Universitat Politècnica de Catalunya

Criptografía

Práctica 3

Joan Marc Pastor Mackenzie

1. BlockChain

Para esta sección, la implementación del BlockChain se puede encontrar en el directorio **/BlockChain**. Las clases están separadas por ficheros junto a tests sobre éstas.

- /BlockChain/rsa_key.py
- /BlockChain/rsa_public_key.py
- /BlockChain/transaction.py
- /BlockChain/block.py
- /BlockChain/block_chain.py

Comparación de eficiencia entre firmar con y sin el Teorema Chino del Resto (TXR):

	#bits n = 512	**#bits n = 1024**	**#bits n = 2048**	**#bits n = 4096**
Con TXR	114 ms	633 ms	4135 ms	28164 ms
Sin TXR	316 ms	2082 ms	14030 ms	96554 ms

Procesador: Intel i7-6700K @ 4.2 GHz

Como se puede observar, sin usar TXR hay una exponenciación más rápida ya que hay que calcular $\boldsymbol{mensaje^{d}\ mod\ n}$. Con TXR el tiempo de ejecución viene determinado por el cálculo de los exponenciales $\boldsymbol{mensaje^{d1}\ mod\ p}$ y $\boldsymbol{mensaje^{d2}\ mod\ q}$ siendo los exponentes *d1* y *d2* mucho más pequeños que *d*. El código utilizado para generar esta tabla se puede encontrar en **/BlockChain/_SignExecutionTimes.py**.

Cadenas de 100 bloques generadas y serializadas con *pickle*:

- Cadena válida: /BlockChain/out/BC_100blocks_100valids.block
- Cadena válida hasta el 90: /BlockChain/out/BC_100blocks_90valids.block

Los ficheros utilizados para generar las cadenas anteriores y verificarlas son los siguientes:

- /BlockChain/_BC_100blocks_100valids.py
- /BlockChain/_BC_100blocks_90valids.py

DNI: 49765**90**

Por tal de romper la cadena de bloques en el índice 91, se ha remplazado el mensaje de la transacción del bloque 91 por un mensaje al azar. Para esta sección, se ha considerado que el índice del primer bloque de la cadena es 1, no 0.

2. RSA

2.1 Ron was wrong, Whit is right

Por tal generar el exponente privado, se ha comparado el módulo n con el de los otros estudiantes. Siendo *n1* y *n2* módulos, si se cumple que *gcd(n1, n2) != 1*, entonces, estos dos módulos comparten un factor común. Éste se puede calcular mediante *gcd(n1, n2)*.

Realizando esta búsqueda, se han encontrado dos primos distintos factores del módulo *n* tal que *n = p * q*. Dada esa igualdad, se puede calcular φ(n) mediante la siguiente fórmula:

$$\varphi(n) = (\mathrm{p} - 1) * (\mathrm{q} - 1)$$

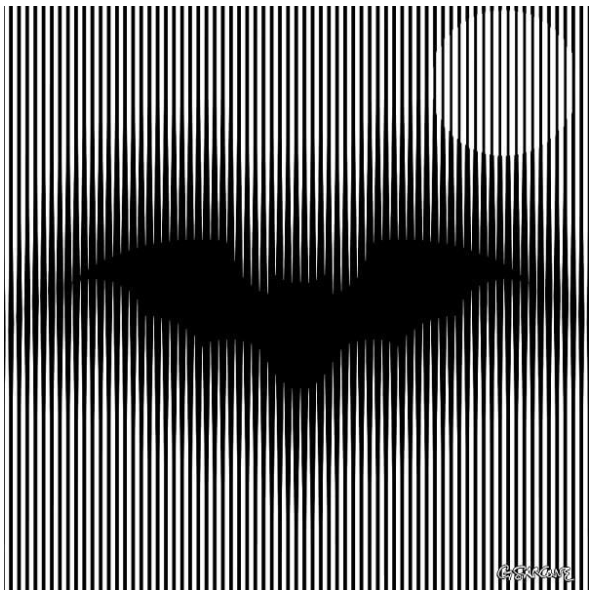
Directorios y ficheros:

- **RSA_RW/factorize.py**: Script para generar la llave privada RSA
- **RSA_RW/decryptCommands.txt**: Comandos openssl utilizados para desencriptar
- **RSA_RW/bin**: Ficheros encriptados y llaves públicas RSA
- **RSA_RW/out**: Ficheros desencriptados y llave RSA privada
    - **RSAprivateKey.pem**: Llave privada RSA
    - **AESkey.dec**: Llave AES
    - **AESmessage.dec**: Mensaje desencriptado
    - **AESmessage.dec.jpg**: Mensaje desencriptado en formato JPEG

Comandos openssl utilizados:

- **Desencriptar llave AES:** openssl rsautl -decrypt -inkey ./out/RSAprivateKey.pem -in ./bin/joan.marc.pastor_RSA_RW.enc -out ./out/AESkey.dec
- **Desencriptar mensaje:** openssl enc -d -aes-128-cbc -pbkdf2 -kfile ./out/AESkey.dec -in ./bin/joan.marc.pastor_AES_RW.enc -out ./out/AESmessage.dec

Fichero desencriptado (Formato JPEG):



2.2 Pseudo RSA

Para generar la llave privada se ha utilizado la función totient(n) de la librería *SimPy*. Este método factoriza el módulo n y calcula φ(n) con la siguiente implementación:

```
t = 1
for p, k in factors.items():
    t *= (p-1) * p**(k-1)
return t
```

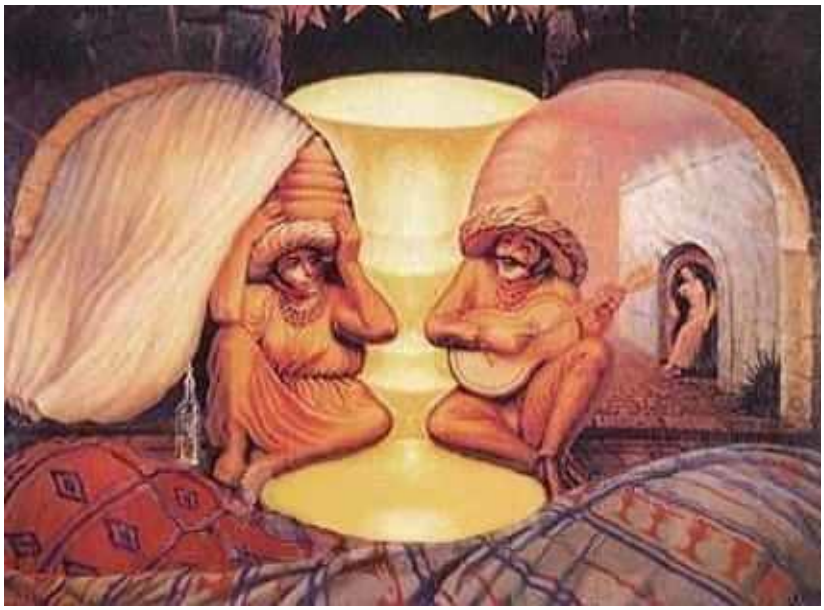
$$\varphi(n) = \prod_{i=1}^{\#factores\,primos} \left((p_i - 1) * {p_i}^{k_i - 1}\right)$$

Siendo *p* los primos resultantes de la factorización de *n* con sus respectivos exponentes *k*.

Directorios y ficheros:

- **RSA_Pseudo/factorize.py**: Script para generar la llave privada RSA
- **RSA_ Pseudo/decryptCommands.txt**: Comandos openssl utilizados para desencriptar
- **RSA_ Pseudo/bin**: Ficheros encriptados y llave pública RSA
- **RSA_ Pseudo/out**: Ficheros desencriptados y llave RSA privada
    - **RSAprivateKey.pem**: Llave privada RSA
    - **AESkey.dec**: Llave AES
    - **AESmessage.dec**: Mensaje desencriptado
    - **AESmessage.dec.png**: Mensaje desencriptado en formato PNG

Fichero desencriptado (Formato PNG):



Referencias Consultadas

Wikipedia – Blockchain

Wikipedia – Euler's totient function

SymPy – Number Theory & documentation

SymPy – Totient Implementation