Acuerdo de claves Diffie-Hellman

Recuerda, el protocolo es:

1. Acuerdan g y p primos entre sí
2. Escogen números en secreto a y b
3. Se envian entre ellos:
    - $Alice \rightarrow Bob : A = g^a \mod p$
    - $Bob \rightarrow Alice : B = g^b \mod p$
4. Calculan en secreto:
    - $Alice: s = B^a \mod p = g^{ab} \mod p$
    - $Alice: s = A^b \mod p = g^{ab} \mod p$
5. Y usan s como clave de cifrado un algoritmo simétrico

A continuación está el código de la librería https://github.com/amiralis/pyDH (https://github.com/amiralis/pyDH) de Amirali Sanatinia, que es sencillo de leer y entender.

Aunque no parece haber errores evidentes, **es obligatorio utilizar librerías auditadas**. Seguiremos esta por su valor educativo, no porque sea recomendable su uso.

In [1]:

```
# Apache License
#         Version 2.0, January 2004
#     Copyright 2015 Amirali Sanatinia

""" Pure Python Diffie Hellman implementation

Source: https://github.com/amiralis/pyDH"""

import os
import binascii
import hashlib

# RFC 3526 - More Modular Exponential (MODP) Diffie-Hellman groups for
# Internet Key Exchange (IKE) https://tools.ietf.org/html/rfc3526

primes = {

    # 1536-bit
    5: {
    "prime": 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BE
    "generator": 2
    },

    # 2048-bit
    14: {
    "prime": 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BE
    "generator": 2
    },

    # 3072-bit
    15: {
    "prime": 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BE
    "generator": 2
    },

    # 4096-bit
    16: {
    "prime": 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BE
    "generator": 2
    },

    # 6144-bit
    17: {
    "prime": 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BE
    "generator": 2
    },

    # 8192-bit
    18: {
    "prime": 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BE
    "generator": 2
    }
}


class DiffieHellman:
    """ Class to represent the Diffie-Hellman key exchange protocol """
    # Current minimum recommendation is 2048 bit.
    def __init__(self, group=14):
```

```
        if group in primes:
            self.p = primes[group]["prime"]
            self.g = primes[group]["generator"]
        else:
            raise Exception("Group not supported")

        self.__a = int(binascii.hexlify(os.urandom(32)), base=16)

    def get_private_key(self):
        """ Return the private key (a) """
        return self.__a

    def gen_public_key(self):
        """ Return A, A = g ^ a mod p """
        # calculate G^a mod p
        return pow(self.g, self.__a, self.p)

    def check_other_public_key(self, other_contribution):
        # check if the other public key is valid based on NIST SP800-56
        # 2 <= g^b <= p-2 and Lagrange for safe primes (g^bq)=1, q=(p-1)/2

        if 2 <= other_contribution and other_contribution <= self.p - 2:
            if pow(other_contribution, (self.p - 1) // 2, self.p) == 1:
                return True
        return False

    def gen_shared_key(self, other_contribution):
        """ Return g ^ ab mod p """
        # calculate the shared key G^ab mod p
        if self.check_other_public_key(other_contribution):
            self.shared_key = pow(other_contribution, self.__a, self.p)
            return hashlib.sha256(str(self.shared_key).encode()).digest()
        else:
            raise Exception("Bad public key from other party")
```

Alice

Vamos a generar primero las claves de Alice

In [2]:

```
alice = DiffieHellman()
alice_pubkey = alice.gen_public_key()
```

- Claves de Alice:
  - Clave pública: $\{g, p, g^a\}$
  - Clave privada: a

Esta es la clave privada de Alice, que es lo que le envía a Bob.

En realidad g y p suelen escogerse como valores conocidos, así que Alice y Bob ya los tienen y solo se envía g^a

In [3]:

```python
print(f'g={alice.g}\n')
print(f'p={alice.p}\n')
print(f'g^a={alice_pubkey}\n')
```

```
g=2

p=32317006071311007300338913926423828248817941241140239112842009751400
74170663435422261968941736356934711790173790970419175460587320919502885
37589861856221532121754125149017745202702357960782362488842461894775876
41105928646099411723245426622522193230540919037680524235519125679715870
11700105805587765103886184728025797605490356973256152616708133936179954
13364765591603683178967290731783845896806396719009772021941686472258710
31411336429319536193471636533209717077448227988588565369208645296636077
25026895550592836275112117409697299806841055435958486658329164213621823
107899099944865246826241697203591185250704536 1090559

g^a=16293620591251737670600988972047884202577990124149557761371900437349801
33928454515910712924681651856892677104772700569565709492986664302563006
39134702740055263833627264861033186381213134010883028425512493971047784
73790754088197127754965328494814877176679041298186461604463138921181066
08551221961366505480964691998782872252913880285696270557423062673473264
34265064748764792364521656816603181091347834566385976026481556454312744
60578562328453983585189513331383154297266655403312916327073856948024347
46070778389829690491551018406006174771241884773072600175838939678139520
534688699916937207314480592966136253448352454104
```

Esta es la clave privada de Alice, que nunca sale de su ordenador

In [4]:

```python
print(f'a={alice.get_private_key()}\n')
```

```
a=2839192359643895261306307968449565154054801586251711922422418299077180
6962473
```

Bob

Cálculo de las claves públicas y privadas de bob

In [5]:

```
bob = DiffieHellman()
bob_pubkey = bob.gen_public_key()
print(f'g={bob.g}\n')
print(f'p={bob.p}\n')
print(f'g^b={bob_pubkey}\n')
```

```
g=2

p=32317006071311007300338913926423828248817941241140239112842009751400
74170663435422261968941736356934711790173790970419175460587320919502885
37589861856221532121754125149017745202702357960782362488842461894775876
41105928646099411723245426622522193230540919037680524235519125679715
87011700105805587765103886184728025797605490356973256152616708133936179
95413364765591603683178967290731783845896806396719009772021941686472258
71031411336429319536193471636533209717077448227988588565369208645296
63607725026895550592836275112117409697299806841055435958486658329164213
6218231078990999448652468262416972035911852507045361090559
```

```
g^b=2926552621034834566414063755624918553670583888733439265029362949149
53862248221910585429916658353017215969620116213321060028695101716084260
18215148270845128885052839091473342621561560202222919462811064488536
53222256950848629016888442066689457887539546047819147330906514695236353
98090258614155273434833082170364520549712770648474125578358360869405153
15252924844661624837173970448333805789543703798773038091324807987344
12702359427545183227494420151558896359042635430620158211350367153923474
57703546240830040546307789662376646465698657124645834856970209608909922
2467966204255720977197610748900233975544066558509265039403O
```

Fíjate: la g de Alice y la de Bob es 2. Aunque podría ser cualquier, es común que g sea 2 siempre porque así se aceleran los cálculos. Esto no reduce la seguridad del algoritmo, según los matemáticos

Alice y Bob: cálculo de la clave compartida

In [6]:

```
alice_sharedkey = alice.gen_shared_key(bob_pubkey)
print(alice_sharedkey)
```

```
b'&&\xe9T\x92\xb5Tk85"\x7f\xe47o\x19\xaa\xc0g\x81f\x0f6\xd3\x07\x05,/
\xc2\x02f7'
```

In [7]:

```
bob_sharedkey = bob.gen_shared_key(alice_pubkey)
print(bob_sharedkey)
print(len(bob_sharedkey))
```

```
b'&&\xe9T\x92\xb5Tk85"\x7f\xe47o\x19\xaa\xc0g\x81f\x0f6\xd3\x07\x05,/
\xc2\x02f7'
32
```

Y podemos comprobar que los dos tienen la misma clave compartida

In [8]:

```
print(bob_sharedkey == alice_sharedkey)
```

```
True
```

Ejercicios

- Ya tenemos una "clave compartida", pero aún hay que adaptarla para poder usarla en AES-256 o ChaCha20. ¿Cómo lo harías?
- Los parámetros p y g de la librería son muy antiguos (RFC3526). ¿Puedes buscar otros más modernos?