17 DE ENERO DE 2023

CREACIÓN DE UN RANSOMWARE

TOMÁS GARCÍA TOBARRA

Universidad de Almería

INDICE

TAREA 1: PROYECTO DE CREACIÓN DE UN RANSOMWARE

Para la realización de nuestro malware utilizaremos:

-**Sistema operativo:** Estará dedicado a Windows por las siguientes razones:

- Es el sistema operativo más utilizado por los usuarios.
- Estamos más familiarizados con él.
- Es más probable que la gente no lo tenga actualizado por lo que tendrán más posibilidad de ser vulnerables.

-Plataforma de desarrollo: Python ya que es un lenguaje de programación con una sintaxis sencilla y es el más utilizado para la creación de malware y contiene muchas funciones para el campo de la ciberseguridad. Permite escribir código en tiempo récord. Además, puede ejecutar prácticamente cualquier acción, desde verificar la integridad de los sistemas corporativos hasta automatizar la mayoría de los programas de piratería.

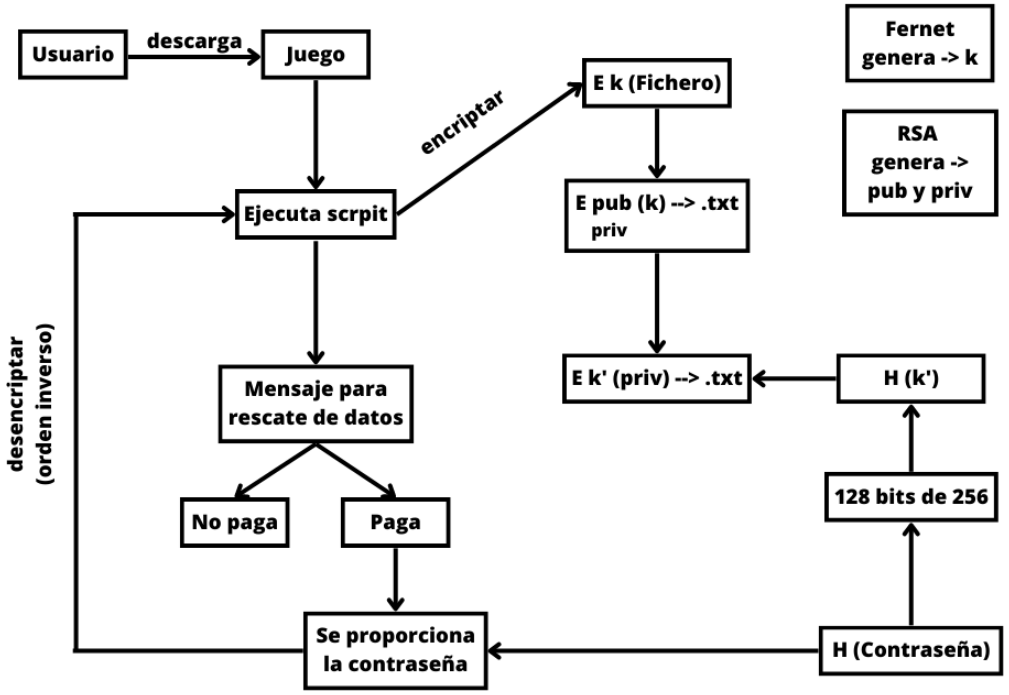
-Librerías: En un principio habíamos pensado utilizar Cryptography y PyCrypto pero conforme fuimos avanzando en el proyecto vimos que deberíamos utilizar más las cuales se mostrarán en las siguientes tareas.

-Ficheros objetivos: El script cifrará todo tipo de archivos de un directorio concreto que será Documentos.

-Modo ejecución (activo o pasivo): La primera idea fue a través de un pdf con el cual de forma activa se ejecutaría en segundo plano el malware, pero finalmente lo hemos hecho a través de un juego, que será el tetris, y mientras el usuario esté jugando, el malware se ejecutará en segundo plano así evitamos que el usuario al descargar el juego y ver que su ordenador vaya más lento pueda intuir que le hemos introducido un malware.

TAREA 2: DIAGRAMA DE BLOQUES DE LA APLICACIÓN

Primero explicaremos la ejecución del malware desde la vista del usuario y después nos adentraremos en las claves que usan y cómo se cifran los mensajes.



El usuario descargará el juego que contiene el malware, una vez lo abra y comience a jugar se ejecutará el script en segundo plano y empezará a cifrar todos los archivos del directorio documentos. Nuestro malware no se detendrá hasta haber cifrado todos los archivos, aunque el usuario se salga de la ventana del juego y una vez finalizado abrirá una ventana con el mensaje de rescate de datos el cual pedirá que se abone una cierta cantidad de dinero para que el usuario pueda recuperar sus datos.

En primer lugar, se encripta el fichero con una clave K la cual la generaremos con el método Fernet. Con RSA se genera una clave pública y otra privada. Luego se encripta la clave K con la clave pública y se depositará en un archivo .txt, para desencriptarlo se usará la clave privada. A continuación, crearemos una contraseña a nuestra elección y le aplicaremos un hash con SHA256 donde obtendremos una clave K'. Con esta clave se encriptará la clave privada y se depositará en otro archivo .txt.
Todo esto ocurre en el proceso de encriptación cuando el usuario ejecuta el juego, para desencriptar simplemente habría que seguir el proceso inverso. Si el usuario ingresa mal la contraseña que le hemos proporcionado, el proceso de desencriptación se realizará pero se obtendrán claves erróneas por lo que no podrá recuperar sus archivos.

TAREA 3: PRIMERA VERSIÓN. ENCRIPTADO CON CRIPTOSISTEMA SIMÉTRICO

Cómo se produce la infiltración

Método

Para la infiltración necesitamos tener el código subido a la nube junto con el juego. El sitio web escogido es mega, uno de los más usados por la gente a la hora de subir juegos para que otros los descarguen. Una vez el juego con el código subido, mandaremos el enlace de los archivos a la víctima por correo electrónico.

Parece una tarea fácil, pero hay que conocer muy bien a la víctima y hacer un correo creíble a primera vista. El correo debe parecer lo más creíble y profesional posible para que la víctima no sospeche de él.

Otros métodos

Otros métodos de infección usuales son los ficheros de clase P2P. Un ejemplo que puede comprender todo el mundo es el torrent. Torrent no utiliza un modelo cliente servidor si no que los computadores están comunicados entre sí para aprovechar el máximo ancho de banda posible a la hora de intercambiar archivos. El problema es que es muy fácil ser infectado con cualquier tipo de malware si usamos este sistema.

Cómo se consigue la ejecución.

Para la ejecución del script vamos a utilizar un juego muy popular como lo es el tetris. El script comenzará una vez se comience la partida. De momento la idea es que el algoritmo comience a encriptar desde el momento de ejecución. Solo nos quedaría por investigar un poco más las opciones a las que podríamos recurrir en caso de que el usuario termine de jugar y por alguna razón no se haya encriptado todos los archivos deseados. También nos falta por determinar que ocurriría en caso de que se produjera algún corte repentino del suministro de luz y se tuviera que reiniciar el computador.

Qué tipos de ficheros se encriptan.

Se encriptan documentos de texto, libros de excel, documentos word, pdfs e imágenes. En el caso de los ficheros de texto, el contenido del mismo se encuentra cifrado mientras que en los demás ficheros lo que ocurre es que al abrirlos no se muestra el contenido, si no que se indica que se ha producido un error al abrir esos archivos.

Criptosistema

Descripción

El criptosistema utilizado es el criptosistema simétrico. Este criptosistema utiliza una única clave que deben conocer tanto emisor como receptor. La clave es producto de un acuerdo mutuo entre el emisor y el receptor. Una vez acordada la clave, el mensaje se cifra y descifra con ella.

Con este sistema no nos importa que algún atacante conozca el algoritmo ya que la seguridad recae completamente en la clave. A su vez, debemos de tener un abanico lo más amplio posible de posibilidades a la hora de crear la clave para que esta sea lo más difícil posible de adivinar.

Algoritmos

AES es un algoritmo de cifrado por bloques. Se toma un bloque de texto plano y se le aplican rondas alternas de cajas de sustitución y permutación. Estas cajas están en 128, 192 o 256 bits; esto determina la fuerza del cifrado.

Durante el proceso de sustitución-permutación se genera una clave de cifrado.

Algunos ejemplos más son el 3DES, DES, RC5…

Detalles de la clave generada; dónde se almacena la clave y cómo se accede a la misma

El tamaño en bits de la clave puede tener tres longitudes (128, 192 o 256 bits).
La clave se almacena en la máquina de la víctima, concretamente en el mismo directorio en el que se encuentra el script. Para acceder a ella basta con abrir el fichero que la contiene y nos la muestra.

Medios y/o librerías utilizados

Fernet

Descripción

Fernet es una librería que pertenece al paquete Cryptography. Esta librería nos asegura que el mensaje cifrado no puede ser manipulado a menos que se tenga la llave con la que se cifró. Es una librería que se basa en criptografía simétrica, perfecta para desarrollar nuestro malware.
La librería nos da la opción tanto de encriptar como de desencriptar.

Métodos

Los métodos principales de esta clase son:

- generate_key(): Con este método podemos generar una clave con la que podemos encriptar y desencriptar los archivos. Podemos manipular donde queremos guardar la clave. Las características de la clave ya han sido descritas anteriormente.
- encrypt(data): Este método encripta los datos pasados por parámetro y genera un token (que contiene la información en texto plano del archivo original) totalmente seguro el cuál es necesario para el proceso de recuperación.
- decrypt(token): Se desencripta el token con la clave. Si todo ha funcionado correctamente, obtendremos la información cifrada en el token.

OS

Descripción

La librería os con la que podemos manipular rutas, directorios, archivos… Una librería ideal para poder indicar qué ficheros vamos a encriptar.

Métodos

Algunos de los métodos más interesantes y que hemos utilizado son:

- open(): Con este método podemos abrir el “file” pasado por parámetro. También hay que indicar por parámetro si el archivo que vamos a abrir va a ser de escritura, de lectura, de escritura y lectura, binario…
- close(): Cierra el archivo abierto con open().

TAREA 4: SEGUNDA VERSIÓN. CRIPTOSISTEMA SIMÉTRICO CON CLAVE PÚBLICA

Funcionamiento

El funcionamiento del programa consta de un cifrado híbrido haciendo uso de criptografía simétrica para cifrar los ficheros y posteriormente hacer uso de la criptografía asimétrica para encriptar la clave y así asegurarnos de que la víctima no puede recuperar sus archivos sin antes obtener autorización de nuestra parte.

Encriptado Simétrico

Para encriptar los ficheros hacemos uso de la ya explicada criptografía simétrica, mediante la librería Fernet de Cryptography para el lenguaje Python. El método está ya explicado en la anterior tarea y por resumir básicamente se hace uso de un algoritmo AES por el cual se genera una clave aleatoria que es guardada en el sistema de la víctima, con esa clave es con la que se cifran y, también, con la que se pueden descifrar los ficheros.

Este método dejaba en el aire bastantes vulnerabilidades, la más llamativa es que la clave se guarda en el dispositivo de la víctima. Por ello es por lo que surge la necesidad de utilizar otros sistemas para ocultar esa clave.

Encriptado Asimétrico

Para poder ocultar esa clave hacemos uso de un encriptado asimétrico. Como introducción, el criptosistema asimétrico es un criptosistema de dos claves, una privada y una pública. Las dos claves son generadas por el destinatario. Éste le comunica la clave pública a la persona con la que se quiere comunicar dándole la posibilidad de encriptar el mensaje que le quiera comunicar. Si esta persona decide cifrar, el mensaje quedará totalmente cifrado y sólo podrá recuperarse con la clave privada que posee el destinatario.

Un ejemplo muy claro es el siguiente:

Supongamos que Ana quiere enviar a David un mensaje secreto que solo él pueda leer.

Primero, David envía a Ana una caja abierta, pero con cerradura, cerradura que se bloqueará una vez se cierre la caja, y que solo podrá abrirse con una llave, que solo David tiene. Ana recibe la caja, escribe el mensaje, lo pone en la caja y la cierra con su cerradura (ahora Ana ya no podrá abrir la caja para acceder de nuevo al mensaje). Finalmente, Ana envía la caja a David y este la abre con su llave. En este ejemplo, la caja con la cerradura es la «clave pública» de David, y la llave de la cerradura es su «clave privada».

1. Ana redacta un mensaje.
2. Ana cifra el mensaje con la **clave pública** de David.
3. Ana envía el mensaje cifrado a David a través de internet, ya sea por correo electrónico, mensajería instantánea o cualquier otro medio.
4. David recibe el mensaje cifrado y lo descifra con su **clave privada**.
5. David ya puede leer el mensaje original que le mandó Ana.

Este procedimiento es el usado para encriptar la clave simétrica de nuestro programa de tal manera que solo nosotros dispondremos de la clave privada para descifrar la clave simétrica y así poder descifrar finalmente los ficheros.

Librerías

Encriptado Simétrico

La librería principal usada para este encriptado es la librería Fernet ya explicada en la anterior tarea.

Encriptado Asimétrico

- PKCS1_OAEP (Paquete Crypto.Cipher):

Método new()

Parámetros:

- **key** (objeto de clave *RSA): objeto de clave* que se va a utilizar para cifrar o descifrar el mensaje. El descifrado sólo es posible con una clave RSA privada.
- **hashAlgo** (objeto hash): función *hash* que se va a utilizar. Puede ser un módulo bajo Crypto.Hash o un objeto hash existente creado a partir de cualquiera de dichos módulos. Si no se especifica, se utiliza Crypto.Hash.SHA1.
- **mgfunc** (*callable*): una función de generación de máscaras que acepta dos parámetros: una cadena para usar como semilla y la longitud de la máscara para generar, en bytes. Si no se especifica, se utiliza el MGF1 estándar consistente con (una opción segura).
- **label** (*bytes/bytearray/memoryview*): etiqueta que se aplicará a este cifrado concreto. Si no se especifica, se utiliza una cadena vacía. Especificar una etiqueta no mejora la seguridad.
- **randfunc** (*llamable*): función que devuelve bytes aleatorios. El valor predeterminado es Random.get_random_bytes.

Método encrypt()

Parámetros:	**Mensaje** (*bytes/bytearray/memoryview*) – El mensaje a cifrar, también conocido como texto plano. Puede ser de longitud variable, pero no más larga que el módulo RSA (en bytes) menos 2, menos el doble del tamaño de salida hash. Por ejemplo, si usa RSA 2048 y SHA-256, el mensaje más largo que puede cifrar tiene una longitud de 190 bytes.
Devuelve:	El texto cifrado, tan grande como el módulo RSA.
Tipo de devolución:	Bytes
Plantea:	**ValueError** – si el mensaje es demasiado largo.

Método decrypt()

Parámetros:	**ciphertext** (*bytes/bytearray/memoryview*): mensaje cifrado.
Devuelve:	El mensaje original (texto sin formato).
Tipo de devolución:	Bytes
Plantea:	• **ValueError**: si el texto cifrado tiene una longitud incorrecta o si el descifrado no supera la comprobación de integridad (en cuyo caso, la clave de descifrado probablemente sea incorrecta).

- **TypeError**: si la clave RSA no tiene la mitad privada (es decir, está intentando descifrar con una clave pública).

- Libreria RSA (Paquete Crypto.PublicKey)

Método generate()

Parámetros:

- bits (int) - Longitud de clave, o tamaño (en **bits**) del módulo RSA. Debe ser un múltiplo de 256 y no menor que 1024.
- **randfunc** (llamable) - Función de generación de números aleatorios; debe aceptar un solo entero N y devolver una cadena de datos aleatorios de N bytes de longitud. Si no se especifica, se creará una instancia de una nueva desde Crypto.Random.
- **progress_func** (llamable) - Función opcional que se llamará con una cadena corta que contiene el parámetro clave que se está generando actualmente; Es útil para aplicaciones interactivas donde un usuario está esperando que se genere una clave.
- **e** (int) - Exponente público de RSA. Debe ser un entero positivo impar. Es típicamente un número pequeño con muy pocos en su representación binaria. El valor predeterminado 65537 (= 0b10000000000000001) es una opción segura: otros valores comunes son 5, 7, 17 y 257.

Devuelve:

Un objeto de clave RSA (`RSAobj`).

Plantea:

- **ValueError** - Cuando **los bits** son demasiado pequeños o no son múltiplos de 256, o cuando **e** no es impar o menor que 2.

Atención:

- Siempre debe utilizar un generador de números aleatorios criptográficamente seguro, como el definido en el módulo Crypto.Random; **No** solo use la hora actual y el módulo aleatorio.
- Exponent 3 también es ampliamente utilizado, pero requiere un cuidado muy especial al rellenar el mensaje.

Método importKey()

Importa una clave RSA (mitad pública o privada), codificada en forma estándar.

Consulte `RSAImplementation.importKey.`

Parámetros:

- **externKey** (cadena): clave RSA que se va a importar, codificada como cadena.
  Una clave pública RSA puede tener cualquiera de los siguientes formatos:
  - X.509 DER SEQUENCE (codificación binaria o PEM)subjectPublicKeyInfo
  - PKCS#1 DER SEQUENCE (codificación binaria o PEM)RSAPublicKey
  - OpenSSH (solo clave pública textual)
- Una clave privada RSA puede tener cualquiera de los siguientes formatos:
  - PKCS#1 DER SEQUENCE (codificación binaria o PEM)RSAPrivateKey
  - PKCS#8 DER SEQUENCE (codificación binaria o PEM)PrivateKeyInfo
  - OpenSSH (solo clave pública textual)
- Para obtener más información sobre la codificación PEM, consulte RFC1421/RFC1423.
  En el caso de la codificación PEM, la clave privada se puede cifrar con DES o 3TDES de acuerdo con una determinada frase de contraseña. Solo se admiten frases de contraseña compatibles con OpenSSL.

frase de contraseña (cadena): en el caso de una clave PEM cifrada, esta es la frase de contraseña de la que se deriva la clave de cifrado.

TAREA 5: MÉTODO DE FIRMA Y VERIFICACIÓN

Funcionamiento

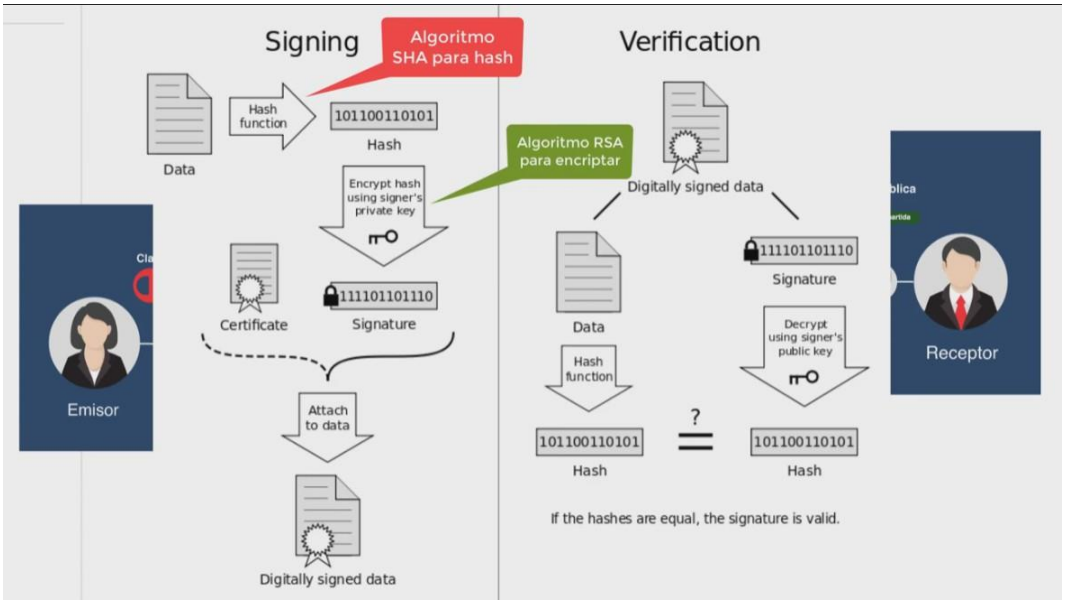
El funcionamiento del programa consta de un cifrado haciendo uso del algoritmo SHA para hash y el algoritmo RSA para encriptar y así asegurarnos de que el documento es auténtico y existe certeza sobre la persona que lo ha elaborado, manuscrito, firmado, o cuando exista certeza respecto de la persona a quien se atribuya el documento.

Método Firmar

El primer paso es hacerle un hash al archivo objetivo (en nuestro caso utilizaremos la función SHA).
Cuando ya tengamos nuestro hash lo que demos hacer es encriptarlo con nuestra clave privada mediante RSA.
Con esto ya tendríamos la firma, ya solo bastaría con adjuntarla al archivo y tendríamos nuestro archivo firmado.

Método Verificar

Una vez tengamos el archivo firmado, para verificar que no ha sido modificado lo que debemos hacer es separar los datos de la firma
La firma la desencriptaríamos con la clave pública
A los datos le haríamos el hash
Teniendo el hash de los datos y la firma desencriptada compararíamos los hashes y si son iguales significa que la firma es válida y no han sido modificados.



Librerías

· La librería Crypto para la importación de RSA y PKCS1_v1_5

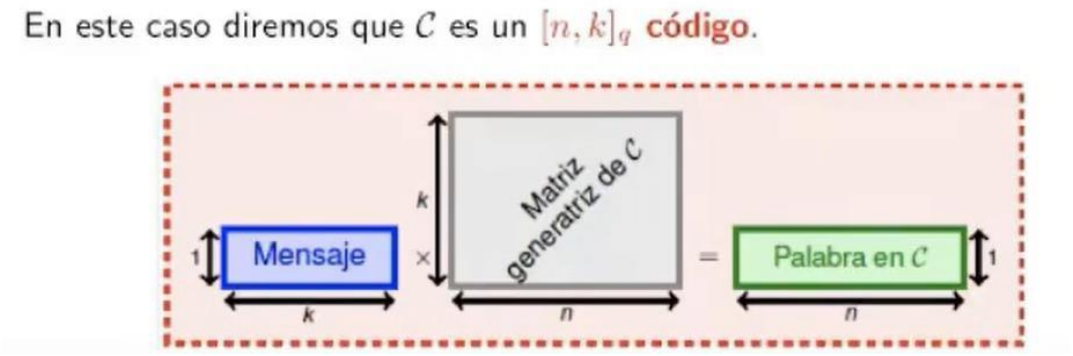
· Hash para el método SHA

· La librería “os” con la que podemos manipular rutas, directorios, archivos…

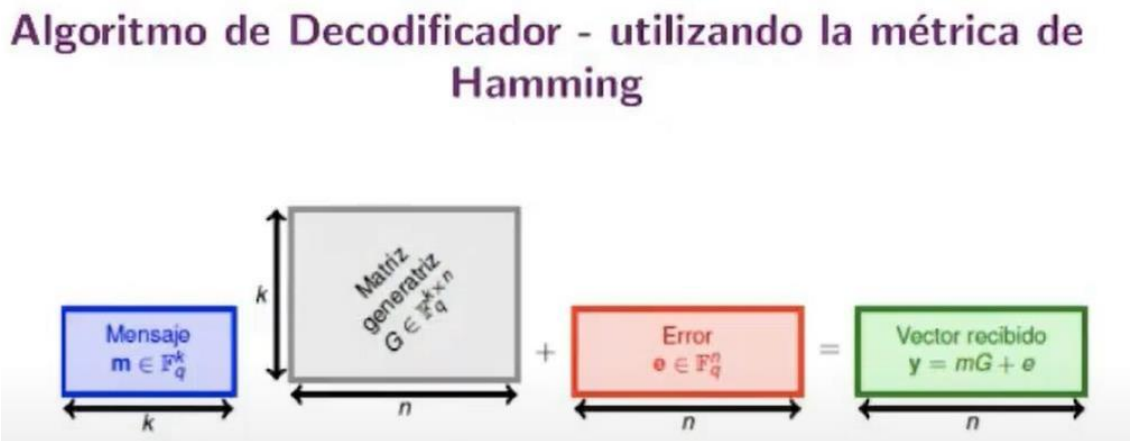
TAREA 6: DEFINICIÓN DE UN CÓDIGO CON CAPACIDAD DE DETECCIÓN Y CORRECCIÓN

Funcionamiento

El programa es un código Hamming que permite la codificación y decodificación de un mensaje compuesto por dígitos binarios, además de poder corregir 1 bit de error en el mensaje. Para ello se introduce el mensaje en binario y se multiplica por la matriz generadora para obtener la palabra código.



Una vez codificada la información el siguiente paso es detectar y corregir los errores en la transmisión, en el proceso conocido como decodificación. Esto consiste en encontrar la palabra del codigo que se parece más a la palabra recibida.



Métodos utilizados

def calcRedudant Bits(m)

Utiliza la fórmula 2 ^ r >= m + r + 1, donde "r" es el número de bits redundantes y "m" es el número de bits en el mensaje original.
La función itera sobre los valores de 0 a m, y devuelve el primer valor de "r" que satisface la ecuación.

m: el número de bits en el mensaje original antes de agregar bits de redundancia.
i: un contador que se utiliza para iterar sobre los valores de 0 a m.
La función devuelve el primer valor de "i" que satisface la ecuación 2 ^ i >= m + i + 1. Este valor se utiliza como el número de bits de redundancia necesarios para codificar el mensaje original.

def posRedundantBits(data, r)

Inserta bits de redundancia en posiciones específicas dentro de una cadena de datos.
Los bits de redundancia se insertan en las posiciones que corresponden a las potencias de 2 (es decir, 1, 2, 4, 8, etc.).
Los bits de datos se insertan en las posiciones restantes
La función devuelve la cadena de datos con los bits de redundancia insertados en las posiciones correctas

data: la cadena de datos original en la que se insertarán los bits de redundancia.
r: el número de bits de redundancia que se insertarán en la cadena de datos.
j: un contador que se utiliza para iterar sobre las posiciones de los bits de redundancia.
k: un contador que se utiliza para iterar sobre las posiciones de los bits de datos.
m: el número de bits de datos en la cadena original.
res: la cadena de datos resultante con los bits de redundancia insertados en las posiciones correctas.

def calcParityBits(arr, r)

Calcula los bits de paridad necesarios para una cadena de datos con bits de redundancia insertados.
La función itera sobre cada bit de paridad (de 0 a r-1, donde r es el número de bits de redundancia) y
realiza una operación OR bit a bit con los valores de la cadena de datos en las posiciones correspondientes.
Luego concatena el valor del bit de paridad en la cadena de datos en la posición correcta y devuelve la cadena resultante.
arr: la cadena de datos con los bits de redundancia insertados en las posiciones correctas.
r: el número de bits de redundancia en la cadena de datos.
n: el número total de bits en la cadena de datos (incluyendo bits de redundancia y datos).
i: un contador que se utiliza para iterar sobre los bits de paridad.
j: un contador que se utiliza para iterar sobre los bits en la cadena de datos.
val: el valor del bit de paridad calculado en cada iteración.

def detectError(arr,nr)

Realiza un segundo cálculo de los bits de paridad en la cadena de datos y luego convierte el resultado a un número decimal.
Si el número decimal resultante es mayor que 0, significa que se ha detectado un error en la cadena de datos y se necesita corregir
Si el resultado es 0, significa que no se ha detectado ningún error

arr: la cadena de datos codificada con bits de redundancia en la que se detectará un error (si existe).
nr: el número de bits de redundancia en la cadena de datos.
n: el número total de bits en la cadena de datos (incluyendo bits de redundancia y datos).
res: el resultado de volver a calcular los bits de paridad en la cadena de datos.
i: un contador que se utiliza para iterar sobre los bits de paridad.
j: un contador que se utiliza para iterar sobre los bits en la cadena de datos.
val: el valor del bit de paridad calculado en cada iteración.

def encode(data)

La función encode() tiene como objetivo codificar los datos utilizando el código de Hamming. Para ello, primero se calcula el número de bits redundantes necesarios con la función calcRedundantBits(). Luego, se determinan las posiciones de los bits redundantes con la función posRedundantBits(). Finalmente, se calculan los bits de paridad con la función calcParityBits() y se devuelve la lista arr, que contiene los bits de datos y los bits redundantes.

data: Es una lista que contiene los datos que se quieren codificar utilizando el código de Hamming.

m: Es el número de bits de datos en data. Se calcula con la función len(), que devuelve el tamaño de la lista.

r: Es el número de bits redundantes que se necesitan para utilizar el código de Hamming. Se calcula con la función calcRedundantBits().

arr: Es una lista que contiene los bits de datos y los bits redundantes. Se determinan las posiciones de los bits redundantes con la función posRedundantBits() y luego se calculan los bits de paridad con la función calcParityBits().

calcRedundantBits(), posRedundantBits() y calcParityBits() son funciones que se utilizan para calcular el número de bits redundantes necesarios, determinar las posiciones de los bits redundantes y calcular los bits de paridad, respectivamente.

def decode(encoded_data)

La función decode() tiene como objetivo decodificar una secuencia de bits codificada con el código de Hamming. La secuencia de bits se pasa como argumento a la función a través de la variable encoded_data.

La función comienza inicializando la variable arr con encoded_data. Luego, se calcula el número de bits redundantes necesarios con la función calcRedundantBits() y se determinan las posiciones de los bits redundantes con la función posRedundantBits().

A continuación, se eliminan los bits de paridad de arr iterando a través de un rango que va desde 0 hasta r (que es el número de bits redundantes) y eliminando cada bit de paridad de la secuencia de bits.

Por último, se eliminan los bits redundantes de la secuencia de bits original y se devuelve la variable data, que es la secuencia de bits original sin los bits redundantes ni los bits de paridad.

encoded_data: Es una lista que contiene la secuencia de bits codificada con el código de Hamming. Esta secuencia de bits incluye tanto los bits de datos originales como los bits redundantes y los bits de paridad.

arr: Es una lista que se inicializa con encoded_data y luego se modifica a medida que se procesa la secuencia de bits. Al final de la función, arr contiene la secuencia de bits original sin los bits redundantes ni los bits de paridad.

m: Es el número de bits de datos originales en la secuencia de bits. Se calcula con la función len(), que devuelve el tamaño de la lista.

r: Es el número de bits redundantes que se utilizaron para codificar la secuencia de bits. Se calcula con la función calcRedundantBits().

calcRedundantBits() y posRedundantBits() son funciones que se utilizan para calcular el número de bits redundantes necesarios y determinar las posiciones de los bits redundantes, respectivamente.

TAREA 7: TERCERA VERSIÓN. CRIPTOSISTEMA DE MCELIECE

Funcionamiento

Para empezar, haremos una breve explicación del funcionamiento que tiene cada clase del código que hemos implementado para el funcionamiento del criptosistema de McEliece.

Clase McEliece.py

Este código implementa un sistema de cifrado y descifrado utilizando el código de Hamming y la eliminación de Gauss.

El código comienza definiendo cuatro matrices, g, h, s y p. Luego, realiza algunas operaciones con estas matrices y las imprime. La matriz g es la matriz generadora del código de Hamming, la matriz h es la matriz de paridad, la matriz s es una matriz de permutación y la matriz p es una matriz invertible.

Luego, se define un mensaje m y un vector de error e. Se realiza el cifrado del mensaje sumando el mensaje y el vector de error y se imprime el resultado. A continuación, se calcula la inversa de p y se imprime.

A continuación, se calcula la clave privada con el error utilizando la inversa de p. Luego, se calcula el síndrome multiplicando la clave privada con el error por la matriz de paridad traspuesta y se imprime el resultado.

Finalmente, se llama a la función eliminaciónGauss para resolver el sistema de ecuaciones y se imprime el resultado. La función eliminaciónGauss se utiliza para calcular el vector de error a partir del síndrome y, a continuación, se suma el vector de error al mensaje cifrado para obtener el mensaje original.

Clase Encriptar.py

Este código implementa el cifrado de clave simétrica de Fernet de la biblioteca de cryptography de Python para cifrar y descifrar archivos.

El código comienza definiendo varias funciones. La primera función, generate_key(), genera una clave y la guarda en un archivo. La segunda función, cargar_key(), carga la clave del archivo y la devuelve. La tercera función, encriptarClaveFernet(), cifra la clave utilizando el código de Hamming y el cifrado de clave simétrica de Fernet. La cuarta función, desencriptarClaveFernet(), descifra la clave utilizando el código de Hamming y el cifrado de clave simétrica de Fernet.

Luego, se define una ruta de acceso para guardar la clave y se llama a la función generate_key() para generar y guardar la clave. A continuación, se carga la clave y se cifra utilizando la función encriptarClaveFernet(). Finalmente, se descifra la clave utilizando la función desencriptarClaveFernet().

Clase Desencriptar.py

Este código implementa el algoritmo de cifrado Fernet de la biblioteca de cryptography de Python para descifrar archivos encriptados previamente.

La función cargar_key() se utiliza para leer la clave privada desde un archivo en el disco y devolverla. Luego, la función desencriptarClaveFernet() lee el contenido del archivo que contiene la clave privada, la divide en bloques de 7 caracteres y los convierte en una lista de listas de dígitos binarios. A continuación, utiliza la función McEliece.decode() para decodificar cada uno de estos bloques y almacenar el resultado en una lista listaDef. Después, convierte listaDef en un vector y utiliza el módulo base64 de Python para convertir los datos binarios en una cadena de caracteres en base 64. Finalmente, imprime la cadena decodificada de la clave privada por consola.

La función decrypt() se utiliza para descifrar una lista de archivos utilizando la clave privada decodificada. Recorre la lista de archivos, lee cada archivo, descifra los datos utilizando la clave privada decodificada y escribe los datos descifrados de vuelta al archivo.

El código principal, al final del archivo, llama a la función desencriptarClaveFernet() y luego a la función decrypt() para descifrar los archivos en un directorio especificado.

Medios y/o librerías utilizadas

Para ser más breves, hemos utilizado los mismos medios y librerías de la Tarea 3 que hemos comentado previamente.

También cabe mencionar, la libreria numpy que nos da soporte para crear vectores y matrices grandes multidimensionales, junto con una gran colección de funciones matemáticas de alto nivel para operar con ellas.

Método eliminacionGauss

Este código implementa una versión de la eliminación de Gauss para resolver sistemas de ecuaciones lineales en el campo modular 2. La eliminación de Gauss es un método para resolver sistemas de ecuaciones lineales de la forma Ax = b, donde A es una matriz de coeficientes y b es el vector de términos independientes.

La función eliminaciónGauss toma dos argumentos, A y b, que son la matriz de coeficientes y el vector de términos independientes, respectivamente. Luego, utiliza un bucle for para recorrer cada fila de la matriz A y realizar el proceso de eliminación de Gauss. Primero, comprueba si el elemento de la diagonal es cero y, en ese caso, intercambia esa fila con otra fila que tenga un uno en esa posición. Si el elemento de la diagonal es uno, elimina la variable de las otras filas restando a cada fila una combinación lineal de la fila actual.

Finalmente, la función utiliza el método de sustitución hacia atrás para resolver el sistema de ecuaciones y devuelve el vector de soluciones x en el campo modular 2.

Bibliografía

1 - Documentación completa de la librería Fernet. Fernet (symmetric encryption) —Cryptography 39.0.0.dev1 documentation
2 - Documentación en castellano de la librería OS. os — Interfaces misceláneas del sistema operativo — documentación de Python - 3.10.8
3 - Información detallada de la Criptografía simétrica en Wikipedia. Criptografía simétrica - Wikipedia, la enciclopedia libre
4 - Apuntes de la asignatura Teoría de Códigos y Criptografía. Disponibles en Aula Virtual.