



Instituto Tecnológico y de Estudios Superiores de
Monterrey

Actividad 2.2. Diffie-Hellmann, DES y AES

Juan Pablo Echeagaray González

A00830646

Análisis de Criptografía y Seguridad

MA2002B.300

Dr.-Ing. Jonathan Montalvo-Urquizo

6 de junio del 2022

1. Diffie-Hellmann

Ejercicio 1

Para el primer ejercicio se utilizan los valores:

$$P = 293$$

$$G = 105$$

$$a = 23$$

$$b = 7$$

Donde a es el número privado de Alice, b es el número privado de , G y P son de carácter público, ambos son privados, pero G debe de ser menor que P y una raíz primitiva de P .

El proceso de intercambio de llaves comienza con Alice y Bob realizando las operaciones:

$$\text{Alice : } x = G^a \quad \text{mód } P = 105^{23} \quad \text{mód } 293 = 251$$

$$\text{Bob : } y = G^b \quad \text{mód } P = 105^7 \quad \text{mód } 293 = 32$$

Alice y Bob intercambian sus llaves y realizan las operaciones:

$$\text{Alice : } K_A = y^a \quad \text{mód } P = 32^{23} \quad \text{mód } 293 = 11$$

$$\text{Bob : } K_B = x^b \quad \text{mód } P = 251^7 \quad \text{mód } 293 = 11$$

Dado que $K_A = K_B$, tanto Alice como Bob pueden estar seguros de que han establecido una comunicación segura entre ellos, ahora pueden usar esta llave para otros procesos.

Ejercicio 2

En esta sección se ejecutarán 3 pruebas diferentes con números distintos en cada iteración. El valor de G será determinado con un calculador de raíces primitivas implementado en WolframAlpha [EdgeOfInfinity, 2017].

Ejercicio 2.1

Se inicializa el intercambio de llaves con los siguientes valores:

$$P = 661$$

$$G = 35$$

$$a = 311$$

$$b = 211$$

Se realizan las operaciones:

$$\text{Alice : } x = G^a \quad \text{mód } P = 35^{311} \quad \text{mód } 661 = 260$$

$$\text{Bob : } y = G^b \quad \text{mód } P = 35^{211} \quad \text{mód } 661 = 530$$

Se envían estos números a través de la red y se realizan las operaciones:

$$\begin{aligned}\text{Alice : } K_A &= y^a \pmod{P = 530^{311}} \pmod{293} = 307 \\ \text{Bob : } K_B &= x^b \pmod{P = 260^{211}} \pmod{293} = 307\end{aligned}$$

Las llaves son iguales, se puede establecer un canal seguro de comunicación.

Ejercicio 2.2

Se inicializa el intercambio de llaves con los siguientes valores:

$$\begin{aligned}P &= 569 \\ G &= 547 \\ a &= 197 \\ b &= 103\end{aligned}$$

Se realizan las operaciones:

$$\begin{aligned}\text{Alice : } x &= G^a \pmod{P = 547^{197}} \pmod{569} = 147 \\ \text{Bob : } y &= G^b \pmod{P = 547^{103}} \pmod{569} = 356\end{aligned}$$

Se envían estos números a través de la red y se realizan las operaciones:

$$\begin{aligned}\text{Alice : } K_A &= y^a \pmod{P = 356^{197}} \pmod{569} = 24 \\ \text{Bob : } K_B &= x^b \pmod{P = 147^{103}} \pmod{569} = 24\end{aligned}$$

Las llaves son iguales, se puede establecer un canal seguro de comunicación.

Ejercicio 2.3

Se inicializa el intercambio de llaves con los siguientes valores:

$$\begin{aligned}P &= 857 \\ G &= 113 \\ a &= 373 \\ b &= 503\end{aligned}$$

Se realizan las operaciones:

$$\begin{aligned}\text{Alice : } x &= G^a \pmod{P = 113^{373}} \pmod{857} = 561 \\ \text{Bob : } y &= G^b \pmod{P = 113^{503}} \pmod{857} = 159\end{aligned}$$

Se envían estos números a través de la red y se realizan las operaciones:

$$\begin{aligned}\text{Alice : } K_A &= y^a \mod P = 159^{373} \mod 857 = 342 \\ \text{Bob : } K_B &= x^b \mod P = 561^{503} \mod 857 = 342\end{aligned}$$

Las llaves son iguales, se puede establecer un canal seguro de comunicación.

Esta implementación de Diffie-Hellmann es vulnerable a un ciberataque del tipo *Man In The Middle*. Un ataque de esta naturaleza consistiría en que al comienzo de la comunicación, un tercero intercepte el mensaje de Alice, y realice la operación que Bob realizaría con el mensaje, se realizaría lo mismo con el mensaje de Bob. Efectivamente

2. DES

Detalles de la implementación modificada y los comentarios agregados pueden ser vistos en la sección A.

Sin duda DES representó un gran avance en el área de la criptografía cuando fue implementado a finales de los 70s, su diseño es bastante elegante y sencillo de explicar incluso a personas sin experiencia en criptografía; sin embargo, su mayor debilidad es la longitud de su clave. En su momento una clave de 56 bits fue lo suficientemente fuerte como para no ser rota por fuerza bruta [Pub, 1999], sin embargo, ya no nos encontramos en ese entorno.

3. AES

Detalles de la implementación modificada y los comentarios agregados pueden ser vistos en la sección A.

El incremento en el nivel de seguridad que brinda AES sobre los estándares anteriores (DES, 3DES) es masivo, ofrece un nivel mayor de seguridad y es más rápido de ejecutar. El modelo matemático es más complicado de entender a comparación de DES, pero es más eficiente de computar [Stallings, 2002].

A. Código desarrollado

diffie-hellmann

June 5, 2022

1 Actividad 2.2. Diffie-Hellmann, DES y AES

- Juan Pablo Echeagaray González
- A00830646
- Análisis de Criptografía y Seguridad
- MA2002B.300
- Profesores:
 - Dr. Alberto F. Martínez
 - Dr.-Ing. Jonathan Montalvo-Urquizo
- 6 de junio del 2022

1.1 Diffie-Hellmann

```
[ ]: def diffie_hellmann(P: int, G: int, a: int, b: int):  
    """  
    Original Author: J. Montalvo-Urquizo (2021-02-10)  
    Modified by: Juan Pablo Echeagaray González (2022-06-04)  
  
    Python: 3.8.10  
  
    Diffie-Hellmann Implementation (In its simple form)  
  
    Args:  
        P (int): Prime number  
        G (int): Primitive root of P  
        a (int): Alice's secret key (an integer)  
        b (int): Bob's secret key (an integer)  
  
    """  
    # Alice and Bob agree on sharing this numbers as public keys:  
    # A prime number P and a primitive root G  
  
    # A primitve root for P, G is taken  
    # Used https://www.wolframalpha.com/widgets/view.jsp?id=ef51422db7db201ebc03c8800f41ba99 para encontrar el primitive root  
  
    print(f'The Value of the selected prime P is: {P}')  
    print(f'The Value of the primitive root G is: {G}')
```

```

    # Alice chooses her private key as
    print(f'The Private Key for Alice is: {a}')

    # Alice gets the generated key as  $G^a \bmod P$ 
    x = int(pow(G, a, P))

    print(f'Alice computes her generated key and sends the result as {x}')

    # Bob chooses his private key as
    print(f'The Private Key for Bob is: {b}')

    # Bob gets the generated key as  $G^b \bmod P$ 
    y = int(pow(G, b, P))
    print(f'Bob computes his generated key and sends the result as {y}')

    # Bob shares his result with Alice, and Alice computes the Shared
    ↪ Secret Key
    ka = int(pow(y, a, P))

    # Alice shares her result with Bob, and Bob computes the Shared Secret
    ↪ Key
    kb = int(pow(x, b, P))

    print(f'Secret key for Alice is: {ka}')
    print(f'Secret Key for Bob is: {kb}')

    if ka == kb:
        print(f'As Bob and Alice are getting the same result {ka}, they can
        ↪ use this value as a Secret Shared Key to exchange information')
    else:
        print('something went wrong!, check your calculations')

```

1.1.1 Ejercicio 1

```

[ ]: P = 293
    G = 105
    a = 23
    b = 7
    print('Ejercicio 1')
    diffie_hellmann(P, G, a, b)

```

Ejercicio 1

The Value of the selected prime P is: 293

The Value of the primitive root G is: 105

The Private Key for Alice is: 23

Alice computes her generated key and sends the result as 251
The Private Key for Bob is: 7
Bob computes his generated key and sends the result as 32
Secret key for Alice is: 11
Secret Key for Bob is: 11
As Bob and Alice are getting the same result 11, they can use this value as a Secret Shared Key to exchange information

1.1.2 Ejercicio 2

Ejercicio 2.1

```
[ ]: print('Ejercicio 2.1')
P = 661
G = 35
a = 311
b = 211
diffie_hellmann(P, G, a, b)
```

Ejercicio 2.1

The Value of the selected prime P is: 661
The Value of the primitive root G is: 35
The Private Key for Alice is: 311
Alice computes her generated key and sends the result as 260
The Private Key for Bob is: 211
Bob computes his generated key and sends the result as 530
Secret key for Alice is: 307
Secret Key for Bob is: 307
As Bob and Alice are getting the same result 307, they can use this value as a Secret Shared Key to exchange information

Ejercicio 2.2

```
[ ]: print('Ejercicio 2.2')
P = 569
G = 547
a = 197
b = 103
diffie_hellmann(P, G, a, b)
```

The Value of the selected prime P is: 569
The Value of the primitive root G is: 547
The Private Key for Alice is: 197
Alice computes her generated key and sends the result as 147
The Private Key for Bob is: 103
Bob computes his generated key and sends the result as 356
Secret key for Alice is: 24
Secret Key for Bob is: 24
As Bob and Alice are getting the same result 24, they can use this value as a Secret Shared Key to exchange information

Ejercicio 2.3

```
[ ]: print('Ejercicio 2.3')
p = 857
G = 113
a = 373
b = 503
diffie_hellmann(p, G, a, b)
```

Ejercicio 2.3

The Value of the selected prime P is: 857

The Value of the primitive root G is: 113

The Private Key for Alice is: 373

Alice computes her generated key and sends the result as 561

The Private Key for Bob is: 503

Bob computes his generated key and sends the result as 159

Secret key for Alice is: 342

Secret Key for Bob is: 342

As Bob and Alice are getting the same result 342, they can use this value as a Secret Shared Key to exchange information

1.2 DES

```
[ ]: def DES(message: str, key: str):
    """
    Original Author: J. Montalvo-Urquiza (2021-02-10)
    Modified by: Juan Pablo Echeagaray González (2022-06-04)

    pyDes: 2.0.1
    Python: 3.8.10

    DES Implementation

    Args:
        message (str): Message to be sent
        key (str): Key used by the DES encryption

    El flujo de la función es:
    1. Se interpreta un mensaje en string normal a su formato bytes, se usa el
    ↪ codificado UTF-8
    2. Se realiza el paso anterior con la llave proporcionada, esta debe de
    ↪ tener una longitud de 8 caracteres
    3. Se crea un objeto de tipo pyDes.des, se le pasa la llave y se usa el
    ↪ modo de operación CBC, como IV usamos un vector de 0s
    4. Se encripta el mensaje con el método .encrypt(), los parámetros
    ↪ necesarios se proporcionaron cuando se instanció el objeto pydes.des
    5. Se imprime el mensaje encriptado
    6. Se crea otro objeto pyDes.des con los mismos parámetros que el original
```


7. Se descripta el mensaje aplicando el método `.decrypt()` al objeto
→ creado en el paso anterior

8. Se imprime el mensaje descriptado y se comprueba que sean iguales
"""

```
import pyDes

# Punto 1
# Inicialización del mensaje
data = bytes(message, 'utf-8')
key = bytes(key, 'utf-8')
print("\n\nOriginal Data: %r" % data)

# Punto 2
# Instancia del objeto pyDes.des en modo CBC con la llave proporcionada, un
→ IV de 0s, y un padding de PKCS5
kAlice = pyDes.des(key, pyDes.CBC, "\0\0\0\0\0\0\0\0", pad=None,
→ padmode=pyDes.PAD_PKCS5)
print("Alice's key is: %r" % kAlice.getKey())

# Punto 3
# Aplicación de DES para encriptar
dataTransferred = kAlice.encrypt(data)
print("Alice's Encrypted Message: %r\n" % dataTransferred)

# Punto 4
# Mismo proceso que en el punto 2, se simula otro usuario
kBob = pyDes.des(key, pyDes.CBC, "\0\0\0\0\0\0\0\0", pad=None,
→ padmode=pyDes.PAD_PKCS5)
print("Bob's key is: %r" % kBob.getKey())

# Punto 5
# Descriptado con la llave del segundo usuario, arroja un error en caso
→ de que el mensaje sea diferente al original
decrypted = kBob.decrypt(dataTransferred)
print("Bob's decrypted message is: %r" % decrypted)

assert decrypted == data

DES("This is a small test for the DES Algorithm Implementation to be used at
→ MA2002B", "MONTALVO")
```

Original Data: b'This is a small test for the DES Algorithm Implementation to be used at MA2002B'

Alice's key is: b'MONTALVO'

Alice's Encrypted Message: b"\xba\xeJX7\\\xf8\x02`\xddI\x1907\xa9(\xb5\xa9\x1b\xfd*\x17't{\x86\x0b~\x08;ti\x93\x88\xe2\xa2\xa3\x06\x82\xf7\xeb\xcdg\xfa(\xbc\x05>g\x1e\xe7\x02P\xec\xce\x0\xb0\x9e\xf7\n\xa8`\xb5\x98\xe2\xb4\xcd[SP,f\xcc\xa8M\xaa\x1b\xfa1"

Bob's key is: b'MONTALVO'

Bob's decrypted message is: b'This is a small test for the DES Algorithm Implementation to be used at MA2002B'

1.3 AES

```
[ ]: def AES_encryption(message: str, key: str):

    data = message.encode('ascii')
    key = key.encode('ascii')

    from Cryptodome.Cipher import AES

    print(f'key: {str(key)}')
    cipher = AES.new(key, AES.MODE_EAX)
    ciphertext, tag = cipher.encrypt_and_digest(data)
    print(f'ciphertext: {str(ciphertext)}')

    file_out = open("encrypted.bin", "wb")
    [file_out.write(x) for x in (cipher.nonce, tag, ciphertext)]
    file_out.close()

    with open('message.txt', 'r') as file:
        this_message = file.read()

    this_key = 'You look lonely '

    AES_encryption(this_message, this_key)
```

key: b'You look lonely '

ciphertext: b'}\xe2\x878+s\x13\$\xb5\xed+#\xb2\xa9\xda%\xa0\x88\xb1\x1e%f\x9c:\x82\x9f\xfe\x1f\xd71\xe0Q\xe0,\xa8_\xb9\x81\xd0\x14\x8f\xc7\x960\xd0\xd6o\x00\xd6\xdc\x810u\xd1M\xfa-\xa3}\x8d\xfa9hdx\xabd\xc9F,\xfdxdb*\xc5\x1fm,R\x84\xbc\xaf\xcc5\xce1\xe9a\x11\x86!\n!(bGj\xa7v\xd1\x07\xd5\xa7\x8e\x00H\xd1V\x0c34\$\xf7\xca1\xa9\x9e\xa2\xe4i\x07\xcb\x8f\x16\x1fu\xfa"\x0f%\x94\xa8)\xf9\xb1@\x00\x1c\x19\xa2\x16\xbb\x136\xe8\x16\x1c\xf1\xda\xf7\xec\xce\xd2,\x14\xe1\xbe-at1\x02\xac\xcd/S\x0e\xei1\xbbGA\xb3\xb8v\xc4g*x\xa8m\xfa0GfN\x96\xca1\xfa(\xfcc<<\xdf\xd9\x14D\x86]8G\x84V3\x8e\x1f!z{\xf7\xd9\x97\x06v~\x1fNk\xeb\''\x08\xc5z\x04\xbb?Y\x89\xe2J\x02>8B\x15\xa9\xe7\x1f\xe5Y\xad\x17A\xde\x84\xba\xa5!Mp\xd1\xde\xfe\x9a\xa6!\xed\\g\x93\xe3\x90\x89d\x16~\xd4Y\xae\xc1Z5\x9e\xcbJ\xfa3\x17_\x0c\x03\x1fS\xd1\x82\x92\x8f<il\xca\xda&\xab\x0cu\x1f\x8d!0\xa1\xad\xeb&%\xc8\x0f\x07\x1fX\xb4\x1dQ\x05\x01\x9a6\x81\x86\xe9\xde\xdb\x10\xff\xa4\xd4PW"\xb1\x17X?\xb7\x82\xca0\n\xe2\xdcZrQ\x08/S\xfc\xe9\x07\xaf\xfe\xe2\xd94\xfa8\xcd\xcf7\xd5\x02b\x1d\xa5|\xff\x

```
df\xa4_\n\x9c|'
```

```
[ ]: def AES_decryption(key: str):  
    from Cryptodome.Cipher import AES  
  
    key = bytes(key, 'ascii')  
    file_in = open("encrypted.bin", "rb")  
    nonce, tag, ciphertext = [file_in.read(x) for x in (16, 16, -1)]  
  
    print('key:\n' + str(key))  
    cipher = AES.new(key, AES.MODE_EAX, nonce)  
    data = cipher.decrypt_and_verify(ciphertext, tag)  
    print(data)  
  
this_key = 'You look lonely '  
AES_decryption(this_key)
```

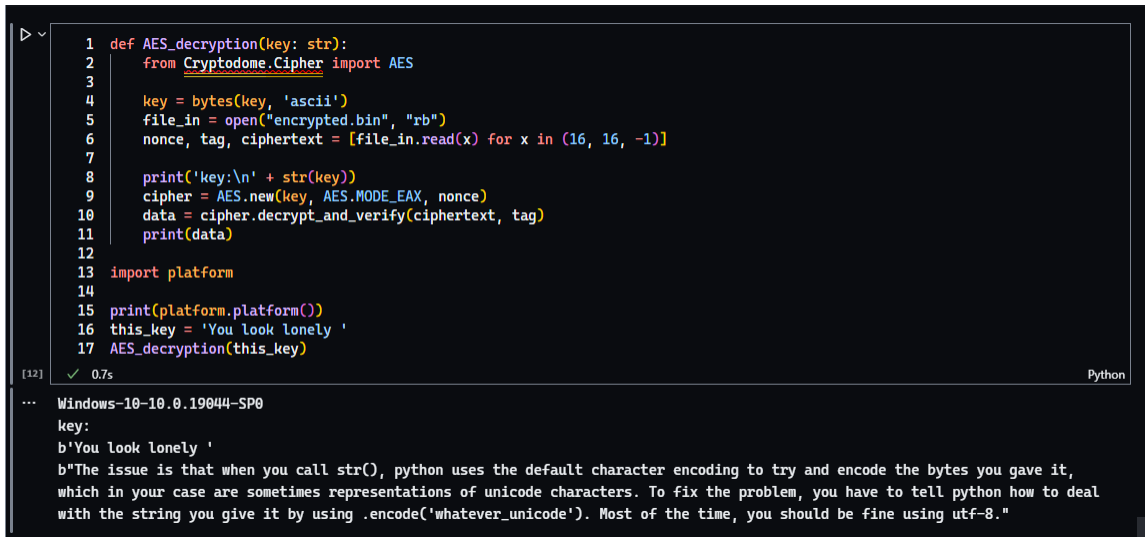
key:

b'You look lonely '

b"The issue is that when you call str(), python uses the default character encoding to try and encode the bytes you gave it, which in your case are sometimes representations of unicode characters. To fix the problem, you have to tell python how to deal with the string you give it by using .encode('whatever_unicode'). Most of the time, you should be fine using utf-8."

B. Pruebas de AES en equipos distintos

B.1. En Windows



```
1 def AES_decryption(key: str):
2     from Cryptodome.Cipher import AES
3
4     key = bytes(key, 'ascii')
5     file_in = open("encrypted.bin", "rb")
6     nonce, tag, ciphertext = [file_in.read(x) for x in (16, 16, -1)]
7
8     print('key:\n' + str(key))
9     cipher = AES.new(key, AES.MODE_EAX, nonce)
10    data = cipher.decrypt_and_verify(ciphertext, tag)
11    print(data)
12
13    import platform
14
15    print(platform.platform())
16    this_key = 'You look lonely '
17    AES_decryption(this_key)
```

[12] ✓ 0.7s Python

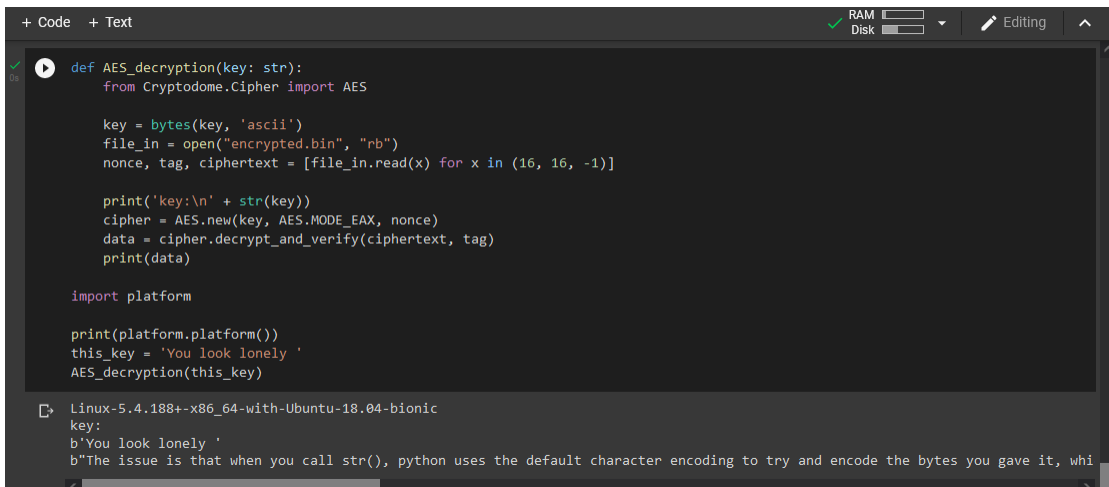
Windows-10-10.0.19044-SP0

key:
b'You look lonely '

b"The issue is that when you call str(), python uses the default character encoding to try and encode the bytes you gave it, which in your case are sometimes representations of unicode characters. To fix the problem, you have to tell python how to deal with the string you give it by using .encode('whatever_unicode'). Most of the time, you should be fine using utf-8."

Figura 1: Evidencia de funcionamiento en máquina con Windows

B.2. En Google Colab



```
+ Code + Text RAM Disk Editing
```

```
def AES_decryption(key: str):
    from Cryptodome.Cipher import AES

    key = bytes(key, 'ascii')
    file_in = open("encrypted.bin", "rb")
    nonce, tag, ciphertext = [file_in.read(x) for x in (16, 16, -1)]

    print('key:\n' + str(key))
    cipher = AES.new(key, AES.MODE_EAX, nonce)
    data = cipher.decrypt_and_verify(ciphertext, tag)
    print(data)

import platform

print(platform.platform())
this_key = 'You look lonely '
AES_decryption(this_key)
```

Linux-5.4.188+-x86_64-with-Ubuntu-18.04-bionic

key:
b'You look lonely '

b"The issue is that when you call str(), python uses the default character encoding to try and encode the bytes you gave it, whi

Figura 2: Evidencia de funcionamiento en máquina con máquina virtual en Google Colab

Referencias

- [EdgeOfInfinity, 2017] EdgeOfInfinity (2017). Wolfram—Alpha Widgets: "Primitive Roots-Free Web & Computer Systems Widget.
- [Pub, 1999] Pub, F. (1999). Data encryption standard (des). *FIPS PUB*, pages 46–3.
- [Stallings, 2002] Stallings, W. (2002). The advanced encryption standard. *Cryptologia*, 26(3):165–188.