

Escrito por Mariana Gallego Rodas y Leonardo Jose Amaris Dominguez
Enlace a video explicativo:

Informe escrito Arquitectura MIPS

Para comenzar con la descripción de este laboratorio es importante tener en cuenta que el código escrito para su desarrollo se escribió implementando una programación estructurada, es decir, se basa en el uso de funciones que realizan una tarea específica.

Primero se hará una descripción detallada de como está escrito es código y qué tarea desarrolla cada función y por último, se mostrarán dos pruebas de escritorio que ejemplificarán como se realizó la encriptación y desencriptación del mensaje de entrada.

Detalle del código:

En lenguaje MIPS, el código se divide en dos partes: *.data* y *.text*

.data: es donde se guarda la información global, las constantes y las cadenas de texto que se utilizarán a lo largo del programa. En el desarrollo de este laboratorio se definieron dentro de esta sección las siguientes constantes:

- **inputFile, outputFile, decodedOutputFile:** contienen la ruta a los documentos a los que tendrá acceso el programa
- **textoClaro:** un buffer con un espacio específico de 1024 bytes en donde se almacenará temporalmente el texto que el sistema leerá del archivo inputFile
- **claveCorta y claveExtendida:** dos buffers con un espacio específico de 1024 bytes cada uno, en donde se almacenará la clave corta y la clave extendida respectivamente
- **longitudTexto y longitudClaveCorta:** dos buffers que almacenaran la cantidad de caracteres que tiene el texto y la clave corta que ingresó el usuario
- **textoCifrado y textoDecifrado:** dos buffers con un espacio específico de 1024 bytes cada uno, en donde se almacenará el texto cifrado y el decifrado respectivamente
- **textoClaveCorta y textoDeErrorDocumento:** almacenan los textos que se le mostrarán al usuario para solicitarle que ingrese la clave corta o para indicarle que no se pudo abrir el documento

.data

```
inputFile: .asciiz "C:/Users/maria/OneDrive - Universidad de Antioquia/Escritorio/Arquitectura/2025 - 1/Arquitectura_Laboratorio 2/input.txt"
textoClaro: .space 1024
longitudTexto: .word 0

textoClaveCorta: .asciiz "Ingrese la palabra que desea utilizar como clave corta (no mayor a 12 caracteres):  "
claveCorta: .space 12 # Cantidad maxima de caracteres en la clave corta
longitudClaveCorta: .word 0

claveExtendida: .space 1024

outputFile: .asciiz "C:/Users/maria/OneDrive - Universidad de Antioquia/Escritorio/Arquitectura/2025 - 1/Arquitectura_Laboratorio 2/criptogram.txt"
textoCifrado: .space 1024

decodedOutputFile: .asciiz "C:/Users/maria/OneDrive - Universidad de Antioquia/Escritorio/Arquitectura/2025 - 1/Arquitectura_Laboratorio 2/decoded.txt"
textoDescifrado: .space 1024

textoDeErrorDocumento: .asciiz "No se pudo abrir el documento"
```

.text: es donde se definen todas las instrucciones que ejecutará el programa, las funciones, las condiciones, etc. Para el desarrollo de este laboratorio esta sección se organizó en tres partes con el fin de obtener un código mucho más ordenado, la descripción de estas secciones se detalla a continuación:

Parte 1 - main: tal como en los lenguajes de alto nivel, es la primera función que se ejecuta por defecto. Para un mejor entendimiento, esta sección se divide en dos partes, cifrado y descifrado, en cada una se estarán invocando las funciones que se necesitan para cumplir con el objetivo del laboratorio.

Desde la parte de cifrado se llaman las siguientes funciones:

- **leerTexto:** a la que a través de \$a1 se le pasa como argumento la dirección de espacio de memoria en donde está almacenada la ruta de acceso al documento que se quiere leer y con \$a3 se le pasa la dirección del buffer en donde debe almacenar los caracteres leídos
- **leerClaveCorta:** le pide al usuario que ingrese la clave que se utilizará para el cifrado
- **crearClaveExtendida:** se encarga de copiar la clave corta y completar con los caracteres del texto
- **cifrarTexto:** se encarga del cifrado del texto usando la técnica de cifrado Autoclave Vigenere
- **escribir:** escribe el texto cifrado o descifrado, dependiendo de cual se le pasa en el argumento \$a3, en el documento que se le especifica a través del argumento que se le pasa con \$a1

Desde la parte de descifrado, se llaman las funciones **leerTexto** y **escribir**, y también la función **descifrarTexto** que se encarga del descifrado del texto usando la técnica de decifrado Autoclave Vigenere.

Más delante de detallará el funcionamiento de todas estas funciones.

```
main:
#----- Cifrado -----#
# Leer texto de inputFile
la $a1, inputFile
la $a3, textoClaro
jal leerTexto

# Leer clave corta
jal leerClaveCorta

# Crear clave extendida
jal crearClaveExtendida

# Cifrar texto
jal cifrarTexto

# Escribir texto cifrado
la $a1, outputFile
la $a3, textoCifrado
jal escribir

#----- Decifrado -----#
# Leer texto de outputFile
la $a1, outputFile
la $a3, textoCifrado
jal leerTexto

# Decifrar texto
jal descifrarTexto

# Escribir texto descifrado
la $a1, decodedOutputFile
la $a3, textoDescifrado
jal escribir
```

Parte 2 - Declaración de funciones: contiene la definición de todas las funciones que se utilizarán a lo largo de la ejecución del programa para obtener la solución. A continuación se dará una descripción de las operaciones que realiza cada una:

- **leerTexto:** cuando se invoca esta función se le deben pasar dos argumentos, a través de los registros **\$a1** y **\$a3**, que le indicarán respectivamente, la ruta del archivo de texto que se debe leer y el buffer donde debe almacenar los caracteres leídos.

Cuando comienza la ejecución, abre el archivo que se le indicó a través del registro **\$a1** utilizando el *Syscall 13*, que le indica al programa que se abrirá un archivo, el valor cero para **\$a1** indica que el archivo se abrirá en modo de solo lectura. Al final, a **\$v0** se le asigna el valor -1 si el archivo no se pudo abrir, luego ese valor se almacena en **\$t0** y se utiliza *bltz* (Branch if less than zero) para llamar al error handler **errorHandlerDocumento** en caso de que sea -1; si **\$v0** es distinto de -1, entonces se continua con la ejecución.

El segundo paso de esta función es leer el archivo que ya se pudo abrir utilizando el *Syscall 14*, que se utiliza para indicar que se leerán datos, luego el valor de **\$t0**, que se obtuvo en el paso anterior, se le pasa al registro **\$a0**, ese es el descriptor de archivo del cual se van a leer los datos; a **\$a1** se le pasa la dirección de memoria en donde se almacenarán los datos leídos, es decir, el valor del registro **\$a3** y a **\$a2** se le pasa el número máximo de bytes que se podrán leer.

Cuando termina de leer el archivo, se mueve el contenido de **\$v0**, que contiene el total de bytes leídos, a **\$s0**, para después utilizar un *StoreWord* para almacenar ese número en el espacio de memoria que se le reservó llamado **longitudTexto**.

Al finalizar, se utiliza el *Syscall 16* para cerrar el archivo.

```

leerTexto:

#----- Abrir archivo -----#
li $v0, 13
la $a0, ($a1)
li $a1, 0
li $a2, 0
syscall
move $t0, $v0

bltz $t0, errorHandlerDocumento

#----- Leer archivo -----#
li $v0, 14
move $a0, $t0
la $a1, ($a3)
li $a2, 1024
syscall
move $s0, $v0

la $t1, longitudTexto
sw $s0, 0($t1)

#----- Cerrar archivo -----#
li $v0, 16
move $a0, $t0
syscall

jr $ra

```

- **escribir:** recibe los parámetros \$a1 y \$a3, que son la ruta de un archivo de texto y la dirección de un buffer respectivamente. Su función es escribir dentro del archivo de texto los caracteres almacenados en el buffer.

El primer paso es abrir el documento en el cual se van a escribir los caracteres, para esto utiliza el *Syscall 13*, y después se comprueba si se abrió correctamente. La descripción de estas dos líneas ya se detalló en procesos anteriores con la única diferencia de que al registro \$a1 se le asigna el valor 1 para indicar que el archivo se va a abrir en modo de escritura.

Después, se utiliza el *Syscall 15* para escribir en el archivo de texto y se mueve el valor de \$t0, que es el que se obtuvo en el Syscall 13, a \$a0; en \$a1 se carga el valor de \$a3 y en \$a2 se almacena la cantidad de caracteres que se van a escribir.

El último paso es cerrar el archivo utilizando el *Syscall 16*, y con eso termina la ejecución de esta función.

```

escribir:

#----- Abrir archivo -----#
li $v0, 13
la $a0, ($a1)
li $a1, 1          # 1 para escritura
li $a2, 0
syscall
move $t0, $v0

bltz $t0, errorHandlerDocumento

#----- Escribir texto cifrado -----#
li $v0, 15
move $a0, $t0
la $a1, ($a3)
li $a2, longitudTexto
syscall

#----- Cerrar archivo -----#
li $v0, 16
move $a0, $t0
syscall

jr $ra

```

- **leerClaveCorta:** esta función se encarga de mostrar el mensaje al usuario para pedirle que ingrese la clave que se va a usar para encriptar el mensaje leído desde el documento.

Utiliza el *Syscall 4* para mostrar el mensaje guardados en textoClaveCorta que le pide al usuario que digite la clave que quiere usar para encriptar y el Syscall 8 para leer la palabra clave.

Después llama a la función ajustarClaveCorta que se encarga de eliminar el carácter ‘salto de linea’ que se almacenó al momento del usuario presionar enter luego de digitar la clave corta. Más adelante se detallará sobre esta función.

Para realizar este paso, primero se debe ‘apartar’ un espacio en memoria para almacenar la dirección a la que se debe regresar a ejecutar después de finalizar el llamado. Esto se hace restándole -4 al \$sp y almacenando \$ra en ese espacio.

Por último, se utiliza un ciclo para contar la cantidad de caracteres que quedaron almacenados en el buffer claveCorta y esta cantidad se almacena en el espacio de memoria reservado llamado longitudClaveCorta.

```
leerClaveCorta:

    li $v0, 4
    la $a0, textoClaveCorta
    syscall

    li $v0, 8
    la $a0, claveCorta
    li $a1, 1024
    syscall

    addi $sp, $sp, -4
    sw $ra, 0($sp)

    jal ajustarClaveCorta

    lw $ra, 0($sp)
    addi $sp, $sp, 4

#----- Contar caracteres -----#
la $t1, claveCorta
li $t2, 0

contarWhile:
    lb $t3, 0($t1)

    beqz $t3, endContarWhile

    addi $t1, $t1, 1
    addi $t2, $t2, 1

    j contarWhile
endContarWhile:

la $t4, longitudClaveCorta
sw $t2, 0($t4)

jr $ra
```

- **ajustarClaveCorta:** como el usuario debe presionar enter cuando termine de digitar la clave para que la ejecución pueda continuar, esta acción se guarda como un salto de línea, o sea, como un carácter más, entonces esta función se encarga de eliminar ese salto de línea del total de caracteres que componen la clave corta.

Para hacer esto, se creó un ciclo llamado encontrarSalto, que recorre el buffer en donde está almacenada la clave corta hasta que llega al carácter nulo; cuando esto pasa, se hace un salto a eliminarSalto en donde se retrocede una posición ya que si se está en el carácter nulo, entonces el salto de línea está en la posición anterior.

Cuando ya se está en la posición del salto de línea, se carga el valor 10 en \$t2, que es el código ASCII del salto de línea; luego se comparan \$t1 que contiene el byte anterior al nulo y \$t2 y si son iguales se hace un salto a reemplazar.

En reemplazar se almacena el valor cero en la posición donde está el salto de línea y finaliza el código de la función.

```
ajustarClaveCorta:

    la $t0, claveCorta

    encontrarSalto:
        lb $t1, 0($t0)
        beqz $t1, eliminarSalto

        addi $t0, $t0, 1
        j encontrarSalto

    eliminarSalto:
        addi $t0, $t0, -1
        lb $t1, 0($t0)
        li $t2, 10

        beq $t1, $t2, reemplazar

        j terminarAjuste

    reemplazar:
        sb $zero, 0($t0)

    terminarAjuste:

    jr $ra
```

- **crearClaveExtendida:** esta función se encarga de copiar, dentro de en buffer claveExtendida, la clave corta y de rellenar el resto de espacios disponibles con caracteres del texto que se va a cifrar.

Para esto utiliza los buffers longitudTexto, longitudClaveCorta, claveCorta, textoClaro y claveExtendida.

Primero llama a la función copiarClaveCorta y luego a la función

completarClaveExtendida. El procedimiento de ambas se detallará más adelante.

```
crearClaveExtendida:

    lw $s0, longitudTexto
    lw $s1, longitudClaveCorta

    la $t3, claveCorta
    la $t4, textoClaro
    la $t5, claveExtendida

    li $t0, 0
    li $t1, 0
    li $t2, 0

    addi $sp, $sp, -4
    sw $ra, 0($sp)

    jal copiarClaveCorta

    lw $ra, 0($sp)
    addi $sp, $sp, 4

    addi $sp, $sp, -4
    sw $ra, 0($sp)

    jal completarClaveExtendida

    lw $ra, 0($sp)
    addi $sp, $sp, 4

    jr $ra
```

- **copiarClaveCorta:** esta función utiliza un ciclo para recorrer el buffer que almacena la clave corta y copiar cada carácter que encuentre en el buffer que guarda la clave extendida.

Cuando comienza el ciclo, se comparan \$t0 y \$s1, que contienen la cantidad de caracteres recorridos y la longitud de la clave corta, respectivamente. Cuando los dos registros son iguales, entonces significa que ya se recorrieron y copiaron todos los caracteres de la clave corta y se termina la función.

```
copiarClaveCorta:

    recorrerClaveCorta:

        beq $t0, $s1, terminarCopiaClave
        lb $t6, 0($t3)
        sb $t6, 0($t5)
        addi $t3, $t3, 1
        addi $t5, $t5, 1
        addi $t0, $t0, 1
        addi $t2, $t2, 1

        j recorrerClaveCorta

    terminarCopiaClave:

        jr $ra
```

- **completarClaveExtendida:** esta función se encarga de rellenar la clave extendida con caracteres del texto hasta que la longitud sea correcta.

Primero resta \$s0 y \$s1, la longitud del texto y la longitud de la clave corta respectivamente, para así saber cuántos caracteres del texto hacen falta para completar la clave extendida. El resultado de esa resta se almacena en \$t7.

Después utiliza un ciclo que al principio compara \$t1, que es un contador y \$t7 para saber si ya se tomaron los caracteres necesarios para completar. Si no es así, entonces se procede a recorrer el buffer del texto y a almacenar cada carácter encontrado en el buffer de la clave extendida. Gracias a que en la función copiarClaveCorta se almacenó en \$t5 la posición en la que está el último carácter que se almacenó en la clave extendida, entonces en esta función \$t5 se encarga de indicar a partir de donde se comienzan a almacenar los caracteres del texto.

El ciclo y la función terminan cuando \$t1 y \$t7 son iguales.

```
completarClaveExtendida:

    sub $t7, $s0, $s1

    recorrerTexto:

        beq $t1, $t7, terminarCompletado

        lb $t6, 0($t4)
        sb $t6, 0($t5)
        addi $t4, $t4, 1
        addi $t5, $t5, 1
        addi $t1, $t1, 1
        j recorrerTexto

    terminarCompletado:

        jr $ra
```

- **cifrarTexto:** esta función hace el cifrado del textoClaro el cual se encuentra dentro del registro salvado \$s0 aplicándole el algoritmo de Vigenere a partir de una clave extendida dentro del registro \$s2 la cual será almacenada dentro de registro temporal \$t6 que indica la dirección de memoria asociada al textoCifrado. Por otro lado, para esta función se aplica un acuerdo para la disposición para el uso de los registros temporal \$t:

- \$t0: contador del bucle
- \$t1: límite del bucle
- \$t2, \$t3: resultado de operaciones aritméticas

- \$t4, \$t5: almacenamiento de caracteres extraídos
- \$t6: dirección base del textoCifrado (salida)
- \$t7: constante 128

- 1) Con el uso de un bucle usando la instrucción beq para recorrer cada elemento del texto claro a partir de la comparación \$t0 = \$t1
- 2) Usando variable temporal de almacenamiento de operaciones aritméticas \$t2 se calcula la dirección de memoria donde se encuentra el carácter \$t0 dentro del textoClaro y se almacena dicho carácter dentro del registro \$t4
- 3) Se hace una instrucción similar para extraer el carácter \$t0 dentro de la claveExtendida y almacenando el carácter dentro de \$t5
- 4) Se ejecuta la operación de suma de los valores numéricos de las letras del textoClaro y claveExtendida almacenándolo dentro del registro temporal \$t3 para aplicarle la operación mod 128 y obtener el encriptado dentro del mismo registro
- 5) Usando la variable temporal de almacenamiento de operaciones aritmética \$t2 se ingresa la dirección de memoria dentro del textoCifrado que se encuentra dentro del registro \$t6 para almacenar el carácter cifrado
- 6) Se repite el ciclo y en caso de que los registros \$t0 y \$t1 sean iguales se retorna a la dirección de registro \$ra.

```

cifrarTexto:

    la $s0, textoClaro
    la $s2, claveExtendida
    la $t6, textoCifrado
    li $t0, 0
    lw $t1, longitudTexto
    li $t7, 128

recorrerTextoClaro:

    beq $t0, $t1, terminarCifrado

    add $t2, $s0, $t0
    lb $t4, 0($t2)

    add $t2, $s2, $t0
    lb $t5, 0($t2)

    add $t3, $t5, $t4
    rem $t3, $t3, $t7

    add $t2, $t6, $t0
    sb $t3, 0($t2)

    addi $t0, $t0, 1

    j recorrerTextoClaro

terminarCifrado:

    jr $ra

```

- **descifrarTexto:** esta función hace el descifrado del textoCifrado el cual se encuentra dentro del registro salvado \$s1 aplicándole el algoritmo de Vigenere a partir de una claveExtendida dentro del registro \$s2 la cual será almacenada dentro de registro temporal \$t6 que indica la dirección de memoria asociada al textoDecifrado. Por otro lado, para esta función se aplica un acuerdo para la disposición para el uso de los registros temporal \$t:

- \$t0: contador del bucle
- \$t1: límite del bucle
- \$t2, \$t3: resultado de operaciones aritméticas
- \$t4, \$t5: almacenamiento de caracteres extraídos
- \$t6: dirección base del textoDecifrado (salida)
- \$t7: constante 128

- 1) Con el uso de un bucle usando la instrucción beq para recorrer cada elemento del texto claro a partir de la comparación \$t0 = \$t1
- 2) usando variable temporal de almacenamiento de operaciones aritmética \$t2 se calcula la dirección de memoria donde se encuentra el carácter \$t0 dentro del textoDecifrado y se almacena dicho carácter dentro del registro \$t4
- 3) Se hace una instrucción similar para extraer el carácter \$t0 dentro de la claveExtendida y almacenando el carácter dentro de \$t5
- 4) Se ejecuta la operación de resta entre los valores numéricos de las letras del textoClaro y claveExtendida almacenándolo dentro del registro temporal \$t3, aplicándole la suma con el entero 128 para garantizar un resultado positivo de la resta seguido de la operación mod 128 y obtener el desencriptado dentro del mismo registro
- 5) Se carga dentro del registro \$t2 la longitudClaveCorta y sumamos la dirección \$s2 y \$t0 para encontrar la posición para ingresar los caracteres descifrados dentro de la claveExtendida y completarla para su posterior uso dentro del desencriptado
- 6) Usando la variable temporal de almacenamiento de operaciones aritmética \$t2 se ingresa la dirección de memoria dentro del textoDecifrado que se encuentra dentro del registro \$t6 para almacenar el carácter cifrado
- 7) Se repite el ciclo y en caso de que los registros \$t0 y \$t1 sean iguales se retorna a la dirección de registro \$ra.

Para controlar que al rellenar la clave extendida con los caracteres que se van descifrando solo se agreguen los suficientes para completar la longitud, se utiliza rellenarClaveExtendida.

descifrarTexto:

```
la $s1, textoCifrado
la $s2, claveExtendida
la $t6, textoDescifrado
li $t0, 0
lw $t1, longitudTexto
li $t7, 128
li $t8, 0
```

recorrerTextoCifrado:

```
beq $t0, $t1, terminarDescifrado
```

```
add $t2, $s1, $t0
lb $t4, 0($t2)
```

```
add $t2, $s2, $t0
lb $t5, 0($t2)
```

```
sub $t3, $t4, $t5
addi $t3, $t3, 128
rem $t3, $t3, $t7
```

rellenarClaveExtendida:

```
lw $t2, longitudClaveCorta
```

```
beq $t2, $t8, claveCompleta
```

```
add $t2, $s2, $t2
add $t2, $t2, $t0
sb $t3, 0($t2)
addi $t8, $t8, 1
```

claveCompleta:

```
add $t2, $t6, $t0
sb $t3, 0($t2)
```

```
addi $t0, $t0, 1
```

```
j recorrerTextoCifrado
```

terminarDescifrado:

```
jr $ra
```

- **errorHandlerDocumento:** utiliza el SYSCALL 4 para mostrarle al usuario los caracteres almacenados en el buffer textoDeErrorDocumento y luego utiliza el SYSCALL 10 para finalizar la ejecución del programa.

errorHandlerDocumento:

```
li $v0, 4
la $a0, textoDeErrorDocumento
syscall
```

```
li $v0, 10
syscall
```

```
jr $ra
```

Pruebas de escritorio Autoclave Vigenere

Cifrado:

CIFRADO									
Buffer	Registros	ITERACIONES							
		1	2	3	4	5	6	7	8
textoClaro	\$s0	0							
claveExtendida	\$s2	0							
	\$t0	0	1	2	3	4	5	6	7
longitudTexto	\$t1	8							
	\$t2	0	0	0	1	1	2	2	2
	\$t3	185	57	206	78	222	94	186	58
	\$t4	108		101	110	71	117	97	74
	\$t5	77		105	112	115	108	101	110
textoCifrado	\$t6	0							
	\$t7	128							

Texto claro							
Posición	0	1	2	3	4	5	6
ASCII	108	101	110	71	117	97	74
Byte	l	e	n	G	u	a	J

Clave extendida							
Posición	0	1	2	3	4	5	6
ASCII	77	105	112	115	108	101	110
Byte	M	i	p	s	l	e	n

Texto cifrado							
Posición	0	1	2	3	4	5	6
ASCII	57	78	94	58	97	70	56
Byte	9	N	^	:	a	F	8

Para representar la técnica de cifrado se utiliza una tabla de Excel. Cada fila de la tabla representa los registros que se utilizan en la función del algoritmo y cada columna representa cada una de las iteraciones que se realizan para cada caracter.

En la tabla se especifican los valores exactos que va tomando cada registro a medida que avanza el algoritmo, exceptuando los valores de \$s0, \$s2 y \$t6, que en Mars son direcciones de memoria y para efectos de entendimiento, se reemplazan por números a partir del cero, que supone el inicio de cada uno de los buffers.

Es importante recordar que el registro \$t2 es el que indica el buffer y la posición en la que se leen o almacenan los caracteres, entonces para una buena ilustración se utilizaron colores que junto con el número que hay en cada celda indican el buffer y la posición respectivamente.

Descifrado:

DESCIFRADO																																																					
Buffer	Registros	ITERACIONES																																																			
		1			2			3			4			5			6			7			8																														
textoCifrado	\$s1	0																																																			
claveExtendida	\$s2	0																																																			
	\$t0	0	1			2			3			4			5			6			7			8																													
longitudTexto	\$t1	8																																																			
	\$t2	0	0	4	4	4	0	1	1	4	4	5	1	2	2	4	4	6	2	3	3	4	4	7	3	4	4	4	4	5			5	5	4	5			6	6	4	6	6			7	7	4	7				
	\$t3	-20	108		108		-27	101		101		-18	110		110		-57	71		71		-11	117		117		-31	97		97		-54	74		74		-27	101		101													
	\$t4	57					78					94					58					97					70					56					44																
	\$t5	77					105					112					115					108					101					110					71																
textoDescifrado	\$t6	0																																																			
	\$t7	128																																																			
	\$t8	0	1			2			3			4																																									

Mensaje cifrado								
Posición	0	1	2	3	4	5	6	7
ASCII	57	78	94	58	97	70	56	44
Byte	9	N	^	:	a	F	8	,

Clave extendida								
Posición	0	1	2	3	4	5	6	7
ASCII	77	105	112	115	108	101	110	71
Byte	M	i	p	s	l	e	n	G

Mensaje descifrado								
Posición	0	1	2	3	4	5	6	7
ASCII	108	101	110	71	117	97	74	101
Byte	l	e	n	G	u	a	J	e

Para representar la técnica de cifrado se utiliza una tabla de Excel. Cada fila de la tabla representa los registros que se utilizan en la función del algoritmo y cada columna representa cada una de las iteraciones que se realizan para cada caracter.

En la tabla se especifican los valores exactos que va tomando cada registro a medida que avanza el algoritmo, exceptuando los valores de \$s1, \$s2 y \$t6, que en Mars son direcciones de memoria y para efectos de entendimiento, se reemplazan por números a partir del cero, que supone el inicio de cada uno de los buffers.

Exceptuando la celda en color amarillo que es el valor de la longitud de la clave corta, es importante recordar que el registro \$t2 es el que indica el buffer y la posición en la que se leen o almacenan los caracteres, entonces para una buena ilustración se utilizaron colores que junto con el número que hay en cada celda indican el buffer y la posición respectivamente.

El siguiente link es el enlace al libro de Excel en el que se realizaron estas pruebas, donde se puede evidenciar el proceso y fórmulas aplicadas:

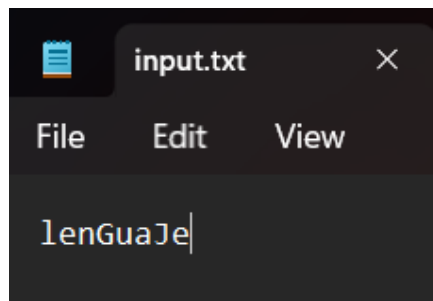
https://docs.google.com/spreadsheets/d/1WeW9vKDTNnASrm6O7Kwe_k-EaAyedS-u/edit?usp=sharing&ouid=117429951034764448322&rtpof=true&sd=true

Resultados de ejecución

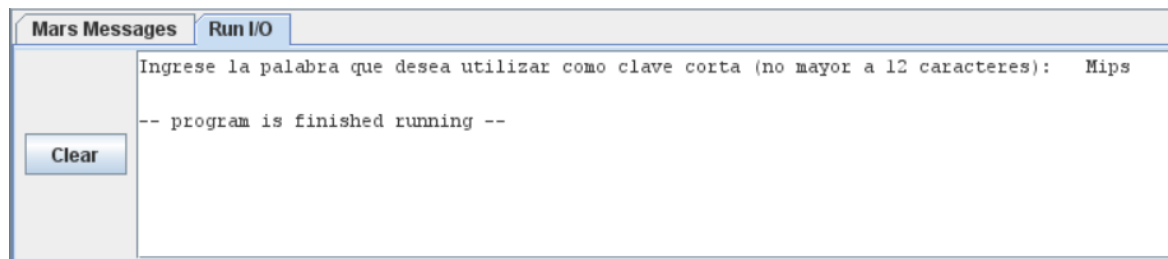
A continuación se mostrarán los resultados de ejecución del código en MIPS con dos ejemplos de texto:

Ejemplo 1:

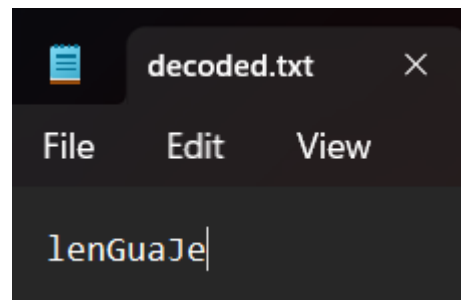
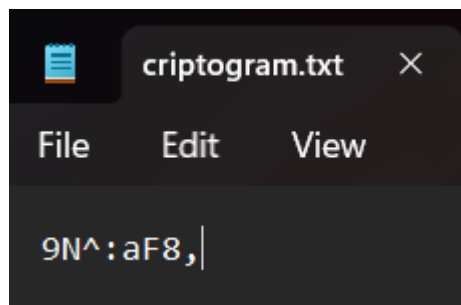
Antes de la ejecución:



Durante la ejecución:

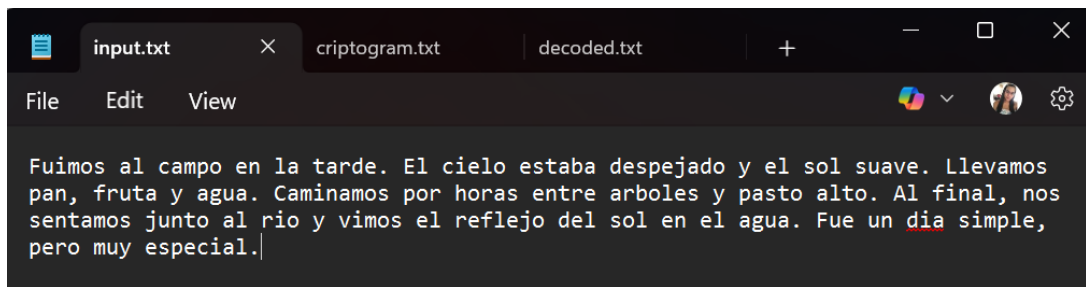


Después de la ejecución:



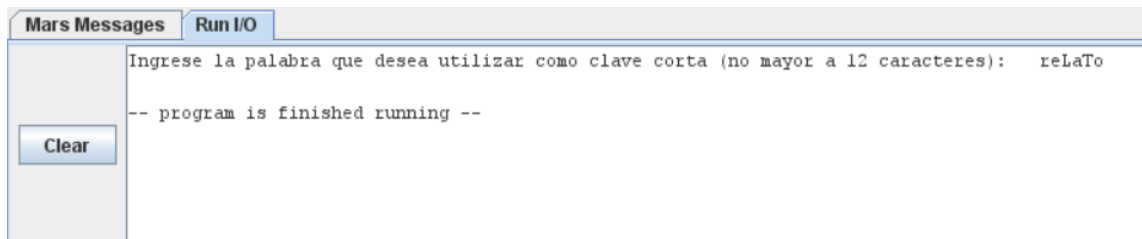
Ejemplo 2:

Antes de la ejecución:



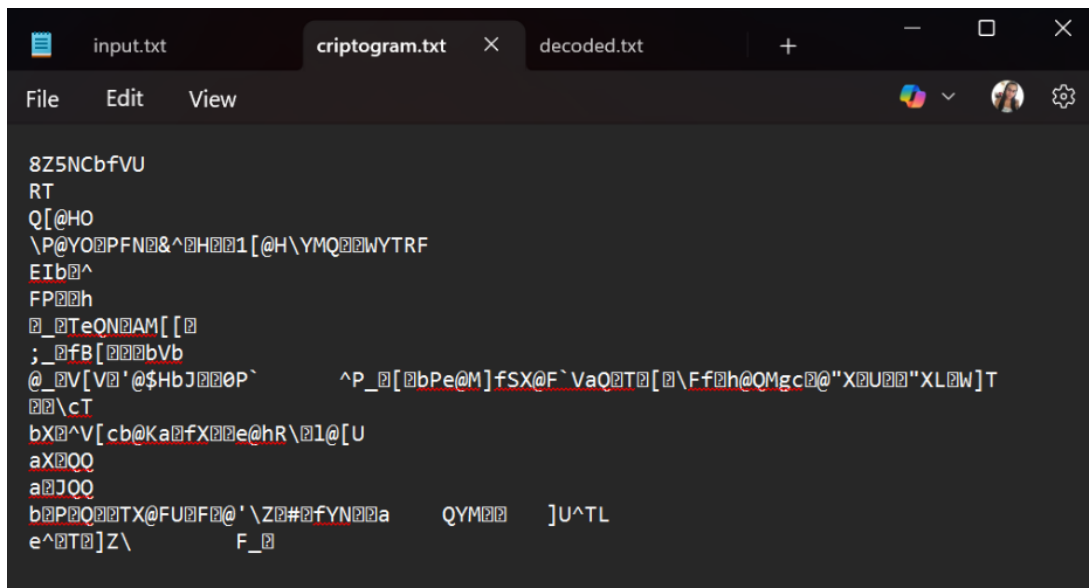
```
input.txt x criptogram.txt decoded.txt +
File Edit View
Fuimos al campo en la tarde. El cielo estaba despejado y el sol suave. Llevamos pan, fruta y agua. Caminamos por horas entre arboles y pasto alto. Al final, nos sentamos junto al rio y vimos el reflejo del sol en el agua. Fue un dia simple, pero muy especial.
```

Durante la ejecución:

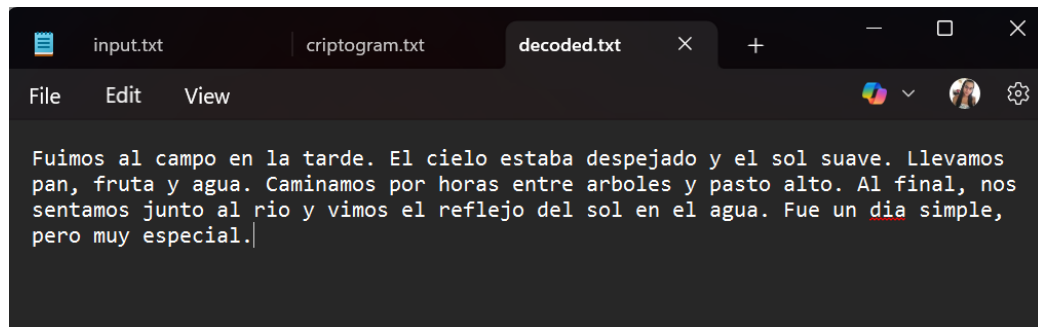


```
Mars Messages Run I/O
Ingrese la palabra que desea utilizar como clave corta (no mayor a 12 caracteres): reLaTo
-- program is finished running --
Clear
```

Después de la ejecución:



```
input.txt x criptogram.txt decoded.txt +
File Edit View
8Z5NCbfVU
RT
Q[@HO
\P@YO@PFN@&^@H@1[@H\YMQ@WYTRF
EIb^
FP@h
@_@TeQ@AM[[@
;_@fB[@@@bVb
@_@V[V@'@$HbJ@P` ^P_@[@bPe@M]fSX@F`VaQ@T@[@Ff@h@QMgc@"X@U@@"XL@W]T
@cT
bX^V[cb@Ka@fX@e@hR\@l@[U
aX@Q
a@JQ
b@P@Q@TX@FU@F@'Z#@fYN@a QYMA ]U^TL
e^@T@]Z\ F_
```



The screenshot shows a text editor window with three tabs: 'input.txt', 'criptogram.txt', and 'decoded.txt'. The 'decoded.txt' tab is active, displaying the following text: 'Fuimos al campo en la tarde. El cielo estaba despejado y el sol suave. Llevamos pan, fruta y agua. Caminamos por horas entre arboles y pasto alto. Al final, nos sentamos junto al rio y vimos el reflejo del sol en el agua. Fue un dia simple, pero muy especial.'

Observaciones

Lo primeros planteamientos del código basados en una longitud estática para la clave corta y el texto claro generaba una reducción de aproximadamente [100,200] líneas de código a ejecutar, pero con una notable reducción en la fidelidad de encriptado, debido a que se toma los primeros 128 elementos de la tabla ASCII como conjunto de los elementos a tratar dentro del encriptado podemos encontrar los elementos vacíos relacionados con elemento nulo.

Texto claro	h	o	l	a	null	null	null	null
	104	111	108	97	0	0	0	0
Clave extendida	s	1	null	null	h	o	l	a
	115	49	0	0	104	111	108	97
Encriptado	á	“”	l	a	h	o	l	a
	91	32	108	97	104	111	108	97

Por lo tanto, podemos encontrar que puede aparecer el texto claro sin encriptar dentro del encriptado, se puede resolver el problema aplicando el algoritmo para clave corta y archivo texto claro que no presenten caracteres no nulos de manera consecutivas debido a que el archivo de texto claro quedaría legible.

Por otro lado, no es necesario hacer una transformación de conjunto de elementos a encriptar para solamente interactuar con caracteres imprimibles, si bien no los caracteres no imprimibles no aparecen de manera visual dentro del archivo de texto si se encuentran como dato oculto y por lo tanto, al momento de extraer los elementos dentro del texto encriptado aparecerá como elemento dentro de la tabla ASCII y como resultado, se le es permitido una operación descripción.

Conclusiones

Sacrificando simpleza del algoritmo podemos obtener una alta fidelidad del encriptado y desencriptado del algoritmo permitiendo la encriptación para cualquier longitud de clave corta y texto claro no mayor a las impuestas dentro del código como espacio solicitado dentro de la memoria, además, la implementación de un conjunto diverso de caracteres imprimibles y no imprimibles nos permite obtener un encriptado menos legible para el ojo humano conservado un proceso de reversibilidad de encriptación al mismo tiempo de obtener una gran reducción de operaciones de transformación de conjunto de los primero 128 elementos de la tabla ASCII a un subconjunto de caracteres imprimibles y obligatorios dentro de un archivo de texto.