Clase Account

Una cuenta es el medio por el que un usuario puede interactuar en la blockchain, tener activos y hacer transacciones hacia otros usuarios.

Vamos a crear un archivo en nuestro directorio de trabajo llamado account.py. En él, vamos a instanciar una clase de nombre Account.

In [1]:
```
class Account():
    pass
```

Cada que se instancie una cuenta, vamos a inicializarla con una cantidad de balance a elección y nickname para efectos de práctica.

In [8]:
```
class Account():
    def __init__(self, balance: int, nickname: str):
        pass
```

Como atributos incluiremos un nickname, un balance de su "dinero" y un historial que registra las transacciones que han realizado.

In [1]:
```
class Account:
    def __init__(self, balance:int, nickname: str):
        self.nickname = nickname
        self.balance = 100
        self.list_of_all_transactions = []
```

En blockchain, y hablando específicamente en el contexto de la seguridad de una cuenta, cada cuenta tiene un cifrado asimétrico, teniendo llaves públicas y privadas.

Cada objeto Account tendrá una llave pública que funge como un identificador público para la cuenta, y una llave privada para autorizar transacciones de la cuenta.

Primero, vamos a importar dos funciones que vamos a usar del módulo Crypto:

- PublicKey.RSA
- Signature.pkcs1_15.PKCS115_SigScheme

In [18]:
```
from Crypto.PublicKey import RSA
from Crypto.Signature.pkcs1_15 import PKCS115_SigScheme

class Account:
    def __init__(self, balance: int, nickname: str):
        self.nickname = nickname
        self.balance = 100
        self.list_of_all_transactions = []
        # Cifrado asimetrico
        self.private_key = RSA.generate(1024) # Llave privada con algoritmo RSA de
        self.public_key = self.private_key.publickey() # Llave publica
        self.signer = PKCS115_SigScheme(self.private_key) # (1)
        self.verifier = PKCS115_SigScheme(self.public_key) # (2)
```

- La RSA nos ayuda a instanciar nuestra llave privada y la pública.
- PKCS115_SigScheme nos ayuda con un objeto de tipo "firmador". Con el, podemos firmar transacciones con nuestra llave privada, y verificar transacciones con nuestra llave publica.

Ejemplo visual.

Vamos a desarrollar un ejemplo visual para ver como se visualizan los objetos y como se ven.

Visualizacion de las llaves

In [22]:
```
from Crypto.PublicKey import RSA
private_key = RSA.generate(1024)
private_key.exportKey()
```

Out[22]:
```
b'-----BEGIN RSA PRIVATE KEY-----\nMIICWwIBAAKBgQDFIL3wCiQSg5OlR8NDmg2xzb8rbg7Tz8w
7c+pw2PCyLPbACbab\np6TXOc8jamUTWPiH2V2F16YVVel+q+mTuB70R7niQEYyNHYVoHiw6wxzLRgWCZX
y\ntml6zd20B5CRdFEkkX7qZk8zp5FopEjvHqsJGLyvCp0J0SpjbcZ5FZ0KEQIDAQAB\nAoGACUlvGVFo/
KRNPuKgHN2zmnZOdM3VDf6CqSRfXZyizkoW/9oVS8T8m5UiIJGB\nHIW7A69mXAlaTjQDcY/Iy8hHUPaRK
++oEkzbpYnfr0bPcjtSH9aPo6ga+UHrlbqs\nVbrhr45FRAv6JeGUY1Z/Nddsx7GS+hp1mGHQfCjWKkhXy
uECQQDLlagFQDU300/O\nz4vEpf9z+ZROg+hiaGJ+EpobiNetAXngtP9oJaRXexlOGfKyNz+umybe2Qwmk
Q3D\nHl2hEx4pAkEA9+GFp+58RGc09v7TRZlmWgcR8itaUiC9B/XqGaDLdoR9IeZKPfMD\ndTXy5V+rpzq
baKpzh8jlYTrwJutXZu85qQJAWkzWSwXw1QMwmg3q892hUkK4qp7N\nm6CrVzpPCrmG2KEX+zitNPfFTlw
2nDLcOHpTD9KXyi7BufWLAinc485ECQJAd+gZ\n1VAhwJ0EG+7MmEBKKIZ/AdxCxrObfTxRz6/efCg+t6V
EiI8DPzGnm5kZ2b0Z7Bl3\n5VkizFPMU6dIEjzqcQJAO+iIIUWmsQFx667W5m8OJHGeK6TG4g/VZ7fZzLC
q8est\nnSf/QYC6QnqUjhDtwkg5ZtOOMzPwPSRggMT59FiCNQ==\n-----END RSA PRIVATE KEY-----
'
```

In [64]:
```
private_key.publickey().exportKey() # LLave publica
```

Out[64]:
```
b'-----BEGIN PUBLIC KEY-----\nMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQC2fTuCQuNf2PWg
mP4Nncfu2hq0\nkEP62lqtm4HYvTiZjvRbmQ5X6gHU3QSpE+x0lVXDV4/kaZQBSO/gCMxWOmqvvaYh\nN+
2GmTH4be4ddyBN41QLXzOmtIGKtkQ69gxg/7GLzu4I4nil7k7cbc+lTCgDM/XI\nJyi/55twZjBwj3iyWw
IDAQAB\n-----END PUBLIC KEY-----'
```

Como vemos, es un bunche aleatorio de carácteres alfanumericos. Aunque...

La llave privada se suele visualizar de otra manera, esta manera la vamos a llamar 'identidad'.

Identidad

Las llaves públicas son el identificador de una cuenta, pero no se presentan o se visualizan como se ve arriba. Hay que convertirlo a hexadecimal para que sea visiblemente más accesible.

In [23]:
```
import binascii
binascii.hexlify(private_key.publickey().exportKey(format="DER")).decode('ascii') #
# este funcion convierte lo que exporta RSA en hexadecimal, haciendolo mas visible.
```

Out[23]:
```
'30819f300d06092a864886f70d010101050003818d0030818902818100c520bdf00a24128393a547c
3439a0db1cdbf2b6e0ed3cfcc3b73ea70d8f0b22cf6c009b69ba7a4d739cf236a651358f887d95d85d
7a61555e97eabe993b81ef447b9e2404632347615a078b0eb0c732d18160995f2b6697acdddb407909
1745124917eea664f33a79168a448ef1eab0918bcaf0a9d09d12a636dc679159d0a110203010001'
```

Signer y Verifier de una cuenta

Cada cuenta va a tener, digamos, dos ítems. Una va a ser una pluma, o una "firmadora", que nos ayudara a "firmar" las transacciones. La otra es una "verificadora", que se asegura que esa firma es legítima y que ni el contenido, ni la firma, ni el remitente, hayan cambiado.

Derivadas de estas llaves, se obtienen estos ítems. (1 y 2)

- Para crear el signer, se necesita la llave privada de la cuenta para autorizar/firmar las transacciones.
- Para crear el verifier, se necesita la llave pública de la cuenta para verificar el contenido, autor y firma digital de la transacción.

Con la biblioteca Crypto podemos usar el modulo Signature.pkcs1_15 para obtener un objeto de tipo PKCS115_SigScheme. Con él, podemos instanciar nuestra firmadora y nuestra verificadora.

In [27]:
```
firmadora = PKCS115_SigScheme(private_key)  # Se instancia con la llave privada
verificadora = PKCS115_SigScheme(private_key.publickey())  # Se instancia con la ll
```

In [28]:
```
firmadora, verificadora
```

Out[28]:
```
(<Crypto.Signature.pkcs1_15.PKCS115_SigScheme at 0x104682fb0>,
 <Crypto.Signature.pkcs1_15.PKCS115_SigScheme at 0x1046823e0>)
```

En la clase Account solo vamos a implementar un método. El codigo completo de la clase Account es el siguiente.

In [1]:
```
import binascii
from Crypto.PublicKey import RSA
from Crypto.Signature.pkcs1_15 import PKCS115_SigScheme

class Account:
    def __init__(self, balance: int, nickname: str):
        self.nickname = nickname
        self.balance = balance
        self.list_of_all_transactions = []
        self.private_key = RSA.generate(1024)
        self.public_key = self.private_key.publickey()
        self.signer = PKCS115_SigScheme(self.private_key)
        self.verifier = PKCS115_SigScheme(self.public_key)

    @property
    def identity(self):
        return binascii.hexlify(self.public_key.exportKey(format="DER")).decode('as
```

Con la clase Account lista, podemos pasar a la clase Transaction.