Acuerdo de claves Diffie-Hellan

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 [2]:
```
# 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": 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B
    "generator": 2
    },

    # 2048-bit
    14: {
    "prime": 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B
    "generator": 2
    },

    # 3072-bit
    15: {
    "prime": 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B
    "generator": 2
    },

    # 4096-bit
    16: {
    "prime": 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B
    "generator": 2
    },

    # 6144-bit
    17: {
    "prime": 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B
    "generator": 2
    },

    # 8192-bit
```

```
    18: {
    "prime": 0xFFFFFFFFFFFFFFFFC90FDAA22168C234C4C6628B80DC1CD129024E088A67CC74020BBEA63B
    "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 [4]:
```
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 [5]:
```
print(f'g={alice.g}\n')
print(f'p={alice.p}\n')
print(f'g^a={alice_pubkey}\n')
```

```
g=2

p=323170060713110073003389139264238282488179412411402391128420097514007417066343542261968941736356934711790173790970419175460587320919502885375898618562215321217541251490177452027023579607823624888424618947758764110592864609941172324542662252219323054091903768052423551912567971587011700105805587765103886184728025797605490356973256152616708133936179954133647655916036831789672907317838458968063967190097720219416864722587103141133642931953619347163653320971707744822798858856536920864529663607725026895550592836275112117409697299806841055435958486658329164213621823107899099944865246826241697203591185250704536109055
9

g^a=107861117620064062759143098233221358758776023392433898304708325087602186825390264327086276278977033643409118751886702142783709340203348505971449433357661421671088415993837180532119365918829462736781891657392901517495273684045306463793737438046857041722199021446323826296536577218370496423050676024434658595990919328339478330867838709869657848004766
```

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

In [6]:
```
print(f'a={alice.get_private_key()}\n')
```

```
a=1152251413054086081650767448710113147432499344940056261602191044144787339967 49
```

Bob

Cálculo de las claves públicas y privadas de bob

In [7]:
```
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=323170060713110073003389139264238282488179412411402391128420097514007417066343542261968941736356934711790173790970419175460587320919502885375898618562215321217541251490177452027023579607823624888424618947758764110592864609941172324542662252219323054091903768052423551912567971587011700105805587765103886184728025797605490356973256152616708133936179954133647655916036831789672907317838458968063967190097720219416864722587103141133642931953619347163653320971707744822798858856536920864529663607725026895550592836275112117409697299806841055435958486658329164213621823107899099944865246826241697203591185250704536109055
9

g^b=224884870968640722928830574372326544072956067516621885282605428715982071321050301741400573943243006601554681297840512269362752595574388027153831141145039622571822401095072722110754516162616058489910817956835579751470713677790461917718787563279234665115370327577746796897104184242556513759959106453677719011833264488985170560498851806909712706159438747077759157445355388111455850643041228629027209576510347210270908269124408702311142243494134048096234478006453387084710143116019933553955153075292975433472516766010898194287580320942516189232087819459631141673837524890900854069021583728007418982441406339643593244831 55
```

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 [8]:
```
alice_sharedkey = alice.gen_shared_key(bob_pubkey)
print(alice_sharedkey)
```

```
b'\xa7\xe9M\x11\xbfy\xf9\x85\xfd\xaa\xa1\x1f\x1a\xa9\x8d4\xf4\xd1\xea\xe2U\xcd\x00ISk\xfa.\x18\x8cb\x88'
```

In [9]:
```
bob_sharedkey = bob.gen_shared_key(alice_pubkey)
print(bob_sharedkey)
print(len(bob_sharedkey))
```

```
b'\xa7\xe9M\x11\xbfy\xf9\x85\xfd\xaa\xa1\x1f\x1a\xa9\x8d4\xf4\xd1\xea\xe2U\xcd\x00ISk\xfa.\x18\x8cb\x88'
32
```

Y podemos comprobar que los dos tienen la misma clave compartida

In [10]:
```
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?

In [ ]:
```

```