UNIVERSIDAD DEL VALLE DE GUATEMALA

Cifrado de información

Sección 10

Ludwing Cano

Laboratorio 2

Parte A

Abner Iván García Alegría 21285

Problemas a resolver

En este laboratorio implementaremos funciones que convierten cadenas a bits y a Base 64. Además, la investigación de una propiedad estadística de la función XOR.

1. Implementar una función para convertir una cadena de caracteres a bits. Por cada carácter de la cadena encontrar la representación en bytes (8 bits) del valor ASCII de dicho carácter. La función debe de devolver la concatenación de todos los bits de la cadena.

a. Muestra 2 ejemplos sencillos de convertir cadenas a bytes

Ejemplo 1

```
Proceso de conversión para: andrea
---------------------------------------
Letra: a | ASCII: 97 | Binario: 01100001
Letra: n | ASCII: 110 | Binario: 01101110
Letra: d | ASCII: 100 | Binario: 01100100
Letra: r | ASCII: 114 | Binario: 01110010
Letra: e | ASCII: 101 | Binario: 01100101
Letra: a | ASCII: 97 | Binario: 01100001
---------------------------------------
Resultado final (concatenado): 011000010110111001100100011100100110010101100001
```

Ejemplo 2

```
Proceso de conversión para: jutiapa
---------------------------------------
Letra: j | ASCII: 106 | Binario: 01101010
Letra: u | ASCII: 117 | Binario: 01110101
Letra: t | ASCII: 116 | Binario: 01110100
Letra: i | ASCII: 105 | Binario: 01101001
Letra: a | ASCII: 97 | Binario: 01100001
Letra: p | ASCII: 112 | Binario: 01110000
Letra: a | ASCII: 97 | Binario: 01100001
---------------------------------------
Resultado final (concatenado): 011010100111010101110100011010010110000101110000001100001
```

2. Implementar una función para convertir una cadena de bytes a caracteres. Por cada grupo de 8 bits encontrar su representante correspondiente en ASCII. La función debe de devolver el texto correspondiente.

a. Muestra 2 ejemplos sencillos de convertir bytes a cadena

Ejemplo 1

```
Proceso de conversión para: 01100001011011100110010001110010011001010110000 1
----------------------------------------
Byte: 01100001 | Decimal: 97 | Letra: a
Byte: 01101110 | Decimal: 110 | Letra: n
Byte: 01100100 | Decimal: 100 | Letra: d
Byte: 01110010 | Decimal: 114 | Letra: r
Byte: 01100101 | Decimal: 101 | Letra: e
Byte: 01100001 | Decimal: 97 | Letra: a
----------------------------------------
Resultado final (concatenado): andrea
```

Ejemplo 2

```
Proceso de conversión para: 01101010011101010111010001101001011000010111000001100001
----------------------------------------
Byte: 01101010 | Decimal: 106 | Letra: j
Byte: 01110101 | Decimal: 117 | Letra: u
Byte: 01110100 | Decimal: 116 | Letra: t
Byte: 01101001 | Decimal: 105 | Letra: i
Byte: 01100001 | Decimal: 97 | Letra: a
Byte: 01110000 | Decimal: 112 | Letra: p
Byte: 01100001 | Decimal: 97 | Letra: a
----------------------------------------
Resultado final (concatenado): jutiapa
```

3. Implementar funciones que permitan convertir una cadena de caracteres a Base64, para esto utilizar la conversión manual (texto a binario, binario a codigo UNICODE).

a. Mostrar 2 ejemplos sencillos de convertir una cadena a base 64.

Ejemplo 1

```
Proceso de conversión para: vida
----------------------------------------
Letra: v | ASCII: 118 | Binario: 01110110
Letra: i | ASCII: 105 | Binario: 01101001
Letra: d | ASCII: 100 | Binario: 01100100
Letra: a | ASCII: 97 | Binario: 01100001
----------------------------------------

Proceso de conversión a Base64:
----------------------------------------
Cadena binaria original: 01110110011010010110010001100001
Agregando padding: 0000
Cadena con padding: 01110110011010010110010001100001 0000

Conversión por grupos de 6 bits:
Grupo: 011101 | Decimal: 29 | Carácter Base64: d
Grupo: 100110 | Decimal: 38 | Carácter Base64: m
Grupo: 100101 | Decimal: 37 | Carácter Base64: l
Grupo: 100100 | Decimal: 36 | Carácter Base64: k
Grupo: 011000 | Decimal: 24 | Carácter Base64: Y
Grupo: 010000 | Decimal: 16 | Carácter Base64: Q
----------------------------------------
Resultado final Base64: dmlkYQ==
```

Ejemplo 2

```
Proceso de conversión para: hola
--------------------------------------
Letra: h | ASCII: 104 | Binario: 01101000
Letra: o | ASCII: 111 | Binario: 01101111
Letra: l | ASCII: 108 | Binario: 01101100
Letra: a | ASCII: 97 | Binario: 01100001
--------------------------------------

Proceso de conversión a Base64:
--------------------------------------
Cadena binaria original: 01101000011011110110110001100001
Agregando padding: 0000
Cadena con padding: 01101000011011110110110001100001000010000

Conversión por grupos de 6 bits:
Grupo: 011010 | Decimal: 26 | Carácter Base64: a
Grupo: 000110 | Decimal: 6 | Carácter Base64: G
Grupo: 111101 | Decimal: 61 | Carácter Base64: 9
Grupo: 101100 | Decimal: 44 | Carácter Base64: s
Grupo: 011000 | Decimal: 24 | Carácter Base64: Y
Grupo: 010000 | Decimal: 16 | Carácter Base64: Q
--------------------------------------
Resultado final Base64: aG9sYQ==
```

4. Implementar funciones que permitan convertir una cadena de base 64 a su texto correspondiente para esto utilizar la conversión manual (texto UNICODE a binario , binario a Códigos ASCII).

a. Mostrar 2 ejemplos sencillos de convertir una cadena de base64 a su texto correspondiente.

Ejemplo 1

```
Proceso de conversión de Base64 a Binario:
--------------------------------------------------
Cadena Base64 original: dmlkYQ==

Conversión caracter por caracter:
Caracter: d | Índice: 29 | Binario: 011101
Caracter: m | Índice: 38 | Binario: 100110
Caracter: l | Índice: 37 | Binario: 100101
Caracter: k | Índice: 36 | Binario: 100100
Caracter: Y | Índice: 24 | Binario: 011000
Caracter: Q | Índice: 16 | Binario: 010000
Encontrado caracter de padding '=', terminando conversión
--------------------------------------------------
Cadena binaria resultante: 011101100110100101100100011000010000

Proceso de conversión de Binario a ASCII:
--------------------------------------------------
Byte: 01110110 | Decimal: 118 | Caracter: v
Byte: 01101001 | Decimal: 105 | Caracter: i
Byte: 01100100 | Decimal: 100 | Caracter: d
Byte: 01100001 | Decimal: 97 | Caracter: a
--------------------------------------------------
Texto final: vida

'vida'
```

Ejemplo 2

```
Proceso de conversión de Base64 a Binario:
--------------------------------------------------
Cadena Base64 original: aG9sYQ==

Conversión caracter por caracter:
Caracter: a | Índice: 26 | Binario: 011010
Caracter: G | Índice: 6 | Binario: 000110
Caracter: 9 | Índice: 61 | Binario: 111101
Caracter: s | Índice: 44 | Binario: 101100
Caracter: Y | Índice: 24 | Binario: 011000
Caracter: Q | Índice: 16 | Binario: 010000
Encontrado caracter de padding '=', terminando conversión
--------------------------------------------------
Cadena binaria resultante: 011010000110111101101100011000010000

Proceso de conversión de Binario a ASCII:
--------------------------------------------------
Byte: 01101000 | Decimal: 104 | Caracter: h
Byte: 01101111 | Decimal: 111 | Caracter: o
Byte: 01101100 | Decimal: 108 | Caracter: l
Byte: 01100001 | Decimal: 97 | Caracter: a
--------------------------------------------------
Texto final: hola

'hola'
```

5. Implementar una función que haga la operación XOR, bit a bit, con dos cadenas de texto.

a. Recuerde que la llave debe ser de menor o igual tamaño que la palabra

b. Si en dado caso la llave es menor complementarla para llegar al mismo tamaño

```
# Ejemplo de uso
texto = b'hola'
clave = b'si'

resultado = encriptar_con_xor(texto, clave)
```

✓ 0.0s

```
Proceso de encriptación XOR:
--------------------------------------------------
Mensaje original: b'hola'
Keystream original: b'si'
Keystream ajustado: b'sisi'

Operación XOR byte por byte:

Posición 0:
Byte mensaje: 01101000 (h)
Byte key:     01110011 (s)
Resultado:    00011011 (0x1b)

Posición 1:
Byte mensaje: 01101111 (o)
Byte key:     01101001 (i)
Resultado:    00000110 (0x6)
```

```
Posición 2:
Byte mensaje: 01101100 (l)
Byte key:     01110011 (s)
Resultado:    00011111 (0x1f)

Posición 3:
Byte mensaje: 01100001 (a)
Byte key:     01101001 (i)
Resultado:    00001000 (0x8)

--------------------------------------------------
Resultado final (bytes): b'\x1b\x06\x1f\x08'
Resultado final (hex): 1b061f08
```