

1 RSA y teorema chino de los restos (TCR)

La primera parte consistirá en comprobar que usar o no el teorema chino de los restos para descifrar/firmar tiene un impacto considerable en el tiempo de ejecución y ver que el coste de cifrar/verificar, fijado el exponente público e , crece cuadráticamente con el tamaño del módulo.

Definid en **Python 3.x** la siguiente clase con los métodos descritos¹:

```
class rsa_key:
    def __init__(self,bits_modulo=2048,e=2**16+1):
        """
        genera una clave RSA (de 2048 bits y exponente público 2**16+1 por defecto)
        """
        self.publicExponent
        self.privateExponent
        self.modulus
        self.primeP
        self.primeQ
        self.privateExponentModulusPhiP
        self.privateExponentModulusPhiQ
        self.inverseQModulusP

    def sign(self, message):
        """
        Entrada: un entero "message"
        Salida: un entero que es la firma de "message" hecha con la clave RSA usando el TCR
        """

    def sign_slow(self, message):
        """
        Entrada: un entero "message"
        Salida: un entero que es la firma de "message" hecha con la clave RSA sin usar el TCR
        """

    def verify(self, message, signature):
        """
        Entrada: dos enteros "message" y "signature"
        Salida: el booleano True si "signature" se corresponde con la firma de "message"
                hecha con la clave RSA;
                el booleano False en cualquier otro caso.
        """

    def __repr__(self):
        return str(self.__dict__)
```

¹Los métodos `__repr__(self)` y `from_dictionary(self, RSAKey)` ya están completamente definidos, no es necesario añadir nada, se usarán para guardar las claves como diccionarios e importar claves.

```

def from_dictionary(self, RSAKey):
    """
    Importa una clave RSA:
    RSAKey = {
        'publicExponent': 65537,
        'modulus': 570131475606908079645289541584935910730810198868796008957155625353952799
        'privateExponent': 4753342970712949550913014106871064919409107720248258835378333364
        'primeP': 6984543319619713760180514011748821810865467260539703382978371694181772714
        'primeQ': 8162759532257435135158099414290415000169780797669900784827582678722199767
        'privateExponentModulusPhiP': 45804459669184686161831989824214201536412548238511972
        'privateExponentModulusPhiQ': 23195305059619789511581135143862306567002125180433428
        'inverseQModulusP': 567233782337109666186598736353187660521719314613992812595580843
    }
    """

    self.publicExponent = RSAKey['publicExponent']
    self.privateExponent = RSAKey['privateExponent']
    self.modulus = RSAKey['modulus']
    self.primeP = RSAKey['primeP']
    self.primeQ = RSAKey['primeQ']
    self.privateExponentModulusPhiP = RSAKey['privateExponentModulusPhiP']
    self.privateExponentModulusPhiQ = RSAKey['privateExponentModulusPhiQ']
    self.inverseQModulusP = RSAKey['inverseQModulusP']

```

Generad una tabla comparativa con el tiempo necesario para descifrar/firmar, usando el TCR y sin usarlo, 1024 mensajes diferentes con claves de 512, 1024, 2048, 4096 y 8192 bits.

Generad una tabla comparativa con el tiempo necesario para cifrar/verificar 1024 mensajes/firmas diferentes con claves de 512, 1024, 2048, 4096 y 8192 bits.

Las tablas deben incluir una explicación detallada de lo que se observa.

2 Votaciones electrónicas UPC

En las votaciones electrónicas que se celebran en la UPC el votante recibe, si lo desea, un comprobante de votación como el de la imagen:

Aquest és el vostre rebut de votació. Copieu-lo per a poder verificar que el vostre vot ha estat emès i emmagatzemat correctament.

Rebut

wsNSSKD8TT

Codi de Control

```
YxL9fF+egpbeCH3lmZhU7GPY/C4VH/Jw
qyW+gG7M0fag3hzdtmiDkGw8cwWt1US
uwK1Dg0i0csiUo0gsXC6Ufh3ofOlpgG
toUxivu5WRCa4QfiC3mv4pdyWcYHmW
AXro7ShWUA1HF2MkyZxDq2QVvxxnxn
87BmYaKYuQUqks0iRhQYD0l0GMvpFhg
MSeyRtKFUcjznuHFOtuTDy89w9MTGIU
0b6Lu18sgUoVurZL5kg7o+wJXwqRYr
Uq/bbFG7dXGV38afCokW4LaMrNr4WAh
86L1NgOIE37z/AegnNm86MyB0q3WBVI
wMv7LvN7Wuz4anLj80x+r32PmCP0kEw
==#N04dsm1p0R6Nvrk0wVkoFFiYxy9y
9Gsj#40289db47d382f52017e63a614
7733a4#40289db47d382f52017e63a5
7cd43385#1642586444563
```

El comprobante consta de dos partes, **Rebut** y **Codi de control**, cuya descripción la encontraréis en el apartado [Guia tècnica per a la verificació de codi de control i rebut](https://seuelectronica.upc.edu/ca/seu-electoral/Informacio-us-plataforma-e-Vot) de <https://seuelectronica.upc.edu/ca/seu-electoral/Informacio-us-plataforma-e-Vot>.

La segunda parte consistirá en definir una función para determinar la validez del **Rebut** y del **Codi de control** incluidos en el comprobante de votación recibido al emitir el voto.

Definid en **Python 3.x** la siguiente función:

```
def comprobante_voto(Rebut, Codi, Certificado):
    """
Entrada:
    Rebut: String de longitud 10
    Codi: String formado por la concatenación de 5 partes usando # como separador,
          podría contener espacios y saltos de líneas que deben ser eliminados.
    Certificado: Fichero que contiene el certificado, en formato PEM, con la clave
                  pública del firmante del comprobante de votación.

Salida: un entero cuyo valor será
        0 si el Rebut y la firma del Codi son correctos,
        1 si el Rebut es incorrecto pero la firma del Codi es correcta,
        2 si el Rebut es correcto pero la firma del Codi es incorrecta,
        3 si ni el Rebut ni la firma del Codi son correctos.
    """
```

Comprobante de voto

El siguiente comprobante de voto es un ejemplo real que podéis usar para validar vuestra implementación.

Rebut

iVoIkFePBM

Codi de Control

```
Ak1TUamCVK4urY+1f00fBtHz7CYt1RE8
0A/xju/qlkKPkj5gUxZ2NNz07FuU1a1
dGztjGJvB3oAfLw6rpAMWNg0c+W+p0m
un8vCEiEuAzPWhflr1ACVRBSoC/vioE
DJ8zoVtzJWEv9a+4vmaxRxaZzZoGFiN
7nNscvJXgqU4pvILQ03VWUve6JJ65KV
rAbxRM8bkBshy3yMGjXLVpZDSsK0wjT
YGArSzX43ITQxbpuitqBujKSw54De1
QDJbv8SGo19aLSMpmZ30/M1PgjAAf1H
WWjzK1unmzv9LDIoh/nLRU6+3a8CWZ9
L5SfV8/MSHQqfHbXJZd9ZcMBvso4NkA
==#0R0En5w1t0cSTjAbvy0gM/9p1Epk
Y3C+#40289dc597e9c1fa0199c2cdf1
29137c#40289dc597e9c1fa0199c2cd
f11c1375#1760076542676
```

3 RSA: puerta trasera

Se han generado unas claves RSA que incluyen en la clave pública una puerta trasera que permite obtener fácilmente la clave privada.

El proceso seguido para generar las claves RSA ha sido el siguiente:

- Generar dos números primos aleatorios diferentes, $p > q$.
- El exponente público e es de la forma $e = \frac{p-q}{2^k} - \epsilon$ siendo:
 - k tal que $\frac{p-q}{2^k}$ sea impar,
 - ϵ el menor entero no negativo tal que $\text{mcd}(e, \varphi(n)) = 1$.

En *Atenea* encontraréis el directorio **RSA_ptE** donde hay una serie de ficheros del tipo:

- **nombre.apellido_AES.enc** que es el resultado de cifrar un fichero determinado con la clave K
- **nombre.apellido_RSA_ptE.enc** que es el resultado de cifrar la clave K con la clave pública RSA que se encuentra en el fichero **nombre.apellido_pubkeyRSA_ptE.pem**.

El fichero cifrado se ha obtenido usando el comando²:

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

El fichero **fichero.key** que contiene la clave se ha cifrado con el comando:

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

Del fichero **nombre.apellido_pubkeyRSA_ptE.pem** hay que extraer la clave pública (**openssl** puede ayudar), factorizar el módulo, calcular la clave privada, escribirla en un fichero en formato PEM (puede ser útil la biblioteca *Crypto.PublicKey.RSA* de *python* aunque podéis encontrar otras para cualquier lenguaje de vuestra preferencia) y, para acabar, descifrar el fichero usando **openssl**.

²**openssl** está disponible en <https://www.openssl.org>. Se instala por defecto en la mayoría de las distribuciones de Linux, por ejemplo en la imagen Linux de la FIB.

4 Entrega

Un único fichero **zip**, **tar**,... con:

- RSA TCR:
 - Un fichero que contenga las clases implementadas.
El nombre del fichero será **RSA_Lab_XX.py** substituyendo XX por el número, con exactamente dos cifras, del grupo de laboratorio elegido
 - Un fichero, en formato PDF o Markdown, con las dos tablas pedidas y sus respectivos comentarios.
- Votaciones UPC:
 - Un fichero que contenga las funciones implementadas.
El nombre del fichero será **ComprobanteVoto_Lab_XX.py**.
- RSA puerta trasera:
 - El fichero descifrado.
 - El fichero descifrado con la clave secreta que se ha usado para el AES.
 - El fichero en formato PEM con la clave privada RSA.
 - El código utilizado para generar los ficheros anteriores.

Referencias

base64 — Base16, Base32, Base64, Base85 Data Encodings <https://docs.python.org/es/3/library/base64.html>

hashlib — Secure hashes and message digests <https://docs.python.org/es/3/library/hashlib.html>

PyCryptodome <https://www.pycryptodome.org/>

Cryptography <https://cryptography.io/>

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>