1 Chacha20

Para hacer esta práctica he escrito el código necesario para aplicar el chacha20 al texto y con la clave que prefiera, y después recopilar datos sobre los cambios de los bits. En este caso, el texto y la clave serán todos bytes 0x00 para poder observar los cambios más claramente.

El código se divide en 3 programas:

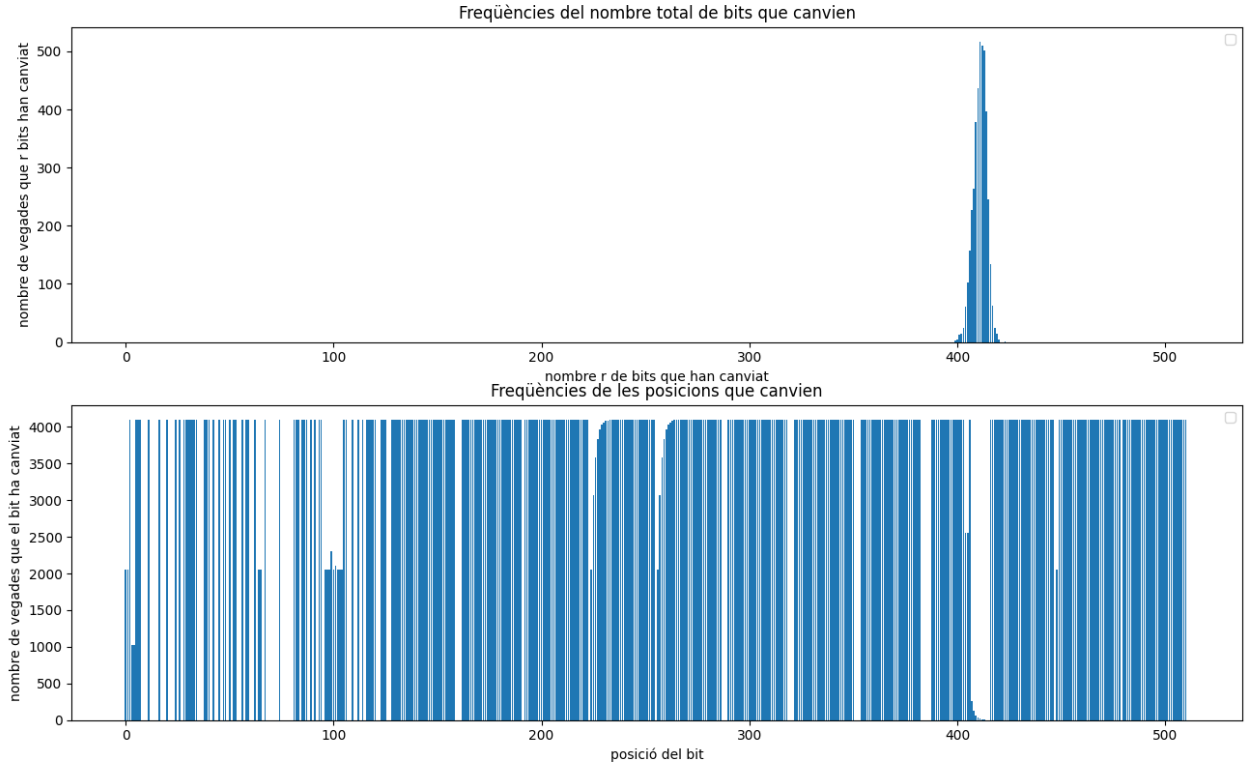
- **quarterround.py** define todas las operaciones necesarias para aplicar cada quarterround según la metodología del chacha20
- **main.py** es el archivo principal que se debe ejecutar desde la consola. Aquí se encuentra todo el procedimiento de crear los estados, aplicar los quarterrounds, analizar los cambios de cada bit y finalmente mostrarlos por pantalla.
- **progress.py** es un código simple utilizado para mostrar una barra de carga en la terminal para indicar el porcentaje de ejecución.

Dependencias del código:

- **matplotlib**, utilizado para mostrar las gráficas de la información analizada.
- **colored**, utilizado simplemente para mostrar la barra de carga con color.

1.1 Propagación de pequeños cambios

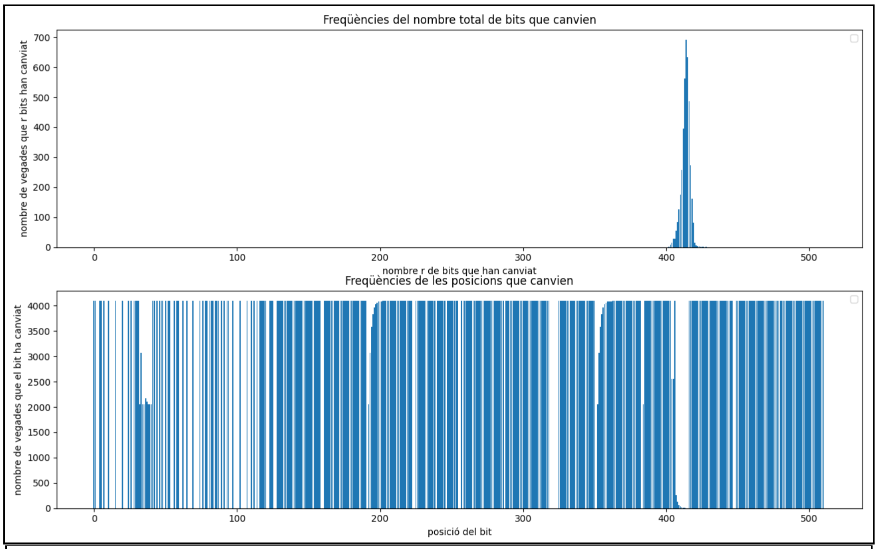
Utilizando una clave y un texto de valor 0x00 y aplicando el chacha20 4096 veces, podemos observar estos cambios en los bits del resultado:



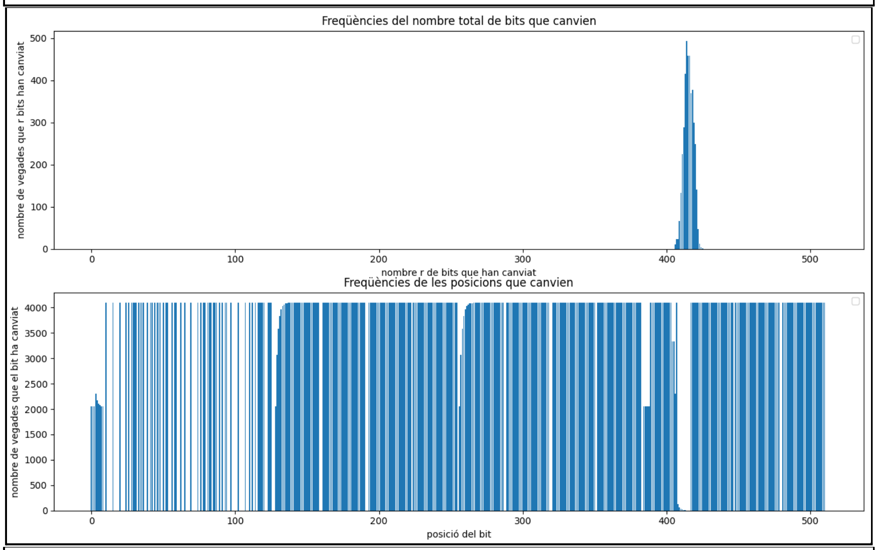
1.2 Efectos de las funciones elementales

Ejecutando sólo algunos de los quarterrounds utilizados antes, podemos observar cómo estos cambios en la ejecución del cifrado afectan al resultado y hacen que los bits finales muestren distintos cambios.

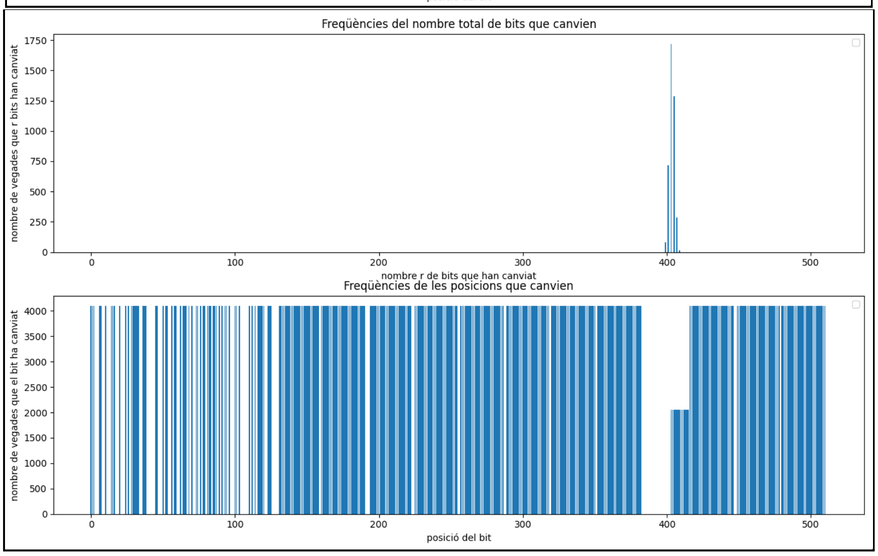
```
QUARTERROUND(state, 0, 4, 8, 12)
QUARTERROUND(state, 1, 5, 9, 13)
QUARTERROUND(state, 2, 6, 10, 14)
QUARTERROUND(state, 3, 7, 11, 15)
QUARTERROUND(state, 0, 5, 10, 15)
QUARTERROUND(state, 1, 6, 11, 12)
QUARTERROUND(state, 2, 7, 8, 13)
QUARTERROUND(state, 3, 4, 9, 14)
```




```
QUARTERROUND(state, 0, 4, 8, 12)
QUARTERROUND(state, 1, 5, 9, 13)
QUARTERROUND(state, 2, 6, 10, 14)
QUARTERROUND(state, 3, 7, 11, 15)
QUARTERROUND(state, 0, 5, 10, 15)
QUARTERROUND(state, 1, 6, 11, 12)
QUARTERROUND(state, 2, 7, 8, 13)
QUARTERROUND(state, 3, 4, 9, 14)
```




```
QUARTERROUND(state, 0, 4, 8, 12)
QUARTERROUND(state, 1, 5, 9, 13)
QUARTERROUND(state, 2, 6, 10, 14)
QUARTERROUND(state, 3, 7, 11, 15)
QUARTERROUND(state, 0, 5, 10, 15)
QUARTERROUND(state, 1, 6, 11, 12)
QUARTERROUND(state, 2, 7, 8, 13)
QUARTERROUND(state, 3, 4, 9, 14)
```



2 El cuerpo finito

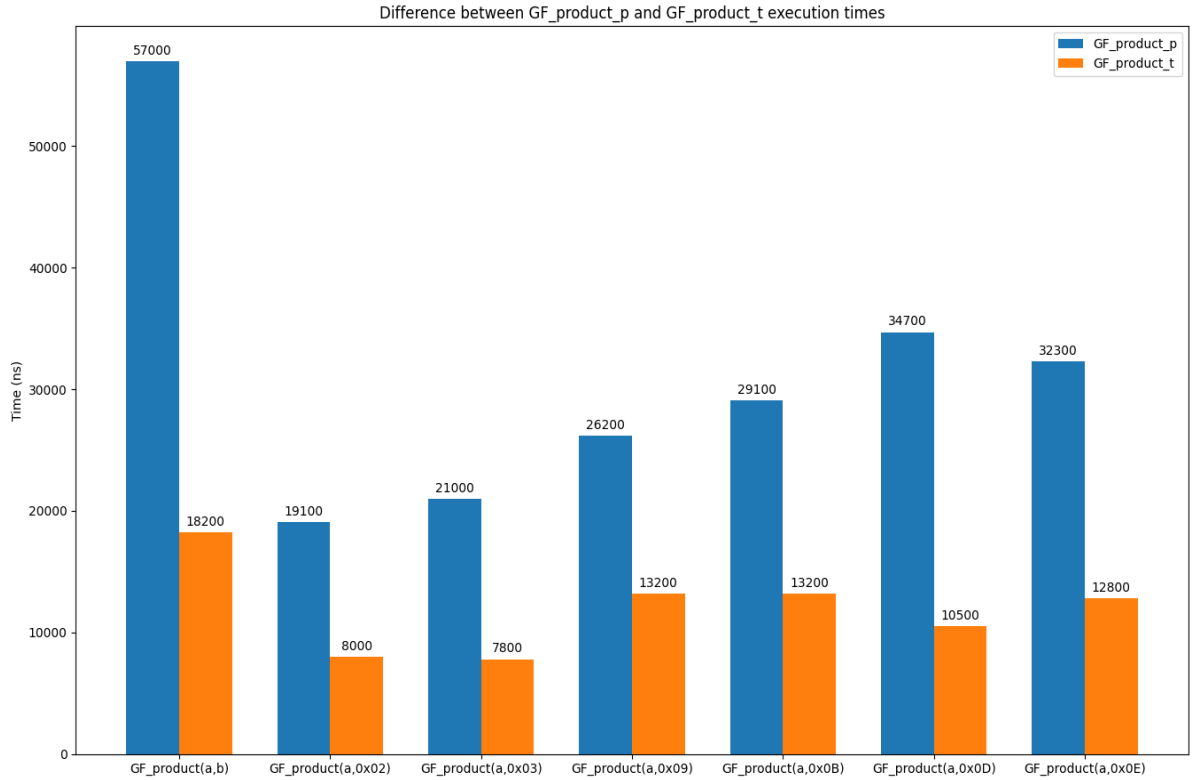
El código de ejercicio está dividido en dos archivos:

- **polynomial.py** define la clase Polynomial, que he creado para poder realizar operaciones entre polinomios, definiéndolos a partir de su valor decimal o binario. Aquí también están definidas todas las operaciones y funciones requeridas para el ejercicio: `GF_product_p(a, b)`, `GF_es_generador(a)`, `GF_tables()`, `GF_product_t(a,b)` y `GF_invers(a)`, donde *a* y *b* deben ser objetos de tipo Polynomial.
- **main.py** es el programa ejecutable. En él se encuentran todas las operaciones para poder obtener las tablas comparativas.

Dependencias del código:

- **matplotlib** se utiliza para generar la tabla.
- **numpy** también se utiliza para organizar la información obtenida con tal de generar la tabla.

Una vez ejecutado el programa, podemos observar que los cálculos utilizando la multiplicación directa de polinomios es mucho más costosa que utilizando una tabla generada anteriormente. Los valores del tiempo de ejecución puede variar entre dispositivos debido a la potencia computacional.



3 Criptografía de clave secreta

Fichero 1

Para el primer apartado tenemos un fichero codificado y su clave de decodificación. Para utilizar este programa, deberemos renombrar el fichero codificado como **fichero1.enc**, y su clave como **clave1.key**. Entonces, una vez ejecutado el programa **descifrar-fichero1.py**, este nos mostrará qué método de decodificación se ha utilizado para obetenr el archivo original, y guardará este archivo con el nombre de **fichero1** y su extensión original.

Para el segundo apartado, el fichero codificado debe ser renombrado a **fichero2.enc**. Entonces, tras ejecutar **descifrar-fichero2.py** el programa nos mostrará con qué clave ha logrado decodificar el fichero y guardará el archivo decodificado con el nombre de **fichero2** y su extensión original.

Dependencias:

- **pycryptodome** ofrece las funciones necesarias para codificar y decodificar con AES y utilizando el método que prefiramos.
- **filetype** permite conocer la extensión original de un fichero.
- **colored**, utilizado simplemente para mostrar la barra de carga con color.