Códigos y criptografía: Curso 2022-2023

Práctica 5: Cifrado RSA. Autentificación de firma.

- Pueden ser de utilidad las funciones de prácticas anteriores *letter_number* y *power_mod*.
- NOTA: El texto que se muestra en los ejemplos no hay que reproducirlo literalmente. Lo importante es que los programas/funciones hagan lo que se pide, y que si se tiene que solicitar algún dato al usuario sean precisos en la descripción de los valores solicitados.

1. Programa *key_generation*

```
key_generation
```

Se trata de un programa con el que se generan las claves pública y privada necesarias para cifrar con RSA. Para ello:

- Se debe pedir al usuario los valores de los primos p y q. Una sugerencia, para facilitar la introducción de estos valores, es pedirle al programa que muestre en primer lugar un listado de números primos hasta un cierto valor.
- Si los números anteriores son lo suficientemente grandes, puede considerarse $e = 1 + 2^{2^4} = 65537$. En caso contrario, o bien se puede solicitar el valor al usuario o se puede generar cualquier valor válido para e de la manera que se considere más oportuna.

Ejemplo:

```
>> key_generation

Introduce the prime number p:
    1187
Introduce the prime number q:
    1171

The value n is n = p q = 1389977

The value e, of the public key, should satisfy gcd(e, ...
    fiden)=gcd(e,(p-1)(q-1))=gcd(e,1387620)=1.
It is selected e = 65537.

d is the inverse of e module fiden: d = 924713.

PRIVATE KEY: (n, d) = (1389977, 924713)
PUBLIC KEY: (n, e) = (1389977, 65537)
```

2. Función *letter_2numbers*

```
function  double = letter_2numbers (text)
```

Se trata de una función que asocia a cada letra su correspondiente valor de $\mathbb{Z}_{27}$ usando siempre dos dígitos por cada letra:

a	b	c	d	e	...	x	y	z
00	01	02	03	04	...	24	25	26

Entrada: una cadena de texto usando el alfabeto español.

Salida: una cadena numérica formada por los números asociados a cada letra del texto.

Ejemplo:

```
>> double = letter_2numbers ('hola, vamos a cifrar con RSA')
double   =   '07151100220012151900020805180018021513181900'
```

3. Función *prepare_num_cipher*

```
function v = prepare_num_cipher (d,double)
```

Se trata de una función que convierte una cadena numérica en bloques de un tamaño dado, convierte dichos bloques en números y los almacena en un vector.

NOTA: Si es necesario, para completar el último bloque se añadirán tantos 30 como sean necesarios y/o un 0.

Entradas:

d: un número natural que representa el tamaño de los bloques.

double: una cadena numérica.

Salida: el vector formado por los números que se corresponden con cada uno de los bloques.

Ejemplos:

```
>> v = prepare_num_cipher (7,'83629486523')
v   =   8362948     6523300
```

```
>> v = prepare_num_cipher (7,'836294806523')
v   =   8362948      652330
```

4. Función *rsa_cipher*

```
function v = rsa_cipher (n, e, text)
```

Se trata de una función que cifra la cadena de texto introducida según el sistema RSA usando la clave pública (n, e).

Entradas:

n y e: los valores de la clave pública para el cifrado RSA.

text: el texto que queremos cifrar.

Salida: el criptograma, es decir, el vector formado por los bloques ya cifrados.

Ejemplo:

```
>> v = rsa_cipher (2726447, 65537, 'cifrando con RSA')
v   =   670406   2123352    740929   1523275   1351881
```

5. Función *rsa_num_cipher*

```
function v = rsa_num_cipher (n, e, blocks)
```

Se trata de una función que realiza el mismo procedimiento que la anterior, salvo que en vez de partir de un texto comienza con los bloques de números.

Entradas:

n y e: los valores de la clave pública para el cifrado RSA.

blocks: un vector numérico.

Salida: el vector formado por los bloques ya cifrados a partir de las entradas.

Ejemplo:

```
>> v = rsa_num_cipher (2726447, 65537, [20805 180013 31502 151318 190030])
v   =   670406   2123352    740929   1523275   1351881
```

6. Función *num_decipher*

```
function text = num_decipher (n, blocks)
```

Se trata de una función que transforma un vector numérico en letras (usando dos dígitos por letra). Para ello debe completar los bloques con 0's a la izquierda para que todos tengan lontigud $d\acute{i}gitos(n) - 1$, concatenarlos, agruparlos de dos en dos, eliminar los posibles 30's y/o 0 que puedan haber al final y pasar a letras.

Entradas:

n: un número natural, necesario para determinar la longitud correcta de los bloques.

blocks: un vector numérico.

Salida: una cadena de texto.

Ejemplo:

```
>> text = num_decipher (2127781,[104 201530])
text    =    'abeto'
```

7. Función *rsa_num_decipher*

```
function v = rsa_num_decipher (n, d, code)
```

Se trata de una función que descifra un vector numérico, usando el sistema RSA a partir de la clave privada proporcionada, y devuelve un vector numérico (el paso previo a convertir los dígitos en caracteres).

Entradas:

n y d: los valores de la clave privada para el descifrado RSA.

code: un vector numérico, que se supone cifrado según el sistema RSA.

Salida: el vector numérico obtenido tras aplicar el descifrado con RSA.

Ejemplo:

```
>> code = [403866  424206  786183  950614  1268222  1245474  747657  1069757]

>> v = rsa_num_decipher (1389977, 924713, code)
v    =    161518    50813   161503    41215   190304   190208    51800   183030
```

8. Función *rsa_decipher*

```
function text = rsa_decipher (n, d, code)
```

Se trata de una función que descifra un vector numérico, usando el sistema RSA a partir de la clave privada proporcionada, devolviendo el texto llano.

Entradas:

n y d: los valores de la clave privada para el descifrado RSA.

code: un vector numérico, que se supone cifrado según el sistema RSA.

Salida: el texto llano.

Ejemplo:

```
>> code = [403866  424206  786183  950614  1268222  1245474  747657  1069757]

>> text = rsa_decipher (1389977, 924713, code)
text    =    'porfinpodemosdescifrar'
```

9. Programa *sign_auth*

```
sign_auth
```

Se trata de un programa que incluya todos los pasos necesarios para cifrar y descrifrar un mensaje mediante el método RSA con autentificación de firma.

- Si se considera conveniente se pueden programar funciones auxiliares.

Ejemplos:

```
>> sign_auth

AGENT A
Introduce the public key of A, (na,ea):
    [27371551   13]
Introduce the private key of A, (na,da):
    [27371551   18941533]

AGENT B
Introduce the public key of B, (nb,eb):
    [492859   179]
Introduce the private key of B, (nb,db):
    [492859   422459]

AGENT A
Introduce the text you want to send to B:
    'el programa funciona'
Introduce your signature:
    'byalma'

The two cryptograms that A sends to B are:
    text_ciph   =   432488   192897   450957   295922   319626    81530   184771 ...
          165686   440500    53020
    sign_ciph_da_eb   =   259007    68799   439509    59081


AGENT B
B starts deciphering the codes.
The text he has received jointly with the signature is:
    text   =   'elprogramafuncionabyalma'
B obtains the signature:
    signature   =   'byalma'

We have succeeded with the signature authentification.
```

```
>> sign_auth

AGENT A
Introduce the public key of A, (na,ea):
[151535011   19]
Introduce the private key of A, (na,da):
[151535011   47845387]

AGENT B
Introduce the public key of B, (nb,eb):
[1389977   179]
Introduce the private key of B, (nb,db):
[1389977   1271339]

AGENT A
Introduce the text you want to send to B:
'buenos dias'
Introduce your signature:
'andrea'

The two cryptograms that A sends to B are:
text_ciph   =   1368412    826348    780471    1058297    286533    797169
sign_ciph_da_eb   =   266522    732743    682143


AGENT B
B starts deciphering the codes.
The text he has received jointly with the signature is:
text   =   'buenosdiasandrea'
B obtains the signature:
signature   =   'andrea'

We have succeeded with the signature authentification.
```

```
>> sign_auth

AGENT A
Introduce the public key of A, (na,ea):
[151535011   19]
Introduce the private key of A, (na,da):
[151535011   47845387]

AGENT B
Introduce the public key of B, (nb,eb):
[492859   179]
Introduce the private key of B, (nb,db):
[492859   422459]

AGENT A
Introduce the text you want to send to B:
'es verano y llueve'
Introduce your signature:
'yeron'

The two cryptograms that A sends to B are:
text_ciph   =   420431    52480    190622    134904    54177    133024    177924 ...
       141441
sign_ciph_da_eb   =   21609    339478    365119    336681


AGENT B
B starts deciphering the codes.
The text he has received jointly with the signature is:
text   =   'esveranoyllueveyeron'
B obtains the signature:
signature   =   'yeron'

We have succeeded with the signature authentification.
```