

Aplicación del Álgebra Lineal: Ofuscación binaria

Universidad Panamericana
Facultad de Ingeniería



Álgebra Lineal
Prof. Miguel Angel Quintero Zamarron

Díaz Barriga Rodrigo Peña		0249221
Osornio López Daniel Alejandro		0244685
Paredes García Ricardo		0241528

Índice

1. Portada	1
2. Objetivos	3
3. Resumen	3
4. Introducción	4
5. Marco Teórico	5
5.1. Transformaciones	5
5.1.1. Transformaciones lineales	5
5.1.2. Matriz asociada	5
5.1.3. Matriz inversa	5
5.2. Computadoras	5
5.2.1. Bit	5
5.2.2. Byte	5
5.2.3. CPU	6
5.2.4. Como funciona un archivo	6
5.2.5. Álgebra modular	7
5.2.6. Módulo	7
6. Aplicación Posible	8
7. Experimentación	9
7.1. Método 1	9
7.1.1. Cifrado	9
7.1.2. Descifrado	14
7.2. Método 2	15
8. Resultados	16
9. Discusión de Resultados	17
9.1. Método 1	17
9.2. Método 2	17
10. Conclusiones y Recomendaciones	18
11. Bibliografía	19
A. Appendix	20
B. Appendix	22
C. Appendix	23
D. Appendix	25

2. Objetivos

1. Diseño de un algoritmo de cifrado que utilice como herramienta principal el álgebra lineal
2. Creación de un ejecutable de ofuscación de archivos
 - 2.1. Implementación de un script de cifrado de archivos con Mathematica
 - 2.2. Empleo de librería para la ofuscación de archivos binarios y de texto
 - 2.3. Verificación de ofuscación con herramientas de ingeniería inversa

3. Resumen

El objetivo de este proyecto es utilizar transformaciones lineales como llaves para cifrar y descifrar bits de información. Se explora el uso de la aritmética modular para mantener la congruencia de la información a la vez que se experimenta con distintos métodos de cifrado por medio de algoritmos varios que emplean las transformaciones en su proceso. La meta es implementar una herramienta de cifrado y descifrado que pueda proteger tanto archivos de texto como ejecutables con un procedimiento que permita su auto descifrado junto con soporte a *passphrases* para obtener mayor seguridad.

Los métodos estudiados son dos.

El primero es una modificación de un cifrado de sustitución. Primero se construye una matriz que contiene todos los valores representables en un byte y modifica la matriz por medio de multiplicaciones matriciales para después sustituir cada byte en el archivo por su nuevo valor.

El segundo, ordena todos los bytes del mensaje a cifrar, como puede ser un archivo, y le aplica una multiplicación matricial a la totalidad del archivo.

Para dar soporte al auto-descifrado de los archivos, cuando se utilice la bandera –auto, se almacenan en el archivo los valores para descifrarse a sí mismo, con bytes de meta información que facilitan el proceso y permiten una mayor seguridad lograda por el factor aleatorio. Este proyecto plantea la implementación de una librería de cifrado de archivos por medio de una ofuscación binaria lograda por transformaciones.

4. Introducción

Como estudiantes de la carrera de Inteligencia de Datos y Ciberseguridad, decidimos desarrollar el proyecto sobre un tema que permite darnos una noción del funcionamiento de las computadoras al desarrollar un sistema que se encargue de cifrar archivos de todo tipo, modificando los bytes del mismo por los que una transformación lineal dicte.

Antes de empezar a programar nada, desarrollaremos el algoritmo de cifrado que emplearemos; describiremos los conceptos necesarios para llegar a dicho procedimiento y los patrones que contiene para analizar si es posible su cifrado de forma sencilla.

Una vez tengamos listo el algoritmo, implementaremos un script de uso general para el cifrado de arreglos de bytes utilizando el lenguaje de programación Rust

Decidimos emplearlo debido a la facilidad que tiene en su sintaxis sumada con las operaciones de bajo nivel que permite realizar con la computadora sin necesidad de producir código complejo como pasaría con C/C++.

Verificaremos que la librería es estable y arroja como resultado el archivo exacto que se tenía antes de ser cifrado. Una ventaja de analizarlos como arreglos de bytes, es que da igual el *encoding* con el que cuente el mismo, por lo que la librería es amigable con ejecutables, imágenes, etc.

La librería contará con el soporte para cifrar ejecutables, ofreciendo una manera de que se autodescifren sin exponer de manera explícita la manera en que lo hacen.

Por último, usaremos la librería para programar una aplicación de la línea de comandos que cifra archivos de todo tipo.

Este es un tema altamente experimental, por lo que se espera encontrar limitaciones en el camino.

5. Marco Teórico

5.1. Transformaciones

En las matemáticas, una transformación puede ser toda función que mapea un conjunto X en otro conjunto o sobre sí mismo.

En algunos casos el conjunto X posee alguna estructura algebraica o geométrica y el término de “transformación” se esta refiriendo a una función X que va a conserva dicha estructura.

Algunos ejemplos de transformaciones geométricas pueden ser transformaciones lineales, transformaciones afines, rotaciones, reflexiones o traslaciones. Estas sueles realizar en un espacio euclidian, en donde la mayoría de las veces suelen ser en \mathbb{R}^2 (dos dimensiones) y \mathbb{R}^3 (tres dimensiones).

Son operaciones en donde se pone en práctica la álgebra lineal y ser descritas de manera explícita utilizando matrices.

5.1.1. Transformaciones lineales

Son funciones en donde se usa la álgebra lineal, y el objetivo que tiene una transformación lineal es el poder transformar de un espacio a otro. Para que se pueda considerar como transformación lineal debe de cumplir con dos condiciones:

1. $T(v + w) = T(v) + T(w)$
2. Multiplicación de un escalar por un vector: $T(cv) = c \times T(v)$

5.1.2. Matriz asociada

Para toda transformación lineal va a existir una matriz asociada de tal manera que para $T(v) = Av$ para toda v en V . Lo que indica el teorema es que cada vez que se tome una base diferente para los espacios vectoriales de una transformación lineal se va a tener una representación matricial diferente.

5.1.3. Matriz inversa

El producto de una matriz por su inversa va a ser igual a la matriz identidad. Se puede calcular la inversa por el método de Guass, pero al que tener en cuenta que para poderlo calcular debe de ser cuadrada la matriz, ya que si no lo hace no se va a poder calcular.

5.2. Computadoras

5.2.1. Bit

Todo en las computadoras es representado por los valores 0 y 1 debido a que funcionan con el estado o no de un componente, con la presencia o no de energía.

A una unidad que puede valer verdadero (1) o falso (0) se le conoce como bit. Los bits son la unidad más pequeña de información en las ciencias de la computación.

El bit es un acrónimo de *Binary digit* lo cual se traduce como “dígito binario” el cual suele identificarse como “b”. De acuerdo con la definición de bit es un dígito del sistema de numeración binaria, que son representados ya sea que se representa con el 0 o 1.

(«¿Qué es un bit?», s.f.)

5.2.2. Byte

A un grupo de 8 bits se les agrupa. A esta agrupación se le conoce como byte.

Es la unidad estándar de información en la informática. Al momento de poder identificar la unidad en la mayoría de ocasiones se suele identificar como “B” pues todavía no existe un símbolo. Al ser un conjunto de 8 bits también se suele llamar como octeto.

(«¿Qué es un byte?», s.f.)

Un byte puede contener 2^8 combinaciones distintas de bits. Por ejemplo:

01000111 01111111 0000000

Los bits en un byte pueden representar cualquier cantidad entre el 0 al 255 si se interpretan como números binarios no firmados. Efectivamente, formando 256 combinaciones diferentes.

5.2.3. CPU

Las computadoras entienden la información de distinta forma a los humanos. Estas interpretan bytes que resultan en instrucciones para la CPU; en caracteres de un archivo de texto, en piezas para armar una imagen, en direcciones de memoria y en cualquier dato que se procese, almacene o ejecute en las mismas.

La forma en que una computadora interpreta se define con las instrucciones que tiene su CPU, su unidad central de procesamiento. Cada una cuenta con instrucciones distintas, como son los procesadores Intel o Ryzen.

La manera en que se moldearon las compuertas lógicas dentro de un procesador hará que el valor, por poner un ejemplo, 47 sea el que indique a la máquina que realizara una suma y que por ende debe tomar los siguientes dos valores para sumarlos.

Así como la computadora necesita entender los bytes con las instrucciones que tiene definidas, los humanos que entiendan las instrucciones y la CPU, pueden entender el funcionamiento de todo lo que sucede dentro de ella.

Por ello es necesario proteger la información, desde archivos de texto como

5.2.4. Como funciona un archivo

Un archivo o también conocido como fichero es un conjunto de unidades de bits almacenados en un dispositivo. Cada archivo se compone de un nombre, un punto y una extensión que va ser el encargado de determinar el tipo de archivo que va ser y sus funciones que debe de cumplir.

Dentro de los archivos existen paquetes pequeños de datos expresados en bits y los cuales se ordenan en registros o en líneas, siendo distintos, pero con algún rasgo en común.

Cabe mencionar que el modo de agrupación depende de quien haga el archivo, por lo que existen varias estructuras de archivos, desde más simples hasta más complejas.

Los archivos pueden tener distintas funciones ya sean contener información como es el caso de los archivos de textos, hasta crear archivos ejecutables que desencadenan ciertas secuencias de acciones que tiene como resultado una acción concreta.

Desde apagar la computadora hasta el iniciar algún videojuego, todo eso funciona a través de archivos interconectados ejecutándose por turno en memoria del computador.

(«Archivo en informática», s.f.)

5.2.5. Álgebra modular

En el álgebra modular los valores que se emplean siguen un ciclo infinito *redondo* desde al 0 hasta un número conocido como **módulo**. Se usa como sistema de equivalencias, también llamado aritmética de reloj o de anillo ya que los números *dan la vuelta* tras alcanzar el módulo.

(«Aritmética modular», s.f.)

Un ejemplo, tal como una de sus denominaciones informales, son las horas que maneja el reloj. En ella tenemos un valor máximo de 12 y uno mínimo de 1.

Matemáticamente podemos representar el conjunto de los valores posibles del reloj como:

$$\{x|x \pmod{12} + 1\}$$

El cual es un conjunto que se construye de los valores $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11\}$ obtenidos con la operación $(\text{mod} 12)$ y a dicho resultado sumarle una unidad.

5.2.6. Módulo

Definición 1. Sea módulo (m) un número entero tal que $m > 1$. Decimos que dos enteros a y b son congruentes entre sí módulo m (congruente “ $(\text{mod } m)$ ” para abreviar). Escribimos

$$a \equiv b \pmod{m},$$

para hacer saber que la diferencia $a - b$ es un múltiplo entero de m . En otras palabras, $a \equiv b \pmod{m}$ cuando $a = b + k \times m$ para algún número entero k (positivo, negativo o cero).

Definición 2. Sea m un entero con $m > 1$. Para un entero arbitrario a , el residuo de un módulo m es el único entero r entre $0, 1, \dots, m - 1$ con el que a es congruente módulo m .

(Eisenberg, 1999)

En la programación hay un operador llamado módulo representado como `%`, que puede causar confusión.

El operador se encarga de devolver el residuo de una operación sin importar si el número es negativo, positivo o 0. El problema radica en que se rompe con la definición 2 puesto que algunos lenguajes de programación suelen devolver valores negativos cuando se realiza la operación `a % m`

De modo en que, en Rust, el siguiente ejemplo no es equivalente $a - 1 \equiv b \pmod{m}$

```
let b: int = -1 % m
```

Pero en Python el siguiente ejemplo si es equivalente a la misma expresión.

```
b = -1 % m
```

Para poder obtener el comportamiento deseado se debe usar otro método con Rust. Es obligación de cada programador entender el funcionamiento de sus operadores para evitar cometer errores.

El siguiente código en Rust si se conforma con las dos definiciones dadas:

```
let b: int = (-1).rem_euclid(m)
```

6. Aplicación Posible

En este caso se decidió aplicarlo en la descifrar de un archivo con el propósito de proteger la información de las personas, ya que es un problema sumamente preocupante pues según Kaspersky en México ocurren aproximadamente 300 ataques por minute, en los cuales en la mayoría de las ocasiones llegan a penetrar la seguridad.

Otro posible caso de su aplicación que se puede hacer al álgebra lineal es el de reconocimiento de imágenes, el cual también es importante porque enseñas a una máquina a reconocer objetos y esto nos puede servir en el caso de que quieras enseñar a un vehículo autónomo a que vaya reconociendo objetos y así vayan siendo mas seguros tanto para el pasajero como para los peatones.

7. Experimentación

7.1. Método 1

7.1.1. Cifrado

Para el primer algoritmo de cifrado tenemos una matriz que contiene todos los valores del 0 al 255, que representan todos los valores posibles representables por un byte de información.

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
32	33	34	35	36	37	38	39	40	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	73	74	75	76	77	78	79	80	81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	
96	97	98	99	100	101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117	118	119	120	121	122	123	124	125	126	127
128	129	130	131	132	133	134	135	136	137	138	139	140	141	142	143	144	145	146	147	148	149	150	151	152	153	154	155	156	157	158	159
160	161	162	163	164	165	166	167	168	169	170	171	172	173	174	175	176	177	178	179	180	181	182	183	184	185	186	187	188	189	190	191
192	193	194	195	196	197	198	199	200	201	202	203	204	205	206	207	208	209	210	211	212	213	214	215	216	217	218	219	220	221	222	223
224	225	226	227	228	229	230	231	232	233	234	235	236	237	238	239	240	241	242	243	244	245	246	247	248	249	250	251	252	253	254	255

Para este método, cambiaremos el orden de los valores aplicándole transformaciones a Bytes. La clave para que se pueda descifrar el mensaje y almacenar en un archivo es no perder ni una variante del valor que representa un byte.

Para lograrlo podemos intercambiar los valores, por fila o columna aplicando transformaciones.

Sea A^T una transformación que se le aplica a Bytes, A^T debe seguir las siguientes reglas:

1. La transformación que afecte a C_x también debe afectar a C_y , donde y es igual a $c - (x - 1)$, donde c es el número de columnas (31, contando desde 0) de la matriz identidad de Bytes
 2. La matriz asociada de la transformación debe ser cuadrada, de 32×32 , para que sea posible obtener su matriz inversa y poder descifrar el mensaje
 3. No se puede afectar a C_1 , porque la forma en que el 0 se comporta con la aritmética modular no nos favorece
 4. A toda transformación que multiplique un escalar tal que $a \neq 1$, se debe aplicar el álgebra modular para obtener un número congruente $b(\text{mod } m)$

Si (1) no se cumple, obtenemos pérdidas de información que hacen imposible el descifrado del *ciphertext*.

Podemos intercambiar filas aplicando una transformación.

Sea

$$T_1 \begin{pmatrix} x_0 & x_1 & x_2 & x_3 & x_4 \\ x_5 & x_6 & x_7 & x_8 & x_9 \\ x_{10} & x_{11} & x_{12} & x_{13} & x_{14} \end{pmatrix} = \begin{pmatrix} x_0 & x_3 & x_2 & x_1 & x_4 \\ x_5 & x_8 & x_7 & x_6 & x_9 \\ x_{10} & x_{13} & x_{12} & x_{11} & x_{14} \end{pmatrix}$$

Su matriz asociada es:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

Para cambiar el orden de las filas solo hace falta intercambiar posiciones de la matriz identidad correspondiente.

Si se quisiera la columna C_3 con la C_4 se puede emplear una transformación descrita con la matriz:

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

Podemos aplicar ésta forma de transformación a `Bytes`, donde haremos un *swap* entre la columna R_1 y $R_{31-(1-1)}$

En la figura 1, se puede apreciar el cambio logrado al intercambiar ambas filas entre sí.

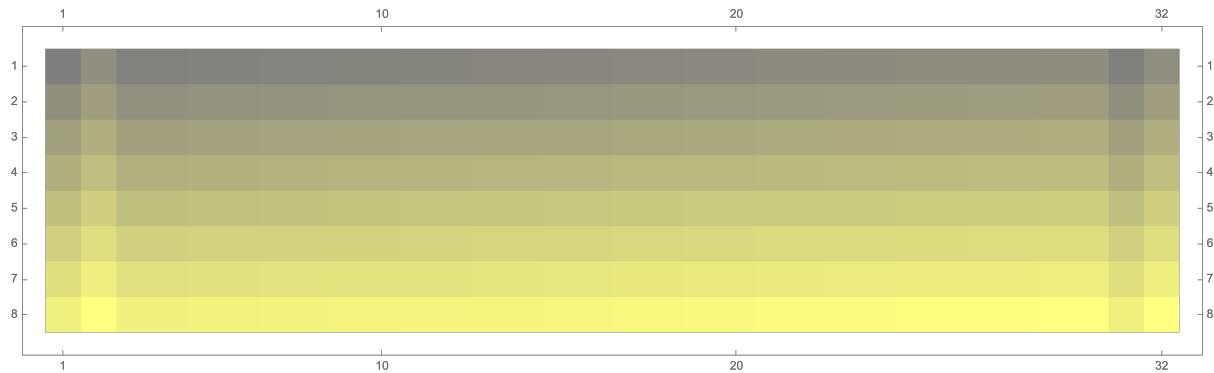


Figura 1: Visualización de Bytes $\times T_1$

Para cambiar el orden de las filas, lo que se puede hacer es multiplicar las filas por -1 en la transformación, eso hace que, al momento de aplicar la misma a la matriz `Bytes` y obtener una matriz congruente por medio de la aplicación del módulo a todos sus elementos, se dé el efecto deseado, en el que el orden de las filas son cambiadas de lugar en ciertas columnas.

Un ejemplo, aplicamos a Bytes una transformación cuya matriz asociada es:

donde solo se cambia el signo de la fila R_2 y la fila $R_{31-(2-1)}$ para no tener pérdidas de información recordando la lista de requisitos de toda transformación aplicada a Bytes. Luego de obtener solo enteros congruentes (mod32), el resultado se aprecia en la siguiente figura 2:

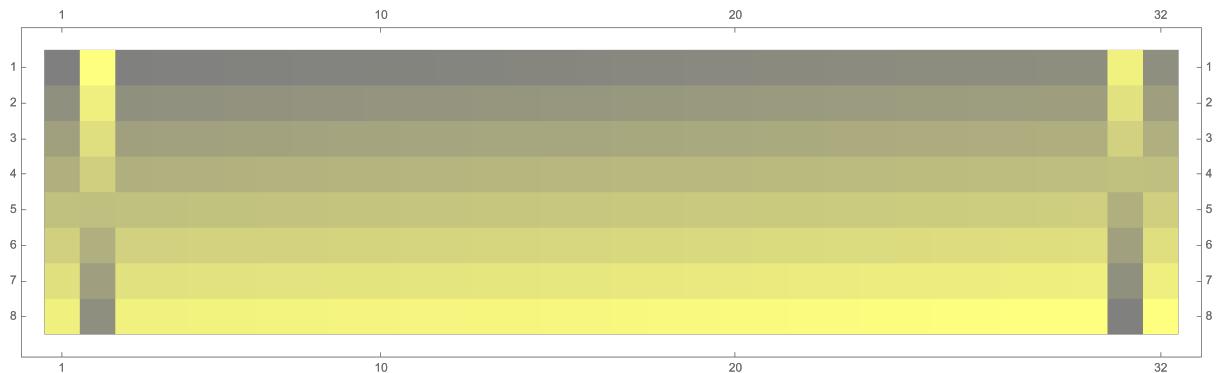


Figura 2: Visualización de Bytes $\times T_2$

Es fácilmente apreciable el cambio. Como mencionado antes, la fila R_2 y la fila $R_{31-(2-1)}$ tienen los

valores verticales intercambiados.

Podemos mezclar los dos tipos de transformaciones mencionadas antes para generar un diccionario que cambia totalmente la forma del mensaje a nivel de bytes de un archivo.

Por ejemplo, con la matriz asociada a T_3 :

Podemos transformar Bytes de forma que obtenemos como resultado:

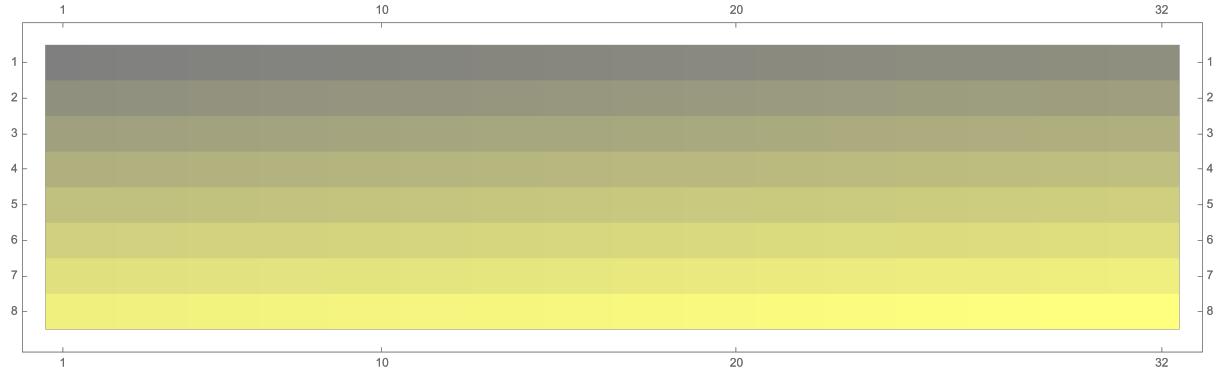


Figura 3: Visualización de Bytes

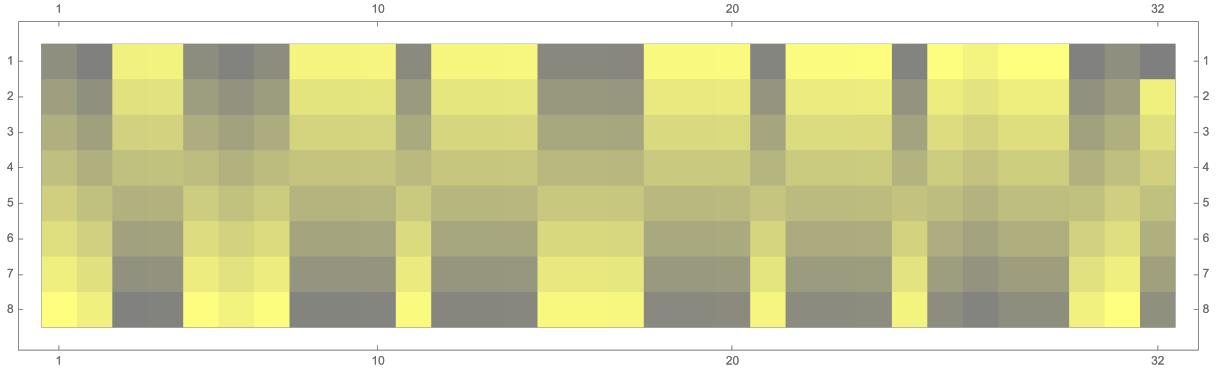


Figura 4: Visualización de $\text{Bytes} \times T_3$

Por todo lo anterior podemos decir que

$$\text{Bytes} \times T \equiv \alpha \pmod{255}$$

Donde Bytes es la matriz de 8×32 que contiene todos los valores representables por un byte y T es una transformación aleatoria formada siguiendo las reglas descritas al inicio de la sección. Siendo α el resultado de la transformación, que puede ser usado como asociación para remplazar los bytes del archivo.

De forma en que, similar al famoso *Caesar Cipher*, representa un mapeo de llave y valor que da la ilusión de desplazamiento o *swap* de valores.

Ejemplo:

$$\begin{array}{ll} 0x00 & 0x00 \\ 0x01 & 0x0F \\ 0x02 & 0x0A \end{array}$$

De forma en que $0x00$ pasará a valer $0x00$, $0x01$ pasará a valer $0xFF$, etc. Mapeo que es realizado en base a la comparación entre Bytes y α .

Luego de lo experimentado, nuestro algoritmo queda así:

1. Solicitar al usuario el archivo a cifrar
2. Comprobar que el usuario ingresó un archivo, no un directorio
3. Comprobar que el usuario ingresó un archivo que existe en el sistema
4. Generar una matriz cuadrada de 32×32 con 0 en ella
5. Popular R_1 con el valor por defecto: $\{1, 0, 0, 0, \dots\}$
6. Generar una matriz asociada de una T de forma aleatoria. Cumpliendo las reglas:
 - a) Si se modifica C_x se debe afectar C_y donde x es un número entero del 0 al 31 y $y = 31 - (x - 1)$
 - b) No se puede modificar a R_1
7. Aplicar la transformación T a Bytes y almacenar el resultado

8. Armar un `HashMap` con los valores del resultado.
 9. Calcular la matriz inversa de la matriz asociada a T
 10. Escribir una firma de archivo para saber cuando el ususario ingresa un archivo que no ha sido cifrado
 11. Escribir la matriz inversa como *metadata* en el archivo cifrado para poder descifrarlo
 12. Escribir el estado de bytes inverso como *metadata* en el archivo cifrado para poder descifrarlo
 13. Reemplazar todos los bytes del archivo objetivo por las asociaciones descritas en el `HashMap`

Se puede observar el algoritmo que genera la matriz asociada a la transformación en el Appendix B

De modo en que los primeros 4 bytes del archivo cifrado deben verse:

42 3C OA FF

Para saber que se trata de un archivo cifrado.

Después se encontrarán 2049 bytes describiendo la inversa de T , en ellos, cada byte inmediante anterior representa si en la matriz se contaba con un número negativo 0x01 o positivo 0x00 seguido del valor (0x00 — 0x01)

Luego, la descripción del estado final de Bytes luego de aplicarle T

```
00 E1 02 FD E4 E5 FA F9 E8 F7 EA F5 F4 F3 F2 F1 F0 EF EE ED EC EB F6 E9 F8 E7 E6 FB FC  
E3 1E FF 20 C1 22 DD C4 C5 DA D9 C8 D7 CA D5 D4 D3 D2 D1 D0 CF CE CD CC CB D6 C9 D8 C7  
...
```

Y por último los bits cifrados del mensaje original.

A8 91 94 81 20 93 85 20 94 94 81 93 91 20 84 81 92 97 85 94

7.1.2. Descifrado

De modo en que:

$$\text{Bytes} \times T \equiv \alpha \pmod{255}$$

Entonces:

$$\alpha \times T^{-1} \equiv \text{Bytes} \pmod{255}$$

Es por esa razón por la que experimentamos almacenando α y T^{-1} dentro del mismo archivo cifrado, de forma en que podemos rápidamente recuperarlos para descifrar los bytes que en él se contengan.

Como decidimos almacenar en el mismo archivo cifrado los datos que nos permiten armar la matriz inversa para regresar del estado cifrado al texto normal, usaremos el algoritmo:

1. Solicitar al usuario el archivo a descifrar
2. Comprobar que el usuario ingresó un archivo, no un directorio
3. Comprobar que el usuario ingresó un archivo que existe en el sistema
4. Generar una matriz cuadrada de 32x32 con 0 en ella llamada `key`
5. Generar una matriz cuadrada de 8x32 con 0 en ella llamada `inv`
6. Popular `key` con los valores almacenados como metadata en el archivo cifrado
7. Popular `inv` con los valores almacenados como metadata en el archivo cifrado
8. Aplicar la transformación T^{-1} (`inv`) a `key` y almacenar el resultado
9. Armar un `HashMap` con los valores del resultado.
10. Reemplazar todos los bytes del archivo cifrado por las asociaciones descritas en el `HashMap`

7.2. Método 2

Este segundo método es similar al primero en el sentido de que usamos las bases:

$$\beta \times T \equiv \alpha \pmod{255} \therefore \alpha \times T^{-1} \equiv \beta \pmod{255}$$

8. Resultados

Luego de la experimentación con Mathematica, se implementaron ambos algoritmos con Rust usando las dependencias:

1. `clap "3.1.17"`: Para tener un manejo y estructura más sencilla de los argumentos y comandos del programa
2. `nalgebra "0.31.0"`: Librería para tener a disposición matrices de tamaño arbitrario. También brinda operaciones como multiplicación de matrices u obtener la inversa.
3. `rand "0.8.5"`: Para la generación de números aleatorios empleados al momento de construir la matriz asociada a la transformación T en cada ejecución,

La utilidad de linea de comandos desarrollada cuenta con dos verbos principales:

1. `cypher` (con alias `c`): Cifra un archivo remplazándolo por uno con la extensión `.lcy`. Elimina el archivo original al terminal.
2. `decipher` (con alias `d`): Descifra un archivo. Crea un archivo con el nombre original, sin extensión `.lcy` donde coloca los resultados. Elimina el archivo cifrado al terminal.

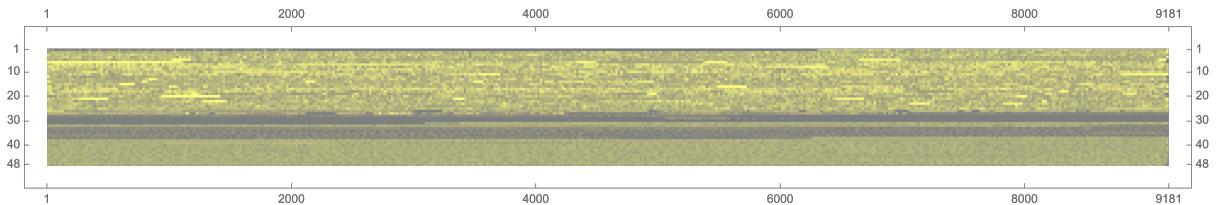


Figura 5: Visualización de los bytes del archivo original. M

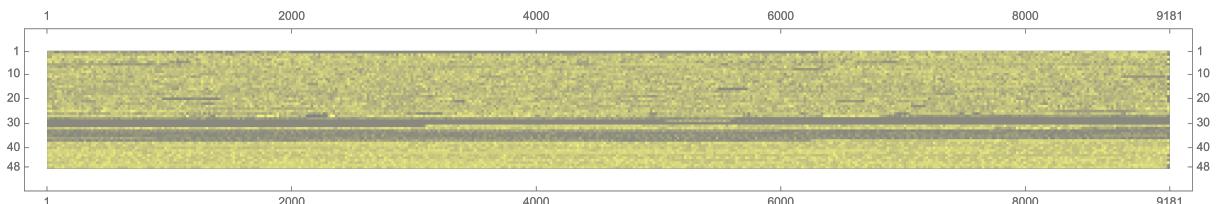


Figura 6: Visualización de los bytes del archivo cifrado. N

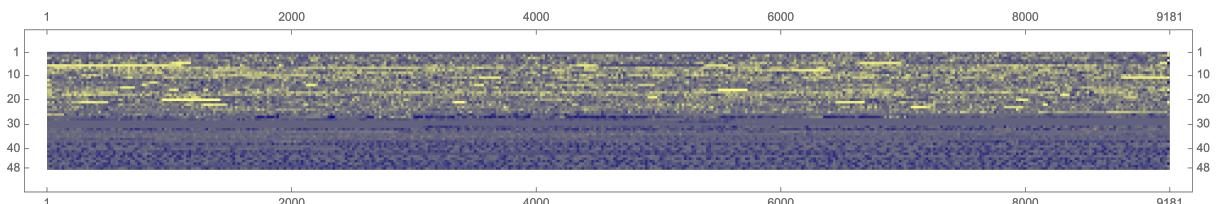


Figura 7: $M - N$

9. Discusión de Resultados

9.1. Método 1

La mayor vulnerabilidad que presenta este método de cifrado es que la frecuencia de bytes presentes es la misma debido a que no estamos sacrificando ninguna pérdida de información y, por lo tanto, solo sustituimos los valores.

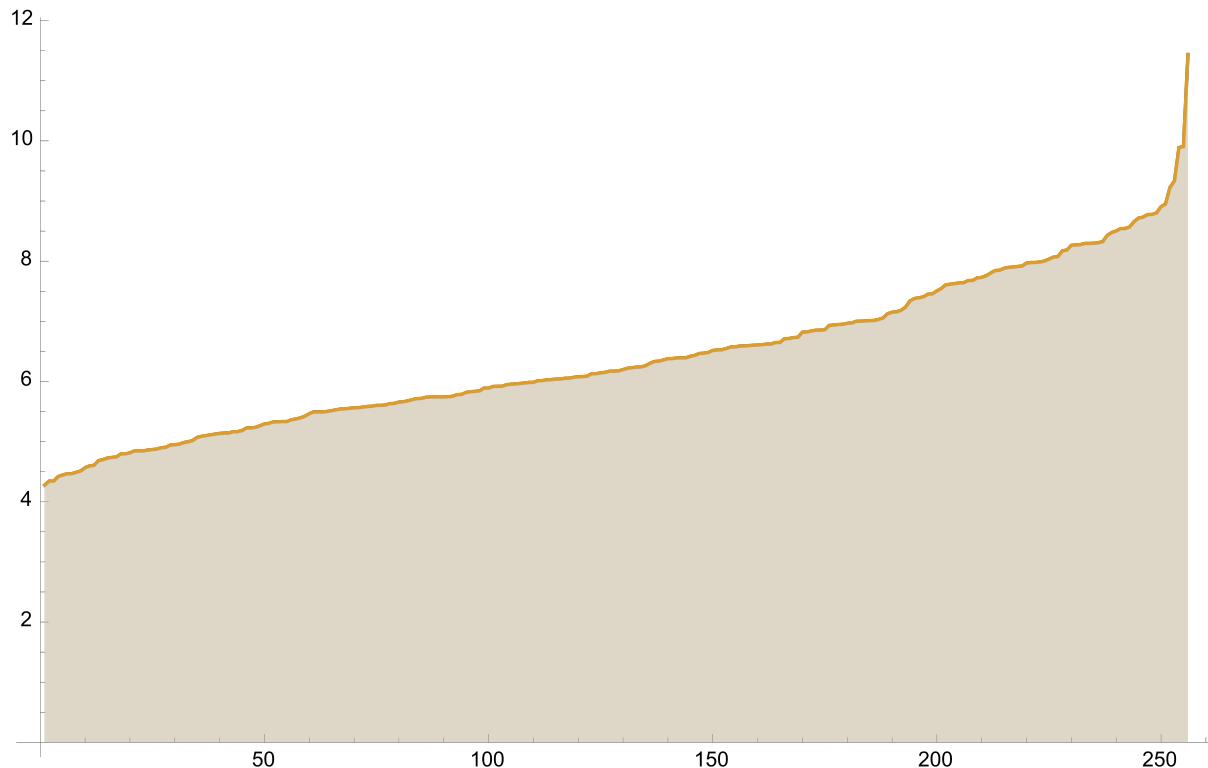


Figura 8: Histograma de los bytes del archivo original y el cifrado.

En la figura 8, se muestra la sobreposición de los histogramas de bytes del documento antes y después de cifrado. Ambos son idénticos. Si el documento está escrito en Inglés o contuviera las instrucciones de cierta arquitectura, y se conoce la frecuencia de bytes, el mensaje puede ser descifrado.

9.2. Método 2

10. Conclusiones y Recomendaciones

11. Bibliografía

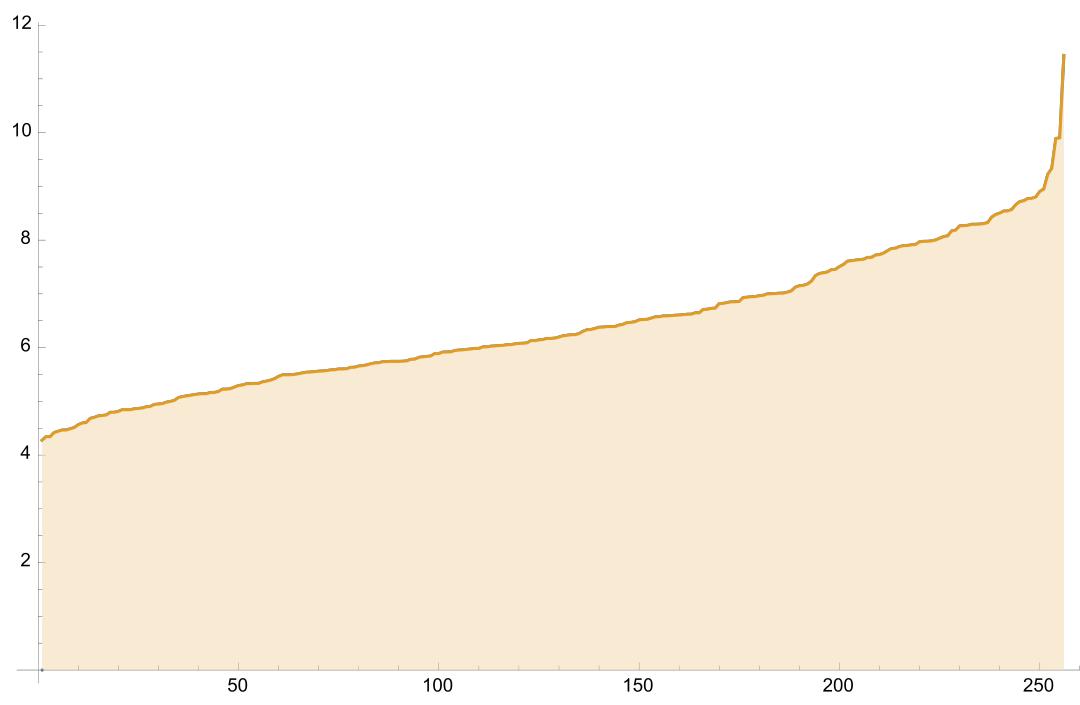
1. *¿Qué es un bit?* (s.f.). Consultado el 28 de abril de 2022, desde <https://www.jvs-informatica.com/blog/glosario/bit/>
2. *¿Qué es un byte?* (s.f.). Consultado el 28 de abril de 2022, desde <https://www.jvs-informatica.com/blog/glosario/byte/>
3. *Archivo en informática.* (s.f.). Consultado el 28 de abril de 2022, desde <https://concepto.de/archivo-informatico/>
4. *Aritmética modular.* (s.f.). https://es.wikipedia.org/wiki/Aritm%C3%A9tica_modular
5. Eisenberg, M. (1999). *Hill Ciphers and Modular Linear Algebra.* <https://apprendre-en-ligne.net/crypto/hill/Hillciph.pdf>

A. Appendix

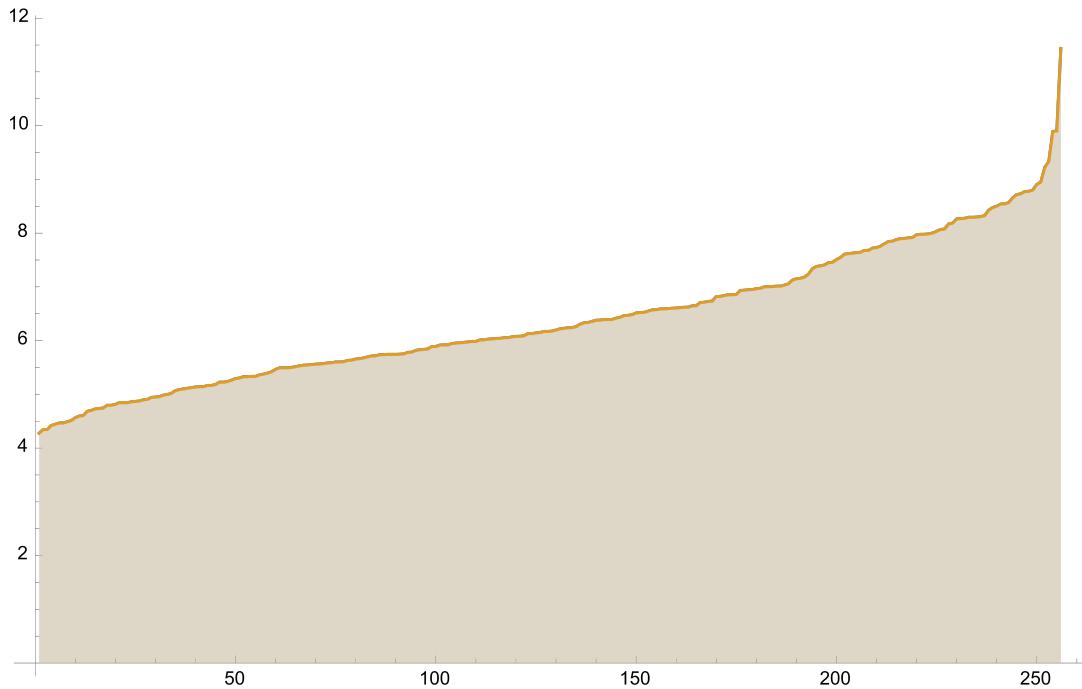
Comparación de histograma de las ocurrencias de bytes entre un arreglo de bytes y su versión cifrada por el Método 1



Histograma de los bytes del archivo original y el cifrado.



Histograma de los bytes del archivo cifrado.



Histograma de los bytes del archivo original y el cifrado.

Gráficas elaboradas en Mathematica 13 con el código:

```

1 Show[
2   With[
3     {
4       assoc = CountRepetitions[baa],
5       assoc2 = CountRepetitions[Flatten[bx]]
6     },
7
8     ListLinePlot[
9       {
10         KeyValueMap[Log[#2] &]@KeySortBy[assoc, assoc[#] &],
11         KeyValueMap[Log[#2] &]@KeySortBy[assoc2, assoc2[#] &]
12       },
13       Filling -> Axis,
14       PlotRange -> Full,
15       ImageSize -> Large
16     ]
17   ]
18 ]

```

donde `baa` es un arreglo de bytes contenido el estado del archivo original y donde `bx` es una matriz que se usa como arreglo de bytes que contiene el estado cifrado de los contenidos de donde `baa`.

B. Appendix

```
1 pub fn met1_armar_matriz(rng: &mut ThreadRng) -> DMatrix<f32> {
2     let mut resultado: DMatrix<f32> = dmatrix![] .resize(32, 32, 0.0);
3
4     let mut switch: [bool; 32] = [false; 32];
5     let mut neg: [bool; 32] = [false; 32];
6
7     for col in 1..32 / 2 {
8         if rng.gen::<u8>() % 2 == 0 {
9             switch[col] = true;
10            switch[31 - (col - 1)] = true;
11        }
12
13        if rng.gen::<u8>() % 5 == 0 {
14            neg[col] = true;
15            neg[31 - (col - 1)] = true;
16        }
17    }
18
19    resultado[(0, 0)] = 1.0;
20
21    for col in 1..32 {
22        if switch[col] {
23            resultado[(col, 31 - (col - 1))] = if neg[col] { 1.0 } else { -1.0 };
24        } else {
25            resultado[(col, col)] = if neg[col] { 1.0 } else { -1.0 };
26        }
27    }
28
29    let switch_mat: DMatrix<bool> = DMatrix::from_vec(1, 32, Vec::from(switch));
30
31    resultado
32 }
```

C. Appendix

```
1 extern crate nalgebra as na;
2
3 use divisors::get_divisors;
4 use lcy::*;
5 use std::collections::HashMap;
6 use std::fs::{remove_file, OpenOptions};
7 use std::io::{Read, Seek, Write};
8 use std::process::exit;
9
10 use na::{dmatrix, DMatrix};
11
12 fn main() {
13     let mut rng = rand::thread_rng();
14     let args: Args = Args::parse();
15
16     let mut map: HashMap<u8, u8> = HashMap::new();
17
18     match args.command {
19         Commands::Cypher { path } => {
20             let bytes: DMatrix<f32> = dmatrix![ ... ];
21
22             if !path.exists() {
23                 eprintln!("Error: El archivo {} no existe!", path.display());
24                 exit(1)
25             }
26
27             let mut file = OpenOptions::new()
28                 .read(true)
29                 .write(true)
30                 .create(false)
31                 .open(&path)
32                 .unwrap();
33
34             let mut file2 = OpenOptions::new()
35                 .write(true)
36                 .create(true)
37                 .truncate(true)
38                 .open(&format!("{}.lcy", &path.display()))
39                 .unwrap();
40
41             let mut contenidos: Vec<u8> = vec![];
42
43             file.read_to_end(&mut contenidos).unwrap();
44
45             let mat = met1_armar_matriz(&mut rng);
46             let mut key = &bytes * &mat;
47             let inv_key = mat.clone().try_inverse().unwrap();
48
49             for i in 0..8usize {
50                 for j in 0..32usize {
51                     key[(i, j)] = key[(i, j)].rem_euclid(256.0);
52                     map.insert(bytes[(i, j)] as u8, key[(i, j)] as u8);
53                 }
54             }
55
56 }
```

```

55
56     for i_byte in 0..contenidos.len() {
57         contenidos[i_byte] = *map.get(&contenidos[i_byte]).unwrap();
58     }
59
60     let mut inv_key_to_write = vec![];
61     let mut bytes_final_to_write = vec![];
62     for i in 0..32 {
63         for j in 0..32 {
64             let indicador = if inv_key[(i, j)] < 0.0 { 1u8 } else { 0u8 };
65             let val = if inv_key[(i, j)] < 0.0 {
66                 -1.0 * inv_key[(i, j)]
67             } else {
68                 inv_key[(i, j)]
69             };
70
71             inv_key_to_write.push(indicador);
72             inv_key_to_write.push(val as u8);
73         }
74     }
75
76     for i in 0..8 {
77         for j in 0..32 {
78             bytes_final_to_write.push(key[(i, j)] as u8);
79         }
80     }
81
82     file2.write_all(&[66u8, 60u8, 10u8, 255u8]).unwrap();
83     file2.write_all(&inv_key_to_write).unwrap();
84     file2.write_all(&bytes_final_to_write).unwrap();
85     file2.write_all(&contenidos).unwrap();
86
87     remove_file(path).unwrap();
88 }
89 Commands::Decipher { path } => { ... }
90 }
91 }
```

D. Appendix

```
1 extern crate nalgebra as na;
2
3 use divisors::get_divisors;
4 use lcy::*;
5 use std::collections::HashMap;
6 use std::fs::{remove_file, OpenOptions};
7 use std::io::{Read, Seek, Write};
8 use std::process::exit;
9
10 use na::{dmatrix, DMatrix};
11
12 fn main() {
13     let mut rng = rand::thread_rng();
14     let args: Args = Args::parse();
15
16     let mut map: HashMap<u8, u8> = HashMap::new();
17
18     match args.command {
19         Commands::Cypher { path } => { ... }
20         Commands::Decipher { path } => {
21             if !path.exists() {
22                 eprintln!("Error: El archivo {} no existe!", path.display());
23                 exit(1)
24             }
25
26             let orig_path = path.as_os_str().to_str().unwrap().replace(".lcy", "");
27
28             let mut file_ci = OpenOptions::new()
29                 .read(true)
30                 .write(true)
31                 .create(false)
32                 .open(&path)
33                 .unwrap();
34
35             let mut file = OpenOptions::new()
36                 .read(true)
37                 .write(true)
38                 .truncate(true)
39                 .create(true)
40                 .open(&orig_path)
41                 .unwrap();
42
43             let mut contenidos: Vec<u8> = vec![];
44
45             file_ci.read_to_end(&mut contenidos).unwrap();
46
47             if contenidos[0..0x4] != [66u8, 60u8, 10u8, 255u8] {
48                 eprintln!("Error: No es un archivo válido cifrado");
49                 exit(1);
50             }
51
52             let mut inv = dmatrix![] .resize(32, 32, 0.0);
53             let mut key = dmatrix![] .resize(8, 32, 0.0);
54 }
```

```

55     let contenidos = &contenidos[0x4..];
56
57     let mut k = 0;
58     for i in 0..32 {
59         for j in 0..32 {
60             let neg = contenidos[k] == 0x1;
61             let val = (contenidos[k + 1] as i32) * if neg { -1 } else { 1 };
62
63             inv[(i, j)] = val as f32;
64             k += 2;
65         }
66     }
67
68     let contenidos = &contenidos[k..];
69
70     let mut k = 0;
71     for i in 0..8 {
72         for j in 0..32 {
73             key[(i, j)] = contenidos[k] as f32;
74             k += 1;
75         }
76     }
77
78     let mut orig = &key * &inv;
79
80     for i in 0..8usize {
81         for j in 0..32usize {
82             orig[(i, j)] = orig[(i, j)].rem_euclid(256.0);
83             map.insert(key[(i, j)] as u8, orig[(i, j)] as u8);
84         }
85     }
86
87     let mut contenidos = contenidos[k..].to_vec();
88
89     for i_byte in 0..contenidos.len() {
90         contenidos[i_byte] = *map.get(&contenidos[i_byte]).unwrap();
91     }
92
93     file.write_all(&contenidos).unwrap();
94     remove_file(path).unwrap();
95 }
96 }
97 }
```