Instituto Tecnológico y de Estudios Superiores de Monterrey

Escuela de Ingeniería y Ciencias

Ingeniería en Ciencia de Datos y Matemáticas

MA2006B: Uso de Álgebras Modernas Para Seguridad y Criptografía

Manual del Programador

Autores:

Antonio Patjane Ceballos - A01657978

Gerardo Barajas Sánchez - A00828760

Ian Timothy Henry Suárez - A01701578

Luis Gabriel Martínez Rentería - A01651812

Miguel Alejandro Salas Reyna - A00827219

Miguel Ángel Chávez Robles - A01620402

Pedro Alan González Arámbula - A01625308

Profesores:

Dr. Alberto Francisco Martínez Herrera

Dr. Daniel Otero Fadul

Socio Formador: Fundación Teletón

Monterrey, Nuevo León

6 de Marzo de 2022

Índice

Índice de figuras

1. Requerimientos mínimos técnicos del sistema.

El desarrollo de la solución se hizo en el lenguaje de programación python3, se usan distintos módulos base de python, y la única librería externa usada en el proyecto es **PySimpleGUI versión 4.59.0**. La máquina usada para desarrollar el sistema cuenta con el sistema operativo Windows 10 Home edition. La solución ha sido probada en los siguientes sistemas operativos:

- Windows 10 Home edition
- Ubuntu 20.04.4 LTS (requiere instalar python3-Tk además de las dependencias)

Los requisitos mínimos de software para el uso de esta implementación son los siguientes:

- Windows 7 or 10
- Mac OS X 10.5
- Python 3.7

Los requerimientos mínimos de hardware para el uso de esta implementación son los siguientes:

- 2GB de RAM
- 5GB de espacio libre en el disco

2. Licencia.

2.1. Licencia de la implementación.

El proyecto está bajo la licencia MIT.

2.2. Licencias de dependencias utilizadas.

- alembic / MIT License
- blinker / MIT License
- certifi / Mozilla Public License 2.0
- cffi / MIT License
- click / BSD License
- colorama / BSD License
- cryptography / Apache Software License, BSD License
- Flask / BSD License
- Flask-Login / MIT License
- Flask-Migrate / MIT License
- Flask-SQLALchemy / BSD License
- Flask-Uploads / MIT License
- Flask-WTF / BSD License
- greenlet / MIT License
- importlib-metadata / Apache Software License
- itsdangerous / BSD License
- Jinja2 / BSD License
- Mako / MIT License
- MarkupSafe / BSD License
- mysql / MIT License
- mysql-connector / GNU General Public License

- mysql-connector-python / GNU General Public License
- mysql-connector-python-rf / GNU General Public License
- mysqlclient / GNU General Public License
- protobuf / BSD-3-Clause
- pycparser / BSD License
- PyMySQL / MIT License
- sentry-sdk / BSD License
- SQLAlchemy / MIT License
- urllib3 / MIT License
- Werkzeug / BSD License
- WTForms / BSD License
- zipp / MIT License

3. Instalación, compatibilidad y dependencias.

Para el correcto funcionamiento de la solución es necesaria la instalación de diversas dependencias lo cual se realiza de la siguiente manera, desde la linea de comandos/shell dentro de la carpeta del código:

Instalación inicial para Windows:

```
pip install -r requirements.txt
```

Instalación inicial para Linux y MacOS:

```
python3 -r pip install requirements.txt
```

La versión de desarrollo más reciente se puede encontrar en GitHub en `https://github.com/mikemachr/firmaDigital` y se puede descargar de la siguiente manera:

1. Ir `https://github.com/mikemachr/firmaDigital`.
2. Buscar y seleccionar el botón verde **Code**, elegir la opción **Download ZIP**.

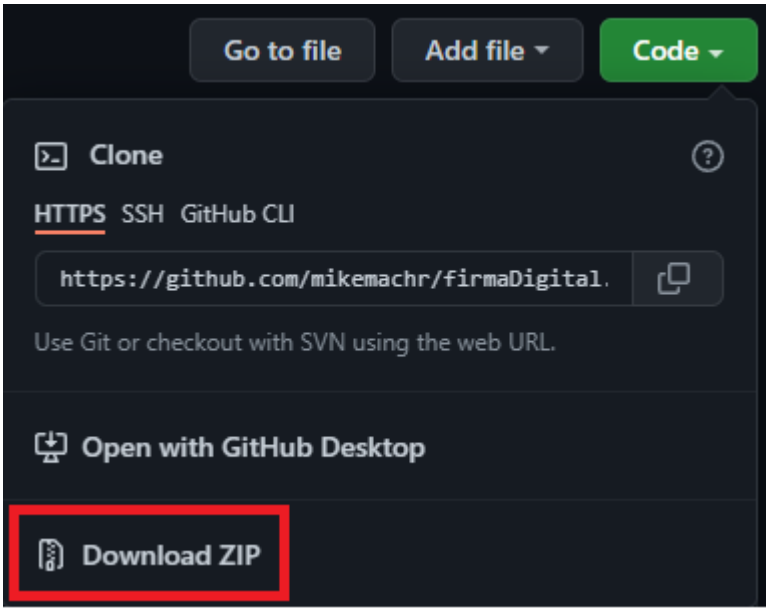

Figura 1: Descarga del ZIP del proyecto.

3. Extraer el archivo zip y abrir la carpeta del proyecto en el compilador de preferencia.

   Cabe destacar que este proyecto se encuentra en un repositorio privado por lo que para tener acceso a el se debe haber otorgado permiso previamente.

Los requerimientos necesarios para que la implementación funcione son:

- Python 3.7

4. Características principales.

4.1. Generación de claves.

Para generar la clave pública se usa el algoritmo Doubling-and-add, mostrado en [1], para computar kP donde P es el generador de la curva y k es la clave privada. El algoritmo se puede apreciar en el algoritmo 1.

Algoritmo 1 Doubling-and-Add

Solicitar k: clave privada, P: punto generador de la curva usada

1: Se convierte k a su representación binaria
2: $Q = P$
3: **for** $i = 1$ to len(k) **do**
4: $\quad Q = 2Q$
5: $\quad$ **if** $k_i = 1$ **then**
6: $\quad\quad Q = Q + P$
7: $\quad$ **end if**
8: **end for**
9: **return** Q

4.2. Generación de firmas.

La generación de firmas usa el algoritmo **EDCSA Digital Signature Generation**, mostrado en [1], el proceso que se sigue se describe en el algoritmo 2.

Algoritmo 2 ECDSA Digital Signature Generation

Solicitar k: clave privada, *curva*: curva usada para clave pública, *mensaje*: mensaje a firmar en bytes, $h(x)$: función hash

Inicializar $r, s = 0$
Obtener el hash del mensaje $e = h(mensaje)$ en su representación hexadecimal
if $len(e) > 64$ **then**
    Truncar e a las primeras 64 posiciones
end if
Convertir e a entero
while $r = 0$ **or** $s = 0$ **do**
    $k' =$**randbelow**$(curva.n)$
    Computar $Q = k' \cdot curva.P$ y obtener Q_x
    Computar $r = Q_x \ mod \ curva.n$
    $s = k'^{-1}(e + k \cdot r) \ mod \ curve.n$
end while
Obtener los datos del firmante asociado a la clave privada k
if La persona tiene vigencia para firmar **then**
    Escribir información del certificado en base de datos, dar vigencia de 3 días a la firma
    return r, s
else
    Terminar proceso
end if

4.3. Verificación de firmas.

La verificación de firmas esta basada en el algoritmo **EDCSA Digital Signature Verification** descrita en [1]. La rutina utilizada se describe en el algoritmo 3.

Algoritmo 3 ECDSA Digital Signature Verification

Solicitar *signature*: firma a verificar (tupla (r, s)), *curva*: curva usada para firmar, *publicKey*: curva que contiene la clave pública del firmante, *mensaje*: mensaje a verificar en bytes, $h(x)$: función hash

if $firma[0] = 0$ **or** $firma[0] > curva.n - 1$ **or** $firma[1] = 0$ **or** $firma[1] > curva.n - 1$ **then**
    Se rechaza la firma
    return False
end if
$w = firma[1]^{-1}$ *mod curve.n*
Obtener el hash del mensaje $u = h(mensaje)$ en su representación hexadecimal
if $len(u) > 64$ **then**
    Truncar u a las primeras 64 posiciones
end if
Convertir u a entero *modcurva.n*
Computar $u_1 = w \cdot u$ *mod curva.n*
Computar $u_2 = firma[0] \cdot w$ *mod curva.n*
Computar $Y = u_1 \cdot curva.P + u_2 \cdot publicKey.Q$ usando doubling and add
Computar $v = Y_x$ *mod curva.n*
Obtener el registro correspondiente a la firma de la base de datos
if El registro no existe en la base de datos **or** El registro tiene vigencia expirada **then**
    Rechazar firma
    return False
end if
if $signature[0] = v$ **then**
    return True
end if
return False

5. Descripción de la API o las funciones de la biblioteca.

La implementación está dividida por 6 módulos, un programa principal y un módulo de tests.

5.1. ECDSA_keyGenerator.py

Este módulo define toda la lógica fundamental para la realización de operaciones con curvas.

La manera de utilizar las funciones de dicho modulo es la siguiente:

```
1 from ECDSA_keyGenerator import *
```

Figura 2: Forma de importar todo el modulo ECDSA_keyGenerator.

5.1.1. Clases del módulo ECDSA_keyGenerator.

El modulo se compone de 3 clases las cuales son mencionadas a continuación.

1. **class Error(Exception)**

   Usada para manejo de excepciones personalizadas.

2. **class CurvaNoExiste(Error)**

   Se levanta cuando la curva elíptica de Weirstrass en el campo finito no existe.

   Ejemplo.

```
from ECDSA_keyGenerator import CurvaNoExiste
2

CurvaNoExiste(10, 10, 10)
```

Figura 3: Ejemplo de uso de la clase CurvaNoExiste

3. **class curve**

   Clase utilizada para el manejo de curvas como objetos.

   Ejemplo.

```
from ECDSA_keyGenerator import curve

'''Curva NIST P-256'''

name="P-256"
p = int(0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff)
a = int(0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc)
b = int(0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b)
G = (int(0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296),
     int(0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5))
n = int(0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551)
h = int(0x1)

curve(name, (a,b), p, G, n, h)
```

Figura 4: Ejemplo de uso de la clase curve

La clase curve tiene consigo funciones las cuales permiten que logre su cometido las cuales se documentan a continuación.

5.1.2. curve.set_Q(self, P: 'tuple[int,int]')

Inicializa el punto Q, el cual es usado para realizar diversas computaciones.

- **Return** None

Ejemplo.

```
from ECDSA_keyGenerator import curve

set_Q((0,0))
```

Figura 5: Ejemplo de uso de la función curve.set_Q

5.1.3. curve.point_doubling(self)

Calcula la operacion de duplicar un punto.

- **Return** None

Ejemplo.

```
1 from ECDSA_keyGenerator import curve

3 curve.point_doubling()
```

Figura 6: Ejemplo de uso de la función curve.point_doubling

5.1.4. curve.point_adition(self, P_1: 'tuple[int,int]', P_2: 'tuple[int,int]', ghost_eval: bool)

Calcula la operacion de sumar 2 puntos $P_1, P_2, P_1 != P_2$.

- **Return** None

Ejemplo.

```
1 from ECDSA_keyGenerator import curve

3 curve.point_adition((1,1),(0,0),False)
```

Figura 7: Ejemplo de uso de la función curve.point_adition

La función anteriormente mostrada es la última utilizada por la clase curve.

5.1.5. secure_random(strenght: int)

Función que genera un numero en el rango [0,n) de una manera segura criptográficamente.

- **Return** int

Ejemplo.

```
1 from ECDSA_keyGenerator import secure_random

3 secure_random(1000)
```

Figura 8: Ejemplo de uso de la función secure_random.

5.1.6. public_key(curve: curve, privateKey: int)

Genera una clave publica a partir de una privada y una curva. Devuelve una curva.

- **Return** curve

Ejemplo.

```
from ECDSA_keyGenerator import public_key, curve

'''Curva NIST P-256'''

name="P-256"
p = int(0xffffffff00000001000000000000000000000000ffffffffffffffffffffffff)
a = int(0xffffffff00000001000000000000000000000000fffffffffffffffffffffffc)
b = int(0x5ac635d8aa3a93e7b3ebbd55769886bc651d06b0cc53b0f63bce3c3e27d2604b)
G = (int(0x6b17d1f2e12c4247f8bce6e563a440f277037d812deb33a0f4a13945d898c296),
     int(0x4fe342e2fe1a7f9b8ee7eb4a7c0f9e162bce33576b315ececbb6406837bf51f5))
n = int(0xffffffff00000000ffffffffffffffffbce6faada7179e84f3b9cac2fc632551)
h = int(0x1)

curva = curve(name,(a,b),p,G,n,h)

private_key = secure_random(curva.n)

public_key(curva, private_key)
```

Figura 9: Ejemplo de uso de la función public_key.

5.2. curve_data.py

En este módulo se instancia la curva NIST P-256 y se guarda como un objeto.

La manera de importar todas las funciones de dicho modulo es la siguiente:

```
from curve_data import *
```

Figura 10: Forma de importar todo el modulo curve_data.

5.2.1. P_256()

Se utiliza para llamar los datos de la curva.

- **Return** curve

```
from curve_data import P_256

P_256()
```

Figura 11: Ejemplo de uso de la función P_256.

5.3. readWrite.py

Este módulo se encarga de escribir la clave privada y pública en un archivo .pem en formato hexadecimal, así como también escribe la firma en un archivo con nombre determinado y en un archivo .pem en formato hexadecimal. También lee los archivos mencionados con anterioridad y los devuelve en una tupla.

La manera de importar todas las funciones de dicho modulo es la siguiente:

```
from readWrite import *
```

Figura 12: Forma de importar todo el modulo readWrite.

5.3.1. write_private_key(private_key: int)

Función que recibe una clave privada como entero. Escribe esta firma a el archivo private.pem en formato hexadecimal. Si la escritura falla, levanta una excepción.

- **Return** None

```
from readWrite import write_private_key

write_private_key(1000)
```

Figura 13: Ejemplo de uso de la función write_private_key.

5.3.2. write_public_key(Q: 'tuple[int,int]')

Función que recibe una clave publica en formato (Qx,Qy) y un nombre de archivo. Escribe esta firma a el archivo public.pem en formato hexadecimal. Si la escritura falla, levanta una excepción.

- **Return** None

```
1 from readWrite import write_public_key

3 write_public_key((0,0))
```

Figura 14: Ejemplo de uso de la función write_public_key.

5.3.3. sig_write(signature: 'tuple[int,int]', filename: str)

Función que recibe una firma en formato (r,s), y un nombre de archivo. Escribe esta firma a el archivo <filename> + signature.pem en formato hexadecimal. Si la escritura falla, levanta una excepción.

- **Return** None

```
1 from readWrite import sig_write

3 sig_write((0,0), 'NombreArchivo')
```

Figura 15: Ejemplo de uso de la función sig_write.

5.3.4. sig_read(filename: str)

Función que recibe un nombre de un archivo generado por la función sig_write. Lee las lineas que contienen la firma y la convierten de vuelta a un entero. Devuelve una tupla (r,s).

- **Return** tuple[int,int]

```
1 from readWrite import sig_read

3 sig_read('NombreArchivo')
```

Figura 16: Ejemplo de uso de la función sig_read.

5.3.5. public_key_read(filename: str)

Función que recibe un nombre de un archivo generado por la función write_public_key. Lee las lineas que contienen la clave y la convierten de vuelta a un entero. Devuelve una tupla (r,s).

- **Return** tuple[int,int]

```
1 from readWrite import public_key_read

3 public_key_read('NombreArchivo')
```

Figura 17: Ejemplo de uso de la función public_key_read.

5.3.6. private_key_read(filename: str)

Función que recibe un nombre de un archivo generado por la función write_private_key. Lee las lineas que contienen la clave y la convierten de vuelta a un entero. Devuelve el entero.

- **Return** int

```
1 from readWrite import write_private_key

3 write_private_key('NombreArchivo')
```

Figura 18: Ejemplo de uso de la función private_key_read.

5.4. signVerify.py

Este módulo genera la firma digital de un archivo y comprueba la validez de la firma que le sea ingresada.

La manera de importar todas las funciones de dicho modulo es la siguiente:

```
1 from signVerify import *
```

Figura 19: Forma de importar todo el modulo signVerify.

5.4.1. sign_doc(private_key: int,curve:curve,message_or_filename,hashFunc:hashlib = hashlib.sha512(), test_k: int = None)

Función que genera la firma digital de un archivo en formato (r,s), recibe la clave privada, la curva a usar, el nombre del archivo (o en caso de testing el texto en bytes), la función hash a usar y en caso de testing la k proporcionada. Por defecto se usa SHA256 y no se da una k.

- **Return** 'tuple[int,int]'

```
from signVerify import sign_doc

sign_doc('firma')
```

Figura 20: Ejemplo de uso de la función sign_doc.

5.4.2. verify(signature: 'tuple[int,int]', curve: curve, publicKey: curve, message_or_filename, hashFunc: hashlib = hashlib.sha512())

Función que verifica que la firma dada sea valida, recibe la firma, la curva usada, la clave publica y el nombre del archivo (o en caso de testing el texto en bytes), y la función hash a usar. Por defecto se usa sha256.

- **Return** bool

```
from signVerify import verify

verify('privada')
```

Figura 21: Ejemplo de uso de la función verify.

5.5. certificate_writer.py

Este módulo contiene las funciones con las que se escriben los certificados.

La manera de importar todas las funciones de dicho módulo es la siguiente:

```
from certificate_writer import *
```

Figura 22: Forma de importar todo el modulo certificate_writer.

5.5.1. write_public_key_certificate(title: String, id_certificate: String, id_key: String, number_of_key_recieved: String, editor: String, date_of_creation: String, date_downloaded: String, validity: String, name_user: String, user: String, public_key: String, save_directory: String)

Función que genera el certificado de generación de llaves.

```
from certificate_writer import write_public_key_certificate

write_public_key_certificate('info')
```

Figura 23: Ejemplo de uso de la función write_public_key_certificate.

5.5.2. write_sign_document_certificate(title: String, id_certificate: String, id_sign: String, document_name: String, date_of_signing: String, validity_of_document: String, name_signer: String, user_signer: String, job_signer: String, public_key_signer: String, save_directory: String)

Función que genera el certificado de firma de un documento.

```
from certificate_writer import write_sign_document_certificate

write_sign_document_certificate('info')
```

Figura 24: Ejemplo de uso de la función write_sign_document_certificate.

5.5.3. write_verification_sign_certificate(title: String, id_certificate: String, id_sign: String, signing_key: String, document_name: String, verificator_name: String, verificator_job: String, validity_of_document: String, name_signer: String, user_signer: String, job_signer: String, public_key_signer: String, save_directory: String)

Función que genera el certificado de verificación de una firma.

```
from certificate_writer import write_verification_sign_certificate

write_verification_sign_certificate('info')
```

Figura 25: Ejemplo de uso de la función write_verification_sign_certificate.

5.6. forms.py

Este módulo se encarga de los formatos de input que se usan en las diferentes páginas de la página web
La manera de importar todas las funciones de dicho módulo es la siguiente:

```
1 from forms import *
```

Figura 26: Forma de importar todo el módulo forms.

Estas son las diferentes clases que se contienen en este módulo:

1. class loginForm(FlaskForm): Usado para el login de los usuarios
2. class userForm(FlaskForm): Usado para el agregado y la modificación de un usuario
3. class keyValidityForm(FlaskForm): usado para cambiar la validez de una llave
4. class uploadDocumentForm(FlaskForm): usado para agregar un documento a la página
5. class signDocumentForm(FlaskForm): usado para firmar un documento

5.7. app.py

Este es el programa principal, desde el cual se corre la página web usando Flask. Dentro de este código se importan las funciones de los otros módulos y se generan las diferentes clases que se van a usar para manejo de base de datos. Queda aclarar que ninguna de las funciones dentro de este programa se llaman por el usuario, estas se llaman desde los HTML con el uso de Jinja2, ya que todas las funciones dentro de este programa llevan el decorador de .ªpp.route", el cual define la ruta desde la cual se va a llamar la función.

5.7.1. Clases dento del programa para SQLAlchemy

1. class Users(SQLAlchemy(Flask(__name__)).Module)

   Esta clase la que define la tabla de usuarios, toma los siguientes nombres como columnas de la tabla: id, name, lastname, job, user, email, date_added, password_hash

2. class Keys(SQLAlchemy(Flask(__name__)).Module)

   Esta clase define la tabla de llaves, toma los siguientes nombres como columnas: id, user, id_user, public_key, private_key_hash, validity, downloaded, number_generated_key, date_modified

3. class Signatures(SQLAlchemy(Flask(__name__)).Module)

   Esta clase define la tabla de firmas, toma los siguientes nombres como columnas: id, name, id_file, id_signer, validity, fingerprint, date_added, checked, signature_filename

4. class Files(SQLAlchemy(Flask(__name__)).Module)

   Esta clase define la tabla de documentos, toma los siguientes nombres como columnas: id, name, pdf_file, all_signers, validity, date_added

5. class Certificates(SQLAlchemy(Flask(__name__)).Module)

   Esta clase define la tabla de certificados, toma los siguientes nombres como columnas: id, display_for, name, topic, pdf_directory, date_added

5.7.2. generate_key()

Genera instancia de clave privada y publica. Esta es la única función que no tiene el decorador de app.route, ya que es llamada desde otra función dentro de la misma página.

- **Return** tuple[int, tuple[int,int]]

```
2 private_key, public_key = generate_key()
```

Figura 27: Ejemplo de uso de la función generate_key.

..

5.7.3. index()

Carga la página principal de un usuario que no sea el admin, en esta página se puede ver los últimos documentos solicitados y los últimos certificados generados. También en esta página se muestra el estado de las llaves del usuario, en caso de que estas estén listas para descargarse se muestra un botón de generar llaves. Al dar click sobre un documento vigente y no firmado te manda a la página de firmado del mismo, al dar click sobre un certificado te lo descarga.

5.7.4. login()

Carga la página de inicio de sesión, en caso de iniciar sesión correctamente te manda a la página principal, en caso contrario te mantiene en la misma y te manda mensaje de contraseña incorrecta.

5.7.5. logout()

Cierra la sesión del usuario, te manda a la página de login

5.7.6. history()

Carga la página de historial de un usuario, en esta se ven todos los documentos que se encuentran en la base de datos en los que el usuario es solicitado para firmar, en caso de que el documento no haya sido firmado todavía y sea vigente, al dar click en un documento te manda a la página de ese documento, en caso contrario no te da el botón de acceso.

5.7.7. certificates()

Carga la página de certificados, desde la que los usuarios pueden ver todos los certificados que han generado y descargarlos

5.7.8. sign()

Carga la página de firmado de documentos, en esta página te muestra todos los documentos que no has firmado que solicitan tu firma, en caso de que un documento sea vigente te da la opción de cargar la página de ese documento, en caso contrario no te deja acceder

5.7.9. sign_document(id)

Carga la página de firma un documento, el cual se obtiene por un query a la base de datos usando el id que se le da a la función. En esta página puedes firmar un documento, para lo que necesitas cargar tu llave privada y escribir tu contraseña de usuario, en el caso de que hagas todo correctamente, te manda a la página de documentos por firmar, en caso contrario te mantiene en la misma página y te marca error.

5.7.10. get_keys(id)

Genera las llaves del usuario, la llave pública la guarda en base de datos y la llave privada se descarga en un archivo .pem, también genera un certificado de generación de llaves, el cual puedes ver desde la página de certificados

5.7.11. download_certificates(id)

Descarga un certificado, este se obtiene haciendo un query a la base de datos usando el id que solicita la función

5.7.12. admin_index()

A esta página solo tiene acceso el admin.

Carga la página de inicio del admin. En esta se puede ver los últimos documentos subidos a la página y los últimos certificados generados para el admin. Si das click sobre un documento vigente puedes ir a la página del documento, si das click sobre un certificado puedes descargar el certificado.

5.7.13. list_users()

A esta página solo tiene acceso el admin.

Carga la página de lista de usuarios, en esta se pueden ver a todos los usuarios que se encuentran en la base de datos. para cada usuario se puede ver nombre, nombre de usuario y si su llave es vigente, se puede modificar un usuario al darle al botón de modificar de un usuario, el cual te lleva a la página de modificar usuario. Se puede modificar la validez de una firma al darle al botón de Actualizar Llave, este último botón

solo se muestra si la firma del usuario está vencida. En la parte superior derecha se encuentra un botón para agregar nuevos usuarios.

5.7.14. add_user()

A esta página solo tiene acceso el admin.

Carga la página de agregar usuario. En esta página se definen las siguientes variables del usuario a agregar en un formato: nombre, apellido, puesto, nombre de usuario, email y contraseña. Al agregar al usuario te regresa a la página de usuarios, en caso de problema te mantiene en esta misma página.

5.7.15. update_user(id)

A esta página solo tiene acceso el admin.

Carga la página de modificar usuario, el usuario a modificar se obtiene por su id. En esta página puedes modificar la información del mismo, puedes borrar al usuario al darle al botón de borrar usuario o puedes modificar la validez de sus llaves.

5.7.16. delete_user(id)

A esta página solo tiene acceso el admin.

Borra a un usuario de la base de datos. Borra el usuario, sus llaves y sus solicitudes de firma.

5.7.17. generate_user_key(id)

A esta página solo tiene acceso el admin.

Carga la página de actualizado de validez de llaves de un usuario, el usuario se obtiene por el id que se le da a la función. En esta página se muestra la información del usuario, se muestra también un formato en el que pones la fecha hasta la que las llaves van a ser válidas.

5.7.18. upload_document()

A esta página solo tiene acceso el admin.

Carga la página de cargado de documento, en esta página el admin define el nombre del documento, los usuarios que la tienen que firmar, la fecha límite para firmarlo y el documento pdf referente al documento que se va a cargar.

5.7.19. documents()

A esta página solo tiene acceso el admin.

Carga la página de documentos. En esta se encuentran todos los documentos que se han subido a la página. Al dar click sobre un documento te lleva a la página del mismo.

5.7.20. document(id)

A esta página solo tiene acceso el admin.

Carga la página de un documento, en esta se puede ver el nombre del documento y se puede descargar el documento solicitado para firmar. También te muestra una lista de todos los usuarios que se solicitaron para firmar el documento, junto con el estado de su firma, en caso de que esta ya este para verificar, te muestra un botón de verificar firma.

5.7.21. download_document(id)

Descarga el pdf de un documento. Este documento se obtiene por un query a la base de datos dependiendo del id que se le de a la función.

5.7.22. verify_signature(id)

A esta página solo tiene acceso el admin.

Verifica una firma. En caso de que esta sea verificada correctamente te genera un certificado de validación de firma y te marca la firma como verificada.

5.7.23. delete_document(id)

A esta página solo tiene acceso el admin.

Borra un documento. El documento a borrar depende del id que se le de a la función.

6. Planes futuros de desarrollo.

En búsqueda de una mejora continua en el proyecto se proponen varias soluciones a futuro que de momento no pueden ser desarrolladas.

1. La base de datos no se encuentra protegida por lo que se espera que esta protección sea otorgada por el administrador de la misma.

2. Para esta implementación se está utilizando una base de datos local por lo que se espera pueda ser implementada una base de datos en servidor.

3. El programa actualmente solo puede funcionar a través de una interfaz gráfica por lo que se buscaría que esta pueda ser utilizada sin un entorno gráfico.

4. Esta implementación está realizada en un lenguaje interpretado por lo que idealmente podría mejor su velocidad al ser reestructurado en un lenguaje de programación compilado.

7. Ayuda y FAQs.

7.1. ¿Cual es el tamaño máximo de clave soportado?

Debido a que se usó Python 3 como base, no existe un límite fijo en el tamaño de los enteros que se pueden manejar, el límite se define por la cantidad de memoria del sistema donde se despliegue la solución. Dicho esto, se debe recordar que el punto de usar curvas elípticas es manejar un tamaño de clave menor ofreciendo una seguridad similar a, por ejemplo RSA. No se recomiendo usar números innecesariamente largos si no se ha probado que el sistema en cuestión pueda manejarlos correctamente.

7.2. Me preocupa la velocidad de la implementación. ¿Cual es el tiempo esperado de ejecución de las 3 fases del algoritmo?

Tomando como referencia el equipo en el que se hicieron las pruebas, usando SHA-512, y la curva NIST P-256 se puede esperar que la generación de clave, generación de firma y verificación de firma sucedan en alrededor de 0.1 segundos (excluyendo el tiempo que tomaría al usuario interactuar con la interfaz gráfica). Dicho esto, los tiempos de ejecución no se ajustan a ninguna distribución en especifico, por lo que no se puede hablar de un tiempo promedio o desviación estándar.

7.3. Me preocupa la seguridad de la implementación. ¿Está protegida contra timming attacks?

Si, el uso de lo que se conoce como evaluación fantasma se uso para balancear las operaciones. Los umbrales de tiempo de cada fase son lo suficientemente bajos para que sea poco factible encontrar información sobre las entradas analizando los tiempos. Se habla de ejecuciones que suceden en milésimas de segundo.

7.4. Nuestra organización usa otra solución de base de datos, ¿Puedo conectar esta solución a otro sistema?

Claro, aunque no proveemos información sobre como realizar esta conexión, si requieren usar una solución distinta pueden cambiar de entorno de base de datos. Basta con diseñar funciones que conectándose a el entorno de base de datos escogido, retornen la misma información que espera el programa. El type hinting del código fuente ayuda con esto.

7.5. Me preocupa la seguridad de la curva utilizada. ¿Puedo usar otra?

Claro, solo agrega los parámetros de la curva en el archivo **curve_data** y reemplaza las instancias de la función P_256() por la curva de tu elección.

7.6. Encontré una vulnerabilidad en el código fuente. ¿Como puedo reportarlo?

Nos tomamos la seguridad muy en serio. Por favor escríbenos a A01620402@tec.mx con el asunto: Vulnerabilidad Firma digital, e incluir datos de contacto en el cuerpo de correo. Favor de **no incluir el reporte de la vulnerabilidad encontrada en el cuerpo del correo**. Tomaremos tus datos y nos pondremos en contacto por un medio seguro para tomar reporte de la vulnerabilidad.

Referencias

[1] F. Rodriguez-Henriquez, N. Saqib, A. D. Pérez, and C. K. Koc, *Cryptographic Algorithms on Reconfigurable Hardware*. New York, Estados Unidos: Springer Publishing, 2007.