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

[ ]:
```python
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 primitive 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 mod 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 mod 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

[ ]:
```python
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

[ ]:
```python
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-Urquizo (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 desencripta el mensaje aplicando el método .decrypt() al objeto␣
 ↪creado en el paso anterior
    8. Se imprime el mensaje desencriptado 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
    dataTransfered = kAlice.encrypt(data)
    print("Alice's Encrypted Message: %r\n" % dataTransfered)

    # 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
    # Desencriptado con la llave del segundo usuario, arroja un error en caso␣
 ↪de que el mensaje sea diferente al original
    decrypted = kBob.decrypt(dataTransfered)
    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\x8eJX7\\\x7f\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\xf6\x8a
(\xbc\x05>g\x1e\xe7\x02P\xec\xce\xe0\xb0\x9e\xf7\n\xa8`\xb5\x98\xe2\xb4\xcd[SP,f
\xcc\xa8M\xaa\x1b\xf1"

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

[ ]:
```python
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$\xb5\xed+#\xb2\xa9\xda%\xa0\x88\xb1\x1e%f\x9c:\x8
2\x9f\xfe\x1f\xd71\xe0Q\xe0,\x8a_\xb9\x81\xd0\x14\x8f\xc7\x96O\xd0\xd6o\x00\xd6\
xdc\x81Ou\xd1M\xfa-\xa3}\x8d\xf9hdx\xabd\xc9F,\xfd\xdb*\xc5\x1fm,R\x84\xbc\xaf\x
c5\xce1\xe9a\x11\x86!\n!(bGj\xa7v\xd1\x07\xd5\xa7\x8e\x00H\xd1V\x0c34$\x7f\xc1\x
a9\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-atl\x02\xac\xcd/S
\x0e\xe1i\xbbGA\xb3\xb8v\xc4g*x\xa8m\xf0GfN\x96\xc1\xfa(\xfc<<\xdf\xd9\x14D\x86]
8G\x84V3\x8e\x1f!z{\xf7\xd9\x97\x06v~\x1fNk\xeb\'\x08\xc5z\x04\xbb?Y\x89\xe2J\x0
2>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\xf3\x17_\x0c\x03\x1fS\xd1\x82\x9
2\x8f<il\xc6l\xca\xda&\xab\x0cu\x1f\x8d!0\xa1\xad\xeb&%\xc8\x0f\x07\x1fX\xb4\x1d
Q\x05\x01\x9a6\x81\x86\xe9\xde\xdb\x10\xff\xa4\xd4PW"\xb1\x17X?\xb7\x82\xc0\n\xe
2\xdcZrQ\x08/S\xfc\xe9\x07\xaf\xfe\xe2\xd94\xf8\xcd\xcf7\xd5\x02b\x1d\xa5|\xff\x
```

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

[ ]:
```python
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."
```