UNIVERSIDAD DE SANTIAGO DE CHILE
FACULTAD DE INGENIERÍA
DEPARTAMENTO DE INGENIERÍA INFORMÁTICA

Laboratorio 3: Criptografía

Integrantes:	Alex Muñoz
	Alberto Rodríguez
Curso:	Sistemas de Comunicación
Sección	0-L-1
Profesor:	Carlos González
Ayudante:	Catalina Morales

25 de Septiembre de 2020

Tabla de contenidos

1. Introducción

Dentro de la informática y la telecomunicación, anteriormente vimos los protocolos de comunicaciones, estas consisten como un conjunto de reglas que especifican el intercambio de datos u órdenes durante la comunicación entre sistemas (algunos son el modelo OSI y TCP/IP). Nace ahora la pregunta de como los usuarios envían estos datos de forma segura sin que en el transcurso del envío hayan ataques o interacción de terceros que puedan afectar los datos. De aquí nace lo que se conoce como la seguridad informática, la cual es una disciplina que se encarga de proteger la integridad y la privacidad de la información almacenada en un sistema informático (Pérez, 2008).

Una forma de proteger los datos es mediante la técnica de la criptografía, que consiste en escribir procedimientos o claves secretas para brindar autenticidad del emisor, confiabilidad e integridad de la información. El emisor emite un mensaje, el cual es "escondido" mediante un cifrador con la ayuda de una clave o llave, para crear un texto cifrado. Este texto cifrado, por medio del canal de comunicación establecido, llega al descifrador que convierte el texto cifrado, apoyándose en otra clave, para obtener el texto original. Las dos claves implicadas en el proceso de cifrado/descifrado pueden ser o no iguales dependiendo del sistema de cifrado utilizado.

Uno de los sistemas es el cifrado simétrico, el cual consiste de que existe una única llave que es capaz de encriptar y desencriptar los datos o mensaje

Los objetivos principales es poder diseñar e implementar un cifrado simétrico por bloques, este tiene que ser capaz de cifrar y descifrar. Además se deben realizar pruebas como son el *efecto avalancha*, y para poder evaluar el desempeño,que en los sistemas de encriptación se suele definir como *Throughput*.

Para poder desarrollar todo lo anterior a cabo, se utiliza el lenguaje de programación Python 3 y algunos módulos de utilidad. Estos son:

- PyCrypto: Es una colección de funciones hash seguras y varios algoritmos de cifrado.
- Matplotlib: Es una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays de Python.

2. Marco teórico

2.1. Criptografía simétrica

Es una de la técnicas mas criptográficas mas antiguas y que aún en la actualidad ofrece un alto nivel de seguridad para hacer frente a distintas situaciones de uso. También se le conoce como criptografía de clave secreta o criptografía de una clave, esto se debe a que este método emplea la misma clave tanto para el cifrado cmomo el descrifrado de un mensaje (Mejía, 2020). Por lo que el emisor y el receptor deben estar previamente de acuerdo y en conocimiento de la clave a utilizar. 1.

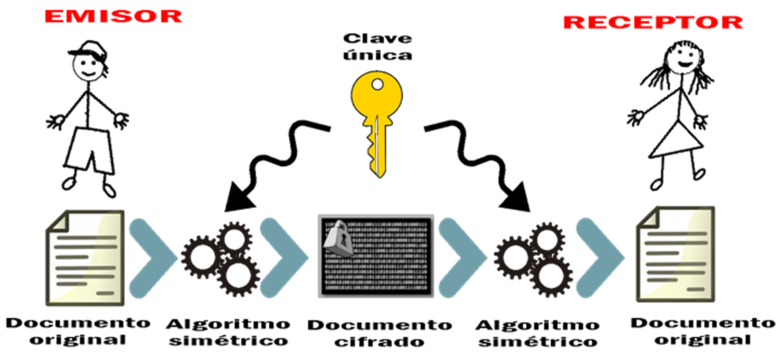

Figura 1: Estructura de cifrado simétrico.

En esta técnica, toda seguridad esta centrada en la clave o llave. Por lo que debe ser secreta y que no sea fácil de adivinar por una tercera persona. Desde el nacimiento de los ordenadores, hoy en día existen muchos algoritmos de cifrado empleados en los correos, en bases de datos e incluso en discos duros. Algunos de los mas conocidos son Data Encryption Standard (DES), cifrador AES o el próximamente nombrado cifrador Feistel.

2.2. Cifrador Feistel

2.2.1. Cifrado de producto

Los algoritmos de cifrado simétrico se apoyan de conceptos de confusión y difusión que se combinan para dar lugar a los denominados *cifrados de productos*. Estas técnicas consisten en básicamente en dividir el mensaje en bloques de tamaño fijo y aplica.r la función

de cifrado en cada uno. La difusión consiste en disipar la estructura estadística del texto plano en el texto cifrado, es decir en ir modificando los bits en repetidas permutaciones, para que cada vez se parezcan menos. El concepto de confusión pretende hacer la relación entra las estadísticas del texto cifrado y el valor de la clave lo más compleja posible, para evitar el descubrimiento de la clave

2.2.2. Cifrado Feistel

Muchos de los cifrados de producto tienen en común que dividen un bloque de longitud n en dos mitades, L y R. Se define entonces un cifrado de producto iterativo en el que la salida de cada ronda se usa como entrada para la siguiente según la relación. Este tipo de estructura se denomina Red de Feistel, y es empleada en multitud de algoritmos, como DES, Lucifer, FEAL, CAST, Blowfish, etc. Para descifrar bastará con aplicar el mismo algoritmo, pero con las K_i en orden inverso. La propuesta de Feistel propuso alterna sustituciones y permutaciones, es una aplicación práctica de una propuesta de Claude Shannon en 1945 para desarrollar un cifrado producto que alterna funciones de confusión y difusión. (Acuayate, 2012).

3. Desarrollo y resultados

Para la creación del sistema de cifrado solicitado se busca la encontrar mayor seguridad posible sin tener mucho en cuenta el tiempo en el cual el texto plano será cifrado en su totalidad. Debido a esto, se buscará un cambio debido al efecto avalancha de aproximadamente un 50 % de bits diferentes al cambiar un solo bit del mensaje original.

Para lograr esto, se implementará una mejora del cifrador Feistel, utilizando cifrado AES (Advanced Encryption Standard) presente en la librería PyCrypto como la función F(K,R) del modelo general de Feistel presente en la figura 2 en conjunto con una cantidad de rondas alta, la cual corresponde a 54 para esta implementación.

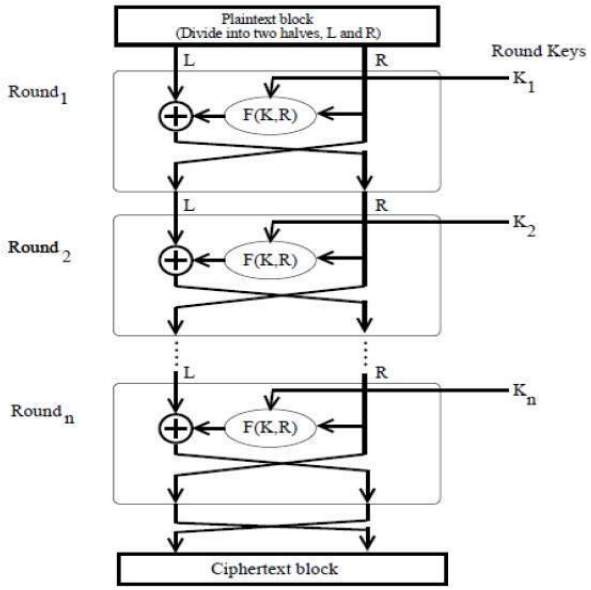

Figura 2: Estructura cifrador Feistel

El algoritmo utilizado sigue exactamente el mismo esquema que en la figura 2, siguiendo el siguiente orden de ejecución

1. Separar texto de entrada en bloques de 32 bytes.
2. Para cada bloque:
   a) Se separa en dos mitades de 16 bytes.
   b) Se encripta con AES la mitad derecha con la llave K_i..

c) Se realiza una operación XOR entre la mitad izquierda y la parte encriptada.

d) Se rotan los bits de la llave hacia la izquierda para generar una nueva llave K_{i+1}.

e) Se concatena la mitad derecha con la parte que realizo la operación XOR.

f) Se repite *a)* un total de 54 veces (la cantidad de rondas estipuladas anteriormente).

g) Se realiza una permutación entre la mitad izquierda y derecha al terminar las 54 rondas.

Para el correcto funcionamiento del algoritmo se necesita lo siguiente:

- Tamaño para la llave: 16 bytes.
- El mensaje a encriptar puede ser de cualquier tamaño distinto de cero.
- Tamaño de bloque: 32 bytes..
- Función de encriptación: Cifrado AES.
- No necesita función de desencriptar, pues debido a las propiedades de la operación XOR, las encriptaciones se cancelan entre sí.

Para probar el efecto avalancha (la función avalanche_test en el código) se creará un bloque que contenga 32 veces el numero uno (11111111111111111111111111111111) y se modificará el ultimo bit del primer byte (dejando como resultado 21111111111111111111111111111111) creando un bloque modificado, luego ambos bloques pasarán por la encriptación utilizando la llave definida en el *main* del código fuente y se contará la cantidad de bits que son distintos entre ambas encriptaciones, con ello se calculará el porcentaje de bits diferentes obteniendo el efecto avalancha.

En el caso de los valores por defecto es el efecto avalancha es igual a el 50 % exacto esperado, pero al cambiar la llave el resultado varía entre 45 % y 55 %.

```
El cambio por el efecto avalancha según la cantidad de bits diferentes (con 54 rondas) es de: 46.09375 %
El cambio por el efecto avalancha según la cantidad de bytes diferentes (con 54 rondas) es de: 100.0 %
```

Figura 3: Salida por defecto del test de efecto avalancha.

Para calcular el throughput simplemente se toman los tiempos de cada bloque, se colocan todos en un arreglo y se le calcula el promedio de ellos (la función *plot_throughput* en el código). A partir de esto, se calcula el throughput promedio al aumentar el tamaño de bloque de 32 en 32 hasta llegar a aumentarlo 100 veces tanto para la encriptación como la desencriptación, obteniendo el resultado de la figura 4. Cómo se puede ver en dicho gráfico, el throughput del algoritmo no es muy alto, debido a la cantidad de vueltas y al cifrado AES que utiliza.

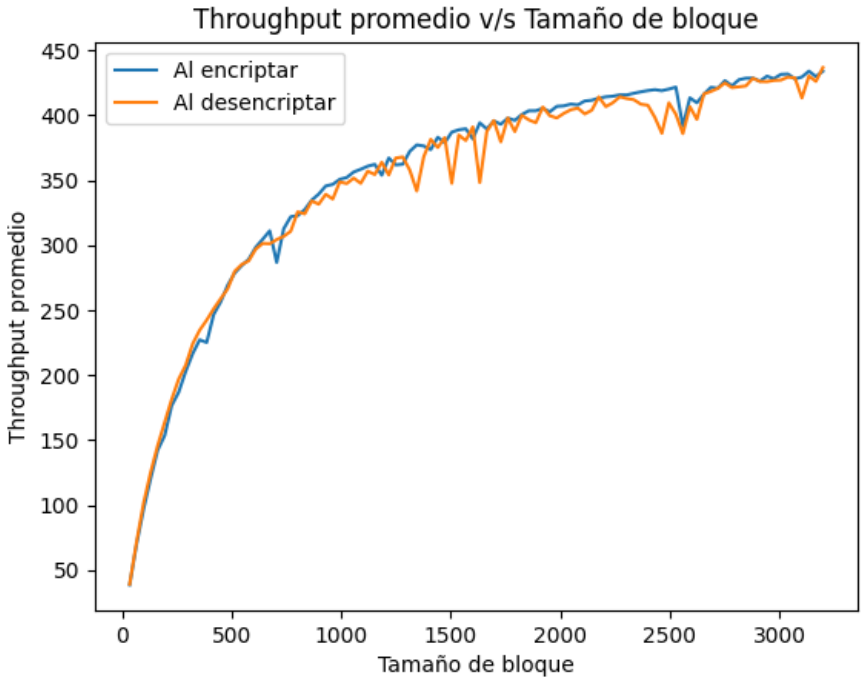

Figura 4: Gráfico de throuhput (Tamaño bloque en KB / tiempo de encriptación) v/s tamaño de bloque (en bytes) en encriptación y desencriptación.

Para comprobar qué sucede si se utiliza una llave incorrecta al desencriptar, se utiliza la función *change_key_test* del código fuente, la cual cambia el último bit del primer byte de la llave y almacenando el resultado en el archivo ÇambioDeLlave.txt". Cómo se puede ver en las figuras 5 y 6 el simple hecho de cambiar un sólo bit de la llave provoca que el texto descifrado sea ininteligible.

A partir de todo lo desarrollado anteriormente, se logra alcanzar el propósito de llegar a un algoritmo capaz de proveer seguridad debido a lo demostrado anteriormente por el

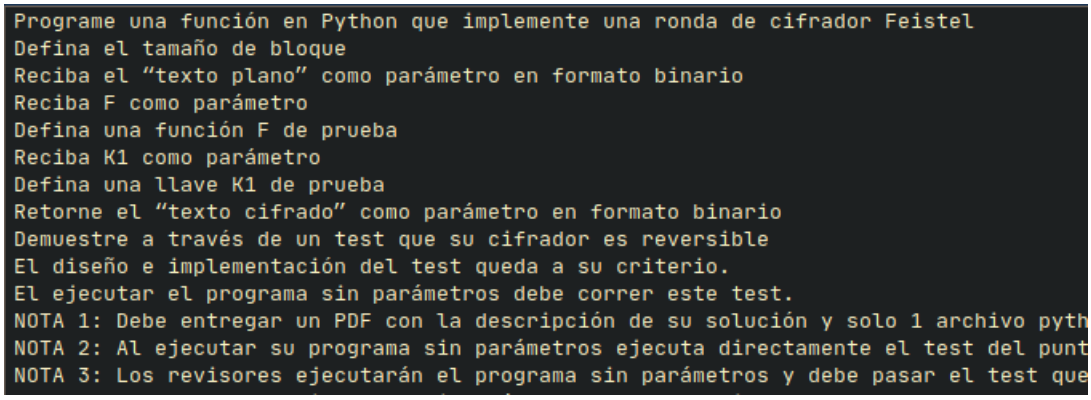

Figura 5: Resultado de la desencriptación con llave correcta (Salida.txt).

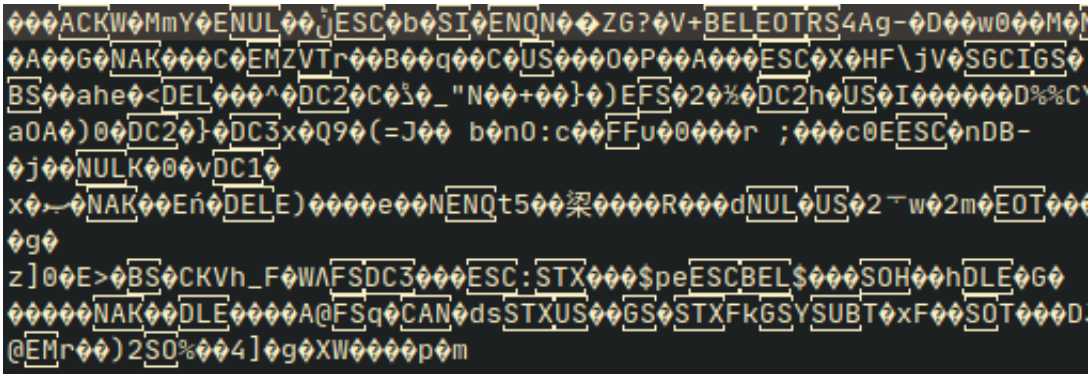

Figura 6: Resultado de la desencriptación con llave incorrecta (CambioDeLlave.txt).

efecto avalancha, dando a cambio eficiencia y tiempo de ejecución para lograr dicho objetivo. Esta implementación, cumple con la misma estructura que un cifrador Feistel, pero con una gran desventaja, la cual es que la función F corresponde a otro cifrador (AES), incrementando de gran manera los tiempos de encriptación y desencriptación según el throughput (que corresponde a aproximadamente 32 para el código fuente por defecto), valor que es muy bajo.

4. Análisis de resultados

4.1. Análisis del algoritmo Feistel

Primero se utilizó un cifrado visto en clases para realizar el cifrado simétrico, el cual es el cifrado Feistel con alguna modificaciones como se explicó anteriormente. Se puede analizar que el algoritmo fue realizado que gran manera, primero porque que cumple con la estructura mencionada (cifrado Feistel), es decir el de ir dividiendo el texto por bloques, para después a cada uno de estos separarlo en dos y realizar la función F (cifrador AES) y la operación XOR. Segundo, por los resultados que se obtuvieron, se pudo crear un texto cifrado ininteligible, como también el texto descifrado es el mismo que el que fue enviado inicialmente, por lo tanto, el algoritmo realiza correctamente el proceso de cifrado y descifrado con las rondas que se le indiquen.

Si algún tercero malicioso quisiera averiguar lo que dice el texto cifrado le costaría bastante, porque, tendría que saber que algoritmo se utilizó, saber cuantas iteraciones hizo el algoritmo para cifrar, y debe tener un conocimiento exacto de la llave. Como se vio en el desarrollo un minúsculo cambio en la llave, el impostor no podría obtener el mensaje oculto.

4.2. Análisis del efecto avalancha y Throughput

Al realizar el efecto avalancha se puede ver analizar lo efectivo que es el algoritmo al encriptar, porque, al ver el porcentaje de bits diferentes que arroja el efecto a avanlancha se ve que tan distintos (porcentaje) son el texto real con el cifrado.

Por otra parte, en el throughput al mirar el gráfico, se llega al resultado lógico de que a medida que se aumenta el tamaño del bloque el throughput aumenta. Este también es muy bajo debido a que como se analizó anteriormente, al implementar un cifrador AES como función F, incrementa de gran manera los tiempos de encriptación y desencriptación.

5. Conclusiones

Luego de la experiencia realizada pudimos analizar y entender de mejor manera los conceptos de criptografía, así como cifrado simétrico. Además del funcionamiento e implementación del algoritmo de Feistel, utilizando las herramientas mencionadas a lo largo del laboratorio.

Al obtener los resultados del algoritmo, pudimos comprender la eficiencia y precisión que este tiene a pesar de que puede iterar muchas veces, si se realiza el algoritmo correctamente, devuelve el texto inicial de forma exacta. Incluir también la importancia de la llave, ya que, sin esta o si esta varía, no se puede obtener el mensaje, dando seguridad a nuestro mensaje.

Pudimos ver de forma "real" como funciona un proceso de cifrado simétrico, y que al relacionarlo con lo visto en clases, de como con un algoritmo específico y una llave privada, se puede encriptar correctamente un mensaje.

Todo lo anterior se pudo analizar gracias al lenguaje de programación Python, juntos con sus librerías muy completas (PyCrypto y Matplotlib), que gracias a ella se puede realizar un cifrado correcto y poder ver de mejor manera los resultados obtenidos.

Finalmente, podemos decir que gracias al este tercer laboratorio hemos reforzado nuestro conocimientos acerca de lo que sería un inicio a la seguridad informática y como al utilizar la criptografía se pueden ocultar los mensajes que se quieren enviar. Se espera que en un futuro curso de la carrera, poder profundizar más los temas de seguridad informática vistos junto con otros nuevos, ya que, es un área muy interesante que nos interesa conocer más.

Bibliografía

Acuayate, L. (2012). *Algoritmo de Feistel*.

Mejía, J. P. (2020). *Qué es la criptografía simétrica*.

Pérez, J. (2008). *Definición de seguridad informática*.

Stallings, W. (2007). *Criptografía y Seguridad de la Red Principios y Práctica*, volume 5. Prentice Hall.