INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

Escuela de Ingeniería en Ciencias Aplicadas

Ingeniería en Ciencia de Datos y Matemáticas

Documentación de esquema de firma digital de Schnorr para administración de compras.

MA2006B.400 -USO DE ÁLGEBRAS MODERNAS PARA SEGURIDAD Y CRIPTOGRAFÍA

Integrantes Equipo:	*Matrícula:*
María del Carmen Vargas	A00828570
Renata Uribe Sánchez	A01274629
Jaime Guzman Parada	A00827834
Gerardo del Valle Cuéllar	A01284200
Álvaro David López Chávez	A01570493
Sebastián Alberto Neri Pérez	A01750190
Jesus A. Marroquin Escobedo	A00827670

Docentes:

Daniel Otero Fadúl | Alberto F. Martínez

Monterrey, Nuevo León. 2 22 de Abril del 2022

Contents

List of Code Listings

1 Introducción

En el siguiente documento son descritos los aspectos técnicos de la implementación del esquema de firma digital multiusuario a partir del algoritmo Schnorr en documentos para generar un sistema de administración de compras.

Sin perder de vista la dimensión ética que se involucra principalmente en la seguridad informática que protege las transacciones digitales de datos sensibles, se debe enfatizar en la validación de integridad y autenticidad. Para garantizar su efectividad, y además confiando en la precisión de las matemáticas, se pueden implementar algoritmos criptográficos basados en el uso de Álgebras Modernas para generar y operar firmas digitales resistentes a potenciales intentos de fraude.

Con esto en mente, se puede impulsar la contribución en torno al Objetivo de Desarrollo Sostenible 9, Industria, innovación e infraestructuras a través de la tecnología para desarrollar soluciones al servicio de los demás, fortaleciendo el crecimiento de la seguridad informática enfocado en protocolos criptográficos de clave pública.

2 Objetivo

Indicar e instruir la implementación adecuada del esquema de firma digital Schnorr que permite a la organización social formadora introducir un sistema de administración de compras a través de la firma de documentos.

3 Requerimientos del sistema

Un equipo de cómputo, preferentemente que cuente con los siguientes softwares:

- OS: Windows 11/10
- VS Code
- Python 3
- GitHub
- PySimpleGUI

Y como requerimientos mínimos de Hardware:

- Equipos de cómputo con al menos Windows 10 1903 (may 2019)

4 Licencia

La licencia que respalda la implementación del esquema de firma digital Schnorr, descrito en el siguiente documento, fue la siguiente:

GNU GENERAL PUBLIC LICENSE
Version 3, 29 June 2007

GNU GENERAL PUBLIC LICENSE es una licencia gratuita con copyleft para software y otros tipos de obras sujeta a lo siguiente:

Copyright (C) 2007 Free Software Foundation, Inc. (página) Cualquier individuo tiene permitido copiar y distribuir copias textuales de este documento, sin embargo no se permiten modificaciones.

5 Instalación, compatibilidad y dependencias

5.1 Instalación de Python

Dirigirse a la página web de descarga y descargar el instalador de Python, según sea el caso para el sistema operativo (OS X, Linux, Ubuntu, Windows).

Descargar el archivo del instalador y dar clic para su ejecución

La instalación siempre puede ser verificada al ingresar a la terminal y ejecutar el siguiente comando:

$ python3 –version
>> Python 3.6.1

La terminal debe desplegar la versión de Python como se muestra.

5.2 Instalación de VS Code

A pesar de que el código puede ser ejecutado en otros ambientes virtuales, se recomienda la utilización del editor de código fuente VS Code, que permite el control integrado de GitHub, el resaltado de sintaxis según el lenguaje, la implementación de shortcuts y algunas otras funciones.

Para la descarga de este ambiente de programación, favor de seguir los siguientes pasos:

1. Ingresar a la página de Microsoft Visual Studio Code y descargar el archivo de instalación de la plataforma del equipo en el que se desea instalarlo.
2. Seleccionar en **Descargar Visual Studio Code.**
3. Una vez que el archivo de instalación haya sido descargado, dirigirse a la carpeta donde yazca y abrirlo.
4. Configurar la instalación como sea deseado, siguiendo las instrucciones del instalador.
   (a) En caso de utilizar MacOS, basta con abrir el archivo descargado y el usuario estará en la aplicación directamente. (Se recomienda que el archivo descargado se mueva a la carpeta “Aplicaciones”.)
5. Finalmente, seleccionar **Terminar** para completar la instalación.

5.3 Configuración de Python3 en VS Code

Una vez instalado VS Code, se debe asegurar de que la aplicación tenga instalada las extensiones necesarias para ejecutar archivos escritos en Python (con extensión .py). Para ello, siga las siguientes instrucciones:

1. Diríjase al panel izquierdo de la aplicación, donde se encuentran 5 iconos (incluida una lupa, como referencia).
2. Seleccione el quinto icono que corresponde a la sección **"Extensiones"**.
3. En ella encontrará un listado de las extensiones instaladas ya en la aplicación.
4. En la barra de búsqueda, escriba *"Python"*, lo que dará como resultado varias extensiones con el nombre de la búsqueda.
5. Seleccione el primer resultado, llamado *"Python"* y elaborado por Microsoft.
6. En esta extensión, seleccionar **"Instalar"** (botón azul).
   (a) En caso de que la opción no se encuentre en pantalla y en su lugar se encuentran las opciones "Deshabilitar" y "Desinstalar", la extensión ya está instalada y no es necesario llevar a cabo las instrucciones siguientes.
7. Esperar a que la extensión se descargue y aparezca la notificación de que el paquete ha sido instalado correctamente.

5.4 Instalación de PySimpleGUI

Si se desea correr la versión del código que incluye la interfaz gráfica de usuario, por medio de la cual podrán ser firmados los documentos y posteriormente verificados, es necesario descargar esta librería. Para descargar la última versión es necesario abrir la terminal en el ordenador e introducir el siguiente comando:

```
$ pip install PySimpleGUI
```

5.5 Descarga de Repositorio GitHub

El código de firma digital que es explicado en el presente documento se encuentra en el repositorio de GitHub

1. Abra la terminal para introducir los comandos que lo llevarán a clonar el repositorio:

```
$ git clone https://github.com/WizenPainter/reto-ma2006.git
```

6 Características principales: esquema de firma digital

Buscando la certificación de contratos, además de una distribución y transacción segura de claves públicas, los esquemas de firma digital, se basan sustancialmente en tres algoritmos polinomiales probabilísticos: el generador de claves, el de firma digital y finalmente, el algoritmo de verificación. Es esencial considerar los tres algoritmos para cualquier tipo de esquema a implementar.

Funciones de Interés

En el proceso de programar la implementación del esquema de firma digital, se tuvo que realizar la inclusión de distintas funciones que proveen procesos necesarios para, entre todos, completar al esquema en sí y su funcionamiento correcto. Ya que son muchas funciones implementadas que hicieron esto posible, la más esencial de mencionar es:

- Sección 6.0.1. Función hash

Ahora pasando a las **funciones principales del programa**, como se ha mencionado, abordan 3 fases primordiales enlistadas a continuación:

- Sección 6.1 Generación de claves (keypair)
- Sección 6.2 Generación de firmas

  Es importante resaltar que dentro de esta ventana de la interfaz correspondiente a la generación de firmas, se encuentra su organización en 4 botones los cuales direccionan a las 4 funciones principales de la generación de la firma MuSig, siendo:

  - Subsección 6.2.1. Generación de claves iniciales
  - Subsección 6.2.2. Juntar Documentos
  - Subsección 6.2.3. Generar mi propia Firma MuSig
  - Subsección 6.2.4. Juntar todas las Firmas MuSig
- Sección 6.3 Verificación

Por otro lado y para otorgar la oportunidad de mayor comprensión del funcionamiento del código, también la sección presente enlista las funciones implementadas junto con una descripción de sus procedimientos, importancia del rol y resultados de éstas.

6.0.1 Función hash

La función que obtiene el hash del documento es la siguiente:

```
def hashPDF(file, BLOCK_SIZE):
    # hash=sha256()
    with open(file, 'rb') as f: # Open the file to read it's bytes
        fb = f.read(BLOCK_SIZE) # Read from the file.
        # Take in the amount declared above
        M = sl.sha256(fb)
    return M
file = "C:/Users/guzma/OneDrive/Documents/TEC/S6/MA2006/reto/test.pdf"
size = os.path.getsize(file)
```

Listing 1: **Función hash**

Al abrir el archivo se leen sus bytes y sobre este se aplica el hash criptográfico SHA256. Nota: Solo se puede utilizar una función hash en strings o arreglos en formato de bytes. Referencia: Pag 35

Parámetros:

- *file*: Archivo PDF. La restricción del archivo debe ser de formato PDF.
- *BLOCK_SIZE*: Tamaño de bloque del documento en bytes. Este se calcula con la función **getsize** del módulo **path** de la librería nativa de python **os**.

El parámetro de salida es simplemente el hash del documento guardado en una variable llamada *M*, haciendo referencia al mensaje.

6.1 Generación de claves

Como primer algoritmo del esquema, la generación de claves parte de la emisión de una clave pública y otra de carácter privado a través de las curvas elípticas, considerando los siguientes parámetros:

- Una curva elíptica E y su respectivo orden N
- Una función hash h
- Un generador G
- La clave secreta d, seleccionada por el usuario
- A través del algoritmo, se procesa el punto P que resulta de efectuar el generador G y la clave secreta d. Es decir, $P = dG = (P_x, P_y)$, en donde la coordenada P_x corresponde a la clave pública.

De esta manera, se generan las claves a través del generador G y la clave secreta d.
A continuación se muestra el código de la función en el que primeramente es necesario importar el documento **schnorr_lib.py**, contenido en el repositorio dentro de la carpeta **schnorr-sig**.

La función **create_keypair** se encuentra implementada dentro del mismo archivo **create_keypair.py** y se muestra a continuación:

```
def create_keypair(n_keys: int):
    # Create json
    users = {
        "$schema": "./users_schema.json",
        "users": []
    }

    # Generate n keys
    for i in range(0, n_keys):
        privkey = os.urandom(32)
        privkey_int = int(privkey.hex(), 16) % n

        publickey = pubkey_point_gen_from_int(privkey_int)
    >>>>>>>>>>>W
        # Check if the point P has the y-coordinate even; negate the private key
        # otherwise
        privkey_even = privkey_int if has_even_y(publickey) else n - privkey_int

        hex_privkey = hex(privkey_even).replace('0x', '').rjust(64, '0')
        users["users"].append({
            "privateKey": hex_privkey,
            "publicKey": bytes_from_point(publickey).hex()
        })
    json_object = json.dumps(users, indent=4)
    with open("users.json", "w") as f:
        f.write(json_object)
    return users
```

Listing 2: **Función de la generación de firmas Schnorr**

Como se observa, el principio de la función **create_keypair** crea un archivo JSON en el que serán almacenados los usuarios participantes en el esquema. Posteriormente, según el parámetro **n_keys**, se inicia un ciclo hasta llegar a este número y se genera una clave privada de acuerdo a una transformación hexadecimal. Como parte de la generación de claves públicas, existen diversas funciones según la transformación en la que se encuentra la clave privada: hexadecimal (**pubkey_point_gen_from_hex**) o entero **(pubkey_point_gen_from_int)**. En este caso, el ejemplo corresponde a la segunda. Posteriormente se genera una nueva variable denominada **privkey_even**, en la que se revisa si en el punto P, la clave privada (P_y) corresponde a un número par y en caso de lo contrario negar la clave privada. Por último, las claves públicas y privadas son añadidas y almacenadas en el archivo según el usuario al que corresponda.

6.2 Generación de firmas

Esquema MuSig

Cabe mencionar que la solución de este proyecto utiliza el esquema de firma múltiple denominado como MuSig-1, el cual se encuentra completamente basado y compuesto en las firmas Schnorr [1], esto quiere decir que este esquema solo funciona con este algoritmo criptográfico. Esta implementación añade las claves públicas y las firmas de cada uno de los partícipes, y a partir de esto genera una sola clave de tipo pública que verifica a todas en conjunto acelerando el proceso y optimizando el almacenamiento.

Así mismo, en la implementación de la generación de firmas se definen los parámetros importantes de los cuales aplican las siguientes condiciones y ecuaciones matemáticas propias de la metodología del esquema Mu-Sig del algoritmo criptográfico Schnorr:

$L = h(P_1 \parallel ... \parallel P_n)$, donde n es el número de usuarios P_i es la llave pública del i-esimo usuario

Cada usuario i calcula la cantidad $a_i = h(L \parallel P_i)$

La llave agregada $\tilde{X} = \sum_{i=1}^{n} a_i P_i$ es un parámetro público.

Cada usuario elige un r_i y calcula $R_i = k_i G$

Cada usuario calcula el punto $R = R_1 + ... + R_n = (R_x, R_y)$

Cada usuario calcula $c = h(R \parallel \tilde{X} \parallel M)$

Cada usuario calcula $s_i = r_i + cd_i a_i \ mod \ N$, y entonces la llave agregada es $s = s_1 + ... + s_n \ mod \ N$

La firma es el par (R_x, s)

Antes de comenzar con las descripciones de las siguientes funciones es de importancia aclarar que cuando se muestre el código fuente y aparezca a continuación al inicio:

- **elif event == 'Ejemplo'**, corresponde al desarrollo de la implementación directamente desde el documento donde se construyó la interfaz de nombre *inerfaz_final.py*.
- **def ejemplo(arg1, arg2)**, se trata de una función definida fuera de la interfaz, localizado en el archivo *schnorr_lib.py* del repositorio.

Se puede continuar con la definición de las funciones seguidamente.

6.2.1 *Generación de claves iniciales para el usuario*

Esta subsección corresponde al primer botón (a su vez, al primer paso) del esquema MuSig dentro de la interfaz orientada al usuario. A continuación se muestra el código correspondiente:

```
elif event == "Crear Claves Esquema":
        # sig_bytes = bytes.fromhex(values["-Firma-"])
        size = os.path.getsize(values["-Archivo7-"])
        M = hashPDF(values["-Archivo7-"], size)

        with open('users.json', 'r') as f:
            data = json.load(f)

        priv_key = data['users'][0]['privateKey']


        di = sl.int_from_hex(priv_key)

        Pi = sl.pubkey_point_gen_from_int(di)

        t = sl.xor_bytes(sl.bytes_from_int(di), sl.tagged_hash("BIP0340/aux", sl.get_aux_rand()))
        ki = sl.int_from_bytes(sl.tagged_hash("BIP0340/nonce", t + sl.bytes_from_point(Pi) + M)) % n
        if ki == 0:
            raise RuntimeError('Failure. This happens only with negligible probability.')

        # Ri = ki * G
        Ri = sl.point_mul(G, ki)

        assert Ri is not None

        #Pi = sl.pubkey_gen_from_hex(priv_key)

        sg.Popup("Clave Publica", Pi, "Ri", Ri, font='Helvetica 16 ', background_color='#362992')

        #cl = str(sl.int_from_bytes(Pi))

        Ri = str(Ri)
        data = {'Clave Publica': str(Pi),'Ri': Ri, 'Ki': ki}

        with open('claves_iniciales.json', 'w') as f:
            json.dump(data, f)
```

Listing 3: **Función de la *generación de claves iniciales***

Abriendo la información del usuario que está contenida en el archivo .JSON, se obtiene la clave privada del usuario, lo convierte a entero del formato hexadecimal y utiliza la función **(pubkey_point_gen_from_int)** y se calcula la llave pública del usuario para el esquema.

Nuevamente, se realizan los cálculos de las siguientes operaciones matemáticas, las cuales son pertenecientes del algoritmo Schnorr:

$$k_i$$
$$R_i = k_i G$$

Son importantes debido a que se utilizarán en las posteriores secciones, para generar la firma digital del esquema MuSig.

Finalmente, la clave pública, k_i y $R_i = k_i G$ se guardan en un archivo de tipo JSON.

6.2.2 *Juntar Documentos*

Esta segunda subsección corresponde al botón siguiente del que se explicó anteriormente. En esta ventana, se tiene como propósito dar al usuario una forma de concatenar los archivos JSON, es decir, aplica para "juntar" las claves iniciales y, por otro lado, las firmas individuales de cada usuario. A continuación se muestra el código correspondiente para el funcionamiento de la ventana resultante al dar clic al botón:

```
def merge_JsonFiles(filename):
    result = list()
    for f1 in filename:
        with open(f1, 'r') as infile:
            result.append(json.load(infile))
    with open('claves_firma.json', 'w') as output_file:
        json.dump(result, output_file)

    return True

 elif event == 'Agregar Documento':
        path_archivo = values["-Archivo3-"]
        lista_documentos.append(path_archivo)
        print(lista_documentos)
        sg.PopupScrolled("Los siguientes archivos son los que se van a juntar: \n",
        f"{lista_documentos}",
        font='Helvetica 16',
        background_color='#362992')


    elif event == 'Descargar Documento':
        sl.merge_JsonFiles(lista_documentos)
        sg.Popup('Se ha descargado el documento',
        font='Helvetica 16',
        background_color='#362992')

    elif event == 'Borrar Documentos':
        lista_documentos = []
```

Listing 4: **Función de *concatenación de documentos***

6.2.3 *Generar mi propia Firma MuSig*

Este tercer botón (recordando: corresponde al entorno de la interfaz de la generación de firma en general) pertenece al tercer paso del esquema MuSig a su vez. El desempeño operativo de la función definida **schnorr_musig_firmar()** se localiza en la librería **schnorr_lib.py** situada en la carpeta **schnorr_sig** del repositorio. El código es el siguiente:

```
def schnorr_musig_firmar(path_clavesp, M, n_usario):
    with open(path_clavesp) as json_file:
        data = json.load(json_file)
    L = b''
    for i in data:
        Pi = ast.literal_eval(i['Clave Publica'])
        byte_clave = bytes_from_point(Pi)
        L += byte_clave
    L = sha256(L)
    print(L)

    Rsum = None
    X = None

    lista_ai = []
    lista_ki = []
    for u in data:
        Pi = ast.literal_eval(u['Clave Publica'])
    #    byte_clave = bytes_from_int(int_clave)
        int_ri = ast.literal_eval(u['Ri'])
        ki = u['Ki']
        ai = int_from_bytes(sha256(L + bytes_from_point(Pi)))
        lista_ai.append(ai)
        X = point_add(X, point_mul(Pi, ai))
        Rsum = point_add(Rsum, int_ri)
        lista_ki.append(ki)
    if not has_even_y(X):
        cont = 0
        for i in lista_ai:
            lista_ai[cont] = n- i
            cont +=1
    if not has_even_y(Rsum):
        cont = 0
        for i in lista_ki:
            lista_ki[cont] = n- i
            cont +=1
    c = int_from_bytes(tagged_hash("BIP0340/challenge", (bytes_from_point(Rsum) + bytes_from_point(X) + M))) % n
    with open('users.json', 'r') as f:
        data_1 = json.load(f)
    priv_key = data_1['users'][0]['privateKey']
    priv_key = int_from_hex(priv_key)

   # ki = int(data[n_usario]['Ki'])

    s = (c * priv_key* lista_ai[n_usario]+ lista_ki[n_usario]) % n

    datos_firma = {'usario': n_usario, 'firma': s, 'firma agregada': X, "Sumatoria R": Rsum}
    with open('data.json', 'w') as f:
        json.dump(datos_firma, f)
    return s, Rsum
```

Listing 5: **Estructura completa de la *función MuSig-1***

En la implementación de la generación de firmas, además de realizar el proceso de la creación de la firma en sí junto con las condiciones matemáticas propias de la metodologia del algoritmo Schnorr, se puede ver que se exporta como resultado un archivo JSON con los datos de la firma generada. Este archivo JSON es de suma importancia para el funcionamiento de la implementación y funcionamiento del programa, dado que en ello se basa el manejo de la propiedad multiusuario del esquema Mu-Sig.

Algorithm 1 Pseudocodigo MuSign

Require: G ▷ Valores curva eliptica
Require: $n \geq 0$
Require: $N \leftarrow n$
Require: M
  for $i \leq N$ **do**
    $C_{pr} \leftarrow r_i$ ▷ r_i es un entero aleatorio
    $C_{pu} \leftarrow r_u$ ▷ r_u es un entero aleatorio basado en la clave privada
    Usuarios $\leftarrow C_{pr}, C_{pu}$
  end for
  for all Usuarios **do**
    $H \leftarrow hash(C_{pu}$
    $L \leftarrow L + H$
  end for
  for all Usuarios **do**
    $a_i \leftarrow hash(C_{pu})$
    $Usuario \leftarrow a_i$
  end for
  for all Usuarios **do**
    $C_{agg} \leftarrow C_{agg} + a_i * C_{pu}$
  end for
  $r_i \leftarrow r_{int}$
  for all Usuarios **do**
    $R_i \leftarrow r_i * G$
    $Usuarios \leftarrow R_i$
    $R \leftarrow R + R_i$
  end for
  for all Usuarios **do**
    $C \leftarrow hash(R) * hash(C_{agg} * hash(M)$
    $s \leftarrow r_i + c * C_{pu} * a_i \mod n$
    $Usuario \leftarrow s$
  end for
  for all Usuarios **do**
    $S \leftarrow S + s \mod n$
  end for
    return (S, C_{agg})

6.2.4 *Juntar todas las Firmas MuSig*

En este último y cuarto paso que concluye la fase de la generación de firma conjunta, lo que se realiza en rasgos generales es combinar todas las firmas individuales de cada parte involucrada obtenidas y detalladas en la subsección 6.2.3. Primeramente, se abre y descarga el archivo JSON generado previamente, el cual el número de usuario, la firma individual, la firma agregada general de todos, y la sumatoria de los valores R_i que se define como R.

Se define la sumatoria de las firmas y se comienza un ciclo que haga toda la operación, para que al valor que quede se le aplique el módulo de la curva n. Y por último para obtener la firma completa se extrae el valor de la coordenada en x de la R, y se junta con el valor de la sumatoria previamente calculado para definir la firma como el punto (R_x, s). Y por último la firma se guarda en formato string de los bytes, y la clave agregada en un entero en un archivo igual de tipo JSON.

```
  elif event == 'Generar La Firma MuSig':

        path_claves = values['-Archivo5-']
        with open(path_claves, 'r') as f:
            data = json.load(f)

        Rsum = data[0]['Sumatoria R']

        s_suma = 0
        for i in data:
            s_i = i['firma']
            s_suma += s_i

        s_suma = s_suma % n

        signature_bytes = sl.bytes_from_point(Rsum) + sl.bytes_from_int(s_suma)


        X = sl.bytes_from_point(data[0]['firma agregada'])

        sg.Popup("Firma", signature_bytes.hex(), "Firma Agregada", X.hex(),
            font='Helvetica 16',background_color='#362992')

#        signature_bytes = sl.int_from_bytes(signature_bytes)

        X = sl.int_from_bytes(X)

        signature_bytes = str(signature_bytes)


        datos_firma = {'Firma': signature_bytes, 'Firma Agregada': X}

        with open('firma_digital.json', 'w') as f:
            json.dump(datos_firma, f)
```

Listing 6: **Estructura de la implementación *Juntar todas las Firmas MuSig***

6.3 Verificación de firmas

La verificación de firmas se encuentra codificada igualmente en *"schnorr_lib.py"*. En éste, se utilizan principalmente las funciones **schnorr_verify_musig()** y **hashPDF()**, así como funciones adicionales que se encuentran en el mismo archivo previamente mencionado.

El código usado para la implementación de este paso es el que se muestra a continuación.

```
# Verify Schnorr signature
def schnorr_verify_musig(M, archivo_verificacion) -> bool:
    if len(M) != 32:
        raise ValueError('The message must be a 32-byte array.')

    with open(archivo_verificacion) as json_file:
        data = json.load(json_file)

    clave_p = data['Firma']
    sig = ast.literal_eval(clave_p)

    clave_agregada = int(data['Firma Agregada'])

    pubkey = bytes_from_int(clave_agregada)


    if len(pubkey) != 32:
        raise ValueError('The public key must be a 32-byte array.')
    if len(sig) != 64:
        raise ValueError('The signature must be a 64-byte array.')
    P = lift_x_even_y(pubkey)
    r = get_int_R_from_sig(sig)
    s = get_int_s_from_sig(sig)
    if (P is None) or (r >= p) or (s >= n):
        return False
    e = int_from_bytes(tagged_hash("BIP0340/challenge", get_bytes_R_from_sig(sig)
        + pubkey + M)) % n
    R = point_add(point_mul(G, s), point_mul(P, n - e))
    if (R is None) or (not has_even_y(R)):
        # print("Please, recompute the sign. R is None or has even y")
        return False
    if x(R) != r:
        # print("There's something wrong")
        return False
    return True
```

Listing 7: **Estructura completa de la *verificación de firmas Schnorr***

De manera general, el proceso de verificación de firmas empieza por la función donde primeramente se encarga de asegurar que las longitudes de los valores del hash del archivo, la clave pública y la firma sean las correctas. Además, procesa el archivo PDF idealmente ya firmado por los usuarios legítimos y obtiene la clave pública de la "Firma Agregada" (Agregada: combinada) y extrae los bytes de estos datos para posteriormente realizar el proceso de operaciones modulares con la función **lift_x_even_y()** y las comparaciones necesarias para cerciorarse de las firmas y terminar su verificación. Si ésta fue exitosa, se regresa *True* para que se le afirme al usuario que la firma es válida. Si el caso es el contrario, causado por una incoherencia resultante de las comparaciones que se realizaron en la función, se le notifica al usuario de que las firmas no son válidas. Con ello, se da por terminada la fase de verificación de firmas.

Una manera sencilla de comprobar el funcionamiento de la verificación es:

1. Seleccionar el archivo PDF que se utilizó en todo el proceso para generar la Firma Musig.
2. Seleccionar la Firma MuSig obtenida.
3. Resultado esperado: *La firma es VÁLIDA para este mensaje y clave pública*.

Y por otro lado, para probar a la función si reconoce un archivo no firmado:

1. Seleccionar un archivo PDF distinto al que se utilizó en todo el proceso para generar la Firma Musig.
2. Seleccionar la Firma MuSig obtenida.
3. Resultado esperado: *La firma es **no** VÁLIDA para este mensaje y clave pública*.

7 Descripción de la API o las funciones de la biblioteca con ejemplos.

7.1 Acerca del despliegue de la interfaz

Para ejecutar finalmente la interfaz gráfica, lograr firmar archivos PDFs y además verificar su validez, es posible hacerlo de dos maneras: a través de la Terminal o directamente en Visual Studio Code.

7.1.1 Desde la terminal

Es necesario cambiar el directorio actual dentro de la terminal de la siguiente manera:

1. Cambiar el directorio actual para continuar con el despliegue de la interfaz con el algoritmo de firma digital en Figura 1:

   $ cd schnorr-sig

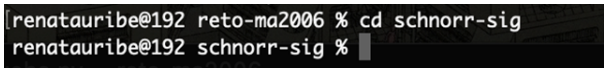

Figure 1: Función de la generación de firmas Schnorr

2. Dentro del directorio, se corre el archivo prueba.py con el siguiente comando:

   $ python3 interfaz_final

3. Inmediatamente se despliega la interfaz, como lo muestra la Figura 2

Figure 2: Interfaz orientada al usuario

7.1.2 Desde VS Code

Para desplegar la interfaz gráfica desde VS Code:

1. Primeramente se abre el repositorio completo en la aplicación VS Code. Esto es posible al arrastrar la carpeta del reto dentro de la ventana de la aplicación.
2. Buscar el archivo **interfaz_final**, incluido dentro de la carpeta **schnorr-sig**
3. Da clic en el botón de correr código, ubicado en la esquina superior derecha como se indica en la Figura 3.

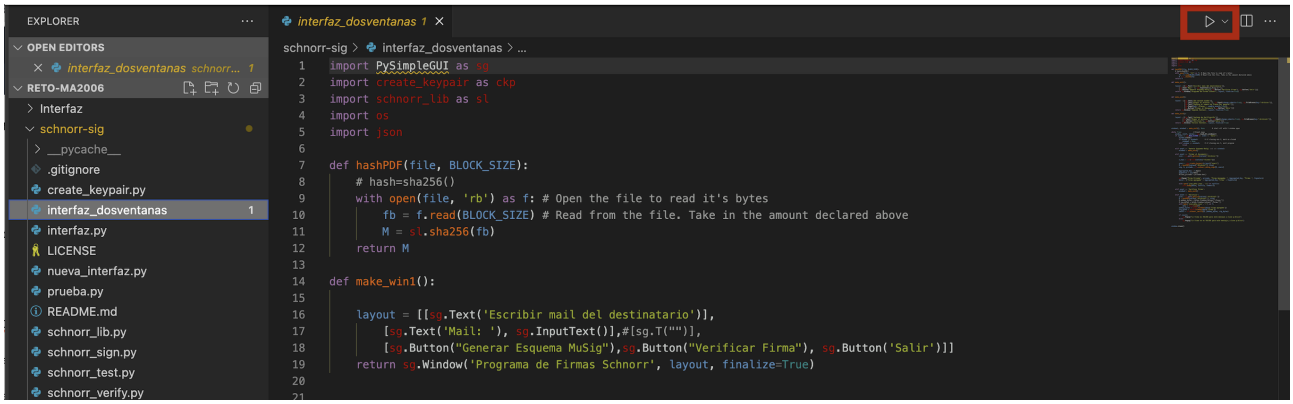

Figure 3: Entorno de VS Code

4. Al dar clic, se despliega automáticamente se despliega la misma ventana, Figura 4.

Figure 4: Interfaz gráfica desde VS Code

Dentro de la ventana principal, además de observar la leyenda con el título del proyecto y el logo de la fundación Teletón, se muestran tres diferentes botones: **Esquema MuSig, Verificar**

Virma y Cambiar mis claves. A continuación se describen los procesos y las ventanas correspondientes a cada botón:

7.1.3 Esquema MuSig

Seleccionar Esquema MuSig despliega una nueva ventana para realizar tanto la generación de claves iniciales como la firma del documento de modo que es posible efectuar cuatro diferentes acciones enlistadas a continuación, como se presenta en la Figura 5.

- Generar mis claves iniciales
- Juntar documentos
- Generar mi propia Firma de Esquema MuSig
- Juntar firmas de esquema MuSig

Figure 5: Ventana de opciones para generar el esquema multi firma

Una vez en la ventana de opciones para la generación del esquema MuSig, es posible comenzar el proceso de generación de claves y firma siguiendo los siguientes pasos:

1. Para generar las claves iniciales es necesario dar clic en **Generar mis Claves Iniciales** y así desplegar la ventana Figura 6.

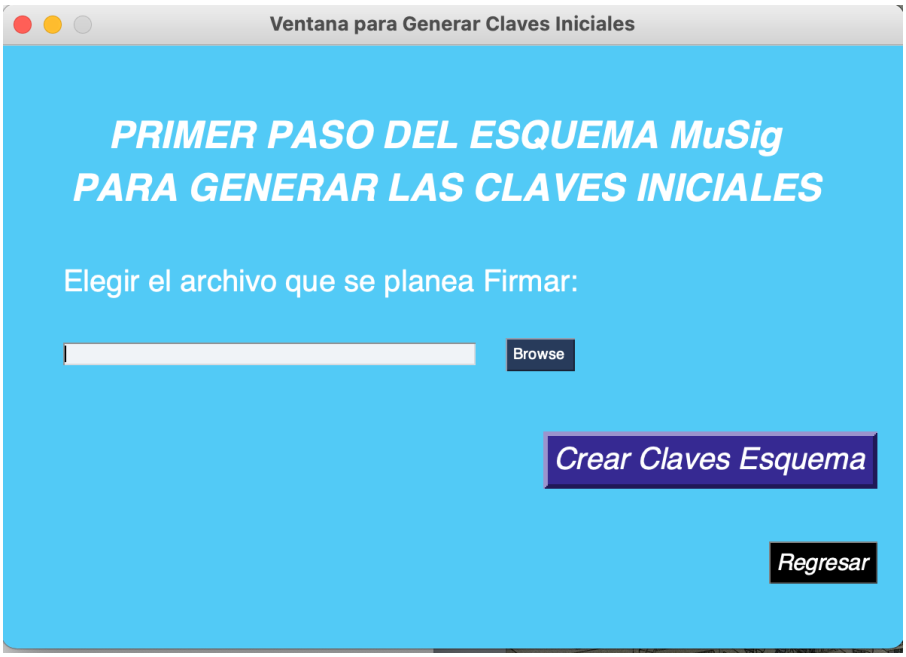

Figure 6: Ventana de generación de claves iniciales por usuario

2. Posteriormente se selecciona el archivo que se desea firmar y se oprime el botón inferior derecho de color azul **Crear Claves Esquema** (Para regresar al menú principal basta con dar clic en **Regresar**.

3. Para continuar con el proceso es necesario que cada usuario comparta con el usuario líder el archivo .JSON que contiene las respectivas claves iniciales después de ejecutar el paso 1.

4. Es obligación del usuario líder, juntar las claves iniciales de cada área al dar clic en **Juntar documentos (claves/firmas)**, desplegando la interfaz de la Figura 7.

Figure 7: Ventana de recolección de claves

5. Una vez el líder con los archivos .JSON de los usuarios involucrados, es posible seleccionar uno a uno, los archivos de claves iniciales que le refirieron. Para seleccionar el archivo se da clic en **Browse**, **Agregar Documento** (en caso de cometer algún error **Borrar documento**) para posteriormente descargar el archivo con la colección de claves por usuario en **Descargar documento**. Al finalizar, regresar al menú principal.

6. El usuario líder comparte el archivo de claves a los usuarios y a partir de ahora, como usuario, es posible ingresar a **Generar mi propia firma MuSig** para desplegar la Figura 8.

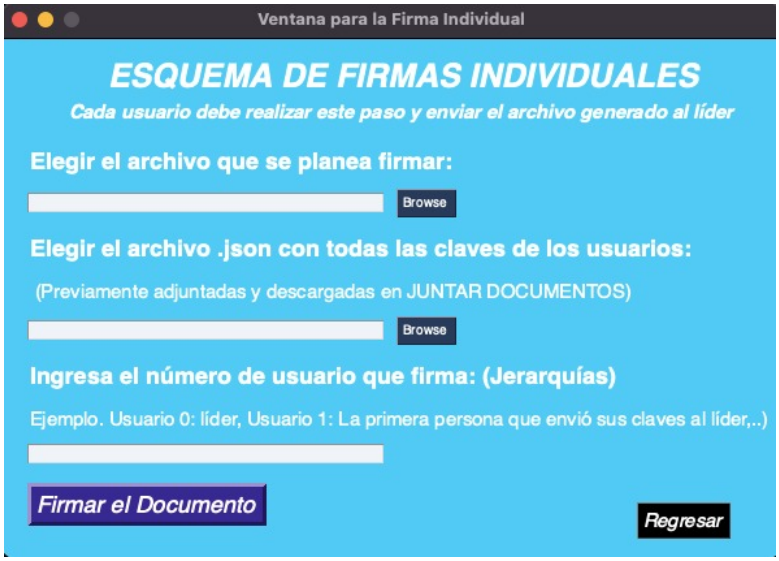

Figure 8: Ventana de esquema de firmas individuales

7. Para este paso se deben tener presentes el archivo que se desea firmar, el conjunto de archivo con las firmas y el número de usuario de quien ejecuta el proceso. De este modo, los dos primeros archivos son cargados en la interfaz y además se escribe el número de usuario. Finalmente se da clic en **Firmar el documento**, descargando inmediatamente un archivo para después **Regresar** al menú principal. (Es necesario que cada uno de los usuarios comparta nuevamente el archivo individual con su firma al administrador)

8. El administrador da clic nuevamente en **Juntar Documentos (claves/firmar** para coleccionar esta vez, no las claves, sino las firmas y así lograr posteriormente la creación de una firma única a partir de los diversos usuarios (Figura 9. Para realizar la recolección de firmas, realizar el **Paso 4**, ahora con los archivos de firma.

Figure 9: Ventana de recolección de firmas

9. Una vez en el menú principal y después de descargar el archivo que contiene la única firma, creada a partir de la colección de firmas de usuario, se da clic en **Juntar todas las firmas MuSig**, en donde únicamente es necesario cargar el archivo recién mencionado e inmediatamente dar clic en **Generar La Firma MuSig** para la generación de la firma oficial en Figura 10

Figure 10: Ventana de generación de firma conjunta MuSig

7.1.4 Verificar Firma

Para la verificación de firma, únicamente es necesario dar clic en **Verificar Firma**, como lo muestra la Figura 11

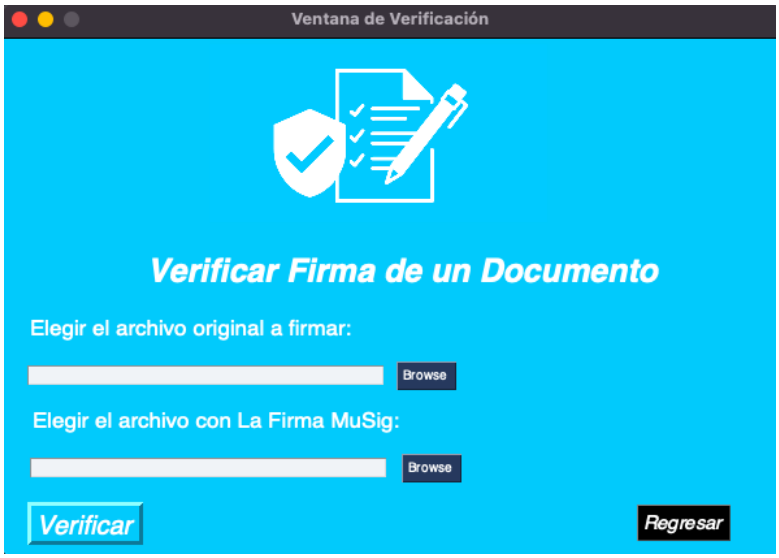

Figure 11: Ventana de generación de firma conjunta MuSig

Posteriormente se carga el PDF a verificar y el archivo de firma anidada, resultante del último paso. Se da clic en **Verificar**

En caso de resultar exitoso se despliega la Figura 12

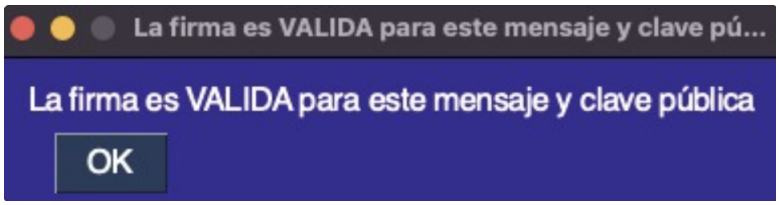

Figure 12: Ventana de verificación exitosa

De lo contrario el mensaje de la Figura 13 se mostrará:

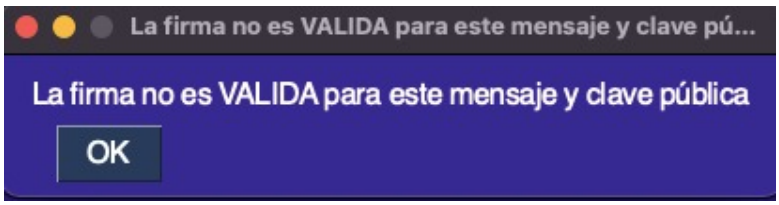

Figure 13: Ventana de verificación fallida

7.1.5 Cambiar mis claves

Si se da clic en **Cambiar mis claves** como se muestra en la Figura 14.

Figure 14: Ventana de cambio de claves

Es suficiente con posicionar el cursor en el botón del centro para trasmutarlas y de cambiarse exitosamente se muestra (Figura 15)

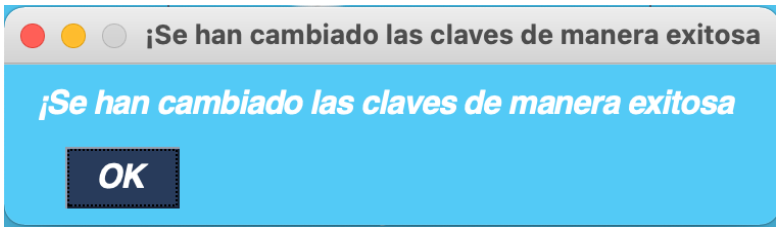

Figure 15: Ventana de generación de firma conjunta MuSig

El proceso recientemente descrito también se puede ejemplificar gráficamente a través del diagrama correspondiente a la Figura 16

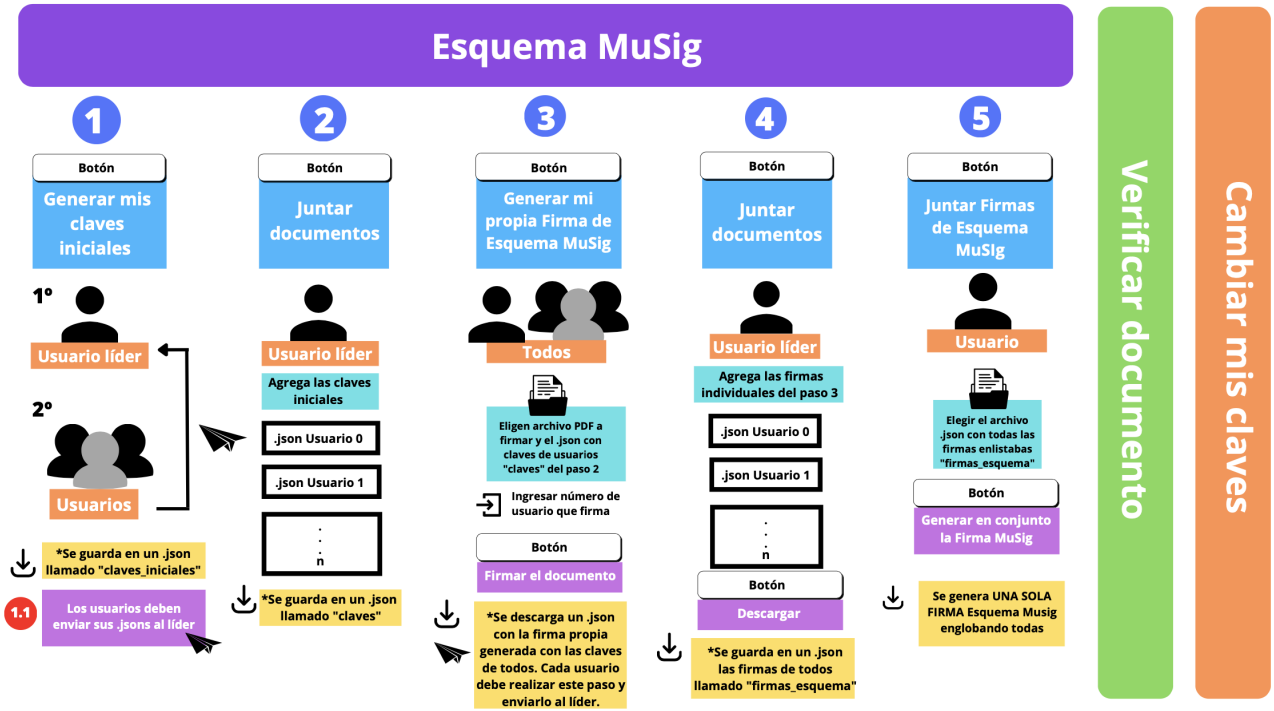

Figure 16: Diagrama con proceso de interfaz

8 Planes futuros de desarrollo

El futuro desarrollo del programa puede incluir diversas etapas e implementaciones para mejorar la facilidad de uso de la solución. De entre las posibles futuras implementaciones, una opción de interés puede ser la adición y vinculación del uso de infraestructuras de bases de datos en un servidor SQL para manejar y registrar de manera más visible las firmas realizadas, usuarios registrados y demás. Otras opciones serían apoyar la experiencia del usuario/cliente para un manejo más automatizado de la solución, así como trabajar en la generación de un certificado *.crt* para validar al propietario de la firma y el estado inalterable del documento firmado.

De la misma forma, es de nuestro interés, escalar las posibilidades de esta implementación de modo que la verificación de documentos pueda ser utilizada en el ámbito médico, es decir, validar las consultas a través de las recetas emitidas por los profesionales de la salud que asisten a la fundación.

9 Ayuda y FAQs

- **¿Se puede correr el programa en cualquier computadora?**
  El programa **SÍ** se puede correr desde cualquier computadora, siempre y cuando esta contenga los requisitos de instalación. Para conocer estos requisitos, véase la sección 3 Requerimientos del Sistema.

- **¿Se será capaz de consultar el código fuente de la solución?**
  SÍ será posible consultar el código fuente de la solución por medio del repositorio de

GitHub pues la licencia que lo respalda estipula que está permitida su copia y distribución siempre y cuando su contenido no se modifique.

- **¿Qué se recomienda hacer en caso de que el programa presente fallas?**
  Para evitar el surgimiento de diversas fallas en el programa se han llevado a cabo diversas pruebas en distintas condiciones por lo que la solución no debería de presentar inconvenientes. En caso de presentarlos,acceda al repositorio de GitHub y comuníquese con los desarrolladores.

  Álvaro López (mail) | Gerardo del Valle (mail) | Jaime Guzmán (mail) | Jesús Marroquín (mail) | Marycarmen Vargas López (mail) | Renata Uribe (mail)

- **No es posible correr el código pues indica que falta la instalación de una librería para su funcionamiento, ¿qué hacer en este caso?**
  La librería que en ocasiones presenta fallas es **PySimpleGUI** por lo que si su instalación a través de VS Code no es suficiente, consulte **PySimpleGUI** 5.4