Python y Linux 1
Nombre del alumno: Nayeli Gissel Larios Pérez
Proyecto: ObjetoSeguro

Objetivo.

Desarrollar la infraestructura básica para un sistema seguro de intercambio de mensajes.

- Los requerimientos de ejecución son los siguientes:
- Se debe asignar un hilo de ejecución a cada conversación, ya sea entrante o saliente.
- Cada objeto debe tener un socket servidor para recibir peticiones de conversación.
- Cuando un objeto desee iniciar una conversación comenzará una conexión a un socket de otro objeto utilizando un socket cliente.
- Cada objeto debe tener un archivo de BD que utilizará para almacenamiento de mensajes y de llaves, este archivo debe estar en la misma carpeta que el código.py que crea el objeto. Las direcciones y puertos del servidor de otros objetos son introducidos de manera manual. Cuando los sockets establecen comunicación, el socket cliente envía su nombre y llave pública al socket servidor, este responde con su nombre y llave pública.
- Cuando se ha llevado a cabo el intercambio de llaves públicas, se comienza la comunicación segura de acuerdo al diagrama de comunicación.

Introducción.

En un contexto de envío y recepción de mensajes, utilizar las propiedades de protección de atributo/método en Python para las funciones básicas de seguridad: Cifrado, descifrado y manejo adecuado de la llave.

Una de las principales vulnerabilidades en los esquemas de cifrado/descifrado es el manejo de llave, es por esto que es conveniente tener la posibilidad de esconder la llave privada.

Desarrollo.

Los mensajes enviados y recibidos se almacenan automáticamente en un archivo de texto llamado: RegistoMsj_nombreDelObjetoSeguro.txt.

Clase: ObjetoSeguro(nombre: str)
Descripción: Creación de un objeto seguro, se recibe el nombre en formato String.

Método: gen_llaves()
Descripción: Genera la llave privada y su correspondiente llave pública.

Método: saludar(name: str, msj: str)
Descripción: Método público accedido por el objeto que quiere comenzar la comunicación, los parámetros de entrada son el nombre del objeto que comienza la comunicación y el mensaje cifrado con la llave pública del destinatario.

Método: responder(msj: str) -> byte.

Python y Linux 1
Nombre del alumno: Nayeli Gissel Larios Pérez
Proyecto: ObjetoSeguro
Descripción: Este método se ejecuta automáticamente al recibir un saludo de otro objeto seguro, el parámetro de entrada es el nombre del objeto al que se responderá. La respuesta deber ser el mensaje recibido, concatenado con el string “MensajeRespuesta”.

Método: llave_publica() -> str.
Descripción: Este método sirve para obtener la llave pública del objeto seguro.

Método: cifrar_msj(pub_key: str, msj: str) -> bytes.
Descripción: Este método sirve para cifrar un mensaje con la llave pública del destinatario, el retorno es el mensaje cifrado.

Método: descifrar_msj(msj: byte) -> byte.
Descripción: Este método sirve para descifrar un mensaje cifrado, el retorno es el mensaje en texto plano codificado en base64.

Método: codificar64(msj: str) -> bytes.
Descripción: Este método sirve para codificar un mensaje en texto plano en base64, el retorno es el mensaje en texto plano codificado en base64.

Método: decodificar64(msj: bytes) -> str.
Descripción: Este método sirve para decodificar un mensaje en base64 un mensaje en texto plano, el retorno es el mensaje en texto plano.

Método: almacenar_msj(msj: str) -> dict.
Descripción: Este método sirve para almacenar un mensaje en texto plano en un archivo de texto y le es asignado un ID para identificarlo. El retorno es el ID asignado en el formato {“ID”:}. Nota: el ID y el mensaje es la mínima información que deberá tener el registro, el alumno puede asignar más campos de acuerdo con su criterio.

Método: consultar_msj(id: int) -> dict.
Descripción: Este método sirve para consultar un mensaje del registro en el archivo de texto utilizando el ID asignado. El retorno de esta función es el mensaje en el siguiente formato {“ID”:”id”, MSJ”:”Mensaje_consultado”, “campo1”:”valor1”, ...}

Método: esperar_respuesta(msj: byte)
Descripción: Este método sirve para esperar una respuesta cifrada con llave pública que se desencadena de un hacer un saludo a otro objeto. Este método debe llamar al método para almacenar automáticamente el mensaje de respuesta recibido en texto plano.

Python y Linux 1
Nombre del alumno: Nayeli Gissel Larios Pérez
Proyecto: ObjetoSeguro

Diagrama.

Diagrama UML

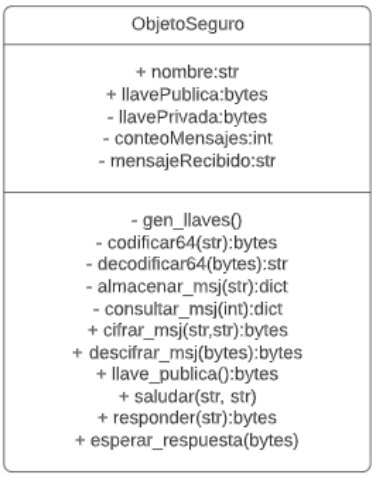
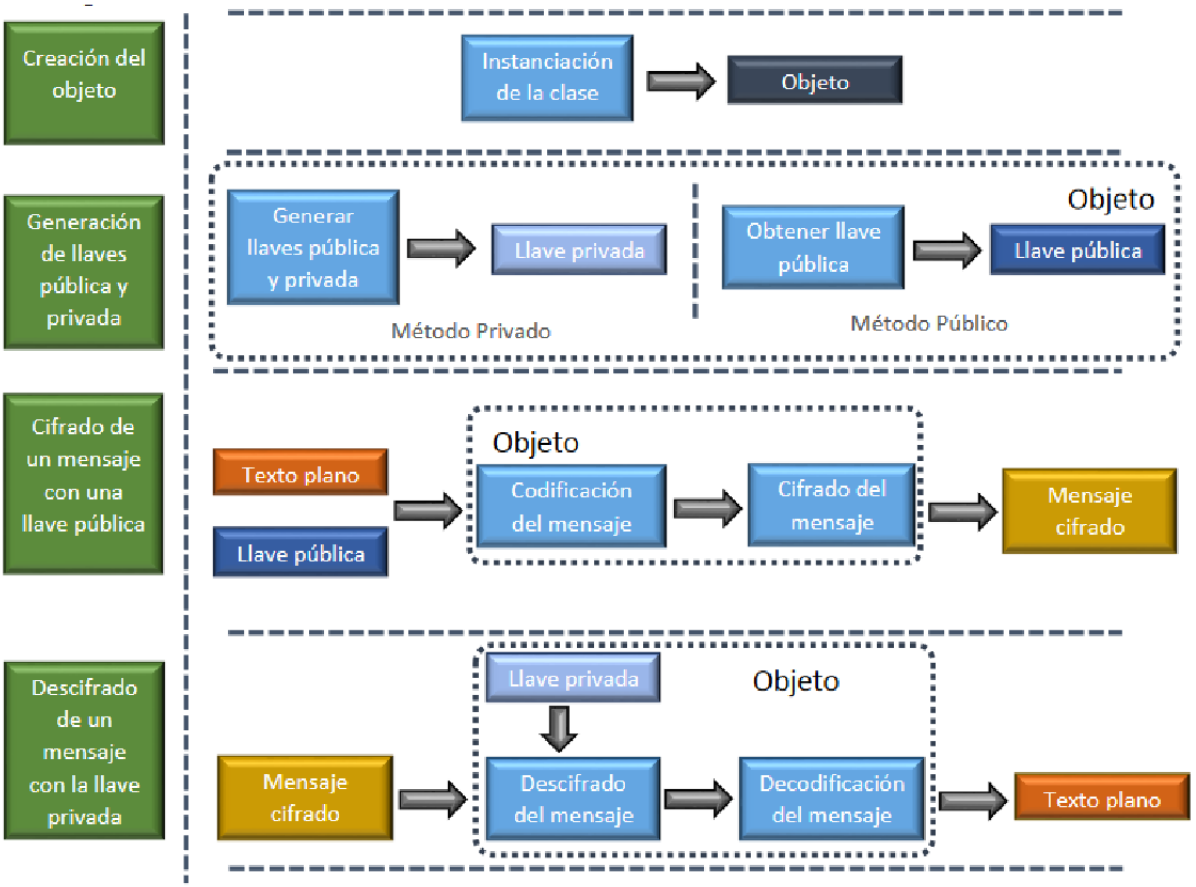


Diagrama de flujo



Python y Linux 1
Nombre del alumno: Nayeli Gissel Larios Pérez
Proyecto: ObjetoSeguro

Código(s).

```
#   Desarrollo de talento especializado 2021-2: Ciberseguridad
#   Python y Linux
#   PL_Proyecto1 - main
#   Nayeli Gissel Larios Pérez

from objetoSeguro import ObjetoSeguro

if __name__ == '__main__':
    print("CREACION DE OBJETO SEGURO")
    nombre = input("Nombre del objeto : ")
    mi_puerto = int(input("Mi puerto : "))
    puerto_destino = int(input("Puerto destino : "))
    Mensajero = ObjetoSeguro(nombre, mi_puerto, puerto_destino)
    Mensajero.espera_conexion()
    Mensajero.inicia_comunicacion()
    Mensajero.termina_comunicacion()



#   Desarrollo de talento especializado 2021-2: Ciberseguridad
#   Python y Linux
#   PL_Proyecto1 - Objeto seguro
#   Nayeli Gissel Larios Pérez

from concurrent.futures import ThreadPoolExecutor
from Crypto.PublicKey import RSA
from Crypto.Cipher import PKCS1_OAEP
import base64
import logging
from socket_cliente import SocketClient
from socket_servidor import SocketServer

logging.basicConfig(format='\tDEBUG : %(message)s', level=logging.DEBUG)

KEY_SIZE = 32  # Para AES-256


# Clase ObjetoSeguro
class ObjetoSeguro:
    def __init__(self, nombre: str, puerto_servidor: int, puerto_cliente: int):
        # Atributos de la clase ObjetoSeguro
        logging.debug("OBJETO : {}".format(nombre))
        self.nombre = nombre
        self.llavePublica, self.__llavePrivada = self.__gen_llaves()
```

Python y Linux 1
Nombre del alumno: Nayeli Gissel Larios Pérez
Proyecto: ObjetoSeguro

```
        self.__conteoMensajes = 0
        self.__mensajeRecibido = ""
        self.socketcliente = SocketClient(puerto_cliente)
        self.socketservidor = SocketServer(nombre, puerto_servidor)
        self.comunicacion = ThreadPoolExecutor(max_workers=3)
        self.llavePublicaReceptor = bytearray()
        self.write = None
        self.read = None
        self.captura = None

    # Métodos del ObjetoSeguro
    # Metodo privado que, al instanciar el objeto genera la llave publica y privada
    @staticmethod
    def __gen_llaves():
        secreto = RSA.generate(1024)
        privada = secreto.exportKey()
        publica = secreto.publickey().exportKey()
        return publica, privada

    # Metodo privado que codifica el str ascii ingresado a byte de base64
    @staticmethod
    def __codificar64(msj: str) -> bytes:
        aux = base64.b64encode(msj.encode("ascii"))
        return aux

    # Metodo privado que decodifica byte de base64 a string ascii
    @staticmethod
    def __decodificar64(msj: bytes) -> str:
        aux = base64.b64decode(msj)
        return aux.decode("ascii")

    # Metodo privado que almacena un mensaje recibido en un archivo de texto
    @staticmethod
    def __consultar_msj(identificador: int) -> dict:
        archivo = open("RegistoMsj_{self.nombre}.txt", "r")
        with archivo as f:
            for texto in f:
                linea = texto.split(",", maxsplit=1)
                extrae_id = linea[0].split(":")
                extrae_txt = linea[1].split(":")
                if int(extrae_id[1]) == id:
                    print(f"ID {identificador}: {extrae_txt[1]}")
        aux = dict(
            ID=id,
            MSJ=extrae_txt[1]
        )
        return aux
```

Python y Linux 1
Nombre del alumno: Nayeli Gissel Larios Pérez
Proyecto: ObjetoSeguro

```
    # Metodo publico que cifra un mensaje con una llave publica
    def cifrar_msj(self, pub_key, msj: str) -> bytes:
        llave_publica = RSA.importKey(pub_key)
        llave_publica = PKCS1_OAEP.new(llave_publica)
        texto_cifrado = llave_publica.encrypt(self.__codificar64(msj))
        return texto_cifrado

    # Metodo publico que descifra un mensaje con una llave publica
    def descifrar_msj(self, msj: bytes) -> bytes:
        llave_privada = RSA.importKey(self.__llavePrivada)
        llave_privada = PKCS1_OAEP.new(llave_privada)
        texto_descifrado = llave_privada.decrypt(msj)
        return texto_descifrado

    # Metodo publico con el que se obtiene la llave publica del objeto
    # No es string por que la biblioteca crea las llaves con otro formato
    def llave_publica(self):
        return str(self.llavePublica)

    def esperar_respuesta(self):
        print("Procesando una respuesta")
        # mensaje_respuesta = self.__codificar64("MensajeRespuesta")
        # mensaje_codificado = msj.rstrip(mensaje_respuesta)
        # print(self.decodificar64(self.descifrar_msj(mensaje_codificado)))
        # self.__almacenar_msj(self.__decodificar64(self.descifrar_msj(mensaje_codificado)))
        return

    def espera_conexion(self):
        inicializa = self.socketservidor.inicializa_socket()
        conectar = self.socketcliente.connect()
        while not conectar.done() and not inicializa.done():
            pass
        logging.debug("OBJETO : Conexion completa")

    def inicia_comunicacion(self):
        self.write = self.comunicacion.submit(self.socketcliente.write)
        self.read = self.comunicacion.submit(self.socketservidor.read)
        self.intercambia_llaves()
        logging.debug("OBJETO : ----------inicia conexion segura----------")
        self.captura = self.comunicacion.submit(self.captura_mensaje)

    def intercambia_llaves(self):
        aux = 1
        self.socketcliente.write_text(self.llave_publica())
        while aux:
            if '-BEGIN PUBLIC KEY-' in self.socketservidor.ultimo_mensaje:
```

Python y Linux 1
Nombre del alumno: Nayeli Gissel Larios Pérez
Proyecto: ObjetoSeguro

```
            logging.debug("OBJETO : Llave recibida!")
            self.llavePublicaReceptor = self.socketservidor.ultimo_mensaje
            aux = 0

    # Metodo para capturar mensaje de la terminal
    def captura_mensaje(self):
        while True:
            mensaje = input()
            # llave2 = str.encode(self.llavePublicaReceptor)
            # mensaje_cifrado = self.cifrar_msj(llave, mensaje)
            # self.socketcliente.write_text(mensaje_cifrado)
            self.socketcliente.write_text(mensaje)

    def termina_comunicacion(self):
        while not self.write.done(): # and not self.write.done():
            pass
        self.read.cancel()
        self.captura.cancel()
        self.socketcliente.close()
        self.socketservidor.close()

#   Desarrollo de talento especializado 2021-2: Ciberseguridad
#   Python y Linux
#   Proyecto final - socket_cliente
#   Nayeli Gissel Larios Pérez
from concurrent.futures import ThreadPoolExecutor
import logging
import socket

LOCAL_HOST = '127.0.0.1'
logging.basicConfig(format='\tDEBUG : %(message)s',
                    level=logging.DEBUG)


class SocketClient:
    def __init__(self, puerto: int):
        # atributos de la clase SocketClient
        # Crea el socket de comunicacion de tipo stream
        self.node = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        # Se especifica la ip y el puerto de conexion
        self.port_and_ip = ('127.0.0.1', puerto)
        # Administrador de threads un hilo para recepcion y otro para transmision
        self.tpe_comunicacion = ThreadPoolExecutor(max_workers=4)
        # Atributo donde se almacena la respuesta
        self.resp = ""
        logging.debug("CLIENTE : socket creado {}".format(self.port_and_ip))
```

Python y Linux 1
Nombre del alumno: Nayeli Gissel Larios Pérez
Proyecto: ObjetoSeguro

```
    # Metodo que buscara servidor al que se quiere conectar hasta encontrarlo
    def busca_servidor(self):
        while True:
            try:
                self.node.connect(self.port_and_ip)
                break
            except ConnectionRefusedError:
                pass
        logging.debug("CLIENTE : conectado a {}".format(self.port_and_ip))

    def connect(self):
        # Conecta el socket a la direccion especificada
        return self.tpe_comunicacion.submit(self.busca_servidor)

    # Metodo que cierra el socket
    def close(self):
        self.node.shutdown(socket.SHUT_RDWR)
        self.node.close()
        logging.debug("CLIENTE : socket cerrado")

    # Metodo que envia el mensaje por el socket
    def send_sms(self, sms):
        self.node.send(sms.encode())
        self.resp = ""

    # Metodo que procesa la informacion que se va a mandar
    def write(self):
        while self.resp != "exit":
            if self.resp == "":
                pass
            else:
                aux = self.resp
                self.send_sms(aux)
        self.send_sms("exit")
        logging.debug("CLIENTE : exit")
        return

    # Metodo que recibe un texto a enviar desde le programa
    def write_text(self, texto):
        self.resp = texto
```

Python y Linux 1
Nombre del alumno: Nayeli Gissel Larios Pérez
Proyecto: ObjetoSeguro

Conclusión.

Para el desarrollo de este proyecto se utilizaron todos los conceptos revisados en clase. Se utilizó programación orientada a objetos para modelar el objeto seguro y sus componentes como los socket cliente y servidor.
Una vez establecida la conexión y ejecutado el intercambio de llaves se utilizaron threads para la comunicación entre los sockets, la sincronización de estos fué muy importante por lo que se implementó un tercer thread que sólo se encargó de capturar el texto desde la terminal.
También se utilizaron funciones para manejo de archivos para almacenar el historial de la comunicación. Y como mejora pendiente se queda la implementación de una base de datos mediante algún controlador de python para mySQL.