Encriptación y Desencriptación de contraseñas e información

¿De qué manera se puede programar un protocolo de seguridad para una aplicación de notas, usando estructuras de datos, el cifrado de Elgamal y OpenCV?

Asignatura: Informática Aplicada

Código: knn768

N° de Palabras: 3996

Índice

1. Introducción……3
2. Planteamiento del problema…… 3
3. Justificación……4
4. Objetivo……... 4
5. Marco teórico……...4
6. Metodología……6
7. Discusión y Análisis……6
8. Conclusiones……..17
9. Bibliografía…….. 18
10. Anexos……... 20

Introducción

La encriptación ha sido un método usado desde tiempos antiguos para esconder información que haya sido interceptada por un tercero y para lograrlo se utilizan diferentes métodos, pero en este siglo XXI se usan los famosos algoritmos de encriptación y protocolos de seguridad y en esta monografía se creará uno de ellos, usando estructura de datos, el cifrado Elgamal y OpenCV.

Pero ¿Por qué hacer un protocolo de seguridad?, Siempre me ha intrigado como una red social, plataforma, etc.., cada día se vuelve una parte más de nosotros y para interactuar con ella, requerimos una contraseña, que es almacenada en una base de datos, pero el inconveniente ocurre cuando esa base de datos es copiada, hackeada o vista por el propio creador, etc.., y es en esos momentos cuando nuestra contraseña queda expuesta a la vista de alguien que puede usarla para acceder a nuestra cuenta.

Para solucionar este problema, un algoritmo de cifrado de contraseña en la base de datos es lo idóneo, pero a veces esto no es suficiente y se necesitan implementar más protocolos de verificación, por lo que el uso de estructura de datos, el cifrado de Elgamal y OpenCV son idóneos para este problema.

Planteamiento del problema

Un problema clave es el time complexity y el space memory de el cifrado de Elgamal y la demás ejecución de todo el código, por lo que el uso de estructuras de datos como Linkedlist, permitiría bajar en algo la ejecución total. No obstante no sirve encriptar una contraseña si hay keyloggers indetectables capaces de copiar los movimientos de tu teclado, para solucionar esto se utilizara los módulos de OpenCV python para hacer un registro de la cara del usuario y hacer uso del registro para cuando inicie sesión, de esta forma no se permitirá entrar si la cara que registra la contraseña no es la misma a la registrada en el sistema. De esta forma, se plantea la siguiente pregunta de investigación:

¿De qué manera se puede programar un protocolo de seguridad para una aplicación, usando estructuras de datos, el cifrado de Elgamal y OpenCV?

Justificación

La web, un gran mundo que tiene como objetivo guardar y compartir la información de los usuarios, esto respetando que son conscientes de lo que hacen, pero qué pasa, cuando la información que se almacena es delicada, como puede el usuario estar seguro de lo que guarda en la red y no quiere compartir, en verdad está siendo vista por alguien más. En principio gran parte de lo usuarios creen que la red es algo totalmente seguro y que no tienen que preocuparse, pero por mi parte, yo si me preocupe al leer del ataque del Bundestag alemán, donde se filtró información financiera, chats, etc.., o conocer a John the Ripper, un software programado para romper con los algoritmos de encriptación, keyloggers, capaces de registrar tus movimientos de teclado. Con todo esto, anexado por mi pasión por la programación, se decide crear un protocolo de seguridad, que sea capaz de evitar la desencriptación por parte del administrador o de algún tercero que haya interceptado el mensaje.

Objetivo

- Programar un protocolo de seguridad para una aplicación, usando estructuras de datos, el cifrado de Elgamal y OpenCV

Marco teórico

Entrando en tecnicismos, para programar un algoritmo, primero se tiene que hablar de varios conceptos relacionados a su rendimiento.

El primer concepto es el Time Complexity de un algoritmo, el cual define (Pandey, 2022), como la cantidad de tiempo en el que un algoritmo se tarda en ejecutar una función de la longitud de entrada, sin tener en cuenta la ejecución real de la máquina. En pocas palabras, es el tiempo en el cual los datos se demoran en pasar por todo el algoritmo, ignorando la capacidad de ejecución de la computadora.

Otro concepto importante es el Space Complexity el cual representa el espacio de memoria RAM que usa un algoritmo para ser ejecutado con respecto al tamaño de entrada y espacio auxiliar si es necesario, explica (GeeksforGeeks, 2022).

El tercer concepto es el Big O Notation que representa qué tan rápido es un algoritmo para procesar todos sus datos, para hacer la clasificación utiliza la estructura de O(n), donde n es el número de operaciones que tiene que hacer el algoritmo en su peor caso (Salton, 2019).

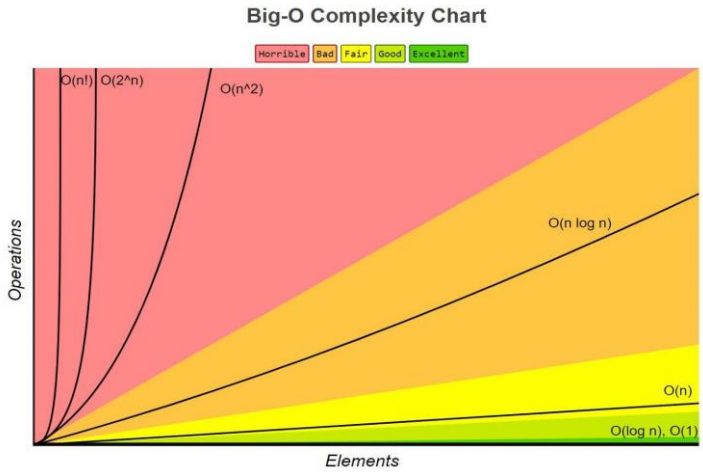


Figura 1: Comparación de Big O Functions.
Imagen de (Salton, 2019)

Estructuras de Datos

Un término bastante importante y que está ligado a la pregunta de investigación son las estructuras de datos y se definen como algoritmos especializados en almacenar y organizar grandes cantidades de datos, con el propósito de realizar operaciones más eficaces al momento de buscar, insertar, actualizar y eliminar de valores, expone (Mallawaarachchi, 2020).

La estructura de datos que se va usar es linked list, definido por (Programiz, 2021), como una estructura de datos lineal[1], donde no se almacena en ubicaciones de memoria contiguas, en cambio usa nodos que almacenan datos y la dirección del siguiente nodo.

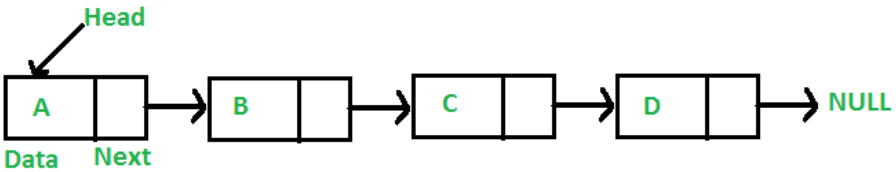


Figura 2: Representación de Linked List.
Imagen de (GeeksforGeeks, 2022)

[1] Estructura de datos lineal: "*Una estructura de datos lineal tiene elementos de datos conectados entre sí, de modo que los elementos se organizan de manera secuencial y cada elemento está conectado al elemento que está delante y detrás de él.*", explica (Samad, 2022).

Para empezar con el código de encriptación de elaboración propia, se le pide al usuario que suba 20 fotos de su rostro en diferentes posiciones, y las guarde en una carpeta con su nombre en el archivo images, para qué el script de faces-train.py, tomado de (Coding For Entrepreneurs, 2018), haga el entrenamiento de reconocimiento facial, para saber más sobre cómo funciona este script **ver anexo 6**, posteriormente se ejecuta el archivo main.py, que empieza llamando al código faces.py tomado de (Coding For Entrepreneurs, 2018), para saber más sobre cómo funciona este script **ver anexo 7**, el cual consigue el índice de similitud entre el rostro en cámara y el rostro registrado. Posteriormente se crea una matriz de A * A, siendo A, el índice de similitud. Pero para completar los valores que están dentro de la matriz, se hace uso de la variable b y c, adicionalmente se guarda las ubicaciones X y Y de la matriz con los valores a encriptar, para ser enviadas a b.py, donde se hace su encriptación con el cifrado de Elgamal, tomado de (Panda, 2021), **ver anexo 2** para entender su funcionamiento, y los valores retornados, se convierten en un linkedlist que rotara k posiciones, siendo k el índice de similitud.

Para la desencriptación, se hará todo un proceso inverso, pero con una autenticación de ±2 puntos de error en el índice de similitud.

Metodología

Para el desarrollo de este trabajo monográfico, se implementará una metodología con enfoque Triangular o mixto siguiendo una ruta específica para la creación del código.

Discusión y Análisis

Partiendo con el código del protocolo de seguridad, el proceso de encriptación, se realiza con el archivo main.py, donde primero se importa la librería Pickle, con la finalidad de serializar y deserializar la estructura de objetos de la variable conf, la cual después de haber llamado al script faces.py, contiene el índice de similitud entre el rostro de la persona registrada y el rostro que aparece en cámara, la idea de usar esta librería, fue para contener el objeto y que este fuera llamado en otro script para reconstruir el objeto en la desencriptación, sin incurrir en cambios frecuentes de la variable conf, que se producen cada vez que se llama al archivo de desencriptación. En las siguientes líneas se esquematiza lo mencionado anteriormente:

```
with open('{path_file}/cache/conf.conf', 'wb') as data:
   pickle.dump(conf, data)

with open('{path_file}/cache/conf.conf', 'rb') as data:
   conf = pickle.load(data)
   conf = conf.decode('ascii')
   conf = int(conf)
```

Para los siguientes pasos de encriptación, se creó la variable pasw, que vendría a ser la unión de la contraseña con el mensaje a encriptar, y su creación se justifica para la autenticación en la desencriptación más adelante y para su almacenamiento en la variable b, la cual va a pasar todo el formato string a integer para su posterior encriptación. Continuando con las variables, la variable power, guarda la longitud del input del usuario elevado la longitud del input de la contraseña, esta acción se realizó a fin de tener un valor constante con el cual multiplicar los ítems del ciclo for en pasw, de esta forma en el primer condicional, se usa la función .isdigit() para identificar si el ítem es un dígito, y la razón de su uso, está en que la función atraviesa cada ítem usando el valor ASCII, permitiendo que su iteración tenga un time complexity de O(n), explica (Saxena, 2022). Posteriormente se revisa que si el ítem es igual que cero, ya que si lo es, se sobreescribirá como la longitud del usuario, debido a que el almacenamiento en la variable de tipo array llamada b, se va a guardar la multiplicación entre el carácter tipo int multiplicado por power, aunque en el else statement, en caso de que el carácter no sea un dígito, se usa el método ord(), el cual convierte un ítem de tipo string, en un carácter de tipo int, a partir del estándar global Unicode, siempre y cuando el ítem sea de 8 bits, expone (Przywóski, 2015), por lo que en la variable b, se guardará el Unicode del ítem multiplicado por power. Con todo esto, ya se tiene almacenado una primera distorsión de la contraseña, la cual posteriormente va a ser guardada en una matriz con más valores.

```
usr = input("Ingrese nombre de usuario: ")
pass1 = input("Ingrese Contraseña: ")
msg1 = input("Ingrese Mensaje: ")

pasw  = pass1 + ": " + msg1
power = len(usr) ** len(pasw)

b = np.array([], dtype = int)
c = np.array([], dtype = int)
```

```
for i in pasw:
    if i.isdigit():
        a = int(i)
        if a == 0:
            a += len(usr)
        b = np.append(b,(a*power))
    else:
        b = np.append(b,(ord(i) * power))
```

Ahora bien en la variable curr_time se almacena el tiempo en milisegundos multiplicando por mil, redondeado al valor entero más cercano, con el propósito de tener un rango aleatorio, ya que en el ciclo for loop, el rango de repetición, va ir desde (conf *conf), representando al tamaño de la matriz, menos len(pasw), debido a que se necesita valores que complementen lo que falta para llenar la matriz, sabiendo que la longitud de pasw, ya hace parte de la matriz, por lo que continuando con curr_time, éste será puesto en la variable a, el cual escoge de manera aleatoria un número entero entre 1 y curr_time, de esta forma se asegura que cada vez que el código es ejecutado, se obtenga valores totalmente aleatorios que se guardan en la variable c de tipo array. Ahora en la variable array, se van a concatenar las dos variables b y c, puesto que de no solo tener ya los datos que van a hacer parte de la matriz, se hace uso del método shuffle() para reorganizar los elementos del array de manera aleatoria, explica (W3schools, 2022), con el fin de que a la hora de hacer uso del método reshape en la variable array y pasados los parámetros, se crea una matriz de tamaño conf por conf, con el objetivo de contener de forma aleatoria, las distorsiones de los valores de la variable pasw, junto con los valores aleatorios de la variable c, un ejemplo del resultado se puede ver en el **anexo 1**.

```
curr_time = round(time.time()*1000)

for i in range((conf*conf)- len(pasw)):
    a = random.randint(1, curr_time)
    c = np.append(c,a)

array = np.concatenate((b,c))

random.shuffle(array)
array = np.reshape(array, (round(conf), round(conf)))
```

Terminando con la variable array, para que quede guarda para su posterior desencriptación, se hace uso del módulo pickle para serializar el array en un documento con la dirección path_file, con el nombre times1, que refiere a un variable que contiene la fecha de ejecución del programa, con una extensión .data para hacer una diferenciación con otro archivo más adelante que va a tener el mismo nombre pero con otra función.

```
with open(f'{path_file}/cache/{times1}.data', 'wb') as data:
    pickle.dump(array, data)
```

Una vez que se tenga la matriz con los datos de la contraseña y valores aleatorios, se procederá con un for loop en la variable b, donde se usa el método where() en cada ítem iterado, para guardar en la variable index_x e index_y, la position que estas tienen en la matriz, de tal forma que se encripta la dirección de X y Y de los ítems de la contraseña en la matriz, con el propósito de desencriptarlas posteriormente y facilitar su ubicación. Aunque para evitar combinaciones entre estos dos elementos, se usa la variable str1, para unir ambos elementos con un espacio intermedio entre ambos, de esta forma se garantiza que no importa si la posición del elemento X sea 1 y la del elemento Y sea 13, y que al unirlos quede 113, que por lo que vemos no se puede distinguir si la posición en X es 1 o es 11, por lo que en la separación con espacio en la tupla, quedaria ‘1 13’, haciendo distinción entre ambos valores. Continuado la encriptación, se va a hacer uso de main(), que es una función importada del archivo b.py, la cual retorna cuatro parámetros después de la encriptación con el algoritmo Elgamal, finalmente, estos elementos son guardados en la variable d de tipo array.

```
c = ''
d = []
for i in b:
    index_x, index_y = np.where(array == i)
    c = str(index_x[0]),str(index_y[0])
    str1 = ' '.join(c)
    m = main(str1)
    d.append(m)
```

Con la variable *d* y *conf*, son mandados como parámetros en a la función linked que es importada de c.py, con la finalidad de convertir una lista tipo array en una tipo linkedlist, con el cual se pueda mover conf posiciones adelante, es decir que si la lista es [1,2,3,4,5,6] con conf 3, ahora será 4,(5,(6,(1,(2,(3,None))))).

```
linked1 = linked(d, conf)
```

Para lograr el cometido, primero se inicia con la función linked, la cual al recibir dos parámetros, el primero es el array a convertir y el segundo el número conf que moverá el linkedlist. Pero para hacer la conversión, primero se pensó en hacer uso del un ciclo while, el cual al iterar por cada elemento, madaria a la función Node() todo lo iterado, pero el problema iba en que en vez de hacer que cada Nodo fuera un valor diferente por cada ítem del primer parámetro, se obtiene un solo Nodo con toda la información, haciendo que fuera imposible hacer una iteración independiente a cada elemento, por lo que esta solución fue descartada. Pero la solución llegó con un truco que usa (Joma Class, 2020) en su solución al problema Intersection of Two Linked List, el cual hace uso de dos pointers como una forma para iterar de manera correcta un linked list. De esta manera el primer pointer que es llist, va a ser el Nodo cabeza que tiene como valor el primer ítem del primer parámetro, así mismo el segundo pointer es head y este es igual a llist, por lo que en el ciclo for, el rango va a ser dentro de 1 (porque ya tenemos el primer valor en un Nodo), hasta la longitud del primer parámetro, por consiguiente como el valor .next del primer nodo está vacío, entonces se asigna un nuevo Nodo con el valor de la iteración del primer parámetro, en otras palabras el Nodo pasa de (1,(None)) a (1,(Node(2,(None)))), pero para no hacer que el Nodo reescriba el .next y en vez de eso itere, se asigna qué head sea igual a su siguiente valor y que se continúe con el ciclo for, pasando así de (1,(Node(2,(None)))) a (2,(Node(3,None)))) y finaliza retornando la lista tipo linkedlist y el segundo parámetro a la función rotate de (Joma Class, 2020), la cual va a ser que el linkedlist rote K posiciones hacia adelante. Ver **anexo 3**, para ver cómo funciona la función rotate y ver el **anexo 4** para mirar un ejemplo del resultado obtenido del linked list después de la función rotate.

```
class Node:
    def __init__(self,value1=None,next=None):
        self.value1 = value1
        self.next = next

    def __repr__(self):
        return f"{self.value1}, {self.next}"

def rotate(list1, k):
    length = 0
    current = list1
```

```
    while current != None:
        length += 1
        current = current.next
    k = k % length
    faster, slower = list1, list1
    for _ in range(k):
        faster = faster.next
    while faster.next != None:
        faster = faster.next
        slower = slower.next
    faster.next = list1
    head = slower.next
    slower.next = None
    return head

def linked(a,b):
    llist = Node(a[0])
    head =  llist
    for i in range(1,len(a)):
        head.next = Node(a[i])
        head = head.next
    return rotate(llist, b)
```

Posteriormente con el uso de la librería pickle, se serializa el linkedlist rotado, que fue almacenado en la variable linked1, con la extensión .node y nombre de times1, con la finalidad de tener dos archivos con el mismo nombre, pero con diferente extensión, que hacen referencia a una misma ejecución de código.

```
with open(f'{path_file}/cache/{times1}.node', 'wb') as node:
    pickle.dump(linked1, node)
```

Para terminar por completo con la encriptación, se decidió hacer uso de la base de datos Sqlite3, debido a que (Koulianos, 2020), explica que se ejecuta en un archivo ligero, el cual no necesita de servidor o instalación RDBMS, adicionalmente su lectura y escritura, se llevan directamente en el mismo archivo con extensión .db, y al ser un archivo autónomo, este puede ser movido o copiado y ser compatible con todas los sistemas operativos. Pero hubo un problema y fue que

Sqlite3, no puede guardar clases, incluso si estas son declaradas tipo Blob[2], imposibilitando así almacenar linked1 y array, por lo que se decidió hacer uso de algo común que tienen ambas clases, y es su nombre, ocasionando que la base de datos solo guarde el usuario y el nombre de las clases.

```
connex = sqlite3.connect('user_encrypted_data.db')
try:
   connex.execute('''
            CREATE TABLE DATA
                (ID TEXT          NOT NULL,
                NODO             BLOB     NOT NULL);''')
   print("Tabla creada exitosamente")
except sqlite3.OperationalError:
   print("La tabla DATA ya existe")
connex.execute("INSERT INTO DATA (ID,NODO) VALUES (?,?)", (usr, times1))
connex.commit()
```

Empezando con la desencriptación, se encontró con un problema y fue en el llamado a la función send() de faces.py para obtener el valor de la variable conf, ya que cada vez que se hacía el llamado, se ejecutaba todo el código, provocando que se reescribiera el archivo .conf y que no se pudiera hacer la comparación de la similitud entre el rostro guardado con el rostro que aparece en cámara, por lo que para solucionar el problema, se decidió hacer una copia del archivo faces.py y llamarlo faces1.py, de esta forma cuando se ejecutaba el archivo, se obtiene el índice de similitud y se guardaba en la variable a, mientras que en la variable conf, se deserializa el índice con el que se hizo la encriptación, adicionalmente se crea la variable continue1 con el valor False, con el motivo de hacer la verificación entre ambos índices de similitud.

```
from faces1 import send2

a = send2()

continue1 = False

with open('{path_file}/cache/conf.conf', 'rb') as data:
```

[2] Blob: Hace parte de los tipos de datos de SQLite y almacena datos exactamente como fueron ingresados, sin especificar el tipo de dato; explica (SQLite, 2022)

```
    conf = pickle.load(data)
    conf = conf.decode('ascii')
    conf = int(conf)
print("B: ",str(conf))
```

Para hacer el proceso de comparar ambos índices de similitud, se va a tener un rango, donde la variable b, la cual contiene el inicio del rango con a-2, va a pasar por un ciclo while que tiene como límite máximo a+2, dejando así un rango de tres números, los cuales si alguno coincide con el valor de conf, se asigna True a continue1, dando así seguridad de verificación de similitud y continuando con la desencriptación. Y la razón de porqué asignar un rango, fue debido a que el índice de similitud en la variable a, puede cambiar debido a factores como la iluminación y posición del rostro que aparece en cámara, por lo que después de 20 pruebas, con diferentes iluminaciones y posiciones, se hizo un cálculo de incertidumbres que dio como resultado ±2 unidades de similitud (**ver anexo 5**), ocasionando que si el valor de conf en la similitud de la encriptación, era de 54, en la desencriptación, el índice de similitud con el rostro que aparece en pantalla debe estar dentro del rango de 52, 53, 54, 55 y 56.

```
b = a-2

while b < a+2:
    print(b)
    if b == conf:
        continue1 = True
        break
    else:
        b += 1
```

El uso del while continue, sirve para verificar si se hizo la asignación True en la verificación de la similitud de rostros, en caso de no ser así, el while no se inicializa y finaliza la ejecución de código. Por otra parte se le va a pedir al ejecutor que inserte el nombre del usuario que hizo la encriptación, los datos a desencriptar y la contraseña. De esta forma se almacena en la variable matrix, la deserialización del archivo .data con el nombre seleccionado, obteniendo la matriz a desencriptar, así mismo la variable linked contiene la deserialización del archivo .node del nombre seleccionado, para adquirir las posiciones encriptadas de los ítems contenedores del mensaje. Por último está la variable linked2, la cual pasa linked y conf como parámetros a la función rotate_list_inv del archivo c.py.

```
while continue1:
    conn = sqlite3.connect('user_encrypted_data.db')
    cursor1 = conn.execute("select ID, NODO from DATA")
    for fila in cursor1:
        print("Usuario: ", f'{fila[0]}', "    Data: ", fila[1])
    conn.close()

    usr1 = input("Usuario a usar: ")
    slec = input("Datos a Desbloquear: ")
    pasw1 = input("Contraseña a Desbloquear: ")

    with open(f'{path_file}/cache/{slec}.data', 'rb') as data:
        matrix = pickle.load(data)

    with open(f'{path_file}/cache/{slec}.node', 'rb') as node:
        linked = pickle.load(node)

    linked2 = rotate_list_inv(linked, conf)
```

Como la variable linked tiene items movidos conf posiciones, se necesita devolverlos a su posición original para desencriptarlos y obtenerlos de forma correcta las posiciones con las que se van a buscar los items en la variable matrix, por lo que para lograr el cometido, se buscó una forma en la que los elementos se muevan otra vez k posiciones para que queden en la posición original, pero para saberlo, primero se tiene que saber la longitud del linkedlist que fue pasado, así que se hace uso la variable length para que cada vez que en el ciclo while se este iterando, la variable incrementa por uno hasta que el siguiente ítem del nodo sea None. Una vez con la longitud, podemos ver que para sacar el número de veces que se mueven los nodos en la función rotate, se hace uso de la variable k para contener la operación k %[3] length, donde k es el número de veces a mover y length, la longitud del linkedlist, obteniendo de esta forma que 5 % 10 es igual 5 y que 6%10 es igual a 6, y así sucesivamente para todos los números enteros que sean menores a length, hasta que 10 % 10 es igual a 0 y 11 % 10 es igual a 1 y sigue de la misma forma cuando k es mayor que length, de esta forma se puede saber que tantos puestos hay que moverse para llegar a la posición original, ya que si sacamos la diferencia entre length y k, en la variable k, podemos ver dos casos, donde 2 % 7 es igual a 2, entonces significa que los items se movieron 2 puestos, pero obteniendo la diferencia, se puede ver que se necesitan 4 puestos

[3] %: También conocido como módulo operador, devuelve el residuo de la división de dos números; aclara (Van Schooneveld, 2020)

para volver a su posición original, es decir que si el linked list 6->7->1->2->3->4->5, que se movio 2 puestos, ahora necesita obtener su diferencia para saber que se tiene que mover 5 veces para estar en el origen, aunque ocurre algo similar si k es mayor que length, ya que la diferencia al ser negativa y pasar por k % length, el resultado es igual a la diferencia faltante para que k sea igual a length, por lo que si 6->7->1->2->3->4->5 que ahora se movio 9 veces y su diferencia es -2, entonces -2 % 7 es igual a 5, y es ese resultado el número de veces a mover para estar en el origen, terminando así la función con el retorno del linkedlist que fue movido k veces en la función rotate que tuvo como parámetros k y el linkedlist a mover.

```
def rotate_list_inv(llist1,k):
    length = 0
    current = llist1
    while current != None:
        length += 1
        current = current.next
    k = length - k
    return rotate(llist1, k)
```

Una vez que tenemos el linkedlist en el orden correcto, se tiene que desencriptar con el cifrado Elgamal, las posiciones que necesitamos para encontrar los datos en la variable matrix, para ello se manda los cuatro parámetros a la función main1 de g.py; con los valores de posicionamiento, se separan para no tener errores de búsqueda en matrix, por lo que para lograr el objetivo, se hace un ciclo for loop, el cual al detectar un espacio en blanco, guarda en la variable a2 los datos que son siguientes al índice del espacio y en la variable a1, los valores que están por detrás del índice.

```
    matrix_b = np.array([], dtype = int)
    current = linked2
    while current != None:
        wa = main1(current.data[0], current.data[1], current.data[2],
current.data[3])
        a1 = ''
        a2 = ''
        for id in range(len(wa)):
            if wa[id] == ' ':
                a1 = int(wa[:id])
                a2 = int(wa[id+1:])
```

```
        matrix_b = np.append(matrix_b,matrix[a1][a2])
        wa = ''
        current = current.next
    power1 = len(usr1) ** len(matrix_b)
```

Para terminar de desencriptar, se necesita primero cambiar en la variable pasw1 cualquier valor 0 por la longitud del usuario, debido a que se hizo lo mismo en la encriptación y para no tener discordancias en los datos desencriptados. Finalmente se usa la variable power1, la cual contiene la longitud del usuario elevado a la longitud de la matrix_b, para hacer la división con el ítem a desencriptar, de esta forma se pueden obtener dos posibles valores, uno donde el valor es menor a 9, y significa que el valor a desencriptar es un número, y otro cuyo valor sea mayor a 9, siendo este un valor que debe pasar por la función chr() para ser convertirlo en su valor de string, por último como el string desencriptado contiene la contraseña y el mensaje, se hace una verificación donde el input de la contraseña tiene que ser igual a la desencriptada, para proceder con mostrar el mensaje.

```
    def final_msg(a,b):
        final_msg = ''
        for i in a:
            ay = round( i / power1)
            if ay <= 9:
                final_msg += ''.join(str(ay))
            else:
                final_msg += ''.join(chr(ay))
        if final_msg[:len(b)] == b:
            print("[Mensaje]: ",final_msg[len(b)+1:])
        else:
            print("Contraseña Incorrecta!")

    for i in pasw1:
        if i == '0':
            pasw1 = pasw1.replace('0', str(len(usr1)))
    final_msg(matrix_b, pasw1)
    continue1 = False
```

Finalmente se puede observar todo el código completo en el **anexo 8**, con su respectiva organización de archivos.

Conclusiones

Como consideraciones finales y dando respuesta a la pregunta de esta monografía, la manera en la que se puede programar un protocolo de seguridad para una aplicación, usando estructuras de datos, el cifrado de Elgamal y OpenCV, es en hacer como primer paso una verificación de similitud entre rostro en cámara y el rostro de encriptación, haciendo uso de la librería de OpenCV, posteriormente usar una matriz que contenga de manera distorsionada los datos a encriptar, así mismo las direcciones de los datos en la matriz, son encriptados con el cifrado de Elgamal, y finalmente se hace uso de la estructura de datos linkedlist, para mover k posiciones dependiendo del índice de similitud del rostro; esta fue una propuesta realizada por fuente propia, haciendo uso de los tres elementos anteriormente descritos más códigos de autores ya mencionados; con la finalidad de ofrecer una mayor seguridad con los datos del usuario.

Bibliografía

Coding For Entrepreneurs. (2018, Diciembre 20). *OpenCV-Python-Series*. GitHub. https://github.com/codingforentrepreneurs/OpenCV-Python-Series/blob/master/src/faces.py

Coding For Entrepreneurs. (2018, Abril 17). *OpenCV-Python-Series*. GitHub. https://github.com/codingforentrepreneurs/OpenCV-Python-Series/blob/master/src/faces-train.py

GeeksforGeeks. (2022, Septiembre 7). *What does 'Space Complexity' mean?* GeeksforGeeks. https://www.geeksforgeeks.org/g-fact-86/

GeeksforGeeks. (2022, Octubre 25). *Linked List Data Structure*. GeeksforGeeks. https://www.geeksforgeeks.org/data-structures/linked-list/

Joma Class. (2020, Agosto 1). *Intersection of two linked list*. Joma Class. https://www.jomaclass.com/blog/intersection-of-two-linked-list

Joma Class. (2020, Agosto 12). *Rotate Linked List*. Joma Class. https://www.jomaclass.com/blog/rotate-linked-list

Koulianos, P. (2020, Agosto 7). *5 Reasons to Use SQLite the Tiny GIANT for Your Next Project*. Medium. https://medium.com/swlh/5-reasons-to-use-sqlite-the-tiny-giant-for-your-next-project-a6bc384b2df4

Mallawaarachchi, V. (2020, Febrero 27). *8 Common Data Structures every Programmer must know | by Vijini Mallawaarachchi*. Towards Data Science. https://towardsdatascience.com/8-common-data-structures-every-programmer-must-know-171acf6a1a42

Panda, S. (2021, Octubre 20). *ElGamal Encryption Algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/elgamal-encryption-algorithm/

Pandey, U. (2022, Julio 15). *Time Complexity and Space Complexity*. GeeksforGeeks. https://www.geeksforgeeks.org/time-complexity-and-space-complexity/

Programiz. (2021). *Linked List Data Structure*. Programiz. https://www.programiz.com/dsa/linked-list

Przywóski, J. (2015). *ord — Python Reference (The Right Way) 0.1 documentation*. Python Reference (The Right Way). https://python-reference.readthedocs.io/en/latest/docs/functions/ord.html

Salton, K. (2019, Marzo 4). *Understanding time complexity with Python examples | by Kelvin Salton do Prado*. Towards Data Science. https://towardsdatascience.com/understanding-time-complexity-with-python-examples-2bda6e8158a7

Samad, A. (2022). *What are linear data structures?* Educative.io. https://www.educative.io/answers/what-are-linear-data-structures

Saxena, A. (2022, Agosto 18). *Python String isdigit() Method*. GeeksforGeeks. https://www.geeksforgeeks.org/python-string-isdigit-method/

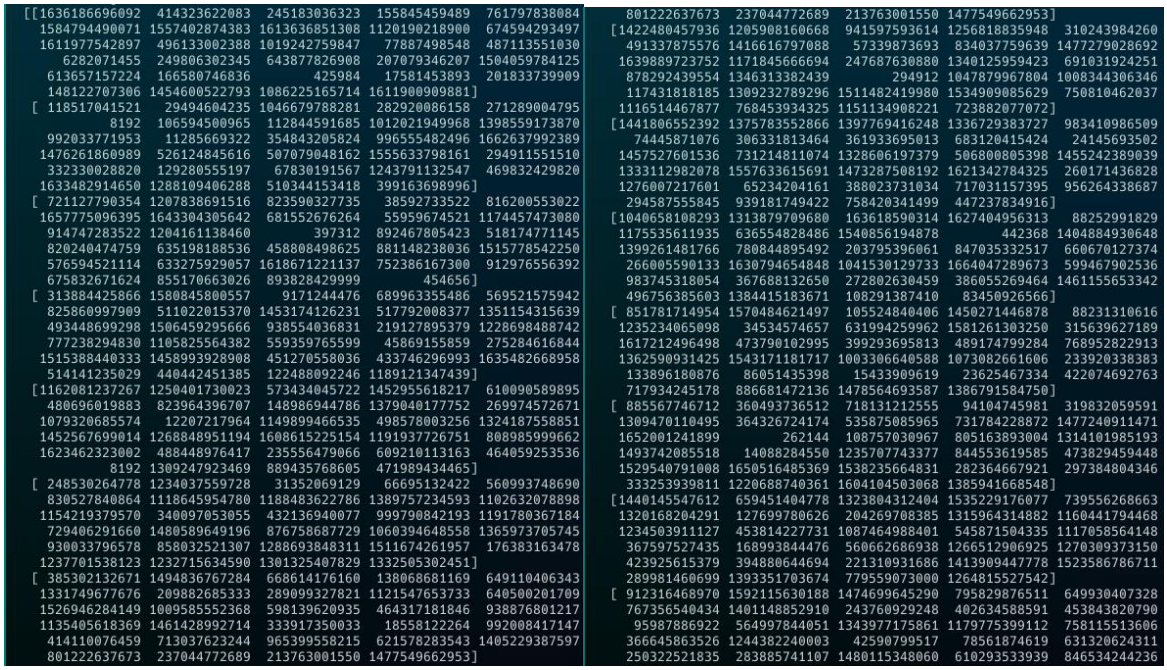
SQLite. (2022, Abril 12). *1. Datatypes In SQLite*. SQLite. https://www.sqlite.org/datatype3.html

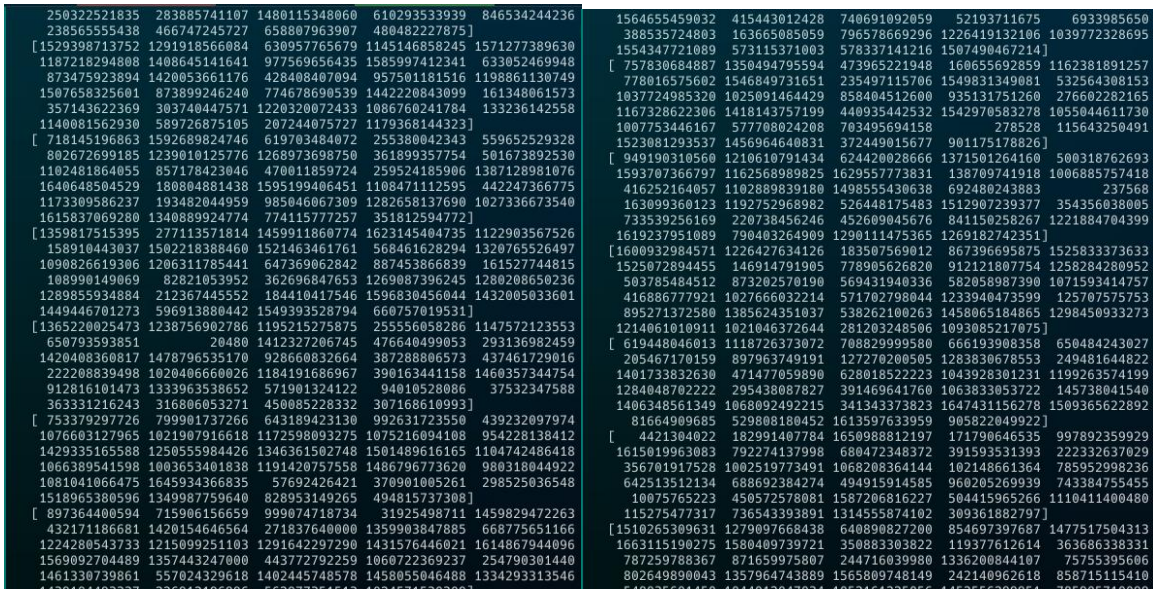
Van Schooneveld, J. (2020). *Python Modulo in Practice: How to Use the % Operator – Real Python*. Real Python. https://realpython.com/python-modulo-operator/

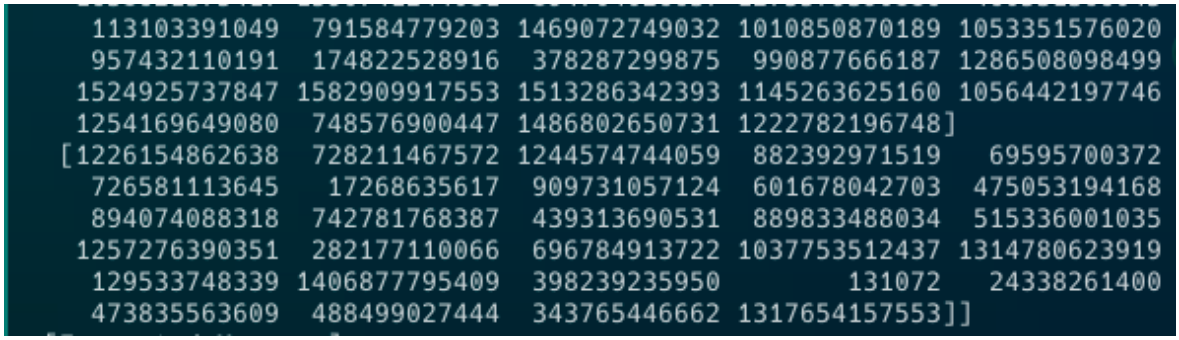
W3schools. (2022). *Python Random shuffle() Method*. W3Schools. https://www.w3schools.com/python/ref_random_shuffle.asp

Anexos

Anexo 1: Matriz de la variable array con parámetros de conf = 29, usr = Dh, pass1 = Dh205@, msg1 = Hola







Anexo 2: Funcionamiento del algoritmo Elgamal tomado de (Panda, 2021)

El algoritmo Elgamal, es un criptosistema de clave pública, que usa la clave simétrica para la dificultad el cálculo de logaritmos discretos en un grupo cíclico, siendo así difícil calcular el valor de g^{ak}, incluso si se sabe g^a y g^k, explica (Panda, 2021).

La idea del funcionamiento del el algoritmo Elgamal, dada por (Panda, 2021) es:

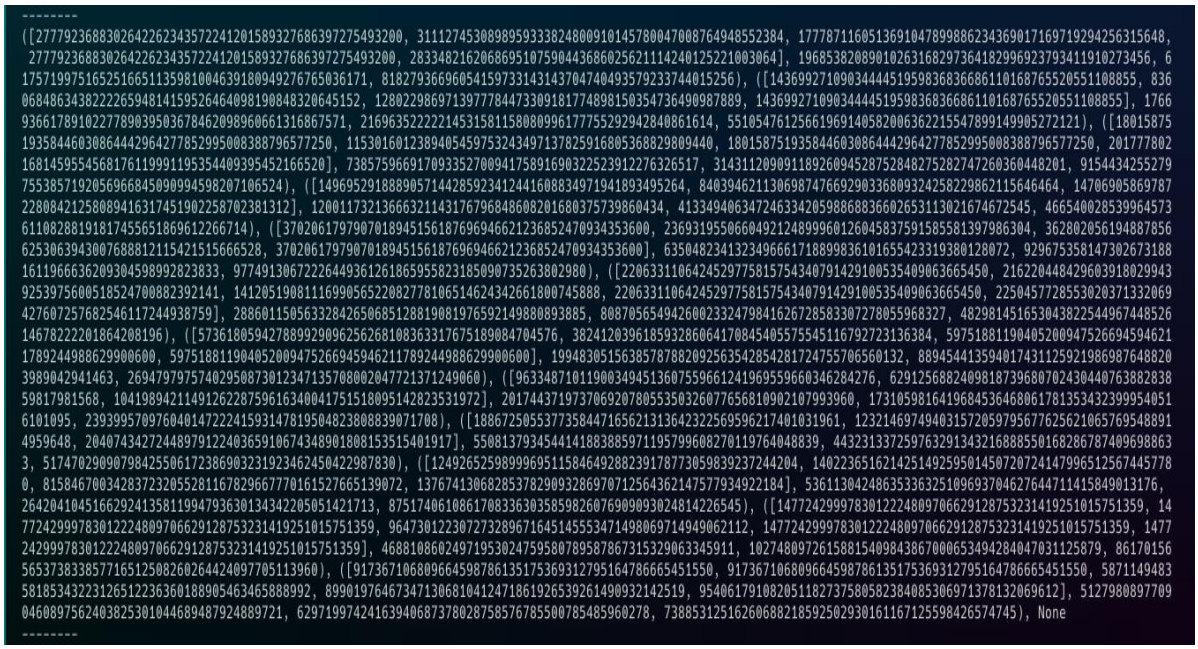
1. Bob genera claves públicas y privadas:
   - Bob elige un número q muy grande y un grupo cíclico F_q.
   - Del grupo cíclico F_q, elige cualquier elemento g y un elemento a tal que gcd(a, q) = 1.
   - Luego calcula h = g^a.
   - Bob pública F , h = g^a, q y g como su clave pública y retiene a como clave privada.
2. Alice cifra los datos usando la clave pública de Bob:
   - Alice selecciona un elemento k del grupo cíclico F tal que gcd(k, q) = 1.
   - Luego calcula p = g^k y s = $h^k = g^{ak}$.
   - Ella multiplica s con M. Luego envía (p, M*s) = (g^k , M*s).
3. Bob descifra el mensaje:
   - Bob calcula s $'$ = $p^a = g^{ak}$.
   - Divide M*s por s $'$ para obtener M como s = s $'$.

Anexo 3: Funcionamiento de la función rotate tomado de (Joma Class, 2020)

Para rotar los ítems del un linkedlist, primero hay que conocer la longitud de este, para ello se atraviesa por todo el linkedlist hasta que el valor de .next sea igual a None, posteriormente se hace uso de la variable k, que con operador % calcula en número de posiciones x a mover dependiendo de la longitud y el número deseado a mover, con estos datos se crean dos pointers el primero avanza hasta la longitud de k, después el primero pointer se mueve hasta el final del linkedlist mientras que el segundo pointer hace lo mismo, una vez termina, el valor .next del primer pointer es igual a la lista original, la cabeza del linkedlist va a ser el valor .next del segundo pointer y al mismo tiempo va a cortar el linkedlist con un None, explica (Joma Class, 2020).

```
def rotate(list1, k):
    length = 0
    current = list1
    while current != None:
        length += 1
        current = current.next
    k = k % length
    fast, slow = list1, list1
    for _ in range(k):
        fast = fast.next
    while fast.next != None:
        fast = fast.next
        slow = slow.next
    fast.next = list1
    head = slow.next
    slow.next = None
    return head
```

Anexo 4: Resultado del linkedlist después de la función rotate con parámetros de conf = 29, usr = Dh, pass1 = Dh205@, msg1 = Hola



Anexo 5: Incertidumbre en el índice de similitud

Tabla 1: Tabla con los resultados del índice de similitud

Prueba 1	34.28274124989236
Prueba 2	33.732949481936195
Prueba 3	33.16145798520633
Prueba 4	30.97051184927079
Prueba 5	34.38371571409794
Prueba 6	35.06895281437445
Prueba 7	32.05323101320842
Prueba 8	31.96332601704973
Prueba 9	31.654678820591368
Prueba 10	34.15722239811593
Prueba 11	32.60107970681687
Prueba 12	31.39216977623138
Prueba 13	30.55597569930113
Prueba 14	30.767472906324052
Prueba 15	32.56845912913467
Prueba 16	38.836783621119174
Prueba 17	32.388311957777105
Prueba 18	30.580305251837522
Prueba 19	32.11849312353868
Prueba 20	32.061164211613715

Fuente: Elaboración propia (2022)

$$n = \text{número de índices de similitud}$$

$$\triangle s = \text{Incertidumbre en el índice de similitud}$$

$$\Delta s = \sqrt{\frac{\sum_{i=1}^{n}(|prueba_i - \underline{prueba}|)^2}{n-1}}$$

$$\Delta s = \sqrt{\frac{72.38142420990395}{20-1}}$$

$$\Delta s = \sqrt{3.809548642626524}$$

$$\Delta s = \pm 1.9518065074762212$$

$$\Delta s = \pm 2$$

Anexo 6: Explicación sobre el funcionamiento del script faces-train.py tomado de (Coding For Entrepreneurs, 2018)

El objetivo general de este código es hacer uso de la librería de cv2 para un archivo .yml que contiene todo el entrenamiento de similitud de todas las caras guardadas en el archivo images, para ello hace uso de la función CascadeClassifier() para cargar una inteligencia artificial entrenada que puede reconocer cuando un objeto es una cara frontal y cuando no lo es, posteriormente revisa si cada archivo en images termina en jpg o png, puesto que estos son las únicas extensiones con las que funciona el entrenamiento, en caso de serlos, son puesto en la variable label, para asignar un id que es guardado en labels_id y con el cual se hace uso de detectMultiScale(), finalmente se identifican las posiciones del rostro en la imagen y se guarda en el id del rostro que está siendo identificado, aclara (Coding For Entrepreneurs, 2018).

```python
import os
import cv2
import numpy as np
from PIL import Image
import pickle

BASE_DIR = os.path.dirname(os.path.abspath(__file__))

image_dir = os.path.join(BASE_DIR, "images")

face_cascade = cv2.CascadeClassifier('cascades/data/haarcascade_frontalface_alt2.xml')

recognizer =  cv2.face.LBPHFaceRecognizer_create()

current_id = 0

label_ids = {}
y_labels = []
x_train = []

for root, dirs, files in os.walk(image_dir):
    for file in files:
        if file.endswith("png") or file.endswith("jpg"):
            path = os.path.join(root, file)
            label = os.path.basename(os.path.dirname(path)).replace(" ","-").lower()
            print(label, path)

            if not label in label_ids:
                label_ids[label] = current_id
                current_id += 1
```

```
            id_ = label_ids[label]
            pil_image = Image.open(path).convert("L")
            image_array = np.array(pil_image, "uint8")
            faces = face_cascade.detectMultiScale(image_array, scaleFactor=1.5, minNeighbors=1)

            for (x,y,w,h) in faces:
                roi = image_array[y:y+h, x:x+w]
                x_train.append(roi)
                y_labels.append(id_)


with open("labels.pickle", "wb") as f:
    pickle.dump(label_ids, f)

recognizer.train(x_train, np.array(y_labels))
recognizer.save("trainner.yml")
```

Figura 3: Código de faces-train.py.
Imagen de (Coding For Entrepreneurs, 2018)

Anexo 7: Explicación sobre el funcionamiento del script faces.py tomado de (Coding For Entrepreneurs, 2018)

Para poder crear el índice de similitud entre el rostro en cara y en rostro registrado por faces-train.py, se hace una carga de la inteligencia artificial entrenada para reconocer rostros en la variable faces_cascade, para hacer que reconozca los rostros entrenados en el archivo .yml, posteriormente se recorren todos los niveles de reconocimiento guardados en labels.pickle. Adicionalmente se convierte la imagen proyectada en cámara a color BGR2GRAY, con la cual se detectan multiescalas que pasan por un for loop para reconocer el rostro en cámara y compararlo con el registrado; y dejando el índice de similitud en la variable conf, finalmente se pone un recuadro con el nombre alrededor del rostro identificado, explica (Coding For Entrepreneurs, 2018).

```
import numpy as np
import cv2
import pickle

face_cascade = cv2.CascadeClassifier('cascades/data/haarcascade_frontalface_alt2.xml')
recognizer =  cv2.face.LBPHFaceRecognizer_create()
recognizer.read("trainner.yml")

labels = {}
with open("labels.pickle", 'rb') as f:
    og_labels = pickle.load(f)
    labels = {v:k for k,v in og_labels.items()}

cap = cv2.VideoCapture(0)
while True:
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    faces = face_cascade.detectMultiScale(gray, scaleFactor=1.5, minNeighbors=1)
    for (x,y,w,h) in faces:
        roi_gray = gray[y:y+h, x:x+w]
        id_, conf = recognizer.predict(roi_gray)
        print(conf)
        def send():
            return round(conf)
        if conf >=29 and conf <= 39 :
            font = cv2.FONT_HERSHEY_SIMPLEX
            name = labels[id_]
            color = (255,255,255)
            stroke = 2
            cv2.putText(frame,name, (x,y), font, 1, color, stroke, cv2.LINE_AA)
```

```
        img_item = "my-image1.png"
        cv2.imwrite(img_item, roi_gray)
        color = (255,0,0)
        stroke = 2
        end_cord_x = x+w
        end_cord_y = y+h
        cv2.rectangle(frame, (x,y),(end_cord_x,end_cord_y), color, stroke)
    cv2.imshow('frame', frame)
    if cv2.waitKey(20) & 0xFF == ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

Figura 4: Código de faces.py.
Imagen de (Coding For Entrepreneurs, 2018)

Anexo 8: Código completo de forma lineal con su respectiva organización de archivos

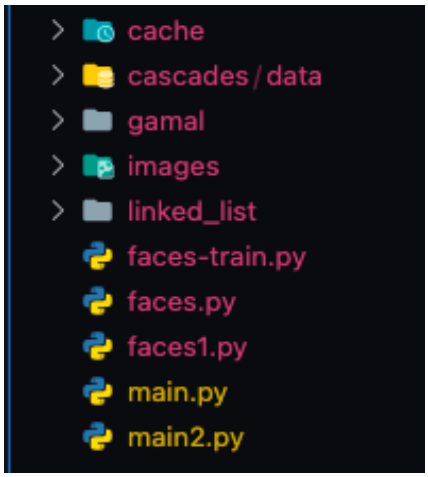


Figura 5: Nombre y ubicación de todos los archivos del código
Imagen de Elaboración propia(2022)

- La carpeta cache es el lugar donde se almacenan los archivos de tipo .data y .node
- La carpeta cascades/data almacena la inteligencia artificial entrenada con la que faces-train.py hace el reconocimiento de cara frontal; es obtenido de (Coding For Entrepreneurs, 2018)
- La carpeta gamal contiene los archivos b.py y g.py

```python
import random
from math import pow

a = random.randint(2, 10)

def gcd(a, b):
    if a < b:
        return gcd(b, a)
    elif a % b == 0:
        return b;
    else:
        return gcd(b, a % b)

# Generación de grandes números aleatorios
def gen_key(q):

    key = random.randint(pow(10, 20), q)
    while gcd(q, key) != 1:
        key = random.randint(pow(10, 20), q)

    return key
# Exponenciación modular
def power(a, b, c):
    x = 1
    y = a

    while b > 0:
        if b % 2 != 0:
            x = (x * y) % c;
        y = (y * y) % c
        b = int(b / 2)
```

```
    return x % c
# Cifrado asimétrico
def encrypt(msg, q, h, g):

    en_msg = []

    k = gen_key(q) # Clave privada para el remitente
    s = power(h, k, q)
    p = power(g, k, q)

    for i in range(0, len(msg)):
        en_msg.append(msg[i])

    for i in range(0, len(en_msg)):
        en_msg[i] = s * ord(en_msg[i])

    return en_msg, p

def main(pasw1):

    msg = pasw1
    q = random.randint(pow(10, 20), pow(10, 50))
    g = random.randint(2, q)

    key = gen_key(q)
    h = power(g, key, q)

    en_msg, p = encrypt(msg, q, h, g)
    return en_msg, p, key, q
```

Figura 6: b.py

Imagen de (Panda, 2021)

```
import random
from math import pow


def power(a, b, c):
    x = 1
    y = a

    while b > 0:
        if b % 2 != 0:
            x = (x * y) % c;
        y = (y * y) % c
        b = int(b / 2)

    return x % c

def decrypt(en_msg, p, key, q):

    dr_msg = []
    h = power(p, key, q)
    for i in range(0, len(en_msg)):
        dr_msg.append(chr(int(en_msg[i]/h)))

    return dr_msg

def main1(en_msg, p, key, q):
    dr_msg = decrypt(en_msg, p, key, q)
    dmsg = ''.join(dr_msg)
    return dmsg
```

Figura 7: g.py

Imagen de (Panda, 2021)

- La carpeta images contiene las 20 imágenes del usuario que usa su rostro para encriptar y desencriptar.
- La carpeta linked_list se encuentra c.py.

```
class Node:
    def __init__(self,value1=None,next=None):
        self.value1 = value1
        self.next = next

    def __repr__(self):
        return f"{self.value1}, {self.next}"

def rotate(list1, k):
    length = 0
    current = list1
    while current != None:
        length += 1
        current = current.next

    k = k % length
    faster, slower = list1, list1

    for _ in range(k):
        faster = fast.next

    while faster.next != None:
        faster = faster.next
        slower = slower.next
    faster.next = list1
    head = slower.next
    slower.next = None
    return head

def linked(a,b):
    llist = Node(a[0])
    head =  llist
    for i in range(1,len(a)):
        head.next = Node(a[i])
        head = head.next
    return rotate(llist, b)
```

```
def rotate_list_inv(llist1,k):
    length = 0
    current = llist1
    while current != None:
        length += 1
        current = current.next
    k = length - k
    return rotate(llist1, k)
```

- El primero archivo en ejecutarse es faces-train.py, ya que empieza entrenando la inteligencia artificial para ser usada en faces.py.
- main.py es el archivo que empieza con la encriptación, donde {path_file} es el lugar donde están ubicados los archivos del protocolo de seguridad

```
import sys
import pickle
sys.path.insert(0, '{path_file}')

from faces import send

conf = str(send())
conf = conf.encode('ascii')
print(conf)

with open('{path_file}/cache/conf.conf', 'wb') as data:
    pickle.dump(conf, data)

with open('{path_file}/cache/conf.conf', 'rb') as data:
    conf = pickle.load(data)
    conf = conf.decode('ascii')
    conf = int(conf)

print(conf)
print(type(conf))

import random
import numpy as np
import getpass
import time
import datetime as dev
import sqlite3
```

```
sys.path.insert(1,'{path_file}/gamal')
sys.path.insert(2,'{path_file}/linked_list')

from b import main
from c import linked

usr = input("Ingrese nombre de usuario: ")
pass1 = input("Ingrese Contraseña: ")
msg1 = input("Ingrese Mensaje: ")

pasw  = pass1 + ": " + msg1
power = len(usr) ** len(pasw)

b = np.array([], dtype = int)
c = np.array([], dtype = int)

for i in pasw:
   if i.isdigit():
       a = int(i)
       if a == 0:
           a += len(usr)
       b = np.append(b,(a*power))
   else:
       b = np.append(b,(ord(i) * power))

times1 = dev.datetime.now()
times1 = times1.strftime("%Y-%m-%d-%H:%M:%S")

curr_time = round(time.time()*1000)

for i in range((conf*conf)- len(pasw)):
   a = random.randint(1, curr_time)
   c = np.append(c,a)

array = np.concatenate((b,c))

random.shuffle(array)
array = np.reshape(array, (round(conf), round(conf)))
print(array)
```

```
with open(f'{path_file}/cache/{times1}.data', 'wb') as data:
    pickle.dump(array, data)

c = ''
d = []
for i in b:
    index_x, index_y = np.where(array == i)
    c = str(index_x[0]),str(index_y[0])
    str1 = ' '.join(c)
    m = main(str1)
    d.append(m)

linked1 = linked(d, conf)
print('[Encrypted Message]: ')

print('--------')

print(linked1)

print('--------')

with open(f'{path_file}/cache/{times1}.node', 'wb') as node:
    pickle.dump(linked1, node)

connex = sqlite3.connect('user_encrypted_data.db')
try:
    connex.execute('''
            CREATE TABLE DATA
                (ID TEXT         NOT NULL,
                NODO            BLOB     NOT NULL);''')
    print("Tabla creada exitosamente")
except sqlite3.OperationalError:
    print("La tabla DATA ya existe")
connex.execute("INSERT INTO DATA (ID,NODO) VALUES (?,?)", (usr, times1))
connex.commit()

cursor = connex.execute("select ID, NODO from DATA")

for fila in cursor:
    print(fila[1])
```

```
connex.close()
```

- main2.py es el archivo a ejecutar para hacer el proceso de desencriptación del archivo seleccionado.

```
import pickle

import sys

sys.path.insert(0, '{path_file}')

from faces1 import send2

a = send2()

continue1 = False

with open('{path_file}/cache/conf.conf', 'rb') as data:
    conf = pickle.load(data)
    conf = conf.decode('ascii')
    conf = int(conf)
print("B: ",str(conf))

b = a-2

while b < a+2:
    print(b)
    if b == conf:
        continue1 = True
        break
    else:
        b += 1

while continue1:

    import sqlite3
    import pickle
    import sys
    import numpy as np
    import getpass
```

```
    sys.path.insert(1,'{path_file}/linked_list')
    sys.path.insert(2,'{path_file}/gamal')

    from c import linked
    from c import rotate_list_inv
    from g import main1

    conn = sqlite3.connect('user_encrypted_data.db')
    cursor1 = conn.execute("select ID, NODO from DATA")

    for fila in cursor1:
        print("User: ", f'{fila[0]}', "    Data: ", fila[1])

    conn.close()

    usr1 = input("Usuario a usar: ")
    slec = input("Datos a Desbloquear: ")
    pasw1 = input("Contraseña a Desbloquear: ")

    with open(f'{path_file}/cache/{slec}.data', 'rb') as data:
        matrix = pickle.load(data)

    with open(f'{path_file}/cache/{slec}.node', 'rb') as node:
        linked = pickle.load(node)

    linked2 = rotate_list_inv(linked, conf)
    print('[Mensaje a Desencriptar]: ')

    print('--------')

    print(linked2)

    print('--------')

    matrix_b = np.array([], dtype = int)
    current = linked2
    while current != None:
        wa   =   main1(current.data[0],   current.data[1],   current.data[2],
current.data[3])
        a1 = ''
```

```
        a2 = ''
        for id in range(len(wa)):
            if wa[id] == ' ':
                a1 = int(wa[:id])
                a2 = int(wa[id+1:])
        matrix_b = np.append(matrix_b,matrix[a1][a2])
        wa = ''
        current = current.next
    power1 = len(usr1) ** len(matrix_b)

    def final_msg(a,b):
        final_msg = ''
        for i in a:
            ay = round( i / power1)
            if ay <= 9:
                final_msg += ''.join(str(ay))
            else:
                final_msg += ''.join(chr(ay))
        if final_msg[:len(b)] == b:
            print("[Mensaje]: ",final_msg[len(b)+1:])
        else:
            print("Contraseña Incorrecta!")

    for i in pasw1:
        if i == '0':
            pasw1 = pasw1.replace('0', str(len(usr1)))
    final_msg(matrix_b, pasw1)
    continue1 = False
```