



**Universidad Popular Autónoma del Estado de Puebla**

**PREPA UPAEP CHOLULA**

**Desarrollo de un código para la protección de datos en dispositivos  
electrónicos**

**Ramiro Rafael Fomperoza Guerrero (Profesor)**

**Gabriel Omar Garcia Salazar**

**Leslie Martínez Bello**

**Lorette Mora Hernandez**

**Miranda Garcia Ugalde**

**Electro Tlalli 2025**

**05/03/2025**

## **Metodología**

### **Introducción**

Este proyecto presenta una propuesta para el desarrollo de un semiconductor dentro del marco del programa Electro Tlalli, que tiene como objetivo capacitar a estudiantes en el diseño de circuitos integrados y semiconductores, fortaleciendo el talento de los participantes.

En este contexto, se propone la implementación de un cifrado César mediante un sumador binario, desarrollado en el lenguaje Python. Este cifrado es un método clásico de sustitución en el que cada letra del texto original es reemplazada por otra situada un número fijo de posiciones más adelante en el alfabeto.

Para este proyecto se implementó un código básico, por lo que su implementación radica en el uso de un sumador binario en operaciones lógicas básicas (AND, OR Y XOR), para realizar el desplazamiento de caracteres.

Este proyecto representa un ejercicio práctico dentro de la iniciativa de Electro Tlalli, proporcionando una base para futuras aplicaciones en el diseño de hardware de cifrado y seguridad digital. En las siguientes secciones se detalla la estructura del código, su funcionamiento, limitaciones y posibles mejoras futuras.

### **Etapas 1: Investigación**

Para la primera etapa del desarrollo del diseño, se tuvieron que analizar problemáticas que se pudieran resolver solamente a través del uso de señales digitales, ya que el uso de señales analógicas requiere el uso de sensores y convertidores de señal tipo ADC, los cuales implicarán el uso de equipo extra.

Es por ello que se decidió trabajar en un producto que optimice la ciberseguridad y protección de la información, pues tan sólo entre los años 2021 y 2022, los delitos cibernéticos aumentaron hasta en un 600%, y se espera que esta cifra se duplique para el año presente, 2025, causando una pérdida total anual de hasta 10.5 billones de dólares (Norman, 2025).

## **Etapas 2: Desarrollo del código**

El objetivo del proyecto es crear un microchip para el encriptado de datos a través del cifrado Caesar, el cual está entre los sistemas de cifrado más utilizados y consiste en recorrer las letras que conforman un texto un número específico de lugares de acuerdo a su posición en el abecedario. El código se desarrolló en Python, por lo que puede ser utilizado en WokWi en los simuladores de MicroPython.

### **Sumador**

El primer paso fue la creación de un sumador, el cuál opera utilizando las compuertas lógicas AND, OR y XOR. Primero, se definió cada uno de los valores de entrada (a y b) con cada una de las compuertas lógicas.

```
def AND(a, b):  
    return a & b
```

```
def OR(a, b):
```

```
return a | b
```

```
def XOR(a, b):
```

```
    return a ^ b
```

Después, se definió la función de suma para 1 bit, lo que establece el modo de operación de cada compuerta lógica dentro de la suma.

```
def sumador_completo_1_bit(a, b, cin):
```

```
    suma = XOR(XOR(a, b), cin)
```

```
    carry_out = OR(OR(AND(a, b), AND(b, cin)), AND(a, cin))
```

```
    return suma, carry_out
```

En este fragmento, se definen las entradas a, b y cin. Esta última es un comando utilizado en el lenguaje de programación C++ para permitir el flujo de datos y la introducción de caracteres desde el teclado. La compuerta XOR se utiliza para definir la suma, pues los datos que pasan por esta compuerta serán verdaderos solamente si son diferentes. Posteriormente, se define la función carry\_out, mediante la cual se utilizará OR como la operación aritmética de suma (+), en donde al menos una entrada o ambas tienen que ser igual a 1 en código binario para obtener una salida verdadera. También, se utiliza la compuerta AND al definir cada entrada, la cual se utiliza como la operación aritmética de multiplicación (\*) y sólo será verdadera cuando todos los valores de entrada lo sean (iguales a 1 en código binario).

Luego, en el siguiente fragmento del código se define el sumador para 8 bits, la magnitud adecuada para el proyecto:

```
def sumador_8_bits(a, b):
```

```
    a = [int(x) for x in f"{a:08b}"]
```

```
    b = [int(x) for x in f"{b:08b}"]
```

Finalmente, en el último fragmento del sumador se definen por el usuario las variables “a” y “b”. A su vez, se organizan las funciones definidas previamente para el correcto funcionamiento del código, añadiendo el rango necesario para la cantidad de bits que se usan.

```
    for i in range(7, -1, -1):
```

```
        s, carry_in = sumador_completo_1_bit(a[i], b[i], carry_in)
```

```
        suma = [s] + suma
```

```
    carry_out = carry_in
```

```
    return suma, carry_out
```

```
a = 1
```

```
b = 3
```

```
suma, carry_out = sumador_8_bits(a, b)
```

```
suma_binaria = "".join(map(str, suma))
```

```
suma_decimal = int(suma_binaria, 2)
```

```
print(f'Suma de {a} y {b} es: {suma_decimal}')
```

## Adición del Cifrado Caesar

Para añadir el Cifrado Caesar en el código, se utilizó el código ASCII, el cual asigna valores numéricos a letras, símbolos e incluso a otros números. Para convertir a este código, se utilizó el comando **ord**. Después, dentro del mismo fragmento de código, se utiliza el comando **int**, el cual revierte los valores literales convirtiéndolos en valores numéricos. Al tener estos nuevos comandos, se asignan funciones nuevas a las ya existentes.

```
def cifrado_cesar(texto, clave):
```

```
    texto_cifrado = []
```

```
    for char in texto:
```

```
        # Convertir el carácter a su código ASCII
```

```
        codigo_ascii = ord(char)
```

```
        suma, carry_out = sumador_8_bits(codigo_ascii, clave)
```

```
        codigo_cifrado = int("".join(map(str, suma)), 2)
```

Luego, se agrega la condición que permite que el cifrado vuelva a empezar en “a” si la suma hace que exceda “z”, esto en caso de que el usuario haga una suma mayor a 26 o la posición

de las letras en la frase ingresada sea cercana a “z”. Se utiliza la condición **if** y el valor previamente asignado, el cual, si es mayor a 122 (“z” en código ASCII), hará que `codigo_cifrado` vuelva a empezar en 97 (“a” en código ASCII) y que se le reste el excedente, al sustraer 123. En este fragmento, también se agrega el comando **chr**, el que, después de que se hayan realizado los procesos anteriores, permitirá valores literales de salida.

```
if codigo_cifrado > 122:
    codigo_cifrado = 97 + (codigo_cifrado - 123) # Esto hace que vuelva a empezar en 'a'
    si pasa de 'z'

texto_cifrado.append(chr(codigo_cifrado))

return ".join(texto_cifrado)
```

Finalmente, el usuario decide el número de posiciones en el abecedario que quiere que sean recorridas las letras de la frase escrita, esto a través de la nueva variable “clave”, la cual será definida por el valor previo de `suma_decimal`, primer resultado dado por el sumador.

Después, el usuario asignará el valor de “texto\_original” ingresando los datos que desee. Por último, se define la variable `texto_cifrado` con el cifrado César que ya había sido asignado, el cual utilizará el texto original y la clave obtenida de la suma. Al final, se mostrará al usuario el mensaje encriptado.

```
clave = suma_decimal
texto_original = "escribir texto"

texto_cifrado = cifrado_cesar(texto_original, clave)
```

```
print(f"Texto Cifrado: {texto_cifrado}")
```

### **Etapla 3: Considerar las aplicaciones de la encriptación de datos**

Las aplicaciones de los sistemas de encriptado digital son extensas, algunas áreas por las que se destacan son:

- **Seguridad y comunicaciones:** El cifrado de datos es responsable de la protección de datos electrónicos, los cuales son propensos a los ciberataques, al encriptarse, se agrega un filtro con una clave de acceso única, la cual al ser aprobada de valida, concede el acceso únicamente a los usuarios pertinentes.
- **Finanzas:** Las tarjetas de débito y crédito son grandes ejemplos de la aplicación de encriptado en áreas relacionadas a las finanzas, se crea una barrera donde se aseguran las transacciones seguras y se protege la información bancaria del usuario.
- **Tecnología de computación:** La protección del *hardware* y *software* es fundamental para evitar el acceso de usuarios no deseados.
- **Medicina y Ciencia:** El uso en áreas médicas se centraliza en la digitalización de datos, lo que favorece a un acceso a historiales clínicos de manera eficiente sin la necesidad de contar con un documentos físico, lo cual ayuda a fomentar una red de salud a lo largo de áreas más grandes, al encriptar los datos se protege el historial clínico de los pacientes.

Debido a las propias limitaciones que existen por la simplicidad del cifrado Cesar, hemos optado por el uso con fines educativos, con los que buscamos el ser capaces de capacitar a futuros estudiantes de una forma didáctica, comprendiendo las implicaciones de la programación en proyectos a gran escala.



#### **Etapla 4: Resultados y conclusiones**

El código fue simulado mediante el sitio Online Python Beta:

<https://www.online-python.com/>

En el ejemplo a continuación, se observa el uso del sumador, en donde la entrada “a” está definida como 1 y la entrada “b” está definida como 15. En este caso, se decidió que cada letra en la palabra escrita recorriera 15 lugares, tomando la posición 16 con respecto a su ubicación actual.



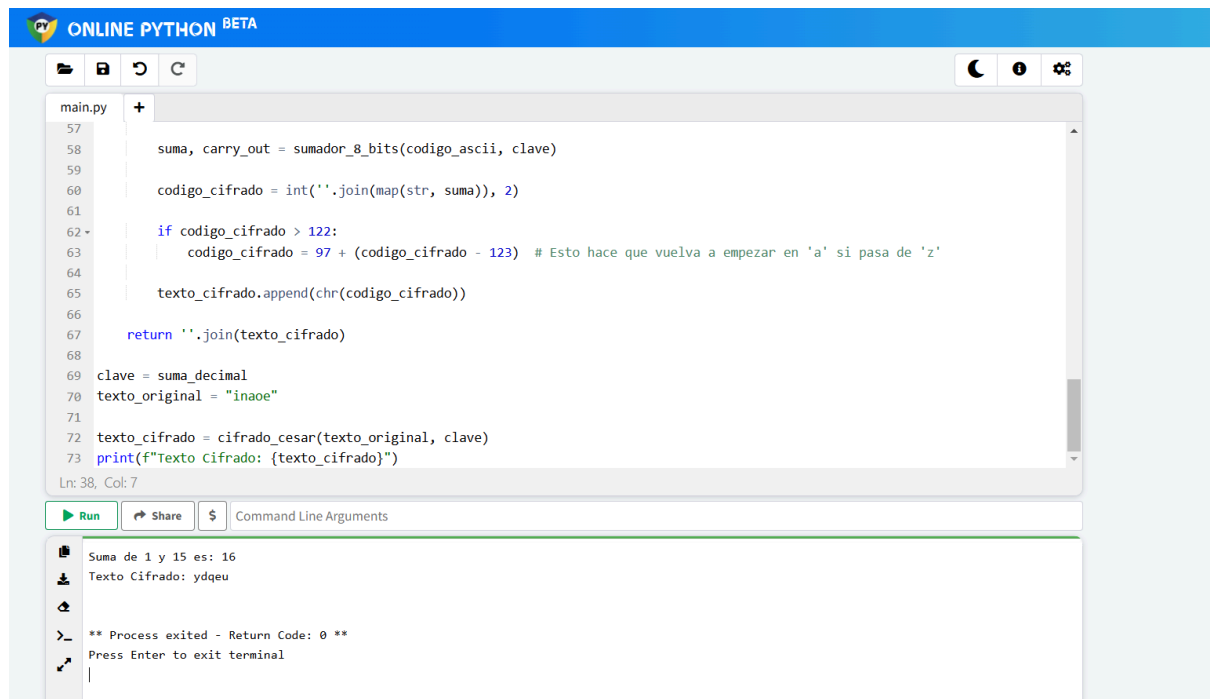
```
main.py +
41
42     carry_out = carry_in
43
44     return suma, carry_out
45
46
47
48
49 a = 1
50 b = 15
51
52
53 suma, carry_out = sumador_8_bits(a, b)
54
55
56
57
Ln: 92, Col: 24

Run Share $ Command Line Arguments

Suma de 1 y 15 es: 16
Texto Cifrado: ydqueu

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Figura 1: Simulación. Introducción de valores “a” y “b” en el sumador.



```
main.py +
57
58 suma, carry_out = sumador_8_bits(codigo_ascii, clave)
59
60 codigo_cifrado = int(''.join(map(str, suma)), 2)
61
62 if codigo_cifrado > 122:
63     codigo_cifrado = 97 + (codigo_cifrado - 123) # Esto hace que vuelva a empezar en 'a' si pasa de 'z'
64
65 texto_cifrado.append(chr(codigo_cifrado))
66
67 return ''.join(texto_cifrado)
68
69 clave = suma_decimal
70 texto_original = "inaoe"
71
72 texto_cifrado = cifrado_cesar(texto_original, clave)
73 print(f"Texto Cifrado: {texto_cifrado}")
Ln: 38, Col: 7

Run Share $ Command Line Arguments

Suma de 1 y 15 es: 16
Texto Cifrado: ydqeu

** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Figura 2: Simulación. Introducción de frase para cifrar en código César.

En base a los resultados obtenidos de las pruebas realizadas en diferentes simuladores, se puede concluir que el programa es funcional, sin embargo tiene oportunidades de mejora, destacando entre ellas la simplicidad del cifrado, aunque en la base el cifrado César es óptimo para la aplicación de este código, se considera importante implementar un cifrado más completo y así conseguir una mayor seguridad, a pesar de esto el cifrado seleccionado cumple correctamente las funciones iniciales del código y brinda una introducción sensata a esta técnica.

El código demostró ser eficaz para cifrar caracteres simples como las minúsculas, no obstante no es aplicable para mayúsculas u otro tipo de caracteres, es por eso que en un futuro se puede considerar agregar estos mismos, igualmente implementar una simplificación del código y conseguir líneas más simples

## **Consideraciones**

Para el funcionamiento adecuado del microchip se tiene que considerar lo siguiente:

- Gracias a que el microchip está diseñado para ser utilizado tanto en celulares como computadoras y tablets se necesitará una tarjeta madre (placa base) para conectarla a los demás componentes del dispositivo.
- Por la simplicidad que el código posee actualmente este solo podrá ser usado en aparatos simples los cuales fueron mencionados anteriormente y son: Computadoras, celulares y tablets.
- Finalmente hay que considerar que el cifrado César es un cifrado de sustitución simple por lo que podría llegar a ser fácil de romper con análisis de frecuencia y por ende un poco inseguro.

### **Código final**

```
def AND(a, b):
```

```
    return a & b
```

```
def OR(a, b):
```

```
    return a | b
```

```
def XOR(a, b):
```

```
    return a ^ b
```

```
def sumador_completo_1_bit(a, b, cin):
```

```
    suma = XOR(XOR(a, b), cin)
```

```
    carry_out = OR(OR(AND(a, b), AND(b, cin)), AND(a, cin))
```

```
    return suma, carry_out
```

```
def sumador_8_bits(a, b):

    a = [int(x) for x in f"{a:08b}"]
    b = [int(x) for x in f"{b:08b}"]

    for i in range(7, -1, -1):

        s, carry_in = sumador_completo_1_bit(a[i], b[i], carry_in)

        suma = [s] + suma

    carry_out = carry_in

    return suma, carry_out

a = 1
b = 3

suma, carry_out = sumador_8_bits(a, b)

suma_binaria = "".join(map(str, suma))

suma_decimal = int(suma_binaria, 2)

print(f'Suma de {a} y {b} es: {suma_decimal}')
```

```
def cifrado_cesar(texto, clave):

    texto_cifrado = []

    for char in texto:

        # Convertir el carácter a su código ASCII

        codigo_ascii = ord(char)

        suma, carry_out = sumador_8_bits(codigo_ascii, clave)

        codigo_cifrado = int("".join(map(str, suma)), 2)

        if codigo_cifrado > 122:

            codigo_cifrado = 97 + (codigo_cifrado - 123) # Esto hace que vuelva a empezar en 'a'
            # si pasa de 'z'

        texto_cifrado.append(chr(codigo_cifrado))

    return "".join(texto_cifrado)

clave = suma_decimal
texto_original = "escribir texto"

texto_cifrado = cifrado_cesar(texto_original, clave)
print(f"Texto Cifrado: {texto_cifrado}")
```

## **Referencias**

- Norman G. (2025) *Estadísticas de ciberseguridad que debes conocer* Prey Project  
<https://preyproject.com/es/blog/estadisticas-seguridad-informatica>
- Kinsta. (2025). *¿Qué es la Encriptación de Datos? Definición, Tipos y Buenas Prácticas* - Kinsta®. Kinsta®.  
<https://kinsta.com/es/base-de-conocimiento/que-es-la-enciptacion/>