

# Códigos y criptografía: Curso 2021-2022

## Práctica 3: Cifrado simétrico con Mochilas.

### Cifrado con Mochilas Trampa.

- Mientras no se diga lo contrario, si es necesario asignar números a letras se hará uso del código UNICODE.

#### 1. Función *knapsack*

```
1 function valide = knapsack(s)
```

Se trata de una función de control que analiza si un vector fila es una mochila simple o supercreciente.

**Entrada:** un vector fila.

**Salida:**

*valide* = -1 si la entrada introducida no representa una mochila.

*valide* = 0 si la entrada representa una mochila, pero no es supercreciente.

*valide* = 1 si la entrada representa una mochila supercreciente.

**Ejemplos:**

```
1 >> valide = knapsack ([5 6 7 2])
2 valide = 0
```

```
1 >>valide = mochila ([5 6 7 20])
2 valide = 0
```

```
1 >> valide = knapsack ([5 6.5 7 2])
2 valide = -1
```

```
1 >> valide = knapsack ([5 6 17 40])
2 valide = 1
```

```
1 >> valide = knapsack ([5 6 17 -40])
2 valide = -1
```

## 2. Función *knapsack\_sol*

```
1 function [v, valide] = knapsack_sol (s,obj)
```

Se trata de una función que comprueba si una mochila dada cumple un determinado objetivo con el algoritmo estudiado para mochilas supercrecientes. Es importante observar que aunque se sabe que el algoritmo siempre funciona para mochilas supercrecientes, pudiera también funcionar para algunos casos concretos de mochilas generales.

### Entradas:

*s*: una mochila. La función debe comprobar que lo sea, así como que sea supercreciente.

*obj*: el objetivo a alcanzar.

### Salidas:

*v*: en caso de que el objetivo se cumpla es un vector indicando los valores de la mochila que permiten obtenerlo (sea o no la mochila supercreciente). En caso contrario  $v = 0$ .

*valide*: misma salida que en la función anterior.

### Ejemplos:

```
1 >> [v, valide] = knapsack_sol ([20 7 4 1],12)
2 v = 0      1      1      1
3 valide = 0
```

```
1 >> [v, valide] = knapsack_sol ([20 5 736 13 2],35)
2 v = 0
3 valide = 0
```

```
1 >> [v, valide] = knapsack_sol ([4 10 20 47 100],53)
2 v = 0
3 valide = 1
```

```
1 >> [v, valide] = knapsack_sol ([4 10 20 47 100],71)
2 v = 1      0      1      1      0
3 valide = 1
```

## 3. Función *knapsack\_cipher*

```
1 function code = knapsack_cipher (s,text)
```

Se trata de una función que cifra un mensaje a partir de una mochila (no necesariamente supercreciente).

### Entradas:

*s*: la mochila que será nuestra clave. La función debe comprobar que es realmente una mochila (no necesariamente supercreciente).

*text*: el texto a cifrar.

**Salida:** el vector numérico que se corresponde con el mensaje cifrado.

### Ejemplos:

```
1 >> code = knapsack_cipher ([2 4 10 19 40], 'hola')
2 code = 54 40 71 31 6 6 73
```

```
1 >> code = knapsack_cipher ([2 4 10 19 40.2], 'hola')
2 Error using knapsack_cipher (line ...)
3 The first input does not represent a knapsack.
```

### 4. Función *decipher\_knapsack\_s*

```
1 function text = decipher_knapsack_s (s,code)
```

Se trata de una función que descifra un criptograma conociendo la mochila supercreciente utilizada como clave.

#### Entradas:

*s*: la mochila usada como clave que debe ser supercreciente.

*code*: el criptograma.

**Salida:** el texto llano.

### Ejemplos:

```
1 >> s = [2 5 10 23 56];
2 >> code = [71 56 91 91 10 7 61 0 15 58 25 25 0 7 86 94 71 68 25 91 66];
3
4 >> text = decipher_knapsack_s (s, code)
5 text = 'hora de comer'
```

```
1 >> s = [2 5 10 23 36];
2 >> code = [71 56 91 91 10 7 61 0 15 58 25 25 0 7 86 94 71 68 25 91 66];
3
4 >> text = decipher_knapsack_s (s, code)
5 Error using decipher_knapsack_s (line ...)
6 The first input does not represent a simple knapsack.
```

- Las siguientes funciones van encaminadas al cifrado con mochilas trampa.

### 5. Función *common\_factors*

```
1 function [factor_c, fact] = common_factors (w,s)
```

Se trata de una función que comprueba si un número *w* tiene factores primos comunes con los elementos de la mochila *s*.

NOTA: En caso de tener no será un buen número que actúe como factor entre la mochila simple y la mochila trampa.

**Entradas:**

$w$ : un número natural.

$s$ : una mochila (aunque para el cifrado deberá ser supercreciente, para implementar esta función no debe serlo necesariamente y no se comprobará).

**Salidas:**

$factor\_c$ : una salida de control de tipo lógica. Vale 0 si no hay factores comunes y 1 si los hay.

$fact$ : un vector con los números de la mochila que tienen factores comunes con  $w$ .

**Ejemplos:**

```
1      >> [factor_c, fact] = common_factors (47,[5 6 81 345 634])
2      factor_c = 0
3      fact = []
```

```
1      >> [factor_c, fact] = common_factors (48,[5 6 81 345 634])
2      factor_c = 1
3      fact = [6    81    345    634]
```

**6. Función *knapsack\_mh***

```
1      function [publ_k,priv_k] = knapsack_mh (s)
```

Se trata de una función que permite al usuario (el receptor) generar una pareja de claves pública y privada adecuada a partir de una mochila supercreciente.

**Entrada:** una mochila supercreciente.

**Salidas:**

$publ\_k$ : la mochila trampa creada a partir de la mochila supercreciente  $s$  y de los valores de  $mu$  y  $w$ . El valor  $mu$  debe introducirlo el usuario, por lo que se lo debe pedir la función y debe comprobar que sea adecuado. El valor  $w$  lo buscará la función asegurándose que tenga inverso módulo  $mu$  y que no tenga factores comunes con los elementos de  $s$ .

$priv\_k$ : un vector fila con dos elementos:  $mu$  y el inverso de  $w$  módulo  $mu$ .

**Ejemplo:**

```
1      >> [publ_k,priv_k] = knapsack_mh ([2 5 8 23 67 131])
2
3      Introduce the value mu, a natural number greater than 236:
4      531
5
6      publ_k = 523    511    499    439    263    7
7      priv_k = 531    398
```

## 7. Función *decipher\_kmh*

```
1 function text = decipher_kmh (s,code,mu,invw)
```

Se trata de una función que permite al usuario (el receptor) descifrar un mensaje cifrado con su clave pública.

### Entradas:

*s*: una mochila supercreciente.

*code*: el criptograma recibido.

*mu* y *invw*: los elementos de la clave privada, obtenidos a partir de la función anterior.

**Salida:** el texto claro.

### Ejemplo:

```
1 >> s = [2 5 8 23 67 131]
2 >> code = [1712 1213 439 523 1449 1736 1979 1724 499 702 ...
            1041 530 1449 1297 0 1297 499 702 518 969 499 702 ...
            1041 530 499 709 518 530 1449 1225 518 969]
3
4 >> text = decipher_kmh (s,code,531,398)
5 text = 'ya son las 2 de la tarde'
```

- Para finalizar vamos a implementar el criptoanálisis de Shamir y Zippel.

## 8. Función *crypt\_shamir\_zippel*

```
1 function s = crypt_shamir_zippel (publ_k,mu)
```

Se trata de una función que realiza el criptoanálisis de Shamir y Zippel. La función debe indicar el rango en el que está buscando el posible primer elemento de la mochila supercreciente.

### Entradas:

*publ\_k*: la clave pública, es decir, la mochila trampa.

*mu*: el módulo de trabajo, que en este criptoanálisis se considera conocido.

**Salida:** la mochila supercreciente asociada al cifrado.

Algunas observaciones importantes para implementar esta función:

- Cada vez que finalice un rango sin éxito se debe pedir al usuario si quiere ampliar o no el rango de búsqueda.
- La función debe mostrar el tiempo que tarda en recorrer cada uno de los intervalos y encontrar la mochila supercreciente.

- Aunque alguno de los dos primeros elementos de la mochila no tenga inverso módulo  $\mu$  la función continuará.

### Ejemplos:

```

1  >> publ_k = [106  265  583  1219  2703  1285  2782  383  1296  3903]
2  >> mu = 5234
3
4  >> s = crypt_shamir_zippel (publ_k, mu)
5  Searching multiples in the rank [1 , 2048]
6  Elapsed time is 0.008956 seconds.
7
8  We have found the simple knapsack:
9  s = 2   5   11  23  51  123  250  501  1012  2345

```

```

1  >> publ_k = [471  785  1413  3611  8007  19468  15349  5384  18165  ...
2  2824  13165]
3  >> mu = 25000
4
5  >> s = crypt_shamir_zippel (publ_k, mu)
6  Searching multiples in the rank [1, 4096]
7  Elapsed time is 0.653591 seconds.
8
9  We have not found the simple knapsack. Do you want to continue with a bigger ...
10 interval? (Y/N): Y
11
12 Searching multiples in the rank [4097, 8192]
13 Elapsed time is 0.610304 seconds.
14
15 We have not found the simple knapsack. Do you want to continue with a bigger ...
16 interval? (Y/N): Y
17
18 Searching multiples in the rank [8193, 16384]
19 Elapsed time is 0.002161 seconds.
20
21 We have found the simple knapsack:
22 s = 3   5   9  23  51  124  257  512  2345  5432  12345

```