size]
cipher2 = AES.new(key, AES.MODE_CFB, iv_enviado)
solo_mensaje=msgc[AES.block_size:]
print('Mensaje cifrado -->', solo_mensaje)
msgd = cipher2.decrypt(solo_mensaje)
print('IV_enviado -->', iv_enviado)
print('\n')
print('Mensaje descifrado -->', msgd)
```

```
('IV -->', '\xc2Q\x13\xd1f\xe8N\xe0\x0ec\xb4\xf50\xfe\x12T')
('Mensaje plano -->', '2222222222222222')


('IV + Mensaje cifrado -->', '\xc2Q\x13\xd1f\xe8N\xe0\x0ec\xb4\xf50\xfe\x12T-\x1fS\xe6\x8e\xf2\xff\x1fB=\xba\xaa\xbc\
xb0\x97\x10')


Envio de mensaje cifrado por canal inseguro
('Mensaje cifrado -->', '-\x1fS\xe6\x8e\xf2\xff\x1fB=\xba\xaa\xbc\xb0\x97\x10')
('IV_enviado -->', '\xc2Q\x13\xd1f\xe8N\xe0\x0ec\xb4\xf50\xfe\x12T')


('Mensaje descifrado -->', '2222222222222222')
```

Probar el AES con tamaños de claves superiores a 128 en modo CFB

```
from Crypto.Cipher import AES
from Crypto import Random

key = b'1111111111111111111111111'

iv = Random.new().read(AES.block_size)
print('IV -->', iv)
cipher1 = AES.new(key, AES.MODE_CFB, iv)
msg = b'2222222222222222222222'
print('Mensaje plano -->', msg)
print('\n')

msgc = iv + cipher1.encrypt(msg)
print('IV + Mensaje cifrado -->', msgc)

print('\n')
print('Envio de mensaje cifrado por canal inseguro')

iv_enviado = msgc[:AES.block_size]
cipher2 = AES.new(key, AES.MODE_CFB, iv_enviado)
solo_mensaje=msgc[AES.block_size:]
print('Mensaje cifrado -->', solo_mensaje)
msgd = cipher2.decrypt(solo_mensaje)
print('IV_enviado -->', iv_enviado)
print('\n')
print('Mensaje descifrado -->', msgd)
```

```
('IV -->', '\xe1?TX\xbf\xb5\xf5\xfb\x1b\x94\x07\xdb\xc1\xc4k\xc5')
('Mensaje plano -->', '2222222222222222222222')


('IV + Mensaje cifrado -->', '\xe1?TX\xbf\xb5\xf5\xfb\x1b\x94\x07\xdb\xc1\xc4k\xc5<+\xfaz`\xf4\xeda\x92\x9f\x94+\x19\
x93{?Q\xe6"s\x9d]\x8d\x1a')


Envio de mensaje cifrado por canal inseguro
('Mensaje cifrado -->', '<+\xfaz`\xf4\xeda\x92\x9f\x94+\x19\x93{?Q\xe6"s\x9d]\x8d\x1a')
('IV_enviado -->', '\xe1?TX\xbf\xb5\xf5\xfb\x1b\x94\x07\xdb\xc1\xc4k\xc5')


('Mensaje descifrado -->', '2222222222222222222222')
```

Probar el DES3 en modo CFB

```
from Crypto.Cipher import DES3
from Crypto import Random

key = b'1111111111111111'

iv = Random.new().read(DES3.block_size)
print('IV -->', iv)
cipher1 = DES3.new(key, DES3.MODE_CFB, iv)
msg = b'2222222233333333'
print('Mensaje plano -->', msg)
print('\n')

msgc = iv + cipher1.encrypt(msg)
print('IV + Mensaje cifrado -->', msgc)

print('\n')
print('Envio de mensaje cifrado por canal inseguro')

iv_enviado = msgc[:DES3.block_size]
cipher2 = DES3.new(key, DES3.MODE_CFB, iv_enviado)
solo_mensaje=msgc[DES3.block_size:]
print('Mensaje cifrado -->', solo_mensaje)
msgd = cipher2.decrypt(solo_mensaje)
print('IV_enviado -->', iv_enviado)
print('\n')
print('Mensaje descifrado -->', msgd)
```

```
('IV -->', '\xb1\x7f\xed~\xcds\xb4\x87')
('Mensaje plano -->', '2222222233333333')


('IV + Mensaje cifrado -->', '\xb1\x7f\xed~\xcds\xb4\x87\xd1\x18\xe1gP\x06\x1af\xd70\x84\x10\xbb\xefzf')


Envio de mensaje cifrado por canal inseguro
('Mensaje cifrado -->', '\xd1\x18\xe1gP\x06\x1af\xd70\x84\x10\xbb\xefzf')
('IV_enviado -->', '\xb1\x7f\xed~\xcds\xb4\x87')


('Mensaje descifrado -->', '2222222233333333')
```

Tu turno 2: Explicar con detalle porque para el cifrado de flujo RC4, como mostramos en el código anterior, la clave de largo plazo se combina complejamente con un Nonce a través de una función Hash ¿Porqué no se utiliza solo la clave de largo plazo? ¿Porqué no se utiliza ninguna función de Padding? ¿Qué relación tiene RC4 con OTP?

¿Porqué en el cifrado de flujo RC4 la clave de largo plazo se combina complejamente con un Nonce a través de una función Hash? ¿Porqué no se utiliza sólo la clave de largo plazo?

La operación de combinación es sencilla y como el RC4 se basa en ella se presta a manipulaciones. Si un atacante invierte un bit en el texto cifrado el receptor recogerá el bit descifrado invertido. Para evitarlo los datos deben firmarse con una función hash para que el atacante no pueda manipular el mensaje.
Pero aun así, si el atacante conoce todo el texto plano del mensaje podría cambiarlo, para evitar esto se usa el "nonce". En el texto plano se incorpora un número pseudoaleatorio, como el nonce no es conocido por el atacante éste no podrá incluirlo y no podrá cambiar el mensaje sin que se note.

¿Porqué no se utiliza ninguna función de Padding?

Las funciones de padding o esquemas de relleno son métodos que introducen información irrelevante con algún objetivo. En los sistemas que operan por bloques de tamaño fijo se usan estas funciones de padding para completar los bloques hasta que tengan el tamaño definido. RC4 no es un sistema basado en bloques, si no un sistema de cifrado de flujo, por lo tanto no tiene la necesidad de utilizar el padding.

¿Qué relación tiene RC4 con OTP?

RC4 es una técnica de cifrado de flujo, con similitudes a OTP, pero con la diferencia de que utiliza un flujo de números pseudoaleatorios como clave y que es orientado a bytes. Usan claves de longitud de mensajes que se cifran, con un cifrado incremental elemento a elemento. Nueva clave para cada sesión, no se debe reutilizar la misma clave. Utilizan la operación XOR.

Tu turno 3: Utiliza RSA para cifrar un mensaje pero invirtiendo el procedimiento: primero utiliza la clave privada y luego la clave pública. Prueba que recuperas el mensaje. ¿Explica que diferencia hay con el procedimiento anterior?

```
# Parte práctica
from Crypto.Util import number
from Crypto.PublicKey import RSA
from Crypto import Random

HEXA= lambda xx: ':'.join(hex(ord(x))[2:] for x in xx)

generador_aleatorio = Random.new().read
numero_bits_clave=1024
key = RSA.generate(numero_bits_clave, generador_aleatorio)

public_key = key.publickey()

msg='Mensaje de prueba para cifrar con la clave privada y descifrar con la pública' # mensaje a cifrar
print('Mensaje sin cifrar --> ' "%s" % msg)
print('\n')
msgc = key.decrypt(msg)
print('Mensaje cifrado --> ', msgc)
print('\n')

msg_descifrado = public_key.encrypt(msgc, '')
print('Mensaje descifrado --> ' "%s"  % msg_descifrado)
```

```
Mensaje sin cifrar --> Mensaje de prueba para cifrar con la clave privada y descifrar con la pública


('Mensaje cifrado --> ', 'Qfq*>@)G\x94c\x06\x89\xc5\xe7QXSR\xe6&l\xd5C|\x9d\xc2\x1b\xcd%:\xb5\xeb\xfaj\x00\xdeAIcO\x0
3/\xb5|\xab\xaa{\xb7\x04\x90\xea(\xa6\x16\x87\n\xb9\xf7\x88 G PsTz\xf5\xf8\xb0w\xfdGq\xe7\xbfW\xcb\xd7\xe5\xac\x11\x9
c(\xfc\x18\xe9|\x10$\xc4\x1e8\xfc&H\xcf\x1d\xa7)\x15\x92\xb4\xec\xa6\xd3\xff\x0e3\\nX\t\x91\xb8\xdd\xfd|\xf8\xb1\x0b=
1\xdf*\xe2b\xa9')


Mensaje descifrado --> Mensaje de prueba para cifrar con la clave privada y descifrar con la pública
```

¿Qué diferencia hay con el procedimiento anterior?

RSA es un sistema critográfico de clave pública válido para cifrar o firmar digitalmente. Es asimétrico, por lo que hay claves distintas para cifrar y descifrar. Los usuarios disponen entonces de clave pública y clave privada. Al enviar un mensaje el emisor cifra el mensaje con la clave pública del receptor, y luego el receptor descifra el mensaje con su clave privada.
La peculiaridad es que sus dos claves sirven tanto para cifrar como para autenticar. La conmutatividad del cifrado y descifrado en RSA, por las propiedades de la exponenciación modular hace que el cifrado y descifrado sean conmutativos y puedan hacerse con la clave pública o privada: si cifrando "M" con la clave pública "e" y luego descifrando el resultado con la clave privada "d" obtenemos de nuevo "M". Y si ciframos "M" con la clave privada "d" y desciframos el resultado con la clave pública "e" se vuelve a obtener el mismo resultado “M".

Tu turno 4: Explicar porque Alice y Bob tienen una clave secreta común, k=53, cuando no se han pasado explicitamente la clave k=53.

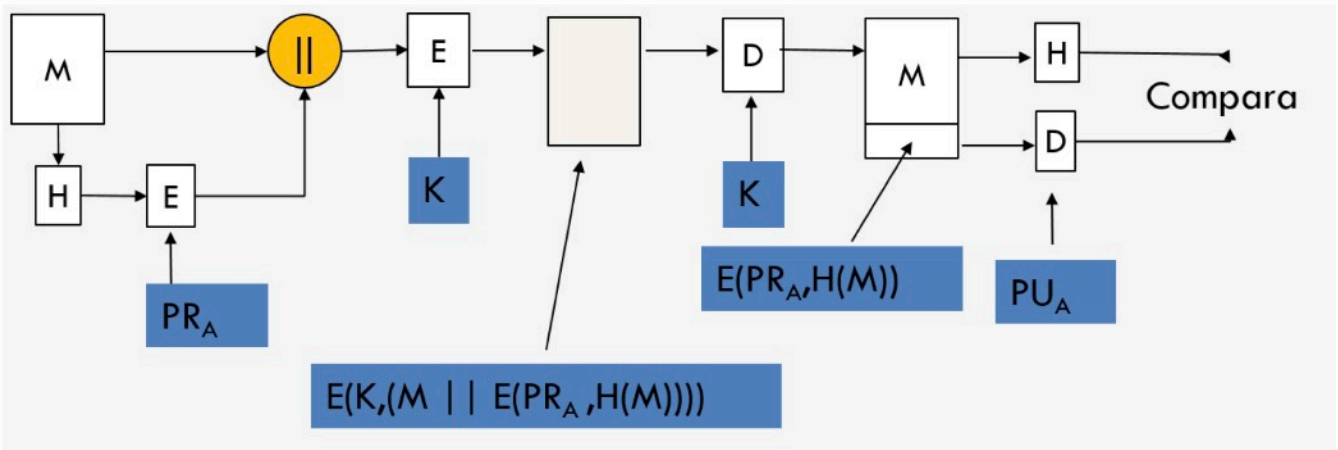
¿Porqué Alice y Bob tienen una clave secreta común?

Diffie-Hellman es un protocolo de establecimiento de claves entre partes que no han tenido contacto previo, usando un canal inseguro y de manera anónima (no autentificada). Se emplea como medio para acordar claves simétricas empleadas para el cifrado de una sesión.
Los pasos del proceso:

1. Alice escoge un número primo p muy grande, y un número aleatorio k menor a p. Estos números se envían a Bob de forma pública.
2. Alice escoge un número cualquiera x menor a p, y lo guarda en secreto. Luego obtiene A, de la forma: A=k^x(mod p). Alice manda a Bob el número A.
3. Bob elige un número secreto y menor que p, y realiza la operación para conseguir B, de la forma: B=k^y(mod p). Luego envía a Alice el número B.
4. Tanto Alice como Bob calculan a partir de A y de B las claves. Ambos llegan al mismo resultado. Clave de Alice = B^x(mod p). Clave de Bob = A^y(mod p).

Alguien que pudiera estar en medio, al ser números tan grandes debería resolver una ecuación enorme para obtener los valores 'x' o 'y' secretos de Alice y Bob. Ahí radica su seguridad.
Al final llegan al mismo resultado, a la misma clave, porque ambos están calculando realmente lo mismo: Clave secreta Alice = Bx%p = kyx%p = k(yx)%p Clave secreta Bob = Ay%p = kxy%p = k(xy) %p

Tu turno 5: Implementa el esquema Firma Digital y Confidencialidad.

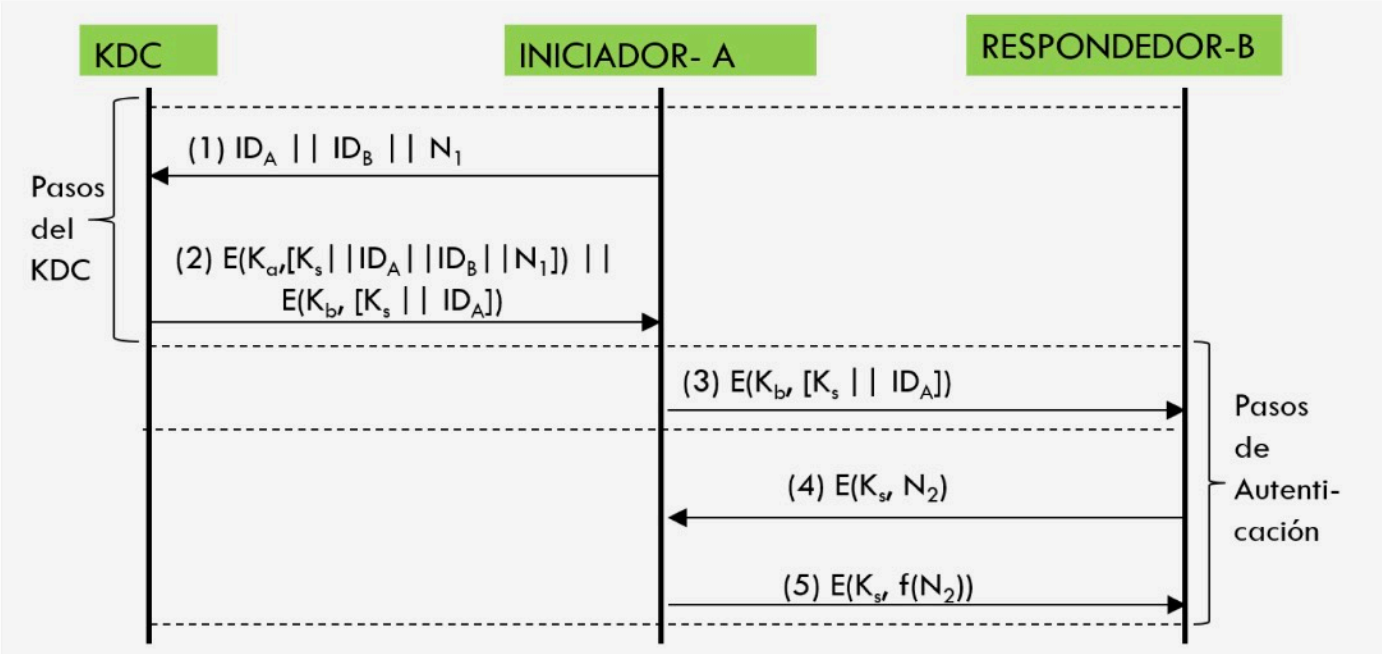


```python
# Implementa el esquema firma digital y confidencialidad
from Crypto.Hash import SHA256
from Crypto.Hash import SHA512
from Crypto.PublicKey import RSA
from Crypto import Random
from Crypto.Cipher import DES

# Definición de la función lambda que comvierte una cadena a hexadecimal.
HEXA = lambda xx: ':'.join(hex(ord(x))[2:] for x in xx)

msg512 = 'Mensaje para el hash 512'
hash512 = SHA512.new(msg512).hexdigest()
print('Mensaje original --> ' "%s" % msg512)
print('hex512 --> ' "%s"  % hash512)
print('\n')

# firma del mensaje
generador = Random.new().read
numero_bits=1024
key = RSA.generate(numero_bits, generador)
p_key = key.publickey()
firma_512 = key.decrypt(hash512)
print('Firma --> ', firma_512)
print('\n')

# sumar mensaje y firma y cifrarlo/descifrarlo con cifrado simétrico
msg512firma = (msg512 + firma_512)
print('Mensaje + Firma --> ',msg512firma)
print('\n')
```

```python
key = b'11111111'
iv = Random.new().read(DES.block_size)
cipher = DES.new(key, DES.MODE_CFB, iv)
msgc = iv + cipher.encrypt(msg512firma)
iv_enviado = msgc[:DES.block_size]
cipher2 = DES.new(key, DES.MODE_CFB, iv_enviado)
solo_mensaje = msgc[DES.block_size:]
print('Mensaje cifrado -->', solo_mensaje)
msgd = cipher2.decrypt(solo_mensaje)
print('\n')
print('Mensaje descifrado -->', msgd)

# separamos mensaje y firma
msg512_new = msgd[:len(msg512)]
print('Mensaje --> ' "%s" % msg512_new)
firma512_new = msgd[len(msg512):]
print('Firma --> ', firma512_new)
```

```
Mensaje original --> Mensaje para el hash 512
hex512 --> 21adf4c0209bf34670c6105746f3d9010954cfcd9fbee048b1b073f4fac205252da46e7248036e52fbe9eba29f0ce040f4256725c5
55f9fc0854e6f6ce7ee488


('Firma --> ', '?\xe3Ig\x04\xbc\x88\xa7\xcd\x98\xa1\x8aY\xe5\xc1\x7f\x17{\xb6\xc0\xafuz\x16Ni]\x0b\xe6\xba]\xa9\xaf$_
Cf\xa5sW\x9aA\xf4\xd2\xd6|\x9a\x13\xbcG\x9dl\x83!QT.\xe79\x02O\x8f\xc88\x1fo)f\x85\xd9xE+0\xfa^\xc5\xa6\xe7D\xd3\n\x1
1\xff\xb1\xbd\xbd\xe6\xad\x86\x86\x8b/A\x04\x93\xaf<\xb5XY\xf7\x86\x1c.\x04\xedKuo\xf5\xb1\xef\x9fH\xdb\x8c:H\xde\xc9
\xe6(\x93\xba\xf1\x01{')


('Mensaje + Firma --> ', 'Mensaje para el hash 512?\xe3Ig\x04\xbc\x88\xa7\xcd\x98\xa1\x8aY\xe5\xc1\x7f\x17{\xb6\xc0\x
afuz\x16Ni]\x0b\xe6\xba]\xa9\xaf$_Cf\xa5sW\x9aA\xf4\xd2\xd6|\x9a\x13\xbcG\x9dl\x83!QT.\xe79\x02O\x8f\xc88\x1fo)f\x85\
xd9xE+0\xfa^\xc5\xa6\xe7D\xd3\n\x11\xff\xb1\xbd\xbd\xe6\xad\x86\x86\x8b/A\x04\x93\xaf<\xb5XY\xf7\x86\x1c.\x04\xedKuo\
xf5\xb1\xef\x9fH\xdb\x8c:H\xde\xc9\xe6(\x93\xba\xf1\x01{')


('Mensaje cifrado -->', "\x06\x95\xd0\x8bkm\xab\xd0+\x80\x95J!\x92\x99\xf2u\xfc\xe5km\x92W=\x18\xfd\xe4H\x1f+\xe6\xce
\xd0\xba\xd0W\xf39D\xd7\xcb\x88P1\xee\t\xc2\xdd\x8b\x86\xe8\r\xdd<\xed\x98]\x99I\rpf\xcc\x12'b\xf9\xc3\xe2PS'\xebzq\x
11]\xc0\x00Yp]\xec\xf1\xef\xf1V\x81\xf3&\x05l'\xc4\xff\xba\x82n\xc8\xe7\xbe\x8d\xf0\xd0\x9cFz\x07\x1d\x10\xa0\x85S\x1
3\xaa\xefGQ\xc1\xec\x039\xe0s\xb2\x8dR\n\x82\x9fm\xb6\nrJ\xc1\xd7\x9c\xa4\x04O\xbaV\xad^a\x1e\x8cZ\xd3\xfcL")


('Mensaje descifrado -->', 'Mensaje para el hash 512?\xe3Ig\x04\xbc\x88\xa7\xcd\x98\xa1\x8aY\xe5\xc1\x7f\x17{\xb6\xc0
\xafuz\x16Ni]\x0b\xe6\xba]\xa9\xaf$_Cf\xa5sW\x9aA\xf4\xd2\xd6|\x9a\x13\xbcG\x9dl\x83!QT.\xe79\x02O\x8f\xc88\x1fo)f\x8
5\xd9xE+0\xfa^\xc5\xa6\xe7D\xd3\n\x11\xff\xb1\xbd\xbd\xe6\xad\x86\x86\x8b/A\x04\x93\xaf<\xb5XY\xf7\x86\x1c.\x04\xedKu
o\xf5\xb1\xef\x9fH\xdb\x8c:H\xde\xc9\xe6(\x93\xba\xf1\x01{')
Mensaje --> Mensaje para el hash 512
('Firma --> ', '?\xe3Ig\x04\xbc\x88\xa7\xcd\x98\xa1\x8aY\xe5\xc1\x7f\x17{\xb6\xc0\xafuz\x16Ni]\x0b\xe6\xba]\xa9\xaf$_
Cf\xa5sW\x9aA\xf4\xd2\xd6|\x9a\x13\xbcG\x9dl\x83!QT.\xe79\x02O\x8f\xc88\x1fo)f\x85\xd9xE+0\xfa^\xc5\xa6\xe7D\xd3\n\x1
1\xff\xb1\xbd\xbd\xe6\xad\x86\x86\x8b/A\x04\x93\xaf<\xb5XY\xf7\x86\x1c.\x04\xedKuo\xf5\xb1\xef\x9fH\xdb\x8c:H\xde\xc9
\xe6(\x93\xba\xf1\x01{')
```

Tu turno 6: Implementa este esquema de distribución de claves.



```python
# Implementa el esquema anterior de distribución de claves
from Crypto.Cipher import AES
from Crypto import Random
from Crypto.Hash import SHA256
# Definición de la función lambda que comvierte una cadena a hexadecimal.
HEXA= lambda xx: ':'.join(hex(ord(x))[2:] for x in xx)

# Paso 1: Mensaje inicial con la identidad de A, la identidad de B, y el id único N1 que sería el nonce
# que sería un número aleatorio.
n1 = Random.new().read(16) # Nonce de 16 bytes
identidad_A = 'id_Aaaaa'
identidad_B = 'id_Bbbbb'
msg_inicial = identidad_A + identidad_B + n1
print('Mensaje inicial -->', msg_inicial)


# Paso 2: A debe verificar que la solicitud original no ha sido alterada, a través del nonce.
# KDC envía mensaje en 2 partes a A, y luego A debe descifrar y comprobar. Se debe usar un algoritmo simétrico.
key = b'11111111'
keyS = b'22222222'
keyB = b'33333333'
cifA = DES.new(key, DES.MODE_ECB)
msg_KDC_1 = cifA.encrypt(keyS + msg_inicial)
cifB = DES.new(keyB, DES.MODE_ECB)
msg_KDC_2 = cifB.encrypt(keyS + identidad_A)
msg_KDC_Final = msg_KDC_1 + msg_KDC_2
print('Mensaje de KDC', msg_KDC_Final)
print('\n')
msg_KDC_A = cifA.decrypt(msg_KDC_Final)
print('A descrifra el mensaje de KDC', msg_KDC_A)


# Paso 3: A envía a B la segunda parte del mensaje original. B lo descifra.
msg_A_B = cifB.decrypt(msg_KDC_2)
print('Mensaje que envia A a B', msg_A_B)
print('\n')


# Paso 4: B envía a A el mensaje, con el nonce n2.
n2 = Random.new().read(16) # Nonce de 16 bytes
cifS = DES.new(keyS, DES.MODE_ECB)
msg_B_A = cifS.encrypt(n2)
print('Mensaje que envia B a A', msg_B_A)
print('\n')


# Paso 5: A le aplica alguna función a n2 y así B podrá saber que no ha sido corrompido.
# A aplica función al nonce2
msg_B_A_descif = cifS.decrypt(msg_B_A)
print('Mensaje de B descifrado por A', msg_B_A_descif)
print('\n')

d256 = SHA256.new(msg_B_A_descif).hexdigest()
msg_A_B_func = cifS.encrypt(d256)
print('Mensaje de A a B tras aplicar funcion hash', msg_A_B_func)
print('\n')

# B descifra y comprueba el nonce2
msg_A_B_func_descif = cifS.decrypt(msg_A_B_func)
print('Mensaje enviado por A tras descifrarlo B', msg_A_B_func_descif)
print('\n')
d256_n2 = SHA256.new(n2).hexdigest()
print('B comprueba que obtenga lo mismo que aplicando al nonce 2 una funcion hash', d256_n2)
```

```
('Mensaje inicial -->', 'id_Aaaaaid_Bbbbb\xff \x94\xee\xe6\xf3\x9a\x86\xb1\x1c\x9f\xa6j\x93\r\xb3')
('Mensaje de KDC', "\xf9'\x14\xe8\x0e\x83.\xdc\x95j7Bw\nh\xa6#)\x84\x98F\xeb\xf2\x98\xf0\x8c\xd5\xb8\x832\xa7\x8eHk\x
d4\xda,\xb1\xcb\xfe\xac\x97\x8c$xc8\x8f^Wy<<SJ\xe0")


('A descrifra el mensaje de KDC', '22222222id_Aaaaaid_Bbbbb\xff \x94\xee\xe6\xf3\x9a\x86\xb1\x1c\x9f\xa6j\x93\r\xb3\x
e9\xbe\xfc\xab\xd4\xe4\xc0\xcdei&\xbc\xa6P\xd42')
('Mensaje que envia A a B', '22222222id_Aaaaa')


('Mensaje que envia B a A', '>\xac\x9d\xb5\xb2\xc0J\xcdE\xc2\x08\xdc\xe7q\xdc\x7f')


('Mensaje de B descifrado por A', '\xd4\xc2\\\x96\xaf\xcc\xa4\xef3a\x93\xe3\xdaD\xadD')


('Mensaje de A a B tras aplicar funcion hash', '^\xac\x8f\xdb\xf2\x8e\xf2\xaf\xbe\x94\x8b\xc0~\xe4RD%\x13W\x94\xcf\x9
0\xcb2\xc1\xcf\x1e\x12\xf1\x9c\x14\xa4l>p\xc9*\xd12[f\x9e\xc4\x8e\xae\x02\x18H\xb7\x87.n\xbd\xf3@&\xafx\x1a\xfc-\xba\
xaa\x84')


('Mensaje enviado por A tras descifrarlo B', '386fe06e761654934d55e14b8da8c7df5666204718dcf99d0a736d615c0787d2')


('B comprueba que obtenga lo mismo que aplicando al nonce 2 una funcion hash', '386fe06e761654934d55e14b8da8c7df56662
04718dcf99d0a736d615c0787d2')
```