Universidad Nacional Autónoma de México

Facultad de Ciencias

Criptografía y Seguridad

Práctica 3: Cifrados Clásicos

Fecha de entrega: 12/09/2022

Equipo:

Criptonianos

Acosta Arzate Rubén - 317205776

Bernal Marquez Erick - 317042522

Deloya Andrade Ana Valeria - 317277582

Gutiérrez Medina Sebastián Alejandro - 318287021

Marco Antonio Rivera Silva - 318183583

1. Introducción

En esta práctica realizamos el descifrado de tres archivos en formato jpg, mp4 y mp3. Para ello realizamos la implementación de tres diferentes tipos de cifrados: cifrado afín, cifrado de César y cifrado base64; o mejor dicho, la implementación del descifrado de estos. Además de se utilizó Python como lenguaje de programación para la lectura y manejo de bytes por las funciones *built-in* que posee.

Este tipo de prácticas, donde nuestros archivos se cifran, pueden ser usadas para secuestro de información, donde el atacante pide una cantidad generosa de dinero para descifrar nustros archivos.

Puede parecer insignificante, pensar "a mí nunca me pasaría eso, yo no visito ese tipo de paginas, mi sistema operativo es Linux, uso arch, btw", pero recordemos que el correo es un canal inseguro y no es necesario visitar páginas de dudosa procedencia para caer en este tipo situaciones, basta con abrir un correo o caer en *phishing*

2. Desarrollo

2.1. file1.lol (Cifrado César)

Descifrado por fuerza bruta, fue el segundo archivo que desciframos.
Como ya tenemos el archivo *file2.lol* descifrado con base 64 entonces nos quedaban 4 posibilidades: Que el archivo *file1.lol* fuera cifrado con cifrado César o afín, y que el archivo *file3.lol* fuera cifrado con cifrado César o afín. Decidimos probar a la suerte cual tendría el cifrado César, sin embargo por cada archivo tendríamos que probar 256 veces, 512 veces en total empezando con $\beta = 1$, luego $\beta = 2$, $\beta = 3$, etc. Por suerte no pasó mucho tiempo (como una hora y media entre descansos) hasta que el resultado nos generó un archivo que se podía visualizar de forma correcta, resultó ser el video.

De esta manera pudimos resolver que el formato *.mp4* se cifró con César cuya llave es -187 al archivo *file2.lol*, para descifrarlo solo tenemos que obtener el inverso de β, es decir, $+187$. y aplicarlo a un cifrado César.

El algoritmo se encuentra en el script *descifrador.py*

2.2. file2.lol (Cifrado Base64)

Descifrado por observación, fue el primer archivo que desciframos.
No sabíamos cómo empezar y hacerlo por fuerza bruta probando todas las combinaciones posibles sería demasiado tardado. Así que nos pusimos a observar detenidamente los bytes de cada archivo con el comando *hexdump* y nos dimos cuenta que los bytes de el archivo *file2.lol* no pasaban del valor 3F en formato hexadecimal, mientras que los otros archivos tenían bytes con valores mayores por ejemplo 70, 90, A9, FF, etc... Indicando así que este archivo estaba "cifrado" en base 64, ya que valores mayores a 3F no existen en esta base.

Sin embargo, aun estaba la pregunta de a cuál de todos los fomatos correspondia (.jpg, .mp3 o .mp4). Decidimos "cifrar" los primeros bytes de los formatos para ver cual hacía "match" con el archivo *file2.lol* y el ganador resultó ser el formato *.jpg*

De esta manera pudimos resolver que el formato *.jpg* se "cifró" con base 64 al archivo *file2.lol*

La explicación del algoritmo en realidad es bastante sencilla ya que es el proceso inverso de convertir base 256 a base 64, en pocas palabras pasamos los bytes base 64 a binario, los juntamos en grupos de 6 bits, luego juntamos toda la cadena de 0 y 1's y los partimos en grupos de 8, por último obtenemos el valor de cada byte en formato decimal [0-255] y obtenemos su valor hexadecimal siendo los primeros FF, D8, FF en el archivo ya descifrado.

El algoritmo se encuentra en el script *descifrador.py*

2.3. file3.lol (Cifrado Afín)

Descifrado por descarte, fue el tercer archivo que desciframos.
Como ya teníamos dos archivos descifrados el único cifrado que faltaba era el cifrado afín, que correspondía a el formato *.mp3*. Pero aun así debíamos obtener la llave para descifrarlo.
Primero debíamos resolver el sistema de ecuaciones en formato decimal base 256

$$73\alpha + \beta = 169$$

$$68\alpha + \beta = 226$$

$$51\alpha + \beta = 215$$

Omitiremos el procedimiento ya que lo que nos interesa es el descifrado. Resolviendo el sistema tenemos que $\alpha = 91$ y $\beta = 182$, para el descifrado basta con "despejar" el multiplicando de α (ya que fue el que se cifró) en términos de α, β y el numero en base 256 (que es el número ya cifrado). Es decir, el despeje es $((byte - 182) * 211)\ mod_{256}$, donde *byte* es el byte en el archivo cifrado, -182 el inverso de la suma β y 211 el inverso multiplicativo de α.
De esta manera obtenemos el número que se cifró en el archivo

Así fue cómo pudimos resolver que el formato *.mp3* se cifró con cifrado afín al archivo *file3.lol* El algoritmo se encuentra en el script *descifrador.py*

2.4. Acerca del código

Como ya se mencionó antes decidimos utilizar el lenguaje *python 3* por las funciones *built-in* que este posee, para la lectura de los archivos tenemos una función *leer_archivo(archivo)* donde leemos los bytes del archivo y los pasamos a formato decimal para que sea más fácil de manipular

Para todas las funciones de descifrado suponemos que las recibimos en base decimal ya que es más fácil de trabajar y regresamos una lista de bytes con la función *bytes(list)*, para que python sepa que deseamos escribir justo en bytes y no los caracteres en sí. Aunque estos siguen estando en base decimal.

Para la escritura tenemos la función *escribir_archivo(des, nombre)* donde el primer parametro es el archivo ya descifrado y el segundo el nombre que le vamos a dar al archivo, este recibe el una lista de bytes ya descifrado en base decimal, sin embargo para poder hacer una correcta escritura en base hexadecimal ocupamos la función *built-in* hex(bytes) que se encarga de regresarnos la representación hexadecimal.

Por último leemos cada archivo y le aplicamos su correspondiente función de descifrado, guardando el archivo con el formato adecuado, pues ya sabemos cómo fueron cifrados cada uno de ellos, y solo nos encargamos de pasarle los archivos correctos a las funciones.

Basta con ejecutar

```
python3 descifrador.py
```

Los archivos descifrados son guardados en la carpeta *archivos-descifrados*

2.5. Preguntas

- ¿Cuántos primos relativos hay en Z_{256} (o bien, en [0, 1, ... 255])?

  Para saber la cantidad de números primos relativos, o co-primos, que hay en un rango basta con aplicar la función de Euler. Aplicando la función obtenemos que hay 128 números primos relativos.

- Explica a qué César se le atribuye este algoritmo y para qué lo usaba.

  El nombre de este algoritmo de cifrado lleva ese nombre por Julio César, el cual fue un líder militar y gobernante de la República Roma. Julio César desarrolló este cifrado para comunicarse con sus aliados por medio del envío de mensajes, utilizando en la mayoría de sus mensajes codificados un desplazamiento de tres posiciones. En esa época este cifrado resultaba ser muy efectivo debido a que la mayoría de personas no sabía leer y/o escribir.

- ¿Cuántas posibles combinaciones hay para cifrar bytes en modo césar?

  El número de combinaciones posibles está dado a partir del número de elementos en el alfabeto. Así, por ejemplo, en nuestro alfabeto en español (sin contar letras como *ñ*, *ll*, *rr*) existen 26 combinaciones. Sin embargo, al tratarse de *bytes* tenemos 256 elementos [00, 01, 02, ..., FD, FE, FF], por lo cual tenemos 256 posibilidades para el cifrado César.

- ¿Cuántas posibles combinaciones hay para cifrar bytes en modo afin?

  Tomemos en cuenta que el cifrado afín es de la forma $\alpha x + \beta$ y por ser *bytes* tenemos 256 elementos. Uno podría pensar que la cantidad de cifrados afines son $256 * 256$, sin embargo no todos son posibles ya que no todas las posibles α's cuentan con inverso módulo 256.

  Por lo cual utilizamos al función de Euler para determinar cuantos primos relativos hay entre 0 y 256 y así obtener la cantidad de inversos.

  $$\phi(256) = 128$$

  De esta manera sabemos que hay 128 elementos con inverso módulo 256. Ahora solo falta multiplicar por la cantidad de β's, es decir, por 256. Así la cantidad total de cifrados afines es:

  $$128 * 256 = 32768$$

- Ya que base64 no es un cifrado seguro, ¿en qué casos sí se puede usar?

  Debido a que el cifrado base 64 no es seguro, en lugar de ser visto como una forma de proteger la información más bien es un método para transportarla. Este cifrado convierte los bytes en formato ASCII. Siendo usado para enviar cualquier dato binario por medio de transmisiones que tratan únicamente con datos textuales, como para el envío de imágenes y archivos adjuntos por correo electrónico.

- Supongamos que estuvieras en Hogwarts y tuvieras que utilizar un búho para comunicarte, ¿cuál crees que sería la mejor opción para ocultar tu mensaje (si es que tu lechuza cae en una

emboscada por ejemplo) y cómo le pasarías la llave al receptor para que los chismosos no se enteren? (no necesariamente deben ser los 3 de esta práctica). Aquí cada integrante deberá justificar brevemente cómo y por qué. Puedes darte una idea con este video.

- ⋆ Rubén:

  La forma que se me ocurriría sería que cuando pudiéramos vernos lo que equivaldría a que en la vida real tuvieras un canal seguro, me pondría de acuerdo en una llave para los 2, entonces haríamos algo similar al three pass control.

  Entonces yo cifraría un mensaje (digamos con cifrado afín) y se lo enviara con mi lechuza a la otra persona, la otra persona, cifraría mi mensaje con otra combinación afín, luego esa persona me mandaría mi mensaje cifrado y al recibirlo le mandaría mi llave con la que cifre mi mensaje por primera vez, pero esta cifrada con la llave que quedamos los 2, de esta manera al enviársela ella sabría como descifrar mi llave, descifraría mi primer mensaje y con la llave ya decoraría el mensaje original, así creo si seria bastante complicado descifrarlo aunque encontraran los mensajes.

- ⋆ Erick:

  Partimos del hecho de que ambas personas conocemos el protocolo, entonces la llave estaría en el mismo mensaje pero de manera oculta. Por ejemplo, que las primeras letras de cada renglón formen alguna palabra, o la primer letra de cada parrafo, la última letra, algún crucigrama, etc. Y serán dos llaves ya que será la misma lechuza quién se encarga de abrir el mensaje dandole la cómida indicada con la cantidad deseada.

  Por ejemplo, en mí mensaje las primeras letras de cada renglón indican qué cómida le tienes que dar a la lechuza, estas primeras letras estan cifradas con el cifrado César donde la llave es el día del mes. Luego, sumando los números que le corresponden a cada letra indicamos el platillo entre 5 diferentes (módulo 5), siendo el 1 girasoles; el 2, semillas; el 3, alpiste, etc...

Después las últimas letras de cada parrafo indican la cantidad de cómida que se le debe dar, esta vez con el cifrado afín donde β es el día del mes y α el resultado de la operación anterior. El resultado de este segundo cifrado será la cantidad de, por ejemplo, semillas que se le tiene que dar a la lechuza.

Si la combinación es la correcta entonces la lechuza te deja el mensaje, de lo contrario te picotea.

⋆ Valeria:

En un inicio, para que la comunicación pase desapercibida a plena vista, en cada envío de mensajes tanto yo como la otra persona utilizaríamos lechuzas diferentes. Por ejemplo, si intercambiamos mensajes durante 2 ocasiones, habrían sido utilizadas 4 lechuzas diferentes. Esto porque si algún intruso conoce cuáles son nuestras lechuzas propias y llega a interceptar el mensaje, también sabría quién es el autor de dicho mensaje pues en los mensajes en ningún momento se mencionarían nombres.

Mientras que para el cifrado de los mensajes, lo que haríamos sería comenzar por mandar un pequeño mensaje cifrado de prueba a la otra persona, esta frase estará en cifrado de César, el desplazamiento de las letras para el cifrado será importante. Digamos entonces que el mensaje cifrado que yo mando tiene un desplazamiento de por ejemplo 7 letras. El receptor va a descifrar el mensaje pero lo importante aquí no será el mensaje sino que el desplazamiento es de 7. Este número va a ser nuestra clave.

Una vez que sabe que el número importante es 7, me mandará casi el mismo mensaje cifrado de regreso, la única diferencia que tendrá con el que yo mandé es que omitirá a propósito la última letra del mensaje. Al yo recibir este mensaje de vuelta sin esa última letra entiendo que es una confirmación de que la persona a la que iba dirigido el mensaje sí lo recibió y ya sabe cuál será el número importante, que en este ejemplo es 7.

Después se realizará el intercambio del verdadero mensaje secreto. Le mando el mensaje,

esta vez el número de desplazamiento en las letras en el cifrado ya no es tan importante, sólo debe ser diferente al del primer mensaje, siguiendo con el ejemplo será de 5 letras el desplazamiento.

Entonces la otra persona va a recibir mi mensaje, lo va a descifrar y va a ir contando cada 7 letras de éste pues es nuestro número clave. El conteo se hará comenzando a contar a partir del 0. Al final lo que hará es que va a juntar esas letras que contó cada 7, en el orden en que las fue encontrando, y éstas formarán un mensaje que dirá lo que en verdad le quiero comunicar a la persona.

De igual manera si quiere responderme, va a mandar un mensaje cifrado con un número de desplazamiento diferente al clave y yo al recibirlo contaré cada 7 letras para conocer el verdadero mensaje. Así se haría el intercambio si quisiéramos seguir mandándonos más mensajes ese día. El número clave cambia cada día y se da a conocer en base al primer mensaje enviado por cualquiera de los dos.

⋆ Sebastián:

Si suponemos que las persona a la que enviamos el mensaje podríamos cifrar el mensaje con elementos tanto como dependientes del mensaje, como externos (que no necesariamente puedan ser enviados), ya sea por ejemplo aplicar un cifrado cesar y acto seguido cifrarlo para que sólo pueda ser leído en alguna hora especifica del día, con información que la persona a la que enviamos el mensaje sepa, coordenadas especificas a las que tenga que ir la otra persona para poder abrirlo, etc, de esta forma solucionaríamos un poco el tener que mandarle elementos previos necesarios para el descifrado.

Por desgracia siempre se corre el riego de mandar las instrucciones de descifrado y que estas sean interferidas, por lo que tendríamos que decirle en persona al destinatario las instrucciones.

Otra forma que podemos usar también como método de seguridad del mensaje, sería que

al no saber uno de los factores de descifrado y se intente leer o forzar el mensaje, este se destruya.

⋆ Marco:

Partiendo del hecho que conozco a la otra persona que le quiero mandar un mensaje cifrado por lechuza, por lo que la clave principal para descifrar el algoritmo estaría partido en múltiples claves (las cuales serán repartidas entre más conocidos de confianza) las cuales obligarían al persecutor a buscar a todas las personas, (así como los horrocruxes), usando un algoritmo que alguna vez me explicaron en otra materia llamado *el Esquema de Shamir*.

De hecho no sería necesario tal vez reunir a todas las personas pues se podría definir una cantidad mínima de claves requeridas.

Obviamente el método no es infalible, sin embargo es capaz de hacer perder mucho tiempo al atacante al tener que buscar a todos los portadores de la clave dando así tiempo de actuar o si no de huir.

Por otro lado nuestro principal problema sería que necesitaríamos transmitir las múltiples claves y en cada una se corre el riesgo de que se captures, por lo que también sería bueno cifrar las llaves de los demás con información que sólo sabrían las personas a las que van dirigidas, tal vez usando un acertijo para seguir la temática de magia.

3. Conclusiones

La práctica nos ha resultado tanto entretenida como complicada de realizar, pues no fue sencillo realizar el descifrado de los archivos, tomó tiempo, paciencia y muchos intentos. Sobre todo porque no sabíamos muy bien como utilizar las funciones *built-in* y eso ocasionaba muchisimos errores al momento de ejecutar el código, por lo cual si fue un verdadero martirio trabajar con bytes, luego pasarlos a listas, luego a hexadecimal, luego a decimal, etc. Nos confundíamos muchos

Algo que aprendimos gracias a esta práctica es que algunos archivos, a pesar de que no tengan los magic bytes coherentes pueden funcionar (cómo fue el caso del file1.lol). De hecho aprendimos que existen los *magic bytes*. También de la existencia de "cifrado" en base 64, entre comillas porque ya vimos que como tal **no** es un cifrado, ya que carece de llave, entre otras cosas.

Además de que contestando las preguntas pudimos saber más del cifrado de César como lo es la importancia que tenía su uso durante época de origen y el porqué del nombre, así como el número de sus posibles combinaciones para cifrar bytes e incluso conocer el uso que se le da al cifrado base64 siendo este un tipo cifrado para nada seguro.

4. Referencias

- GeeksforGeeks. (2023). Caesar Cipher in Cryptography. GeeksforGeeks. https://www.geeksforgeeks.org/caesar-cipher-in-cryptography
- Neoteo. (2010, 5 julio). El cifrado de César. Diario ABC. https://www.abc.es/ciencia/cifrado-cesar-201007050000_noticia.html#:~:text=El%20m%C3%A9todo%20ideado%20por%20C%C3%A9sar,usando%20una%20palabra%20clave%20repetitiva
- Euler Phi Function of 256 - ProofWiki. (s. f.). https://proofwiki.org/wiki/Euler_Phi_Function_of_256
- Fernandes, H. M. (2021). ¿Qué es Base64, para qué sirve y cómo funciona? Henrique Marques Fernandes. https://marquesfernandes.com/es/tecnologia-es/que-y-base64-para-que-serve-y-como-funciona/
- Gonzalez, A. (2020). Base64: la codificación más útil en criptografía. Cybersecurity, Payment Security & Cryptography. https://albertx.mx/base64/