1 BlockChain

La primera part d'aquesta pràctica consisteix en implementar una BlockChain senzilla per comprendre el seu funcionament.

Com a funció *hash* farem servir `SHA256` i RSA per validar les transaccions.

Cada bloc serà de la forma:

Lo mas costoso de calcular

contenido del bloque

Lo mas importante. (

```
class block:
    def __init__(self):
        self.block_hash
        self.previous_block_hash
        self.transaction
        self.seed
```

on

- `block_hash` és el SHA256 del bloc actual representat per un enter,
- `previous_block_hash` és el SHA256 del bloc anterior representat per un enter,
- `transaction` és una transacció vàlida,
- `seed` és un enter.

Cada transacció serà de la forma:

```
class transaction:
    def __init__(self, message, RSAkey):
        self.public_key
        self.message
        self.signature
```

on

- `public_key` és la clau pública RSA corresponent a `RSAkey`,
- `RSAkey` és la clau RSA amb que es signa la transacció,
- `message` és el document que es signa a la transacció representat per un enter,
- `signature` és la signatura de `message` feta amb la clau `RSAkey` representada per un enter.

Les claus privades i públiques RSA seran de la forma:

```
class rsa_key:
    def __init__(self,bits_modulo=2048,e=2**16+1):
        self.publicExponent
        self.privateExponent
        self.modulus
        self.primeP
        self.primeQ
        self.privateExponentModulusPhiP
        self.privateExponentModulusPhiQ
        self.inverseQModulusP
```

on

- `publicExponent, privateExponent, modulus, primeP, primeQ` estan representats per enters,
- `privateExponentModulusPhiP` és congruent amb `privateExponent` modul `primeP`-1 representat per un enter,
- `privateExponentModulusPhiQ` és congruent amb `privateExponent` modul `primeQ`-1 representat per un enter,
- `inverseQModulusP` és l'invers de `primeQ` mòdul `primeP` representat per un enter,

i

```
class rsa_public_key:
    def __init__(self, rsa_key):
        self.publicExponent
        self.modulus
```

on

- `publicExponent` és l'exponent públic de la clau `rsa_key`,
- `modulus` és el mòdul de la clau `rsa_key`.

Una transacció és vàlida si

$$\texttt{signature}^{\texttt{publicExponent}} \equiv \texttt{message} \mod \texttt{modulus}$$

Un bloc és vàlid si la transacció és vàlida i el seu *hash* h satisfà la condició $h < 2^{256-d}$ on d és un paràmetre que indica el *proof of work* necessari per generar un bloc vàlid. Per aquesta pràctica **d=16**.

Per calcular el *hash* h d'un bloc farem el següent:

```
entrada=str(previous_block_hash)
entrada=entrada+str(transaction.public_key.publicExponent)
entrada=entrada+str(transaction.public_key.modulus)
entrada=entrada+str(transaction.message)
entrada=entrada+str(transaction.signature)
entrada=entrada+str(seed)
h=int(hashlib.sha256(entrada.encode()).hexdigest(),16)
```

Definiu, en **Python 3.x** les següents classes amb (com a mínim) els mètodes descrits:

- ```
  class rsa_key:
      def __init__(self,bits_modulo=2048,e=2**16+1):
          '''
          genera una clau RSA (de 2048 bits i amb exponent públic 2**16+1 per defecte)
          '''
          self.publicExponent
          self.privateExponent
          self.modulus
          self.primeP
          self.primeQ
          self.privateExponentModulusPhiP
          self.privateExponentModulusPhiQ
          self.inverseQModulusP


      def sign(self,message):
          '''
          retorma un enter que és la signatura de "message" feta amb la clau RSA fent servir el TXR
          '''
  ```
  ```

```
    def sign_slow(self,message):
        '''
        retorma un enter que és la signatura de "message" feta amb la clau RSA sense fer servir el TXR
        '''
```

- class rsa_public_key:

```
    def __init__(self, rsa_key):
        '''
        genera la clau pública RSA asociada a la clau RSA "rsa_key"
        '''
        self.publicExponent
        self.modulus


    def verify(self, message, signature):
        '''
        retorna el booleà True si "signature" es correspon amb una
        signatura de "message" feta amb la clau RSA associada a la clau
        pública RSA.
        En qualsevol altre cas retorma el booleà False
        '''
```

- class transaction:

```
    def __init__(self, message, RSAkey):
        '''
        genera una transacció signant "message" amb la clau "RSAkey"
        '''
        self.public_key
        self.message
        self.signature

    def verify(self):
        '''
        retorna el booleà True si "signature" es correspon amb una
        signatura de "message" feta amb la clau pública "public_key".
        En qualsevol altre cas retorma el booleà False
        '''
```

- class block:

```
    def __init__(self):
        '''
        crea un bloc (no necesàriamnet vàlid)
        '''
        self.block_hash
        self.previous_block_hash
        self.transaction
        self.seed


    def genesis(self,transaction):
        '''
        genera el primer bloc d'una cadena amb la transacció "transaction" que es caracteritza per:
           - previous_block_hash=0
           - ser vàlid
        '''
```

```
    def next_block(self, transaction):
        '''
        genera el següent block vàlid amb la transacció "transaction"
        '''

    def verify_block(self):
        '''
        Verifica si un bloc és vàlid:
           -Comprova que el hash del bloc anterior cumpleix las condicions exigides
           -Comprova la transacció del bloc és vàlida
           -Comprova que el hash del bloc cumpleix las condicions exigides

        Si totes les comprovacions són correctes retorna el booleà True.
        En qualsevol altre cas retorma el booleà False
        '''
```

- class block_chain:

```
    def __init__(self,transaction):
        '''
        genera una cadena de blocs que és una llista de blocs,
        el primer bloc és un bloc "genesis" generat amb la transacció "transaction"
        '''
        self.list_of_blocks

    def add_block(self,transaction):
        '''
        afegeix a la llista de blocs un nou bloc vàlid generat amb la transacció "transaction"
        '''

    def verify(self):
        '''
        verifica si la cadena de blocs és vàlida:
           - Comprova que tots el blocs són vàlids
           - Comprova que el primer bloc és un bloc "genesis"
           - Comprova que per cada bloc de la cadena el següent és el correcte

        Si totes les comprovacions són correctes retorna el booleà True.
        En qualsevol altre cas retorma el booleà False i fins a quin bloc la cadena és válida
        '''
```

2 RSA

2.1 Ron was wrong, Whit is right

Es recomana llegir l'article:

"Ron was wrong, Whit is right", https://eprint.iacr.org/2012/064.pdf.

En *Atenea* tobareu el directori RSA_RW on hi han una sèrie de fitxers del tipus:

- *nom.cognom_AES.enc* que és el resultat de xifrar un fitxer amb la clau K
- *nom.cognom_RSA_RW.enc* que és el resultat de xifrar la clau K amb la clau pública RSA que es troba a
- *nom.cognom_pubkeyRSA_RW.pem*.

El fitxer xifrat s'ha obtingut fent servir la comanda:

```
openssl enc -e -aes-128-cbc -pbkdf2 -kfile fichero.key -in fichero.txt -out fichero.enc
```

El fitxer `fichero.key` que conté la clau s'ha xifrat amb la comanda:

```
openssl rsautl -encrypt -inkey pubkeyRSA.pem -pubin -in fichero.txt -out fichero.enc
```

openssl està disponible en https://www.openssl.org. S'instal.la per defecte en la majoria de les distribucions Linux, a la imatge Linux de la FIB ho està.

Del fitxer *nom.cognom_pubkeyRSA_RW.pem* heu d'extreure la clau pública (*openssl* pot ajudar), factoritzar el mòdul, calcular la clau privada, escriure-la en un fitxer en format PEM (pot ser útil la biblioteca *Crypto.PublicKey.RSA* de *python* però podeu fer servir qualsevol altra) i, per acabar, desxifrar els fitxers fent servir *openssl*.

2.2 Pseudo RSA

En *Atenea* tobareu el directori RSA_pseudo on també hi han una sèrie de fitxers semblants als anteriors.

Ara el mòdul públic és un enter $n = p\,q$ amb p i q tals que si en binari p és la concatenació de r i s de exactament la meitat de bits de p, llavors q és, en binari, la concatenació de s i r. O sigui que si $p = r||s$, llavors $q = r||s$ amb $\#bits(r) = \#bits(s) = \frac{1}{2}\#bits(p) = \frac{1}{2}\#bits(q)$.

Del fitxer *nom.cognom_pubkeyRSA_pseudo.pem* heu d'extreure la clau pública (*openssl* pot ajudar), factoritzar el mòdul, calcular la clau privada, escriure-la en un fitxer en format PEM (pot ser útil la biblioteca *Crypto.PublicKey.RSA* de *python* però podeu fer servir qualsevol altra) i, per acabar, desxifrar els fitxers fent servir *openssl*, el fitxer.

3 Entrega

Un únic fitxer `zip`, `tar`,... amb:

- BlockChain:
  - El codi python
  - Una taula comparativa amb el temps necessari per signar, fent servir el TXR i sense fer-ho servir, 100 missatges diferents amb claus de 512, 1024, 2048 i 4096 bits.
  - Un fitxer amb una cadena vàlida de 100 blocs.
  - Un fitxer amb una cadena de 100 blocs que només sigui vàlida fins al bloc XX on XX són les dues darreres xifres del vostre DNI.[1]

  Els fitxers amb les cadenes de blocs heu de generar-los amb el següent codi:

```
import pickle
with open(fitxer_de_sortida, 'wb') as file:
pickle.dump(cadena_de_blocs, file)
```

- RSA:
  - Els fitxers desxifrats.
  - Els fitxers desxifrats amb les claus secretes que s'han fet servir per l'AES.
  - Els fitxers en format PEM amb les claus privades RSA.

Referències

Sympy: Number Theory

Sympy is a Python library for symbolic mathematics.
Welcome to SymPy's documentation!
Number Theory

Sage

http://www.sagemath.org

SageMath is a free open-source mathematics software system licensed under the GPL. It builds on top of many existing open-source packages: NumPy, SciPy, matplotlib, Sympy, Maxima, GAP, FLINT, R and many more. Access their combined power through a common, Python-based language or directly via interfaces or wrappers.

https://cocalc.com
http://sagecell.sagemath.org
Sage Quick Reference: *Elementary Number Theory*, William Stein, Sage Version 3.4
http://wiki.sagemath.org/quickref
http://wiki.sagemath.org/quickref?action=AttachFile&do=get&target=quickref-nt.pdf

[1] Si $XX = 00$, llavors descarteu les dues darreres xifres fins que $XX \neq 00$.