

Instituto Tecnológico y de Estudios Superiores de Monterrey  
Escuela de Ingeniería y Ciencias

Ingeniería en Ciencia de Datos y Matemáticas  
Uso de Álgebras Modernas para Seguridad y Criptografía



**MANUAL DE PROGRAMADOR: Caso Fundación  
Teletón.**

Como parte del reto propuesto por:

**Fundación Teletón**



Mauricio Hernández López - A01173961  
Nemesio Navarro Arámbula - A01284430  
Freddy Sebastián Silva Culcay - A00828792  
Bárbara García Quiroga - A01720402  
Roel Adrián De la Rosa Castillo - A01197595  
Fernando Guevara Moreno - A00828723  
Ricardo Díaz Garza Cabello -A00826848

Profesores:

Alberto Francisco Martínez Herrera  
Daniel Otero Fadul

Monterrey, Nuevo León a 04 de mayo del año 2022

# Índice

1. Características principales	3
2. Instalación	3
3. Documentación de la aplicación	4
4. Unión del Código	17
5. Vectores de Prueba	23
6. Licencias utilizadas	24
7. Preguntas frecuentes	25
8. Contacto	26

En este documento, se muestra la implementación segura del algoritmo de generación de llaves, firma digital y verificación de firma. Además, se explica las características del sistema operativo al igual que el algoritmo detalladamente.

## 1. Características principales

Se realizó el algoritmo en el sistema operativo **Laptop Gamer HP Pavilion**. Este dispositivo tiene una memoria **8 de RAM**, con un procesador **Intel(R) Core(TM) i5-9300H** al igual que tiene la edición y versión **Windows 11 Home Single Language 21H2**.

El algoritmo está implementado en el lenguaje de programación **Python Versión 3.9.7** utilizando en el intérprete y compilador de **Visual Studio Code 1.66.2**.

## 2. Instalación

Las librerías implementadas en el algoritmo están sujetas a la última versión de las librerías al día 5 de mayo de 2022, por lo cual, el usuario que busque implementar el algoritmo que se muestra en la próxima sección tiene de responsabilidad actualizarlas para su funcionalidad. Igualmente la versión que fue utilizada en la implementación del código fue la **3.9.7**, por lo que está a disposición del usuario tener una versión actualizada y compatible con las librerías necesarias.

```
import PySimpleGUI as sg
from OpenSSL import crypto
import OpenSSL.crypto
from Crypto.PublicKey import RSA
import hashlib
from Crypto.Hash import SHA256
from Crypto.PublicKey import ECC
from Crypto.Signature import pss
from datetime import datetime
from OpenSSL import crypto, SSL
```

Las librerías que se muestran son las necesarias para el funcionamiento del algoritmo. De las cuales hay librerías que ya vienen incluidas por parte de Python y unas que son necesarias descargar. Si no se tienen las librerías es necesario descargarlas de la siguiente forma en la terminal:

- pip install PySimpleGUI
- pip install pycryptodome
- pip install hashlib
- pip install cffi
- pip install PyQt5
- pip install pycparse

Una vez instalado cada una de las librerías, la aplicación ya es funcional y se puede hacer la verificación y generación de firmas y la firma digital.

### 3. Documentación de la aplicación

```
# Coleccion de usuarios y contraseñas. Debe moverse a base de datos y/o
    archivo contenedor
admin_usernames = ["admin"]
usernames = ["user1", "user2"]
passwords = ["123", "abcd"]

curr_usr = ""
```

En la sección anterior del código se muestra en lista los usuarios y contraseñas para acceder a la plataforma de verificación y generación de firmas. Para una implementación en el mundo real, esta parte debe de estar vinculada con una base de datos segura, de tal manera que para poder acceder al sistema se necesiten tener credenciales válidas. Cabe mencionar que hay distinción entre el administrador y los usuarios puesto que solo el primero consta de permiso para generar llaves.

```
#Funcion que verifica que el certificado corresponda a la firma

def check_certificate(certificate, key):
    try:
        p_key = open(key).read()
        load = crypto.load_privatekey(crypto.FILETYPE_PEM, p_key)

    except crypto.Error:
        sg.popup("Archivo de llave incorrecto")

    try:

        certific = crypto.load_certificate(crypto.FILETYPE_PEM, open(
certificate).read())

    except crypto.Error:
        sg.popup("Archivo de certificado incorrecto")

    verifier = SSL.Context(OpenSSL.SSL.TLSv1_METHOD)
```

```

verifier.use_privatekey(load)
verifier.use_certificate(certific)
try:
    verifier.check_privatekey()
    return True
except SSL.Error:
    sg.popup("Certificado no coincide")
    return False

```

Esta función nos permite verificar que el certificado corresponda a la firma generada. Con el comando Try y Except, permite comprobar que efectivamente cumpla con los requerimientos para pasar como válida o denegada. Al final de esta función se devuelven los siguientes mensajes: Archivo de llave incorrecto, Archivo de certificado incorrecto y Certificado no coincide.

Los tres posibles resultados de la verificación del certificado de firma, parten de que el archivo de llave o certificado no es el correcto, y en caso de que sea el correcto este último, no coincide con la firma.

```

def check_date_validity(certificate):

    try:

        certific = crypto.load_certificate(crypto.FILETYPE_PEM, open(
certificate).read())

    except crypto.Error:
        sg.popup("Archivo de certificado incorrecto")

    expiration_date = certific.get_notAfter().decode()
    y = int(expiration_date[:4])
    m = int(expiration_date[4:6])
    d = int(expiration_date[6:8])

    expiration_date = datetime(y, m ,d)
    today = datetime.now()

    if expiration_date > today:
        return True

    else:

```

```

sg.popup("El certificado ha expirado y no se puede firmar con el.
Por favor solicite uno nuevo.")
return False

```

Ahora bien, esta función devuelve un mensaje de texto bastante simple que nos indica que el archivo de certificado dista de ser el correcto, o bien, que la fecha de este certificado expiró y no es posible firmarse. Finalmente se pide al usuario que solicite uno nuevo.

El funcionamiento es bastante simple, primero se comprueba que el certificado sea el correcto. Después, con la función `expiration date` se extrae el día, mes y año. Finalmente con un condicional, se comprueba si la fecha de expiración es mayor a la del día de hoy, en caso de cumplirse, se devuelve un valor `True`.

```

#Funcion que genera un par de llaves publica-privada siguiendo el
algoritmo RSA
def rsa_key_generation(city="Monterrey", state="Nuevo Leon"):

    priv_name = curr_usr+"_priv.pem"
    pub_name = curr_usr+"_pub.pem"
    cert_name = curr_usr+"_cert.crt"

    #Clave de 2048 bits (o 256 bytes)
    key = RSA.generate(2048)
    #Se exporta en formato PEM para facilitar su lectura
    private_key = key.export_key(format="PEM", pkcs=8)
    file_out = open(priv_name, "wb")
    file_out.write(private_key)
    file_out.close()

    public_key = key.publickey().export_key(format="PEM")
    file_out = open(pub_name, "wb")
    file_out.write(public_key)
    file_out.close()

    certificate_generation(name = curr_usr, country= "MX", country_code="
MX",
    city=city, state=state, organiz="TELETON MTY", serial=0, validity_s=0,
    validity_e=60*20, pu_key=pub_name, pr_key=priv_name,
    output_f=cert_name)

```

En el código anterior se tiene la función `rsa_key_generation()`, la cual está diseñada para que genere las llaves privada y pública que posteriormente serán utilizadas para la generación y verificación de las firmas

digitales. En esta función se generan las dos llaves usando el algoritmo RSA a partir de la librería **PyCryp-**  
**ptodome**. Estas llaves son posteriormente guardadas y exportadas en archivos PEM (Privacy-Enhanced  
Mail). Estas llaves tienen una longitud de 2048 bits o 256 bytes. Como se mencionó anteriormente, solo el  
administrador tiene permiso de generar llaves.

```
#Funcion que lee los bits de un archivo de cualquier tipo y les aplica la
funcion SHA-256
def get_file_hash(file_name):

    file_hash = hashlib.sha256()

    #Se usa especificamente el modo 'rb' para leer en bits
    with open(file_name, "rb") as file:
        ck = 0
        #Se lee de 1024 en 1024 para evitar errores de lectura
        while ck != b'':
            ck = file.read(1024)
            file_hash.update(ck)

    #Devuelve un hash especifico del archivo. No puede haber 2 iguales
    para archivos distintos
    return file_hash.hexdigest()
```

Usando la función *get\_file\_hash()* se trabaja con el documento que se quiere firmar y se transforma dicho  
documento en un objeto hash. Este hash se crea a partir de la función *hashlib.sha256()* de la librería **hashlib**.  
Esta función nos permite crear un objeto hash con formato SHA256 y, con el documento deseado, generar el  
hash que se usará en la generación y verificación de firma digital.

```
#Interfaz y algoritmo de firma de documentos
def sign_document():
    sg.theme("LightBlue2")
    layout = [[sg.T("")],
               [sg.Text("Escoger un documento: "), sg.Input(), sg.FileBrowse(key="-IN-"),
               [sg.Text("Escoger una llave privada: "), sg.Input(), sg.FileBrowse(key="-KEY-"),
               [sg.Text("Escoger un certificado: "), sg.Input(), sg.FileBrowse(key="-CERT-"),
               [sg.Button("Firmar"), sg.Button("Atras")]]]
```

```

window = sg.Window("Firma de documentos", layout)

while True:
    event, values = window.read()
    if event == sg.WIN_CLOSED or event == "Exit":
        break
    elif event == "Firmar":

        if check_certificate(certificate = values["-CERT-"], key=
values["-KEY-"]):
            if check_date_validity(certificate = values["-CERT-"]):
                progress_bar(message="Firmando documento...")
                #Se leen valores de la ventana
                doc = values["-IN-"]
                k = values["-KEY-"]
                file_hash = str(get_file_hash(doc)) #Se hashea el
documento para crear un ID identificador
                key = RSA.import_key(open(k).read()) #Se lee la llave
privada
                hs = SHA256.new(file_hash.encode("utf-8")) #Se hashea
el identificador por motivos de seguridad
                sgnr = pss.new(key)
                signature = sgnr.sign(hs)
                file_out = open("document_signature.pem", "wb") #Se
firma y se escribe dicha firma en otro archivo PEM
                file_out.write(signature)
                file_out.close()
                sg.popup("Firma exitosa")

            elif event == "Atras":
                window.close()
                signing_interface()
                break

```

En la función anterior se diseña la interfaz que tendrá el usuario al momento de firmar los documentos. En esta interfaz se tienen dos cajas de texto, en las cuales el usuario ingresará, ya sea escribiendo manualmente o buscando en las carpetas de su computadora, los directorios del archivo que se desea firmar y la llave



pública con la que se firmará. A su vez también se tienen dos botones, uno para regresar al menú anterior y otro botón con el que se hará el proceso de la firma digital. Si se hace clic al botón de firmado, usando el algoritmo RSA se crea un archivo con formato PEM (Privacy-Enhanced Mail) y en éste se guardan los valores generados para la firma digital. Al momento de realizarse la firma se crea una barra de progreso y cuando ésta termina, se crea un pop-up que imprime 'firma exitosa'.

```
#Interfaz y algoritmo de verificacion de firmas
def verify_signature():
    sg.theme("LightBlue2")
    layout = [[sg.T("")], [sg.Text("Escoger una llave publica: "), sg.
Input(), sg.FileBrowse(key="-IN-")],
    [sg.Text("Escoger una firma:      "), sg.Input(), sg.FileBrowse(key="-
SIG-")],
    [sg.Text("Escoger un documento:   "), sg.Input(), sg.FileBrowse(key=
"-DOC-")],
    [sg.Button("Verificar"), sg.Button("Atras")]]

    window = sg.Window("Verificacion de fimas", layout)

    while True:
        event, values = window.read()
        if event == sg.WIN_CLOSED or event == "Exit":
            break
        elif event == "Verificar":

            progress_bar(message="Verificando firma...")
            #Se leen los contenidos de la ventana
            doc = values["-DOC-"]
            p_key = values["-IN-"]
            sig = values["-SIG-"]

            file_hash = str(get_file_hash(doc)) #Se hashea el documento de
nuevo para verificar la autenticidad de la firma

            sig = open(sig, "rb") #Se lee la firma en bits
            sig = sig.read()

            key = RSA.import_key(open(p_key).read())
```

```

        hs = SHA256.new(file_hash.encode("utf-8")) #Se vuelve a
hashear el ID del documento

        verifier = pss.new(key) #Se inicia una instancia de
verificacion basada en la clave publica

        try: # Manejo de excepciones para comprobar si la firma es
legitima o no
            verifier.verify(hs, sig)
            sg.popup("La firma es autentica.")
        except (ValueError, TypeError):
            sg.popup("La firma no es autentica.")

    elif event == "Atras":
        window.close()

    signing_interface()
    break

```

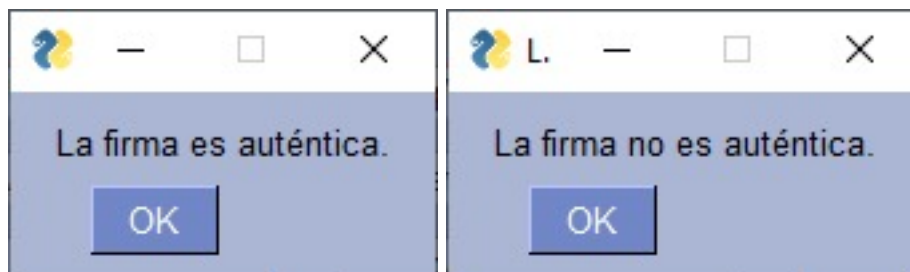


Figura 1: Ventana emergente para comprobar si la firma es legítima o no.

El código anterior demuestra la ventana en donde el usuario podrá verificar la veracidad de la firma. La ventana utiliza la librería PySimpleGUI versión 4.57.0. para su interfaz. El usuario tiene 4 botones a elegir, puede subir un documento que contiene la firma, el documento a firmar y la llave pública para posteriormente seleccionar el botón de "Verificar". Así mismo, el usuario puede elegir "Atrás" para regresar al menú principal. Al momento de seleccionar la opción de verificar, se abre una ventana en el cual se crea una barra de progreso y cuando ésta termina, se crea un pop-up que imprime "**La firma es auténtica**" al ser correcta y "**La firma no es auténtica**" en caso contrario. La librería utilizada es haslib versión 3.9.7 y Crypto.Hash, perteneciente a PyCryptodome versión 3.14.1 en las cuales se encargan de hashear el ID del documento e iniciar una instancia de verificación basada en la clave pública.

```

def certificate_generation(name, country, country_code,

```

```
city, state, organiz, serial, validity_s, validity_e, pu_key, pr_key,
output_f):
```

```
cert = crypto.X509()
cert.get_subject().C = country
cert.get_subject().ST = state
cert.get_subject().L = city
cert.get_subject().O = organiz
cert.get_subject().OU = "unit"
cert.get_subject().CN = name
cert.get_subject().emailAddress = "example@email.com"
cert.get_issuer().C = country
cert.get_issuer().ST = state
cert.get_issuer().L = city
cert.get_issuer().O = organiz
cert.get_issuer().OU = "unit"
cert.get_issuer().CN = name
cert.get_issuer().emailAddress = "example@email.com"
cert.set_serial_number(serial)
cert.gmtime_adj_notBefore(0)
cert.gmtime_adj_notAfter(validity_e)
cert.set_issuer(cert.get_issuer())
with open(pu_key, "r") as p_key:
    p_key_s = p_key.read()
    a = crypto.load_publickey(crypto.FILETYPE_PEM, p_key_s)
cert.set_pubkey(a)

with open(pr_key, "r") as pri_key:
    pri_key_s = pri_key.read()
    k = crypto.load_privatekey(crypto.FILETYPE_PEM, pri_key_s)

cert.sign(k, 'sha256')
with open(output_f, "wt") as f:
    f.write(crypto.dump_certificate(crypto.FILETYPE_PEM, cert).decode(
"utf-8"))
```

La función `certificate generation` recibe distintos parámetros de entrada a los cuales se les asignan

ciertas variables una vez ejecutada la función. Utilizando la función `crypto` de la librería OpenSSL, específicamente `crypto.X509()`. Entre los parámetros que se solicitan al usuario se encuentran el nombre, país, código del país, estado, ciudad, organización, serial.

El certificado que se genera contiene todos y cada uno de los parámetros declarados en la función. Son importantes también la clave pública y el tipo de hash que se estará utilizando al momento de firmar, así como la fecha en la que fue generado el certificado; esto para más adelante comprobar la expiración de este.

Finalmente, se exporta y se guarda el certificado creado como archivo `.crt`.

```
#Animacion de la barra de progreso
def progress_bar(message):
    sg.theme('LightBlue2')
    layout = [[sg.Text(message)],
               [sg.ProgressBar(1000, orientation='h', size=(20, 20), key='
progressbar')]],
               [sg.Button("Cancelar")]]

    window = sg.Window('Espere', layout)
    for i in range(1000):
        event, values = window.read(timeout=1)
        if event == 'Cancelar' or event == sg.WIN_CLOSED:
            break
        window['progressbar'].update_bar(i + 1)
    window.close()
```

El código anterior demuestra la animación de las barras de progreso, utiliza la librería PySimpleGUI versión 4.57.0. para crear el tamaño, color y tiempo. Es una animación sencilla donde se carga la barra de progreso en medida del tiempo que toma la verificación de la firma.

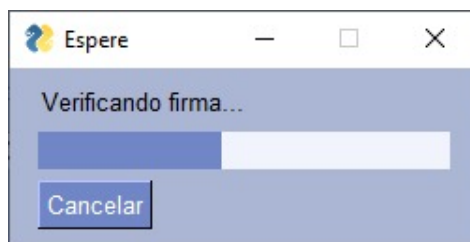


Figura 2: Animación de la barra de progreso

```
#Interfaz de la opcion "Firmar documento"
def signing_interface():

    sg.theme("LightBlue2")
    layout_validation = [[sg.Text('Selecccionar una opcion:'),
```

```

        [sg.Button('Firmar Documento'),sg.Button('Verificar
Firma'), sg.Button("Atras")]]

window = sg.Window("Fima y validacion", layout_validation)

while True:
    event, values = window.read()

    if event == sg.WIN_CLOSED:
        break

    elif event == "Atras":
        window.close()
        menu()
        break

    elif event == "Firmar Documento":
        window.close()
        sign_document()
        break

    elif event == "Verificar Firma":
        window.close()
        verify_signature()
        break

```

La función `signing interface` muestra la ventana en donde el usuario podrá elegir entre firmar o verificar la firma de un documento. Los botones a elegir son "Firmar Documento" en donde mandará al autor a la ventana donde podrá subir los documentos correspondientes para dicha firma, "Verificar Firma" donde el usuario podrá verificar la firma con la respectiva llave y finalmente el botón de "Atras" en la que dirige al usuario al menú principal. Utiliza la librería PySimpleGUI versión 4.57.0.

```

#Interfaz de la opcion "Creacion de llaves"
def gen_signature():
    sg.theme("LightBlue2")

    layout_key_gen = [[sg.Text('Seleccionar una opcion:'),
                        [sg.Button('Generar llave'),sg.Button('Atras')]]]

```

```

window = sg.Window("Generacion de Claves", layout_key_gen)

while True:
    event, values = window.read()

    if event == sg.WIN_CLOSED:
        break

    elif event == "Atras":
        window.close()
        menu()
        break

    elif event == "Generar llave":
        progress_bar(message = "Generando par de llaves...")
        rsa_key_generation()
        sg.popup("Generacion completa")

```

La función previa muestra la ventana en la cual únicamente el administrador puede generar las llaves tanto pública como privada. Se muestran dos opciones, "Generar llave" y "Atras". Al seleccionar la opción generar llave, se desplegará una ventana con la barra de progreso que dicta "Generando par de llaves", al terminar, dos archivos se crearán en los documentos del administrador. Posteriormente aparecerá de nuevo la ventana con las dos opciones previamente mencionadas, el administrador podrá seleccionar la opción "Atras" para ir al menú principal. Se utilizó la librería PySimpleGUI versión 4.57.0.

```

#Logica del menu
def menu():
    sg.theme("LightBlue2")
    layout = [[sg.Text("Menu")],
               [sg.Button("Creacion de llaves"), sg.Button("Firma de documentos"), sg.Button("Salir")]]

    window = sg.Window("Menu principal", layout)

    while True:
        event, values = window.read()

        if event == "Salir" or event == sg.WIN_CLOSED:
            break

        elif event == "Creacion de llaves":

```

```

        window.close()
        gen_signature()

    elif event == "Firma de documentos":
        window.close()
        signing_interface()

```

El código demuestra la ventana del menú principal que le aparece al usuario al correrlo. Le aparecen tres opciones a elegir, "Creación de Llaves", "Firma de Documentos" y "Salir". El usuario deberá elegir la opción que requiera en el momento. A excepción de las opciones Creación de Llaves y Firma de Documentos, que son mencionadas en este manual, la opción de Salir cierra sesión. Se utilizó la librería PySimpleGUI versión 4.57.0.

```

#Logica del sistema de inicio de sesion

def login():
    global usernames, admin_usernames, passwords
    sg.theme("LightBlue2")
    layout = [[sg.Text("Ingresar", size=(15, 1), font=40)],
               [sg.Text("Usuario", size=(15, 1), font=16), sg.InputText(key='
-usrnm-', font=16)],
               [sg.Text("Contraseña", size=(15, 1), font=16), sg.InputText(
key='-pwd-', password_char='*', font=16)],
               [sg.Button('Entrar'), sg.Button('Salir')]]

    window = sg.Window("Ingresar", layout)

    while True:
        event, values = window.read()
        if event == "Salir" or event == sg.WIN_CLOSED:
            break
        else:
            if event == "Entrar":
                global curr_usr
                global var_ad
                if values['-usrnm-'] in admin_usernames and values['-pwd-
] in passwords:
                    sg.popup("Bienvenido administrador!")

                global var_ad

```

```

        curr_usr = values['-usrnm-']
        var_ad = "Y"
        window.close()
        return True
        break

    elif values['-usrnm-'] in usernames and values['-pwd-'] in
passwords:
        sg.popup("Bienvenido!")
        curr_usr = values['-usrnm-']
        window.close()
        return True
        break

    elif values['-usrnm-'] not in usernames or values['-pwd-']
not in passwords:
        sg.popup("Informacion incorrecta. Intente de nuevo.")

window.close()

```

Los usuarios deberán iniciar sesión antes de poder realizar cualquier acción como lo son la generación de llaves, firma de documento y verificación de firma. Las claves predeterminadas son, para el administrador:

- Usuario: admin
- Contraseña: 123

y para el usuario:

- Usuario1: user1
- Contraseña1: 123
- Usuario2: user2
- Contraseña2: abcd

El usuario deberá iniciar sesión, si es realizado correctamente, se arrojará una ventana que dice "Bienvenido!", caso contrario dirá "Información incorrecta. Intente de nuevo.". Utiliza la librería PySimpleGUI versión 4.57.0.

```

#Funcion principal
def main():

    if login():
        menu()

```



```
#Inicio del programa
main()
@ 2022 GitHub, Inc.
```

El código llama las funciones para inicio de sesión y la ventana del menú principal.

## 4. Unión del Código

Como se muestra anteriormente, el código se dividió por cada ventana emergente que le aparece al usuario al querer firmar o verificar la firma de un documento. Esta sección consiste en la explicación de cómo el código visto anteriormente se une con la finalidad de comprender la vista del usuario al correr el programa.

Al iniciar, al usuario se le solicita iniciar sesión con su cuenta de usuario y contraseña como se ve en la siguiente imagen, al llenar los datos correctos, el usuario es dirigido al menú principal. El usuario tiene diferentes opciones a elegir, creación de llaves, firma de documentos y salir.

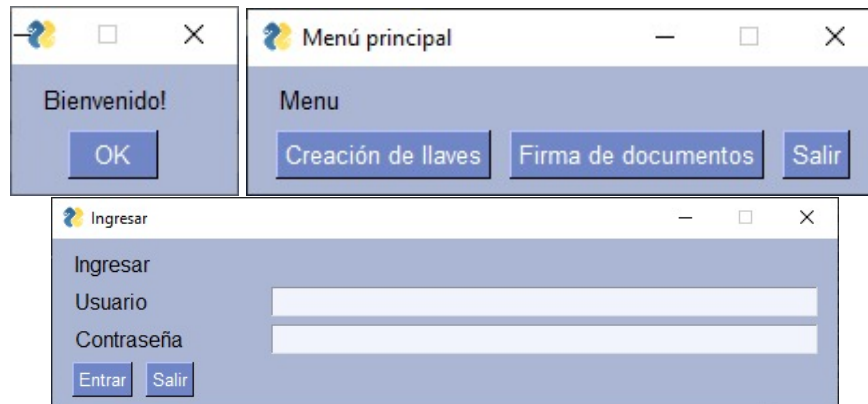


Figura 3: Ventana de Bienvenida, Inicio de Sesión y Menú Principal

Cabe recalcar que, para el usuario, la interfaz se muestra de manera diferente. Mientras que el administrador tiene permisos para crear llaves, el usuario solamente puede firmar documentos como se ve en la siguiente imagen.

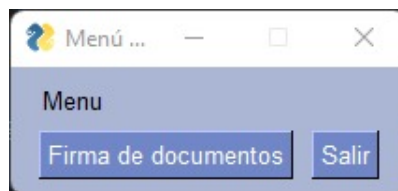


Figura 4: Implementación del código en el menú principal para Usuario

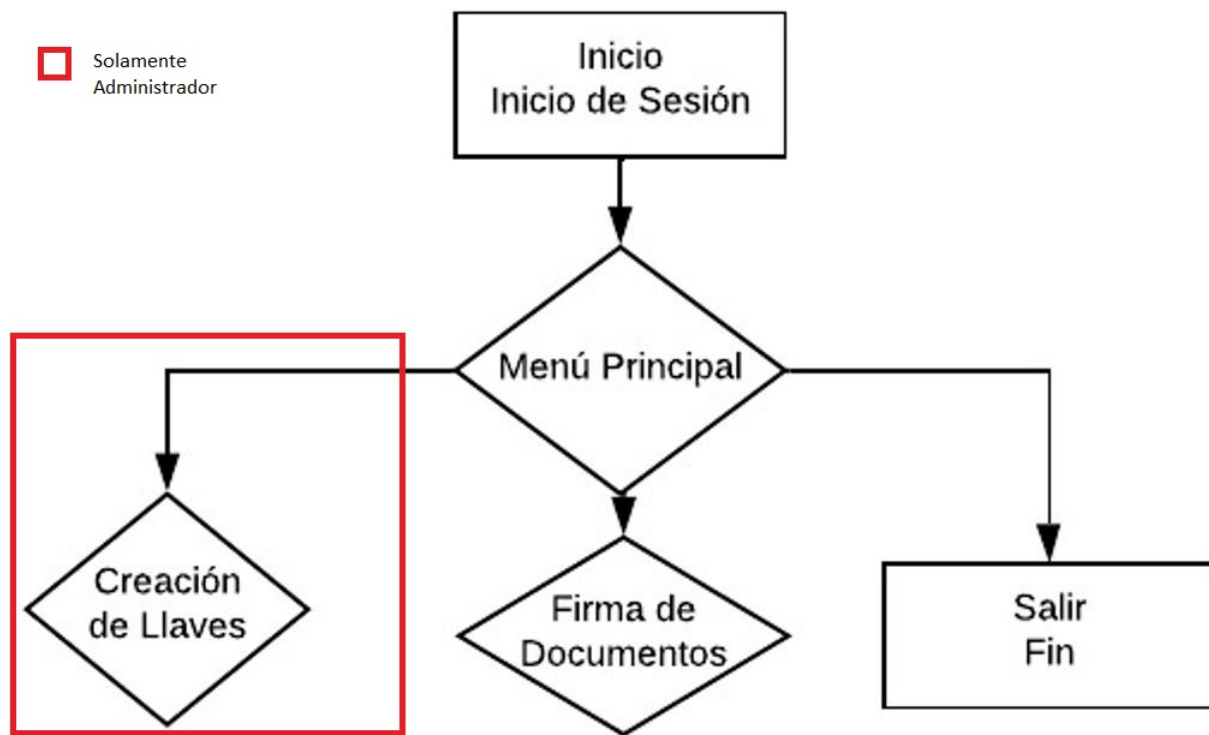


Figura 5: Implementación del código en el menú principal

Dentro de la opción de Creación de Llaves, el administrador podrá elegir si generar llaves o regresar al menú principal. De elegir generar llaves, se abrirá una ventana de progreso la cual arrojará si fue exitoso.

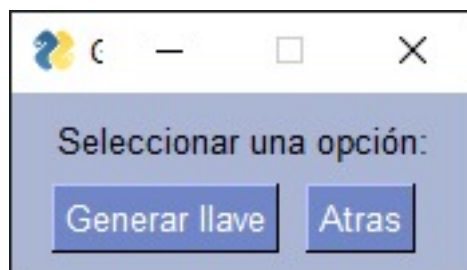


Figura 6: Ventana de opciones dentro de Creación de Llaves

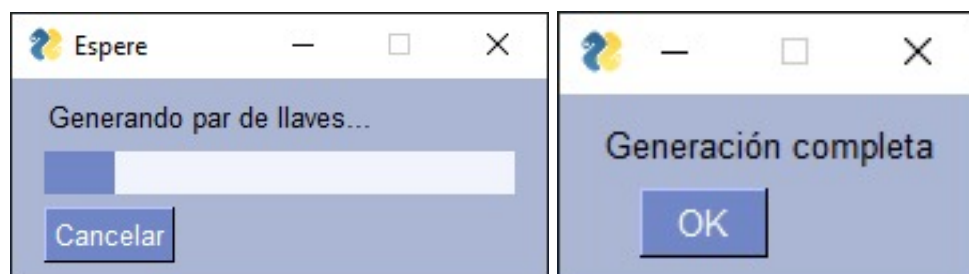


Figura 7: Ventana de progreso y generación completa

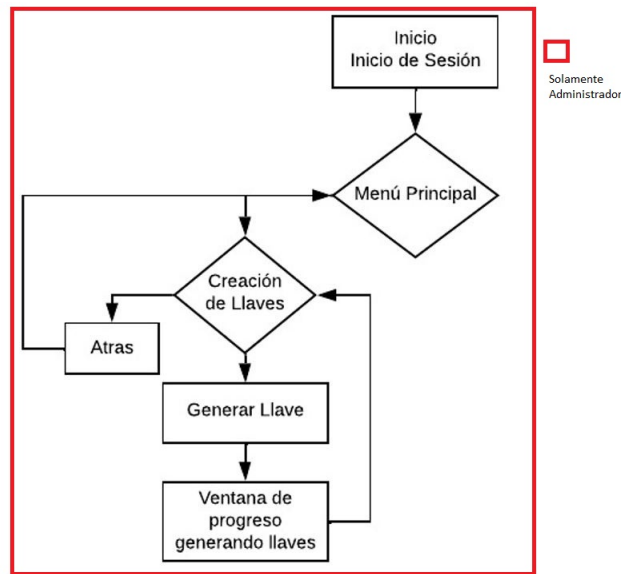


Figura 8: Implementación del código en la ventana de Creación de Llaves

Dentro de la opción de Firma de Documentos, el usuario podrá elegir si Firmar Documento, Verificar Firma o regresar al menú principal.

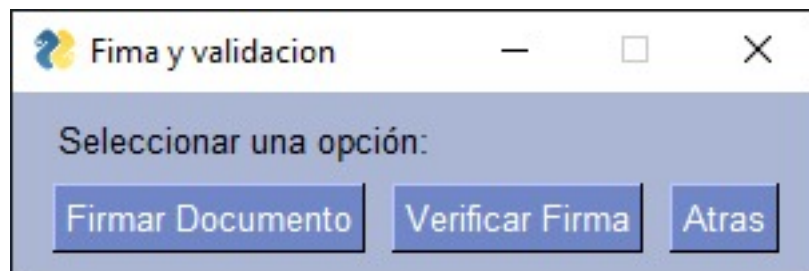


Figura 9: Ventana de generación completa

Al elegir Firmar Documento, el usuario deberá subir un documento, una llave privada y un certificado para posteriormente firmar, o bien, regresar al menú de Firma de Documentos.

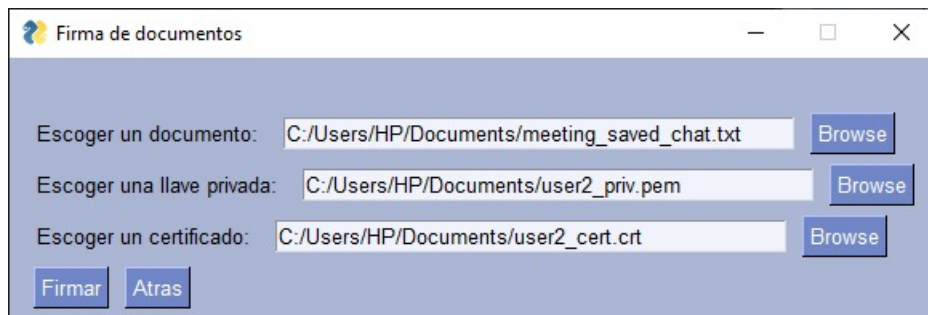


Figura 10: Ventana de Firmar Documento

Al firmar se abrirá una ventana de progreso la cual arrojará si fue exitoso y regresará al menú de Firma de Documentos.

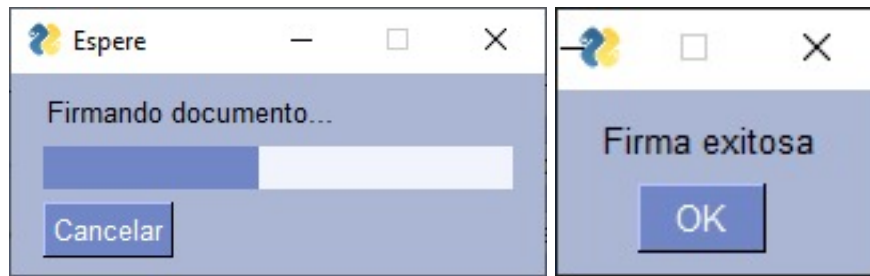


Figura 11: Ventana de progreso y firma exitosa

Al elegir Verificar Firma, el usuario deberá subir una firma, documento y llave pública, o bien, regresar al menú de Firma de Documentos.

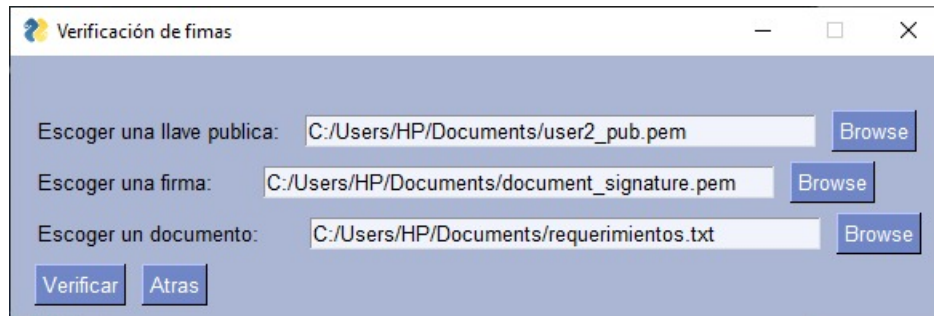


Figura 12: Ventana de verificación de firmas

Al verificar se abrirá una ventana de progreso la cual arrojará si fue exitoso y regresará al menú de Verificar Firma.



Figura 13: Ventana de progreso y estatus de verificación

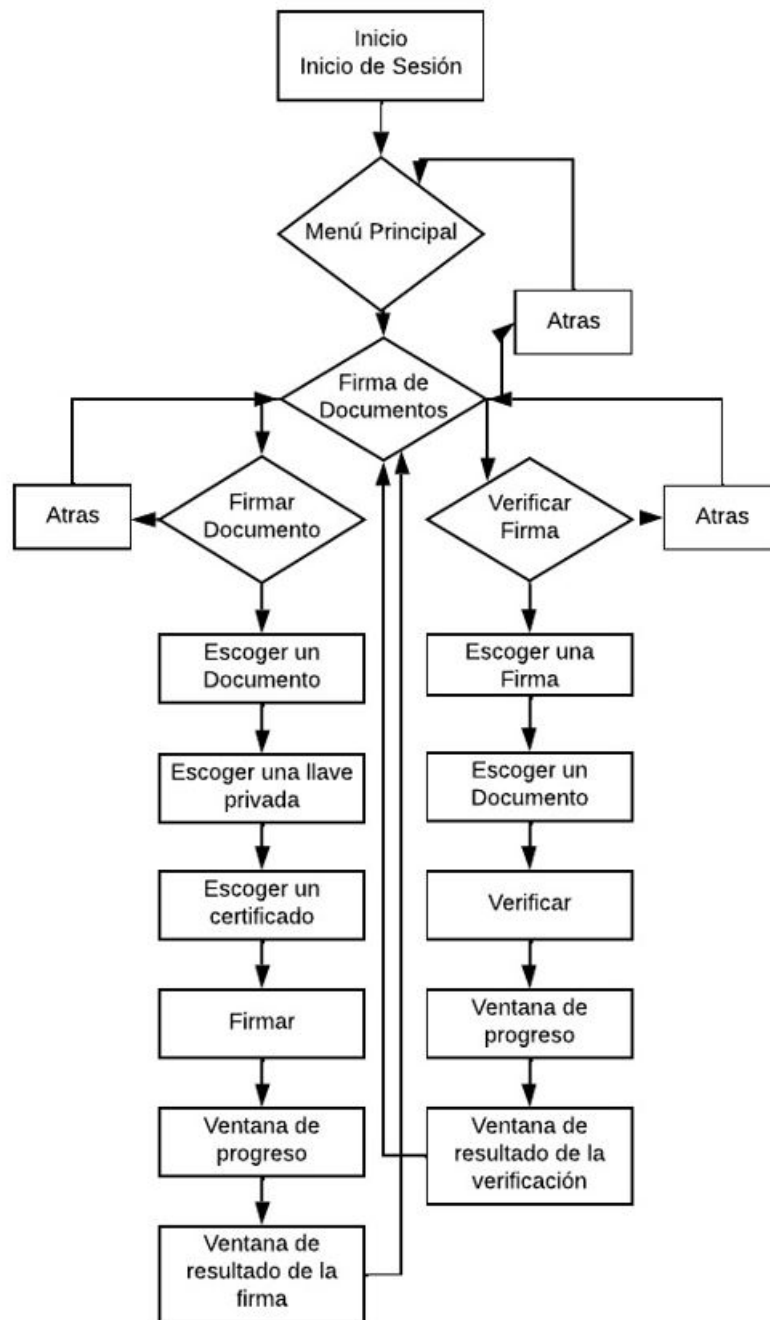


Figura 14: Implementación del código en la ventana de Firmar Documento

Finalmente, para salir del programa se deberá elegir la opción de Salir en el menú principal.



Figura 15: Ventana de Menú Principal

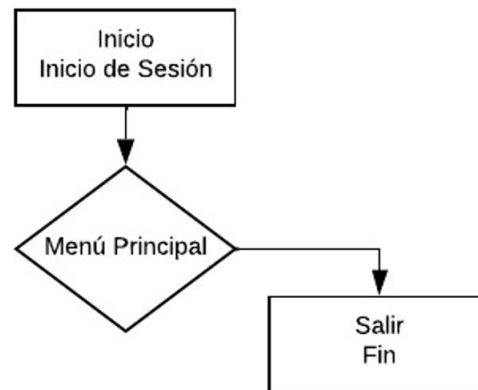


Figura 16: Implementación del código al salir

El diagrama completo se puede ver en la siguiente figura:

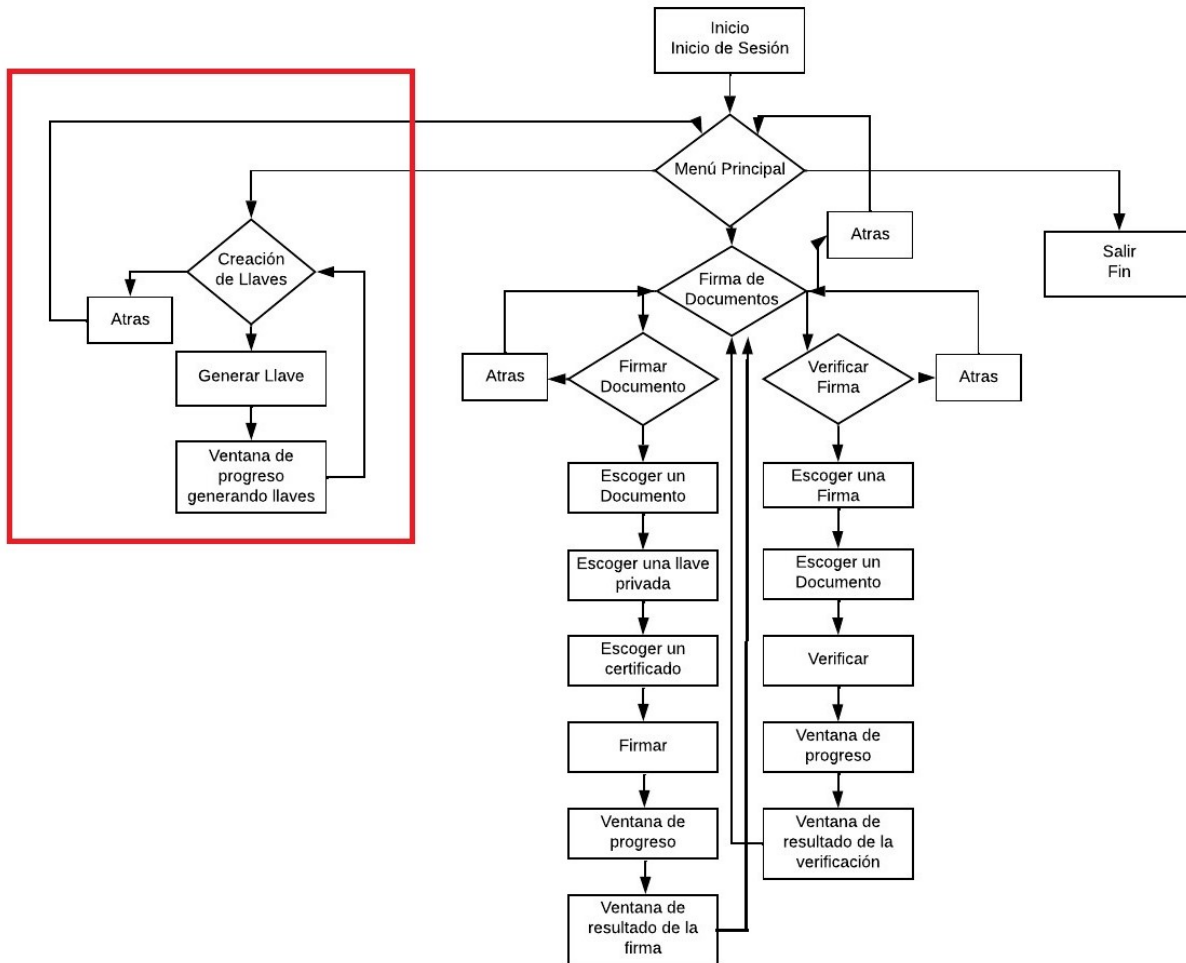


Figura 17: Implementación del código

## 5. Vectores de Prueba

Para la generación y verificación de firmas, se buscó utilizar los vectores ofrecidos por los estándares del NIST, sin embargo, uno de los principales problemas que se tuvo para utilizar los vectores de prueba que nuestros profesores nos habían brindado, es que estos tenían algunos números primos, mensajes ya hashados y llaves y firmas digitales en formatos que no eran los que se utilizaron en nuestra implementación. Por lo que usar estos vectores brindados nos daría problemas. Es por ello que se ha decidido que los vectores de prueba usados serían generados por nosotros. Para la generación de los vectores de prueba se tomaron en consideración las características necesarias planteadas por el NIST en el artículo The 186-4 RSA Validation System (RSA2VS). Estos vectores los hemos generado utilizando las librerías y módulos que se han mencionado anteriormente, usando PyCryptodome.

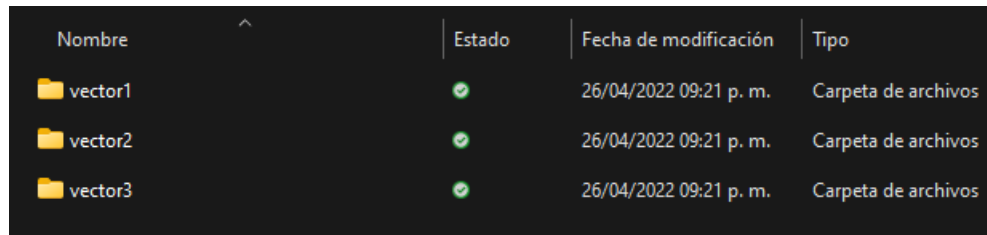
Por el momento se ha decidido que se usarán tres vectores de prueba. Estos vectores consisten en tres

llaves públicas, tres firmas digitales y tres documentos distintos. También se tienen tres llaves privadas, aunque estas no son requeridas para la verificación de la firma digital planteada. A continuación se muestra una captura con todos estos vectores de prueba.

Tras haberlos generado, se utiliza la aplicación de la implementación criptográfica y se utiliza la función para la validación de las firmas digitales. Si se utilizan la llave pública, el documento y su firma digital las cuales pertenezcan al mismo vector de prueba, la aplicación le dice al usuario que la firma es legítima, en caso contrario, se le dice al usuario que la firma no es legítima.

En cada una de las carpetas que se muestran en la anterior figura (Figura 18), se tienen 4 archivos, los cuales son un conjunto de vectores de prueba. En cada conjunto se tienen

- Un par de llaves
  - Llave privada
  - Llave pública
- El documento al que se le realizó la firma digital
- Firma generada



Nombre	Estado	Fecha de modificación	Tipo
vector1	✓	26/04/2022 09:21 p. m.	Carpeta de archivos
vector2	✓	26/04/2022 09:21 p. m.	Carpeta de archivos
vector3	✓	26/04/2022 09:21 p. m.	Carpeta de archivos

Figura 18: Carpeta con los 3 conjuntos de vectores de prueba

Al momento de realizar la prueba de verificación con los vectores de prueba, si se usan los que pertenecen en la misma carpeta y pertenecen a los mismos procesos, la firma digital será válida, en cualquier otro caso la firma digital será no válida.

## 6. Licencias utilizadas

Así como fue mencionado, el algoritmo fue implementado en el lenguaje de programación **Python**, por ende antes de ser utilizado tiene que considerarse y adaptarse a los términos de licencias que conlleva utilizar el lenguaje de programación.

La licencia de Python es llamada "Python Software Foundation License" (PSFL) es una licencia de código o software abierto que significa que es libre. La PSFL tiene de ventajas que es libre de regalías, es decir no



cobra por utilizarse, no es exclusiva es para todas las personas, y es una licencia mundial para reproducirse para que los usuarios tengan acceso a utilizarla.

La misma licencia no tiene representaciones ni garantías. Al aceptar los términos se deberá tener en consideración que la licencia no da permiso a las marcas registradas de hacer promoción de su marca o producto. Se impedirá crear cualquier tipo de relación o acuerdo entre la PFSL y el que usa la licencia ya sea individuo o empresa. Sin embargo, la licencia terminará en cualquier caso de rescindir el incumplimiento del material.

Además, es necesario que para los repositorios públicos se utilice, para poder compartir el software entre la comunidad, se pueda editar y modificar libremente. Solo se puede seleccionar una licencia cuando se crea un proyecto de GitHub y existen bastantes tipos de licencia por lo que se tiene que buscar la adecuada. Las licencias tienen como objetivo proveer un punto de partida para tener una elección informada del motivo que se busca al compartir el código.

Se escogió la licencia para el algoritmo de generación y verificación de firmas y firma digital de **MIT**, ya que ésta es permisible y sencilla hacia la comunidad y no tiene restricciones, salvo una condición: la licencia y el copyright. Se tiene de permisos lo siguientes:

- Uso comercial
- Modificación
- Distribución
- Uso privado

**MIT** no tiene garantía de algún tipo expresada o implicada y el copyright tiene que estar incluido en todas las copias que hagamos del proyecto ya sean pequeñas modificaciones o inclusive algoritmos.

## 7. Preguntas frecuentes

1. **¿Cuáles son los requerimientos del equipo de cómputo para el correcto funcionamiento del sistema?**

Se recomienda ampliamente contar con un dispositivo que cuente con al menos 8 GB de Memoria RAM. Así como un procesador **Intel(R) Core(TM) i5-9300H**, **AMD Rizen 5 2500 U**. Sin embargo, trabajar con especificaciones inferiores a las mencionadas no obstaculizará de manera drástica el desempeño del programa.

2. **¿Hay algún problema si no cuento con la versión mas reciente de cualquiera de los compiladores o sistemas utilizados?**

El algoritmo fue desarrollado utilizando las versiones más recientes del lenguaje de programación **Python (3.9.7)** y **Visual Studio Code (1.66.2)**. Debido a que se trabaja con las librerías y pa-

queterías mas actualizadas, el no contar con la versión mas reciente podría ocasionar dificultades al momento de ejecutar el programa. Se recomienda ampliamente contar con la versión mas reciente.

3. ¿Se deben descargar e instalar las librerías para el correcto funcionamiento del algoritmo?

Así es. Es importante tener instaladas todas y cada una de las librerías mencionadas previamente en este documento.

4. Se tiene una duda específica del funcionamiento del programa, ¿A quién se debe dirigir para la resolución de esta?

En la sección inferior **Contacto** se adjuntan los correos electrónicos de los alumnos que conforman el equipo encargado de realizar este Manual del Programador. En caso de contar con una duda puntual, se invita a enviar un correo.

## 8. Contacto

Contáctenos para preguntas, errores o comentarios. El equipo de trabajo encargado de elaborar este Manual del Programador estará dispuesto a responder sus preguntas. Asimismo, se agradece la retroalimentación y comentarios que nos permitan mejorar.

**Mauricio Hernández López:** A01173961@tec.mx

**Nemesio Navarro Arámbula:** A01284430@tec.mx

**Bárbara García Quiroga:** A01720402@tec.mx

**Freddy Sebastián Silva Culcay:** A00828792@tec.mx

**Fernando Guevara Moreno:** A00828723@tec.mx

**Roel Adrián De la Rosa Castillo:** A01197595@tec.mx

**Ricardo Díaz Garza Cabello:** A00826848@tec.mx