Instituto Tecnológico y de Estudios Superiores de Monterrey
Escuela de Ingeniería y Ciencias
Ingeniería en Ciencias de Datos y Matemáticas

Implementación de un Esquema de Firma Digital para Administración de Compras: Caso Fundación Teletón

Frida Bautista Martínez - *A01235009*
Leyre Carpinteyro Palos- *A01610296*
Samuel Méndez Villegas - *A01652277*
Salette Noemi Villalobos - *A01246619*
Paola Montserrat Vega Ortega - *A01274773*
Ethan Enrique Verduzco Pérez - *A01066955*

Uso de álgebras modernas para seguridad y criptografía
Grupo: 401

Profesores:
Dr. Alberto Francisco Martínez Herrera
Dr. Daniel Otero Fadul

Organización Socio Formadora: Fundación Teletón

Monterrey, Nuevo León
03 de mayo de 2022

Índice

Abstract: *En la actualidad, gracias al creciente desarrollo tecnológico, muchos procesos que solían realizarse de forma manual han pasado a ser digitales como lo es la firma electrónica. Esto implica que se deben de desarrollar esquemas robustos que garanticen la seguridad de las personas. En este reporte se trabajó de la mano con Fundación Teletón para migrar de un esquema de firma autógrafa a uno digital orientado a los procesos de compra. La solución construida consistió en una aplicación web que sigue el esquema del algoritmo de firma digital con curvas elípticas (ECDSA), la cual permite realizar 3 procesos fundamentales: creación de claves públicas y privadas, generación de firma digital y verificación de firma. Las herramientas que se utilizaron para desarrollar la aplicación fueron HTML5, y para vincular la parte del esquema de firma electrónica construido en el lenguaje de programación Python, se utilizó FLASK. Como resultado se obtuvo una aplicación funcional y efectiva que permite a un ente certificado generar las claves correspondientes, así como revocar certificados. Adicionalmente, los usuarios son capaces de subir los documentos que quieran firmar electrónicamente junto con su clave privada, y para verificar, son capaces de subir el documento con la firma digital y el archivo a verificar. En conclusión, gracias al esquema se puede garantizar la autenticidad e integridad de las firmas electrónicas de los usuarios, lo que permite agilizar los procesos que necesiten de esta.*

Keywords: *ecdsa, desarrollo web, firma digital, certificados digitales, curvas elípticas*

1. Introducción

Hoy en día es de suma relevancia contar con almacenamiento y comunicaciones electrónicas que contengan altas medidas de seguridad informática para la protección de datos personales y sensibles de una empresa u organización. Al estar en un mundo prácticamente digitalizado tras el COVID-19, es necesario abordar temas de seguridad informática y criptografía al realizar trámites de carácter oficial, como es el caso de la organización socio formadora *Fundación Teletón.*

Fundación Teletón es una organización sin fines de lucro fundada en 1997. Su propósito es atender a personas con distintas discapacidades, cáncer y autismo (TEA), con el fin de promover el desarrollo e inclusión en México [1]. Dicho objetivo lo hace cumplir a través de sus centros de rehabilitación e inclusión infantil que se encuentran alrededor del país. Además de los centros de rehabilitación y hospitales asociados, Fundación Teletón cuenta con un corporativo ubicado en Estado de México el cual tiene la función de velar por todos los centros de la fundación como unidades independientes. En dicho corporativo se tienen centralizadas las operaciones que van de la mano con la función de procesos de compras y administración. Actualmente, Fundación Teletón realiza estos procesos de una manera autógrafa, es decir que es necesario recabar físicamente las firmas necesarias que autoricen procedimientos y trámites, y debido al COVID-19 esta tarea ha sufrido contratiempos para completar procedimientos y trámites internos. Es por la razón anterior, que en conjunto con Fundación Teletón se busca resolver la problemática con la implementación de un algoritmo de firma digital con múltiples participantes, el cual agilice la firma de dichos documentos, su validación, y su resguardo seguro. De igual forma, se busca que la solución esté orientada al usuario final para que éste sea capaz de firmar digitalmente sus documentos.

En la figura 1 se muestra un diagrama que plantea a grandes rasgos lo que se busca conseguir con la solución de la problemática. En dicha figura se muestra que una entidad, o sistema administrador, se

encargará de generar firmas digitales para los usuarios, y de verificar la validez de la firma a través de un esquema de firma digital. Esto es con el objetivo de agilizar y autenticar de manera segura los procesos de compras de la fundación.

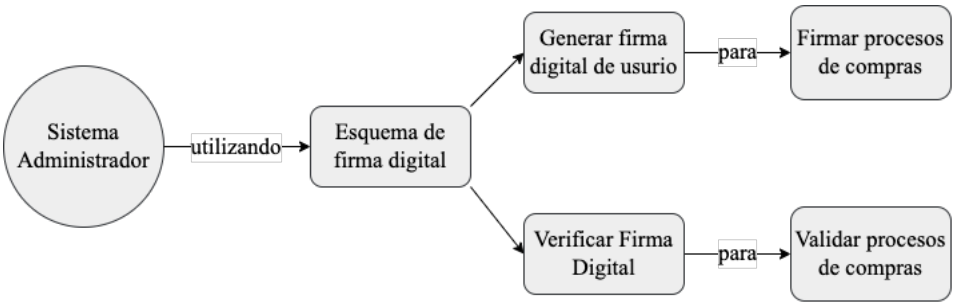

Figura 1: Diagrama de la solución a implementar

De igual forma, en el cuadro 1 se muestra algunas de las diferencias entre el proceso que actualmente se aplica en Fundación Teletón, y el propuesto en la solución.

	Firma autógrafa	**Firma Digital**
Recaudación	Forma física	Forma digital
Autenticidad	Se puede cuestionar la autenticidad del usuario	Garantiza la autenticidad del usuario
Repudio	Repudiable	No repudiable
Integridad	Puede sufrir cambios después de haber sido firmado	Constancia de conservación para mantener la integridad de la firma
Evidencia	No cuenta con evidencia documental de que el archivo fue firmado	Se tiene evidencia documental de que el archivo fue firmado (fecha y tercero confiable).

Cuadro 1: Diferencias entre soluciones de firma

Cabe mencionar que de manera paralela se buscará contribuir al Objetivo de Desarrollo Sostenible 9: Industria, innovación e infraestructuras, el cual habla acerca de desarrollar una industrialización inclusiva y sostenible para poder generar nuevas tecnologías y permitir el uso optimizado de recursos [2]. Lo anterior por medio de la implementación de un esquema para la protección de firmas digitales en el área de operaciones y compras de la fundación.

El algoritmo a implementar requiere del seguimiento de un protocolo que asegure su buen desempeño, su fiabilidad y viabilidad, así como su eficiencia ante posibles ataques cibernéticos. Las firmas a ingresar deben ser verificables para evitar problemas de validación. De igual manera, esta aplicación incluye autenticación, integridad de los datos y el no repudio, sin embargo una de las aplicaciones más significativas de las firmas digitales es la certificación de claves públicas en grandes redes.

Finalmente, a modo de resumen, se propone una solución que involucra un esquema de firma digital orientado al usuario final con el objetivo de beneficiar directamente a Fundación Teletón. Esta solución en un futuro puede ser de utilidad no únicamente en el corporativo de la fundación, sino que también en los centros de rehabilitación, pues ayudaría a los médicos y especialistas en las atenciones médicas que brindan, ya que al utilizar la firma digital se sustituiría las firmas autógrafas y sellos que validen documentos médicos.

2. Estado del Arte

Para poder desarrollar la solución anteriormente descrita, primeramente se investigó en la literatura tanto algoritmos de firma digital como bibliotecas de funciones existentes. Esto con el objetivo de tener

una visión mucho más amplia de lo que ya se hace y cómo se hace, para así, tener un punto de partida para la construcción del esquema.

Primeramente se investigaron diferentes algoritmos de firma digital existentes, los cuales se observan en el cuadro 2.

Referencia	Algoritmo	Función Hash	Tamaño Clave	Patente
[3]	RSA	SHA-256	3072-bit (para 128 bit de nivel de seguridad).	Patente MIT expirada en el año 2000.
[3]	DSA	SHA-1	Mínimo 512 bits, máximo 1024 bits	Patente NIST sin regalías.
[4]	Curvas Elípticas	SHA-2 (SHA-256)	Salida de 256 bits (32 bytes)	Certicom expirada en el año 2017.

Cuadro 2: Referencias de firma digital

Posteriormente, se investigaron algunas bibliotecas que se componen de funciones que realizan los algoritmos correspondientes para lograr firmar documentos de forma electrónica. En el cuadro 3 se muestra lo encontrado.

Referencia	Biblioteca	Lenguaje	Licencia	Clave Pública
[5]	PyCryptodome	Python	Apache Software License, BSD License	Elliptic Curve Cryptography (ECC), RSA
[5]	Botan	C++	BSD License	ECDSA, DSA
[5, 6]	OpenSSL	C	BSD License	RSA,ECC
[7]	pyca/cryptography	Python	Apache Software License, BSD License	ECC, RSA, DSA, etc.

Cuadro 3: Biblioteca de funciones

A continuación, se entrará a hablar en detalle sobre la información recabada en las tablas 2 y 3, describiendo brevemente los algoritmos y bibliotecas.

2.1. Descripción esquemas de firma digital

2.1.1. Esquema de firma digital con RSA

El primer método descubierto para realizar la certificación de claves públicas en grandes redes fue el esquema RSA, el cual sigue siendo uno de los más usados [3]. Consiste en un algoritmo de cifrado asimétrico que se basa en el problema de factorización de enteros, los logaritmos discretos y la exponenciación modular. Puede utilizar claves de tamaños 1024, 2048, 4096, ..., 16384 bits, incluso funciona con claves más grandes pero se vuelve muy lento para un uso práctico. Para un nivel de seguridad de 128-bit, se requiere una clave de 3072-bit [4]. El par de claves consisten en:

- Clave pública $\{n,e\}$
- Clave privada $\{n,d\}$

Los números n y d generalmente son grandes mientras que e tiene un tamaño pequeño [4]. La firma de un mensaje m con el exponente d de la clave probada consiste en:

1. Calcular el hash del mensaje: $h = hash(m)$
2. Encriptar h para calcular la firma: $s = h^d (mod\ n)$

2.1.2. Esquema de firma digital con DSA

En cuanto al algoritmo DSA, este consiste también en un algoritmo de firma digital y fue el primero de estos en ser reconocido [3]. Explícitamente requiere del uso de la función hash SHA-1, para la generación y verificación debe:

1. Se seleccionar un entero aleatorio k,
2. Se calcula $r = (\alpha^k \, mod \, p) \, mod \, q$
3. Se calcula $z = k^{-1} \, mod \, q$
4. Se calcula $u = h(m)$
5. Se calcula $s = z * (u + a * r) \, mod \, q$
6. La firma del mensaje m es el par (r, s)

El tamaño de p puede ser cualquier múltiplo de 64 entre 512 y 1024 bits, donde 768 es el mínimo recomendado. El algoritmo tiene la ventaja de que la exponenciación puede ser precomputarizada y no necesariamente hecha en el momento de generación de la firma, cosa que no es posible con el RSA [3].

2.1.3. Esquema de firma digital con curvas elípticas

Uno de los esquemas de firma digital que se propone en [4] es el uso de criptografía de curvas elípticas. El esquema tiene como base las matemáticas que hay detrás de los grupos cíclicos de curvas elípticas sobre campos finitos y la dificultad del problema del logaritmo discreto. Las curvas elípticas que se utilizan definen tres elementos importantes [8, 9]:

- **E:** Curva elíptica, definida dentro de un campo finito.
- **Orden n:** número primo suficientemente grande (mayor a 160 bits), que divida al orden de la curva. Además define la longitud de las llaves privadas.
- **Un punto generador G:** punto que se utiliza para aplicar el producto escalar en la curva, de orden **n**.

El esquema abarca tres pasos importantes. El primero de ellos, es la generación del par de llaves, es decir la llave pública $\{x, y\}$ y la llave privada. Mientras la pública será representada por un punto de la curva elíptica, la clave privada será un número entero que especifique el total de ciclos realizados.

El segundo paso consiste en la realización de la firma, en donde se toma como entrada un mensaje (msg) y la llave privada $(privKey)$ y se siguen, los siguientes pasos:

1. Se utiliza una función hash (SHA-256) para calcular el hash del mensaje (h).
2. Se genera un número aleatorio k en el rango de $[1, n-1]$
3. Se calcula el punto aleatorio siguiendo la operación: $R = k * G$ y se toma la coordenada del eje x (R_x).

4. Se calcula la prueba de la firma (s). $s = k^{-1} * (h + privKey)$

Como salida se obtiene la firma digital que consiste en un par de enteros $\{r, s\}$.

Finalmente, como tercer paso importante se encuentra la validación de la firma digital en donde se toma como entrada el mensaje firmado (msg) y la firma digital ($\{r, s\}$). Este proceso sigue los siguientes pasos como se describe en [4].

1. Se calcula el hash (h) de msg utilizando la misma función hash durante la firma.
2. Se calcula el inverso modular de la prueba de la firma de la siguiente forma: $s1 = s^{-1} (\text{mod } n)$
3. Se recupera el punto que se usó para firmar: $R^{'} = (h * s1) * G + (r * s1) * pubKey$
4. Se toma la coordenada x de $R^{'}$: $r^{'} = R^{'}x$
5. Se valida la firma comparando que $r == r^{'}$

Como salida se obtiene un valor booleano que valida la firma digital. De esta manera, se puede verificar la autenticación, integridad y no repudio del mensaje enviado.

2.2. Descripción bibliotecas de funciones

Como se puede observar en la tabla 3 existen diversas librerías que nos permiten implementar funciones criptográficas pre-diseñadas. Estas librerías se encuentran en varios lenguajes de programación, como Python, C++ y C, por lo que se podrá decidir trabajar en el lenguaje que resulte más cómodo.

Un aspecto importante al momento de utilizar estas librerías es tomar en consideración la licencia de uso bajo las que es tan protegidas. En este caso se estará trabajando en Python por lo que es de interés conocer los términos de las licencias 'Apache' y 'BSD' que son las que aplican para ambas librerías de Python, 'PyCryptodome' y 'Pyca/Cryptography'.

En el caso de la licencia 'BSD', se da el permiso de uso de la fuente con o sin modificaciones con la condición de que se incluya la información de la licencia en la distribución. De igual manera todo contribuidor acepta que su contribución caiga bajo la misma licencia de uso que la fuente.

Para la licencia 'Apache Software', el código puede ser usado, reproducido y modificado libremente. Sin embargo para cualquier modificación realizada se debe dejar claro que fue una modificación propia y no parte del código fuente. Cualquier contribución o modificación deberá recaer bajo la misma licencia de uso e incluir la misma información de licencia en su distribución.

Ambas licencias permiten la distribución, reproducción y modificación del código fuente ya sea para uso con fines comerciales o no, por lo que se puede asegurar que se cuenta con la libertad de uso de estas librerías. Con el conocimiento de estas condiciones y aceptando los términos se puede proseguir a hacer uso de estas librerías, 'PyCryptodome' y 'Pyca/Cryptography', para los fines de este proyecto con prácticamente ninguna limitación.

El ultimo aspecto a tomar en consideración es el tipo de clave pública que manejan. En el caso de 'PyCryptodome' se trabaja con la implementación *Eliptic Curve Cryptography* ('ECC'), y 'RSA'. Mientras que en 'Pyca/Cryptography' se manejan las implementaciones 'ECC', 'RSA', 'DSA', entre otros. Tomando en cuenta las referencias académicas, 'Pyca/Cryptography' brinda más opciones y recursos para

la implementación de distintos algoritmos de encriptado. La decisión de uso con respecto a cuál librería queda a determinación por el algoritmo seleccionado.

2.3. Algoritmo escogido

Con el previo análisis realizado, considerando los distintos tipos de algoritmos de firmas digitales y los recursos disponibles en los lenguajes de programación, se puede llegar a la selección del algoritmo para desarrollar este proyecto. Entre las opciones analizadas se encuentran las implementaciones de 'DSA', 'RSA' y curvas elípticas 'ECC'. Las tres opciones constan de una firma digital compuesta por dos claves, una privada y una pública ya que son del tipo de criptografía asimétrica.

Comparando los algoritmos tenemos que el 'DSA' y 'RSA' podrían considerase equivalentes en cuestión de rendimiento y seguridad. Sin embargo cuentan con sutiles diferencias como que el 'RSA' es más rápido para el proceso de firmado, pero el 'DSA' es más eficiente para la verificación [10]. Otro aspecto a tomar en cuenta es el Protocolo SSH con el que son compatibles. 'RSA' se considera compatible con SSH y SSH2 [10]. Mientras que el 'DSA' es solo compatible con SSH2 ya que el SSH no es considerado lo suficientemente seguro como su segunda versión.

Por último, comparando el 'ECC' con el 'RSA' y el 'DSA' se puede notar una mayor diferencia. El 'ECC' es un algoritmo más sutil basado en curvas elípticas y comparado con el 'RSA' o el 'DSA' la diferencia principal se encuentra en el tamaño de las clave, siendo el 'ECC' muchos más eficiente manteniendo el mismo nivel de seguridad con un menor tamaño de clave [11]. Para un nivel de seguridad de 128 bits el 'RSA' hace uso de 3072 bits mientras que el 'ECC' ofrece el mismo nivel de seguridad con 256 bits [10]. Esta diferencia vuelve trabajar con el 'ECC' más eficiente para trabajar.

Tomando en cuenta esta comparación, se considera apropiado el trabajar con el algoritmo de criptografía de curvas elípticas 'ECC' para este proyecto. Esto debido a que la eficiencia y tamaño de las claves en el 'ECC' vuelve más sencillo su uso, permitiendo realizar el encriptado y desencriptado con menor poder de procesamiento. El algoritmo 'ECC' implementado correctamente permitirá tener un sistemas rápido y eficiente de firmas digitales manteniendo altos niveles de seguridad digital.

3. Métodos y Desarrollo de la Solución Propuesta

Gracias a la literatura consultada, se pudo establecer un punto de partida para generar la solución a la problemática. La solución que se proporcionará será utilizar distintas bibliotecas de funciones existentes que implementen el algoritmo de firma digital con curvas elípticas. Adicionalmente, la solución será orientada al usuario final, es decir que se desarrollará de forma paralela una aplicación web de fácil acceso y uso. En dicha aplicación, se buscará que el personal autorizado pueda generar el par de claves, firmar documentos y verificar estos últimos. En la figura 2 muestra un diagrama de flujo de la solución propuesta. En la imagen se observa que el flujo comienza con una entidad certificada encargada de la generación y resguardo de las claves de los usuarios. A través de la clave pública, se genera un certificado que valida el estado de la clave, pudiendo tomar diferentes estados, siendo estos activo, inválido, revocado y expirado. Tanto la clave pública generada como el estado del certificado se almacena en una base de datos junto con los datos del usuario. Posteriormente, para que un usuario pueda firmar un documento, es necesario

cargar un documento `pdf` con el material que se quiere firmar, así como un documento `pem` con la clave privada del usuario. Con la clave privada se valida si el certificado se encuentra en un estado activo, si es de esta forma, se continua con el proceso y se genera la firma, mientras que si el certificado se encuentra en cualquier otro estado, el sistema no genera la firma. La firma generada se guarda en un documento `pem` junto con la clave pública. Finalmente, para la verificación de la firma, se carga el archivo `pem` con la firma digital y clave pública, junto con el archivo `pdf` firmado. En caso de que la firma no coincida, se muestra al usuario un anuncio de posible corrupción del archivo, mientras que si coincide, la firma es verificada con éxito y se tiene la certeza de ser una firma auténtica e integra.

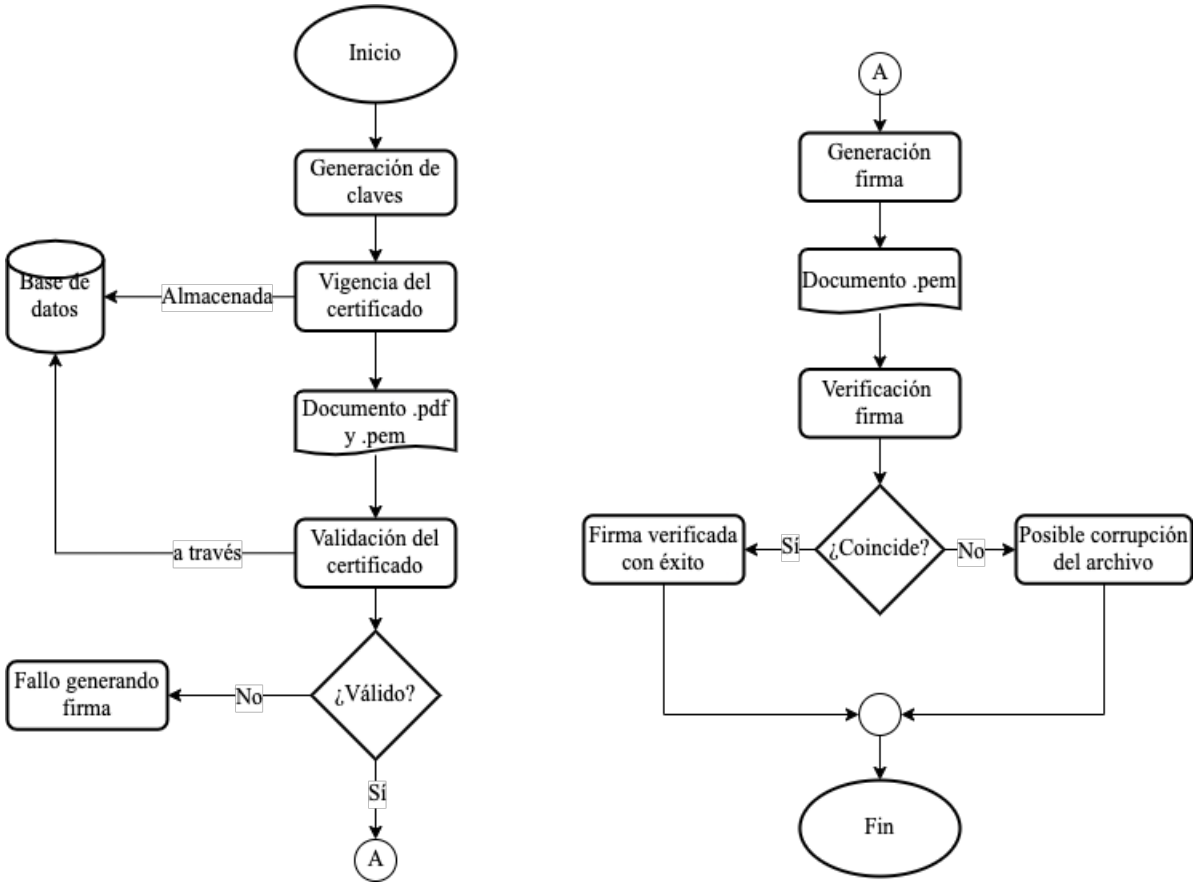

Figura 2: Diagrama de la solución a implementar

A continuación se entrará en detalle a cada una de las etapas fundamentales del esquema de firma digital planteado, siendo estas la generación de claves, la generación de firma digital y la verificación de la misma. Cabe destacar que de igual forma se profundizará en cómo se desarrollo la aplicación web.

3.1. Aplicación web

La solución para la problemática planteada por Teletón consiste en proporcionar dos páginas web, cada una de esta contará con distintos privilegios de acceso. La primera de ellas será accesible exclusivamente a una persona que sea la responsable del departamento de Tecnología e Informática, esto debido a que en esta primer página se realizará la generación de claves para todos los miembros que requieran de una firma digital. Asimismo, en esta se podrá indicar la expiración y revocación de firmas, para los casos que lo requieran. La segunda página será más accesible porque será donde se realice la firma de documentos

y la verificación de firmas.

Para la creación de las páginas web se utiliza en el Front-End el lenguaje HTML (HyperText Markup Language), debido a que es el más utilizado para la creación de páginas. Junto con este, el lenguaje CSS (Cascading Style Sheets) para dar un formato al HTML amigable para el usuario.
Para lo correspondiente al Back-End, este consistirá en el código para generación de firmas, firma de documento y verificación de firma en el lenguaje Python 3. Sin embargo, para poder conectar el Front con el Back, se utiliza Flask para que todos los archivos que suba el usuario puedan ser recibidos y procesados, para el output esperado.

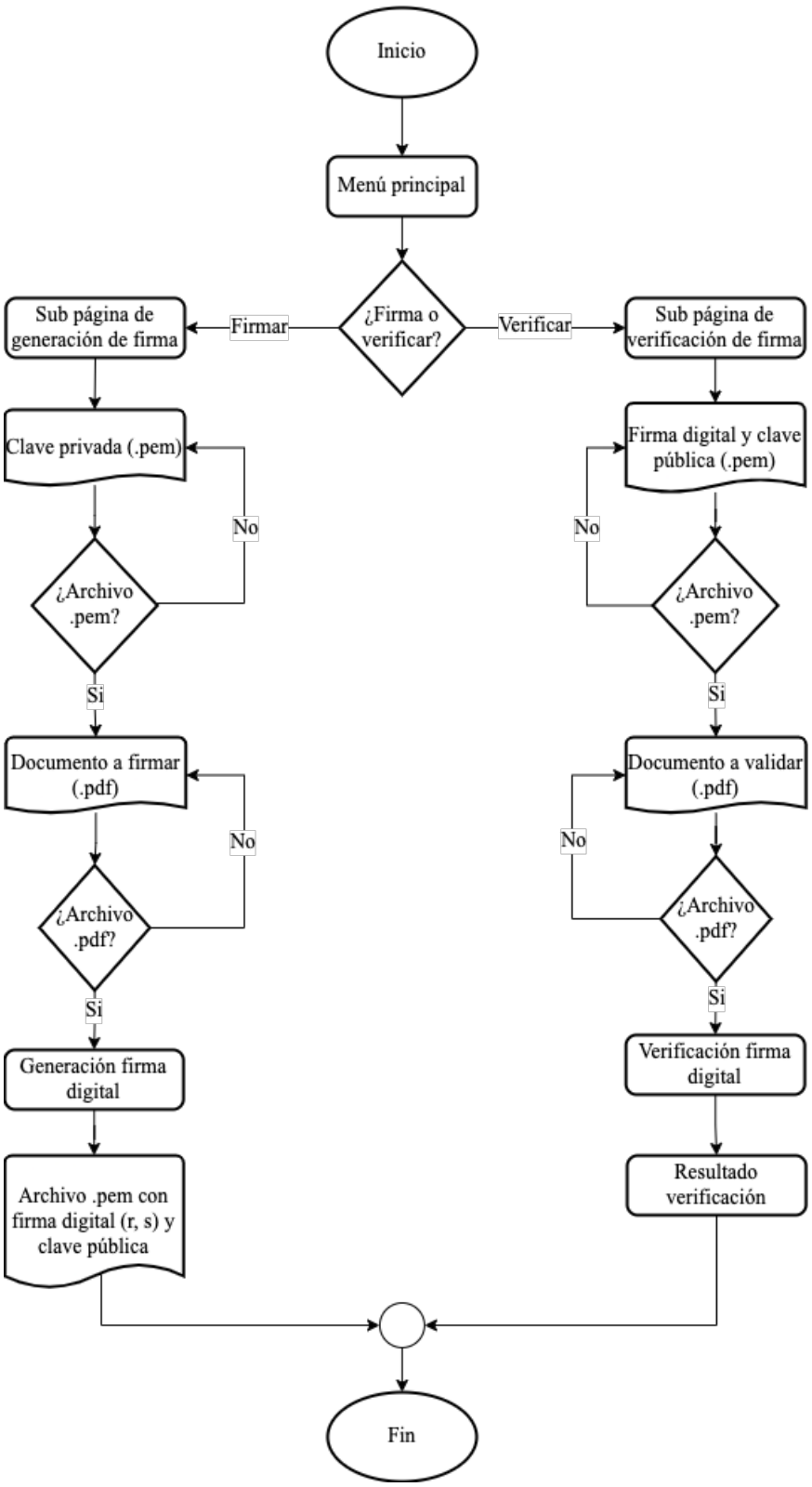

Figura 3: Diagrama del funcionamiento de la página web para la generación de firma digital y su verificación

En la figura 3 se muestra un diagrama que representa el flujo que tendrá el usuario al utilizar la página web de firma y verificación. Este comienza en un menú principal en donde se puede seleccionar dos diferentes opciones, “firmar un documento” o “verificar un documento”. En caso de que se seleccione la primera opción, la página conduce al usuario a una sub-página en donde tendrá que cargar dos docu-

mentos. El primero es la clave privada que el usuario posee, la cual deberá ser subida en un documento `pem`. El otro documento que se debe de subir es el correspondiente al archivo `pdf` que se quiera firmar electrónicamente. La página evalúa que los archivos se encuentren en el formato especificado, y de no ser el caso, deberán de cargarlo con el formato requerido. Una vez cargados los archivos, se genera la firma digital y el usuario como resultado obtiene un documento `pem` con la firma digital y la clave pública del usuario.

Por el otro lado, si se escoge la alternativa de verificar algún archivo, se conducirá al usuario correspondiente a la sub-pagina de verificación. Allí, tendrá que cargar el documento `pem` que contenga la firma digital y la clave pública del usuario que firmó. De igual forma, deberá de cargar el documento que se quiere revisar que fue firmado. Así como sucedió en la parte de generación, si los archivos subidos no tienen el formato adecuado, se deben de volver a cargar. Es en este punto en donde se realiza todo el proceso de la verificación de la firma y finalmente, el usuario obtiene como salida el resultado de la verificación, el cual puede ser que la firma es correcta, o que el archivo podría estar corrupto.

Cabe resaltar que en la sección 6, perteneciente a los anexos, se pueden encontrar diferentes imágenes del funcionamiento de la aplicación web.

3.2. Generación de claves

En la generación de claves se empleó el algoritmo ECDSA (Elliptic Curve Digital Signature Algorithm) con la curva NIST 256 bits. Se asume además que únicamente el administrador o la figura autorizada tendrá acceso al sistema. El propósito general del procedimiento es tener un sistema de certificados basado en la PKI (Public Key Infrastructure), donde las llaves serán plenamente identificadas con los campos: 'ID Algoritmo', 'Emisor', 'No antes de', 'No después de', 'Sujeto', 'Algoritmo de clave pública', 'Clave pública' y 'Estado' [12]. A continuación se muestra en detalle el significado que se le dará a cada uno de estos campos:

- **ID Algoritmo:** número único con el que se identifica el registro de la clave.
- **Emisor:** organización que está emitiendo el certificado.
- **No antes de:** fecha que establece hasta qué día la certificación será activada.
- **No después de:** fecha que establece hasta que día la certificación tiene vigencia
- **Sujeto:** organización a la que se le está emitiendo el certificado.
- **Algoritmo de clave pública:** algoritmo criptográfico que se está utilizando para generar las claves.
- **Clave pública:** clave pública del sujeto guardada en formato de bits.
- **Estado:** indica si las llaves se encuentran activas, inválidas, revocadas o expiradas.

Además, del estado activo de las llaves, están pueden llegar a ser revocadas, expiradas o inválidas según la circunstancia. A continuación se indica el significado de estos términos:

- **Revocada:** El administrador decide revocar una clave determinada.

- **Expirada:** En caso de que se supere la fecha 'No después de', la clave se marcará como expirada.
- **Inválida:** Si la fecha 'No antes de' todavía no se cumple, se registrará como inválida.

Para el almacenamiento y administración de las claves públicas se decidió implementar una base de datos en Python. Dicha base de datos se compondrá únicamente de los campos descritos con anterioridad. Adicionalmente, todos los cambios y nuevos registros se irán almacenando en la base de datos del sistema.

En la figura 4 se observa el diagrama de flujo correspondiente al proceso de generación de claves públicas y certificados por parte de la entidad autorizada para llevar a cabo esta acción.

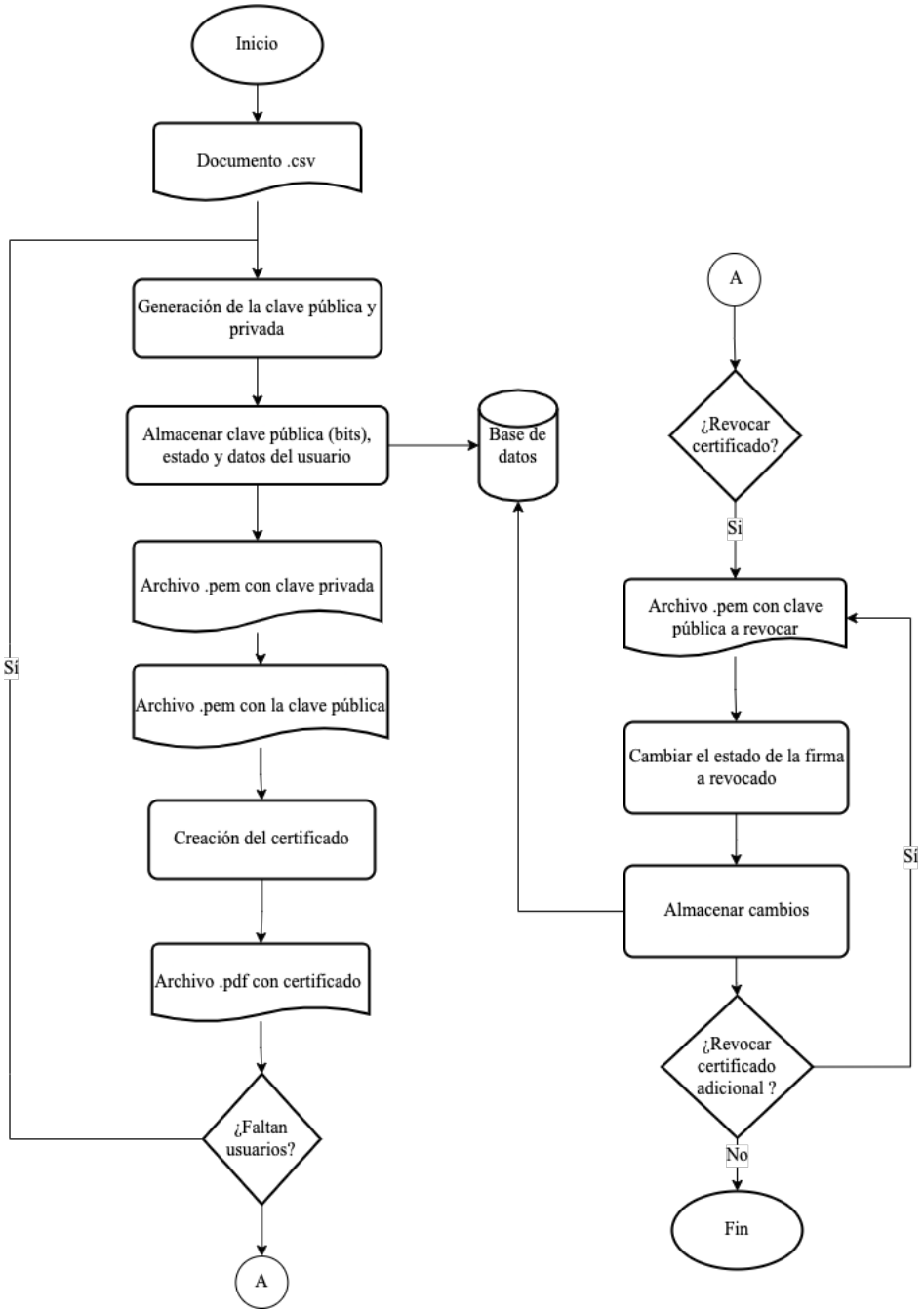

Figura 4: Diagrama de flujo de la generación de claves y certificados para los usuarios

A continuación, se explica con más detalle cada uno de los pasos del diagrama para la generación y almacenamiento de claves.

El primer paso consistirá en la lectura de un archivo `.csv` que será proporcionado por el usuario con los datos de los sujetos que adquirirán una llave pública y privada. Los campos deben de ser 'ID Algoritmo', 'Emisor', 'No antes de', 'No después de', 'Sujeto' y 'Algoritmo de clave pública'. El resto de los campos se agregarán automáticamente durante el proceso. A continuación se puede ver el código de la lectura del archivo:

```
def lectura_csv(nombre_csv):
  data = []
  with open(nombre_csv, mode ='r')as file:
    csvFile = csv.reader(file)
    for lines in csvFile:
      data.append(lines)
  return data
```

Una vez almacenados estos datos, se prosigue a crear las claves públicas y privadas para cada individuo con el uso de la librería ECDSA. Además, se agrega el 'Estado' de los sujetos como 'Activa' y se añade la firma pública de cada sujeto a la lista de datos. Aquí es muy importante destacar que el algoritmo criptográfico con el que se están generando las claves es curvas elípticas. Más concretamente se utiliza la curva `NIST P-256` la cual pertenece al *NIST* (National Institute of Standards and Technology). Adicionalmente, en la librería ECDSA se encuentra una función que crea automáticamente las claves, simplemente se tiene que indicar qué curva se está utilizando. A continuación, se presenta el código:

```
def crear_claves(data, num_sujeto):
  private = ecdsa.SigningKey.generate(curve=ecdsa.NIST256p,)
  privatepem = private.to_pem()
  public = private.get_verifying_key()
  data[num_sujeto-1].append(public)
  data[num_sujeto-1].append('Activa')
  return data, privatepem, public
```

Ya creadas la claves, se crean archivos separados `.pem` para cada clave y para cada uno de los usuarios. Para ello, se convierte el formato de la llave de bits a texto plano. El código es el siguiente:

```
def byte_a_texto(key):
  p_key = key.to_string()
  p_key = codecs.decode(p_key, 'latin1')
  return p_key

def archivo_key(num_archivo, key, name):
  name_a = name + str(num_archivo) + '.pem'
  if name == 'privateKey':
     text_file = open(name_a, "wb")
     text_file.write(key)
     text_file.close()
```

```
  elif name == 'publicKey':
     text_file = open(name_a, "w", encoding="utf-8")
     text_file.write(byte_a_texto(key))
     text_file.close()
```

Posteriormente, se crea un certificado para cada uno de los usuarios en formato `pdf` para que este pueda ser firmado por la autoridad certificadora, en este caso el administrador del sistema, con la Generación de Firma Digital.

Una vez que se ha obtenido la información necesaria, se almacenan todos los campos anteriormente mencionados en la base de datos del sistema con el siguiente código:

```
def cargar_base(data, nom_base):
  for s_data in data:
    with open(nom_base, 'a', newline='') as f_object:
      writer_object = writer(f_object)
      writer_object.writerow(s_data)
      f_object.close()
```

El resultado se observa en el cuadro 4.

ID Algoritmo	Emisor	No antes de	No despues de	Sujeto	Algoritmo CP	Clave Publica	Estado
ECDSA	Teleton	20/04/2022	30/04/2022	Teleton inc	ECDSA	7723cf55a0f4919a84adb1b13301e426bf161a...	Activa
ECDSA	Uber	23/04/2022	30/04/2022	Uber inc	ECDSA	9e5c0dc2ee992173d97e6371028b969873f4e1...	Activa
ECDSA	CEMEX	18/04/2022	19/04/2022	CEMEX	ECDSA	0803b460130a70d9d63e20e4974936e6b5756f...	Activa

Cuadro 4: Base de datos.

Para revocar, será necesario que el administrador proporcione un archivo `.pem` con la clave pública del sujeto que se quiere inhabilitar. El archivo se transforma primero de texto a bits, posteriormente se encuentra en la base de datos y se cambia su estado a 'Revocado'. A continuación se presenta el código para llevar a cabo las tareas:

```
def texto_a_bytes(nom_archivo):
  with open(nom_archivo,'r') as key:
    key_bytes_pk = key.read()
  key_bytes_pk = codecs.encode(key_bytes_pk, 'latin1')
  return key_bytes_pk


def revocar_firma(nom_file, data_frame, base_name):
  public_key_r = str(texto_a_bytes(nom_file))
  data_frame.loc[data_frame.iloc[:,6] == public_key_r,'Estado'] = 'Revocado'
  data_frame.to_csv(base_name, index=False)
  return data_frame
```

Con fines de prueba, se revoca el permiso de la llave otorgada a Teletón Inc. La base de datos se actualiza como se observa en el Cuadro 5.

ID Algoritmo	Emisor	No antes de	No despues de	Sujeto	Algoritmo CP	Clave Publica	Estado
ECDSA	Teleton	20/04/2022	30/04/2022	Teleton inc	ECDSA	7723cf55a0f4919a84adb1b13301e426bf161a...	Revocado
ECDSA	Uber	23/04/2022	30/04/2022	Uber inc	ECDSA	9e5c0dc2ee992173d97e6371028b969873f4e1...	Activa
ECDSA	CEMEX	18/04/2022	19/04/2022	CEMEX	ECDSA	0803b460130a70d9d63e20e4974936e6b5756f...	Activa

Cuadro 5: Cambio de estado a revocado.

Para verificar que la fecha de la clave sea válida, se utiliza la siguiente función:

```
def comprobar_caducidad(data_frame, base_name):

  fecha_actual = datetime.now()

  data_frame.iloc[:,2]= pd.to_datetime(data_frame.iloc[:,2])
  data_frame.iloc[:,3]= pd.to_datetime(data_frame.iloc[:,3])

  data_frame.loc[data_frame.iloc[:,2] > fecha_actual,'Estado'] = 'Invalido'
  data_frame.loc[data_frame.iloc[:,3] < fecha_actual,'Estado'] = 'Expirado'

  data_frame.to_csv(base_name, index=False)
```

Comprueba que la fecha actual sea superior a la fecha 'No antes de', en caso contrario, se asigna un estado de 'Inválido'. Además, se revisa que todavía no se supere la fecha de caducidad, de no ser así, se asigna un estado de 'Expirado'. La base de datos después de esta función, utilizada en la fecha 21/04/2022, se observa en el Cuadro 6.

ID Algoritmo	Emisor	No antes de	No despues de	Sujeto	Algoritmo CP	Clave Publica	Estado
ECDSA	Teleton	20/04/2022	30/04/2022	Teleton inc	ECDSA	7723cf55a0f4919a84adb1b13301e426bf161a...	Revocado
ECDSA	Uber	23/04/2022	30/04/2022	Uber inc	ECDSA	9e5c0dc2ee992173d97e6371028b969873f4e1...	Invalido
ECDSA	CEMEX	18/04/2022	19/04/2022	CEMEX	ECDSA	0803b460130a70d9d63e20e4974936e6b5756f...	Expirado

Cuadro 6: Revisar caducidad.

Aquí es importante mencionar que una vez que se generaron las claves, es necesario pensar en dónde guardará el usuario su clave privada. Esto es muy importante debido a que la llave privada es un elemento sumamente confidencial que nadie mas que el respectivo dueño puede tener acceso. Es por la razón anterior, que al generar las claves, es recomendable guardar el archivo `pem` correspondiente en una USB, y a partir de aquí, la responsabilidad de qué suceda con dicho dispositivo de almacenamiento queda en manos del usuario. Para facilitar esto, la aplicación web cuenta con un botón que permite guardar el archivo correspondiente en la ruta deseada. De igual forma, se debe mencionar que en la sección de anexos se incluyen imágenes que muestran a modo de ejemplo cómo sería el proceso de revocación de un certificado.

3.3. Generación de firma digital

La segunda parte fundamental del esquema de firma digital es el relacionado con la generación de la firma digital. Para su generación se utilizará el algoritmo ECDSA (Elliptic Curve Digital Signature

Algorithm) y la curva NIST con 256 bits, para lo cual se requiere la clave privada del usuario y el documento que se desea firmar. De acuerdo con lo planteado por la organización, la mayoría de los documentos que se manejan para en el Departamento de Compras son en formato `pdf`, por ello en esta solución se considera este formato como el tipo de archivo que el usuario desea firmar.

En la figura 5 se muestra el diagrama de flujo relacionando con el proceso de generación de la firma digital.

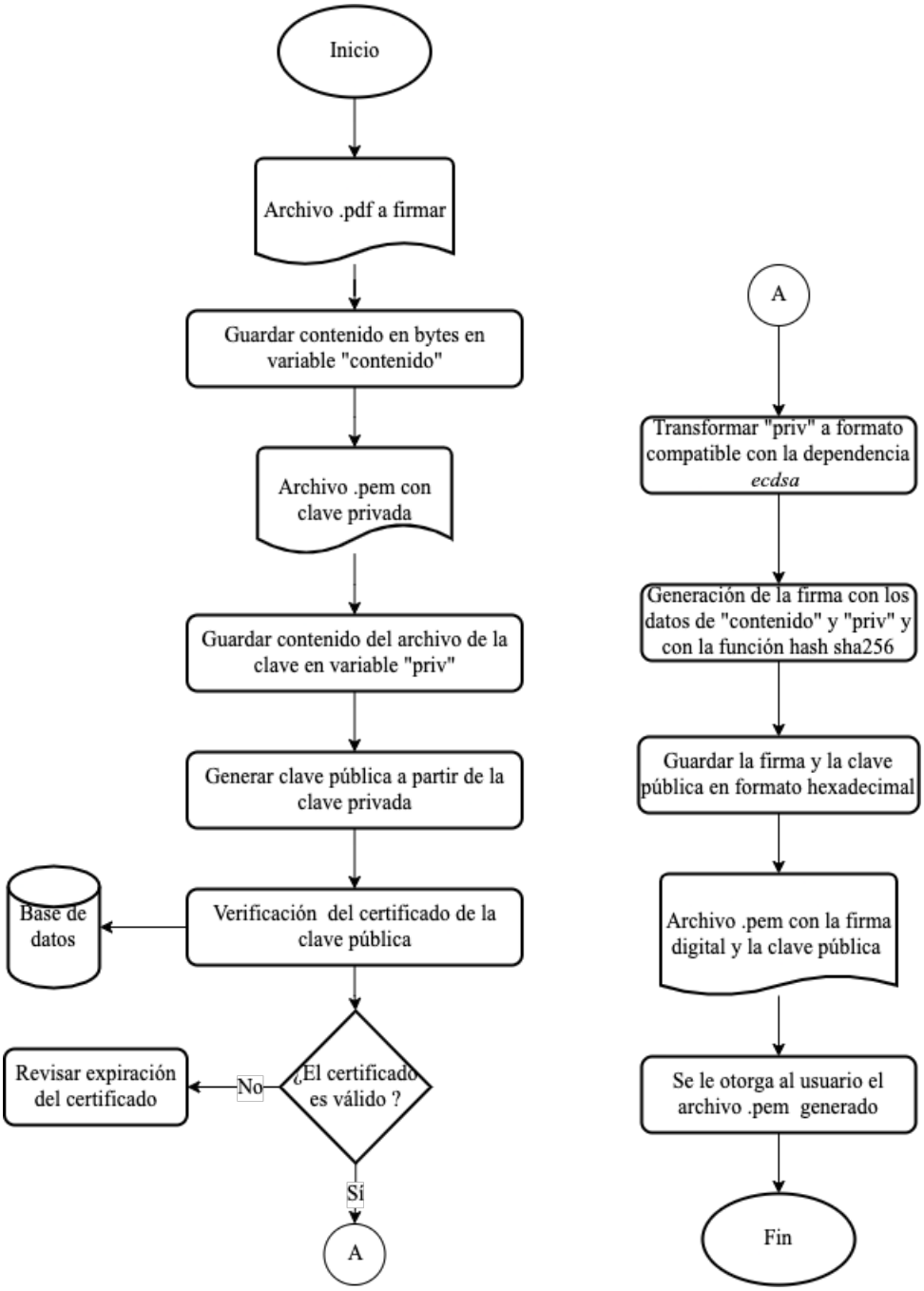

Figura 5: Diagrama de flujo de la generación de firma digital

Los primeros pasos del proceso requieren de lectura del archivo `pdf` a firmar, junto con la extracción de su contenido. Cabe aclarar que se busca que la firma sea sobre el documento, en lugar de que sea sobre el texto o el contenido del archivo. Dado esto, se hace uso de bytes para leer el archivo y que en estos se almacene toda la información que contenga el `pdf` (imágenes, texto, tablas, etc.). A continuación se

muestra el código para ello:

```
archivo = open("Nombre del documento.pdf", "rb")
contenido = archivo.read()
archivo.close()
```

Donde `rb` indica el modo de lectura de bytes del archivo, los cuales son almacenados en `contenido`. Posteriormente, se debe obtener su hash para que, junto con la clave privada, se genere la firma. En este punto se debe enfatizar en la función hash que se utiliza para la solución es SHA-256, pues esta cuenta con un alto nivel de seguridad que garantiza la integridad de la información de los archivos de lectura. Para la generación del hash, se utilizó la biblioteca **hashlib** de Python, la cual implementa varias funciones hash, entre las cuales se encuentra SHA-256 [13].

Para comprobar el funcionamiento de la librería, se realizó una prueba en donde utilizando la herramienta 'Checksum Online' se comparó el hash resultante de un archivo `pdf` de prueba a firmar. Dicho archivo se nombró 'Prueba-Hash' y contiene únicamente texto. Primero se calculó el hash con la herramienta 'Checksum Online', y en la figura 6 se muestra el resultado del hash obtenido.

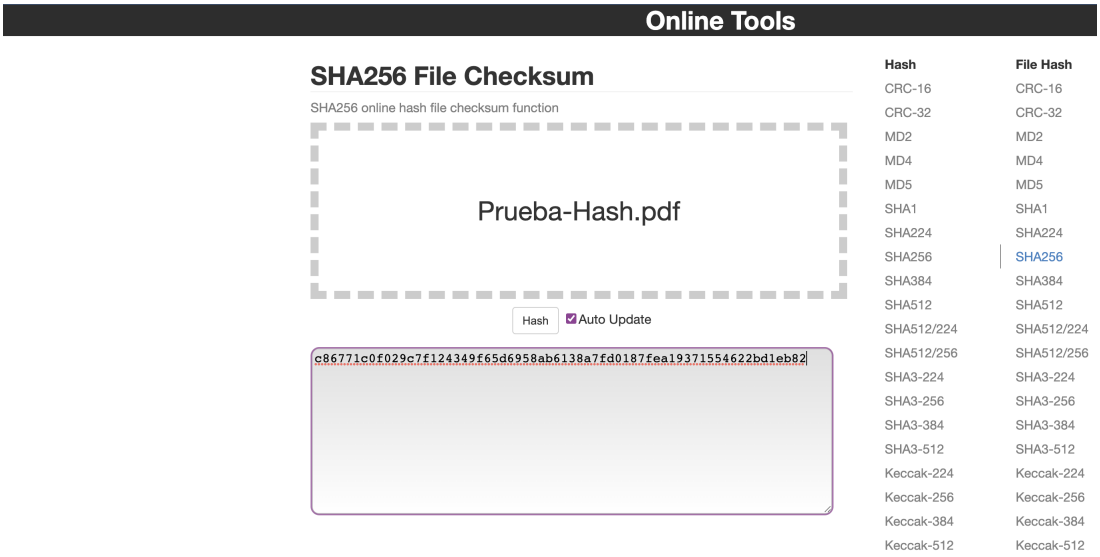

Figura 6: Hash del archivo 'Prueba-Hash.pdf' obtenido de la función SHA256 utilizando 'Checksum online'

Posteriormente, se utilizó el siguiente código para realizar esta prueba utilizando **hashlib** y en la figura 7 se muestra el hash resultante.

```
import hashlib
archivo = open('Prueba-Hash.pdf', 'rb')
contenido = archivo.read()
archivo.close()

h = hashlib.new('sha256')
h.update(contenido)
h.hexdigest()
```

```
Out[76]: 'c86771c0f029c7f124349f65d6958ab6138a7fd0187fea19371554622bd1eb82'
```

Figura 7: Hash del archivo 'Prueba-Hash.pdf' obtenido de la función SHA256 de la librería **hashlib** de Python'.

Como se puede observar, tanto en la figura 6 y 7 el hash obtenido del documento fue exactamente el mismo, comprobando de esta manera el correcto funcionamiento de la librería. Sin embargo, se decidió realizar una segunda prueba utilizando otro documento `pdf`, pero esta vez incluyendo imágenes, tablas y otras estructuras dentro del documento. A dicho archivo se le nombró 'Prueba-Hash2'. El proceso que se realizó para calcular el hash fue exactamente el mismo que en la prueba anterior, por que en las figuras 8 y 9 se muestran los hashes obtenidos.

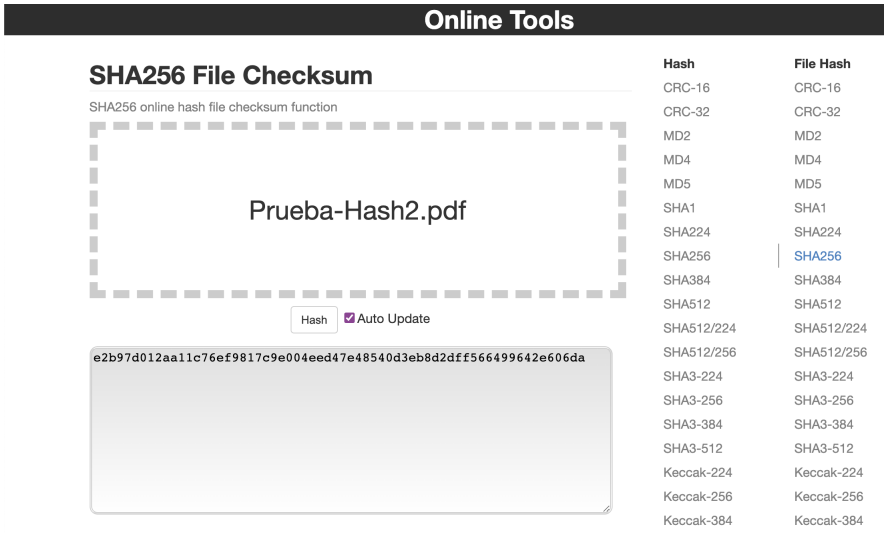

Figura 8: Hash del archivo 'Prueba-Hash2.pdf' obtenido de la función SHA256 utilizando 'Checksum online'

```
Out[77]: 'e2b97d012aa11c76ef9817c9e004eed47e48540d3eb8d2dff566499642e606da'
```

Figura 9: Hash del archivo 'Prueba-Hash2.pdf' obtenido de la función SHA256 de la librería **hashlib** de Python'.

Como se observa en estas dos últimas figuras, el hash obtenido fue idéntico en ambas aplicaciones, dejando aún más claro que la biblioteca de **hashlib** es adecuado para crear hashes a partir de archivos `pdf`.

Regresando al proceso de la generación de firma, esta última, como se mencionó en la sección 3.2, las adquiere el usuario en un archivo con formato `.pem`, debido a que se utiliza generalmente para guardar certificados y claves públicas o privadas. De este archivo, se obtiene la clave privada con el mismo formato `.pem`. Lo anterior se realiza mediante lo siguiente:

```
private = open('private.pem', 'r')
private = private.read()
```

```
private= private.encode()
```

La generación de firma será por medio de la librería **ecdsa**, por lo que se requiere que la clave privada obtenida sea un objeto de esta misma. Para ello, se emplea la función `SigningKey`, como se muestra:

```
private_key = SigningKey.from_pem(private)
```

Una vez siendo objeto de la librería, se puede utilizar el método `.sign()` para firmar, este recibe de parámetros el contenido del PDF (`contenido`) y el tipo de hash que se desea para el archivo (SHA256 en este caso), además se puede establecer un valor k para en el caso de vectores de prueba.

```
signature = private_key.sign(contenido,hashfunc=hashlib.sha256)
```

La firma `signature` tiene el formato automático en bytes, no obstante se optará por regresar al usuario la firma en hexadecimal en un archivo `.pem`. En el mismo archivo se regresará la clave pública del usuario. Este proceso se realiza de la siguiente manera:

```
public_key = private_key.get_verifying_key()
firma_hex = sig.hex()
public_hex = public_key.to_string().hex()


#Creación del archivo .pem
with open('firma_clavepublica.pem', 'w') as file:
    file.write("%s %s" % (firma_hex, public_hex))
```

De esta manera se da por terminado el proceso de generación de la firma digital a partir de la clave privada y el documento a firmar.

3.4. Verificación de firma digital

El último proceso fundamental dentro del esquema de firma digital, es el correspondiente a la verificación de la firma por parte de otro usuario. Dicho proceso se realiza utilizando la misma librería, **ecdsa**. En la figura 10 se muestra el diagrama de flujo correspondiente al proceso de verificación.

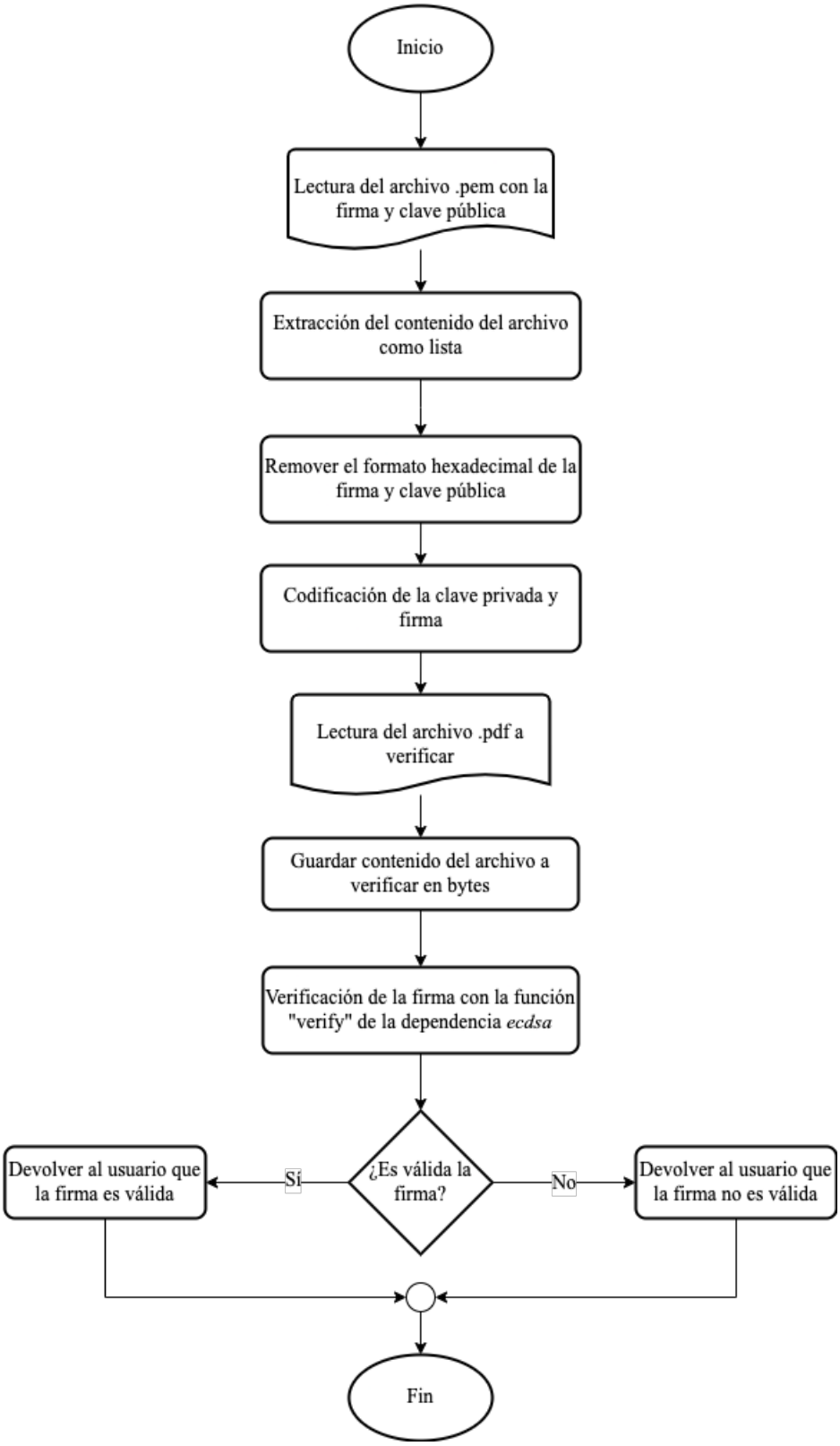

Figura 10: Diagrama de flujo de la generación de firma digital

A continuación se explica con detalle cada una de los pasos descritos en el diagrama. En esta etapa se necesita la clave pública, la firma y el mensaje que se envió, por lo que el primer paso es la lectura de dichos documentos. Para obtener la clave pública y la firma, que previamente se guardaron en un archivo `pem`, se realiza lo siguiente:

```
string = open('firma_clavepublica.pem', 'r')
string = string.read()
lista = string.split()
##  Clave pública de 'string' a 'bytes'
public_verify =lista[1]
public_verify = public_verify.encode()
##  Firma de 'string' a 'bytes'
firma_verify =lista[0]
firma_verify = firma_verify.encode()
```

Como se ve en el bloque de código anterior, se extrajo la información del archivo en forma de lista para que de esta manera de pudiera separar la parte correspondiente a la firma digital, de la clave pública.

Antes de realizar la verificación se debe regresar la firma y la clave pública a sus formatos originales para la función de verificación. Lo que se necesita es cancelar el formato hexadecimal que se les agregó anteriormente. Para esto se utiliza la función `unhexlify` de la librería `binascii`:

```
public_kv = unhexlify(public_verify)
firma_kv = unhexlify(firma_verify)
```

De manera similar se necesita otra modificación más a la clave pública ya que necesita ser un objeto de la librería. Para esto existe una función dentro de la librería que transforma la clave de un `string` a un objeto recibiendo también la curva utilizada.

```
public_kv= VerifyingKey.from_string(public_kv, curve=ecdsa.NIST256p)
```

Posteriormente, el usuario sube el `pdf` que se firmó y el algoritmo realizará lo mismo que se realizó anteriormente para la lectura del documento en la parte de la generación de la firma. Es decir lee el archivo y guarda su contenido en bytes para poder ser utilizado en la función correspondiente.

Finalmente, la verificación también consiste en una sola función que recibe el mensaje, la firma y la clave pública. Se incluye la función `hash` utilizada, la cual debe ser la misma utilizada para la firma. El resultado se presenta como un `True` o un `False`, el primero confirmando que la firma es válida y el segundo lo contrario.

```
public_kv.verify(firma_kv, contenido, hashfunc=hashlib.sha256)
```

De esta manera, termina el proceso de verificación de la firma, y por lo tanto, concluyendo con todo el esquema de firma digital planteado, por lo que en las siguientes secciones se dedicarán a evaluar el desempeño de la solución.

4. Resultados y discusión

En esta sección se presentan los resultados obtenidos a través de distintos vectores de prueba, los cuales se utilizaron para generar las firmas. Adicionalmente, se obtuvieron medidas estadísticas relacionadas al tiempo que se tarda la solución en generar las claves, firmar un documento, y verificar este último.

Sin embargo, antes de dar a conocer los resultados, se especificará el equipo en donde se realizaron las pruebas. El equipo que se utilizo fue una MacBook Pro de 16 pulgadas del año 2019. Dicho equipo cuenta con una memoria 16 GB 2667 MHz DDR4, y un procesador de 2.6 GHz Intel Core i7 de seis núcleos. Sobre el sistema operativo, el equipo cuenta con la versión 12.1 de macOS Monterey. Pasando hablar un poco sobre las características del entorno en donde se realizaron las pruebas, fue utilizando Anaconda, más concretamente Jupyter Notebook, en donde se trabajó con el lenguaje de programación Python 3.7. Sobre las versiones, se utilizó conda 4.10.1 y Pyhton 3. Finalmente, aunque los vectores de prueba se utilizaron en Jupyter Notebook, la plataforma donde se desarrolló la solución, fue en Visual Studio Code.

Dada ya la información anterior sobre el equipo, ahora si se pasará a la sección de vectores de prueba.

4.1. Vectores de prueba

Para poner en prueba el funcionamiento correcto de la solución, se utilizaron en total 10 vectores de prueba, los cuales se obtuvieron de [8] siendo este el RFC 6979 del IETF (Internet Engineering Task Force). En dicha referencia se dan diversos vectores de prueba para distintas curvas elípticas, sin embargo, la que se utilizó, fue la curva del NIST P-256 indicando en su nombre el número de bits con los que trabaja. De igual forma, en este reporte únicamente se mostrarán de ejemplo 3 vectores y los demás estarán disponibles en el manual del programador que se adjunta con este documento. Los 2 primeros vectores implementados fueron los relacionados con la función SHA256, pues es la que se implementó dentro del esquema como función hash predeterminada. Los parámetros que se utilizaron para la curva, la clave privada, y la clave pública se pueden encontrar en la sección de anexos (6).

En dichos vectores, q es un número primo que define el orden de la curva, x es la clave privada y U_x y U_y forman las coordenadas del punto generador, el cual se utiliza junto con la clave privada para generar la clave pública (U), pues recordemos que esta última se obtiene de $U = xG$. Esto parámetros fueron los que se usaron para realizar las firmas de los siguientes 2 mensajes: 'sample' y 'test'. Finalmente, otro factor que se necesitaba conocer, es el valor de k, que representa un número aleatorio del rango entre $[1, ..., q - 1]$. Los valores de k correspondientes a los mensajes a firmar, junto con los resultados que se debe de obtener de la firma, es decir los parámetros r y s, se incluyen de igual forma en los anexos (6).

Ya definidos estos los vectores que se utilizaron por parte del IETF, se pasó a su implementación en la solución. Para esto se tuvo que realizar directamente en el código ya que la página web genera claves aleatorias, y únicamente acepta documentos `pdf` para firmar, no mensajes como tal. Por dicha razón, se utilizó el siguiente código para la primer prueba con el mensaje 'sample':

```
x = 'C9AFA9D845BA75166B5C215767B1D6934E50C3DB36E89B127B8A622B120F6721'
message = b'sample'
k = 'A6E3C57DD01ABE90086538398355DD4C3B17AA873382B0F24D6129493D8AAD60'
private = ecdsa.SigningKey.from_string(string=unhexlify(x), curve=ecdsa.NIST256p)
public = private.get_verifying_key()
sig = private.sign(message, k=int(k, 16), hashfunc = hashlib.sha256)
public = private.get_verifying_key()
public.to_string().hex()
```

El resultado obtenido se puede visualizar en la figura 11, sin embargo, a diferencia del resultado

esperado en el RFC 6979, la biblioteca al realizar la firma, devuelve en una sola cadena de caracteres tanto el valor de r como de s, es decir que están unidos. Esto se menciona porque a simple vista pueden parecer resultados diferentes si se compara el RFC con lo arrojado, sin embargo, si se observa con atención, se puede notar que los valores coinciden, validando de esta forma el funcionamiento de la firma electrónica.

```
+---------------------------+
Algortimo: ECDSA/SHA256
Mensaje firmado: b'sample'
Firma obtenida: efd48b2aacb6a8fd1140dd9cd45e81d69d2c877b56aaf991c34d0ea84eaf3716f7cb1c942d657c41d436c7a1b6e29f65f3e90
0dbb9aff4064dc4ab2f843acda8
+---------------------------+
```

Figura 11: Resultado obtenido del vector de prueba SHA256con el mensaje 'sample' a firmar.

El mismo procedimiento se aplicó para el segundo vector con el mensaje 'test', por lo que el código fue el mismo, solamente cambiando los valores de k y el mensaje a firmar por los correspondientes. El resultado de este último se observa en la figura 12 en donde también se nota que el resultado coincide con el esperado en el RFC 6979.

```
+---------------------------+
Algortimo: ECDSA/SHA256
Mensaje firmado: b'test'
Firma obtenida: f1abb023518351cd71d881567b1ea663ed3efcf6c5132b354f28d3b0b7d38367019f4113742a2b14bd25926b49c649155f267
e60d3814b4c0cc84250e46f0083
+---------------------------+
```

Figura 12: Resultado obtenido del vector de prueba SHA256 con el mensaje 'test' a firmar.

Sin embargo, como se explicó en la introducción del proyecto, este esquema digital será implementado por el socio formador, por lo que en el posible caso de que se quiera hacer uso de cualquier otra función de hash, se realizó otra prueba de vector para la validación de la firma, pero utilizando SHA512. Este nuevo vector de prueba se encuentra dentro la sección de anexos. El código que se implementó fue el mismo que el mostrado anteriormente, con la diferencia de que la función hash que se utiliza es SHA512.

```
x = 'C9AFA9D845BA75166B5C215767B1D6934E50C3DB36E89B127B8A622B120F6721'
message = b'sample'
k = '5FA81C63109BADB88C1F367B47DA606DA28CAD69AA22C4FE6AD7DF73A7173AA5'
private = ecdsa.SigningKey.from_string(string=unhexlify(x), curve=ecdsa.NIST256p)
public = private.get_verifying_key()
sig = private.sign(message, k=int(k, 16), hashfunc = hashlib.sha512)
public = private.get_verifying_key()
public.to_string().hex()
```

En la figura 13 se visualiza que el resultado obtenido en la solución, coincide con el vector de prueba correspondiente.

```
+---------------------------+
Algortimo: ECDSA/SHA512
Mensaje firmado: b'sample'
Firma obtenida: 8496a60b5e9b47c825488827e0495b0e3fa109ec4568fd3f8d1097678eb97f002362ab1adbe2b8adf9cb9edab740ea6049c02
8114f2460f96554f61fae3302fe
+---------------------------+
```

Figura 13: Resultado obtenido del vector de prueba SHA512 con el mensaje 'sample' a firmar.

4.2. Análisis de tiempo

Una vez validado el funcionamiento correcto, se realizaron varias pruebas de la solución pero con el objetivo de poder medir el tiempo (en segundos) que tarda el algoritmo en ejecutarse. Esto tiene mucha relevancia, ya que además de ser una forma de evaluar qué tan eficiente es el proceso, es una manera de evaluar la seguridad. Estas pruebas se realizaron para cada una de las 3 etapas fundamentales en el esquema de firma digital, es decir para la generación de claves, la firma digital, y la verificación de ésta última. En total se realizaron 1000 pruebas y al finalizar, se obtuvo la media, desviación estándar, valor máximo y mínimo de los tiempos en cada una de las etapas de la solución. Dichos resultados se muestran en el cuadro 7. Sin embargo, antes de pasar a la discusión de dicho cuadro, se explicará cómo se obtuvieron tanto la media, la desviación estándar y los valores mínimos y máximos.

Cuadro 7: Resultados de los vectores de prueba de la solución

Algoritmo	Número de pruebas	Media (segundos)	Desviación estándar (segundos)	Tiempo mínimo	Tiempo máximo	Etapa
ECDSA/SHA256	1000	0.003602308232931727	0.0011291558617893913	0.00232	0.024728	Generación de claves
ECDSA/SHA256	1000	0.001890381763527054	0.0004327087557268301	0.001385	0.004801	Generación de firma
ECDSA/SHA256	1000	0.002554206619859579	0.0005343438138225427	0.001966	0.004144	Verificación de firma

4.2.1. Generación de claves

Como se mencionó anteriormente, en total se realizaron 1000 pruebas, por lo que para calcular los 1000 tiempos, se creó una base de datos con este número de registros. Esta base cuenta con el formato `csv` y lleva el nombre de 'datos-Prueba-Claves'. Para realizar todos los pares de claves de manera automática, se realizó una pequeña modificación en el código correspondiente, en donde se añadió un ciclo `for` para generar todo de una vez. Para la obtención del tiempo de cálculo, se utilizó la librería **datetime**, la cual cuenta con objetos del tipo 'datetime' que tienen el método `now()` con el que se obtiene la fecha y hora en ese preciso instante. Se utilizaron en total dos objetos 'datetime', uno para obtener el tiempo una línea antes de empezar con la generación de claves, y otro una línea después de haberlas creado. Finalmente, al final de cada iteración de guardó en una lista la resta de estos objetos 'datetime' para obtener el tiempo del proceso. El código quedó de la siguiente manera:

```
tiempos_claves = []
for i in range(1, len(data)+1):
    then = datetime.now().microsecond
    data, private, public = crear_claves(data, i)
    archivo_key(i, private, 'privateKey')
    archivo_key(i, public, 'publicKey')
    imp_certificado(i, data)
    now = datetime.now().microsecond
    tiempos_claves.append(microsegundoSegundo(now-then))
```

En el bloque de código se muestra la función `microsegundoSegundo()` que transforma los microsegundos en segundos. La lista `tiempo_claves` es la que almacena los valores en segundos de cada una de las iteraciones. Finalmente, para la obtención de la media y desviación estándar de estos tiempos, se

utilizó la librería **statistics**, más concretamente los módulos `mean()` y `pstdev()`. Por otro lado, para el cálculo de los valores máximos y mínimos, se utilizaron los métodos `max()` y `min()` respectivamente.

4.2.2. Generación de firma

Para la generación de firmas se siguió un proceso muy similar, ya que se modificó un poco el código implementado en la solución parar obtener en el tiempo total que el programa tarda en realizar la firma digital. Recordemos que dicho proceso conlleva la lectura de los archivos, la verificación de la vigencia del certificados, y la generación de la firma. Cabe mencionar que los archivos que contuvieron las claves privadas fueron los generados en el paso anterior, mientras que el documento `pdf` que se firmó en cada iteración, fue el ya antes utilizado, es decir 'Prueba-Hash'.

A continuación se presenta el bloque de código utilizado para esta acción:

```
tiempos_firma = []


for i in range(1, 1001):
    then = datetime.now().microsecond
    archivo = open("Prueba-Hash.pdf", "rb")
    contenido = archivo.read()
    archivo.close()
    archivo_pk = nombreArchivo(i)
    priv = open(archivo_pk, 'r')
    priv = priv.read() # esto lo guarda como string
    priv = priv.encode() # esto lo 'codifica' y lo regresa a pem
    private_key = SigningKey.from_pem(priv)
    sig = private_key.sign(contenido, hashfunc=hashlib.sha256)
    public_key = private_key.get_verifying_key()
    firma_hex = sig.hex()
    public_hex = public_key.to_string().hex()
    sig_file_name = 'firma_clavePublica' + str(i) + '.pem'
    with open(sig_file_name, 'w') as file:
        file.write("%s %s" % (firma_hex, public_hex))
    now = datetime.now().microsecond
    tiempos_firma.append(microsegundoSegundo(now-then))
```

Posteriormente a obtener la lista `tiempos_firma`, se pasó a calcular la media, la desviación estándar y los valores máximos y mínimos con **statistics**. Los resultados obtenidos se muestran en el cuadro 7.

4.2.3. Verificación de firma

Finalmente, se siguió utilizando la misma lógica para calcular la media y desviación estándar del proceso de verificación. Es decir se implementó un ciclo `for` con el que se realizó de manera automática la verificación de las firmas. Aquí es importante mencionar que el documento del que se verificaron todas las firmas, fue de igual forma el `pdf` 'Prueba-Hash'. Por otro lado, se utilizaron los archivos generados

cuando se realizó la firma, para obtener los parámetros de la firma y la clave pública del usuario. En el siguiente código de bloque se presenta cómo se realizó la implementación:

```
tiempos_verificacion = []


for i in range(1, 1001):
    then = datetime.now().microsecond
    archivo_firma = nombreArchivoFirma(i)
    string = open(archivo_firma, 'r')
    string = string.read()
    lista = string.split()
    public_verify
    public_verify =lista[1]
    public_verify = public_verify.encode()#Lo codifica para que este en bytes
    public_verify
    firma_verify =lista[0]
    firma_verify = firma_verify.encode()
    firma_verify
    archivo = open("Prueba-Hash.pdf", "rb")
    contenido = archivo.read()
    archivo.close()
    public_kv = unhexlify(public_verify)
    firma_kv = unhexlify(firma_verify)
    public_kv= VerifyingKey.from_string(public_kv, curve=ecdsa.NIST256p)


    public_kv.verify(firma_kv, contenido, hashfunc=hashlib.sha256)
    now = datetime.now().microsecond
    tiempos_verificacion.append(microsegundoSegundo(now-then))
```

Ya con la lista con los tiempos que se tardó la verificación de la firma, se obtuvo la media, desviación estándar, valor máximo y mínimo de estos tiempos. Los resultados se visualizan en el cuadro 7.

Adicionalmente, para cada una de las etapas del esquema, se generó un histograma de frecuencias del conjunto de tiempos de cada sección. Esto se realizó con el fin de observar el comportamiento del histograma e indicar que los datos no se ajustan a una distribución normal. Los histogramas correspondientes a la generación de claves, generación de firmas y verificación se muestran en las figuras 14 y 15.

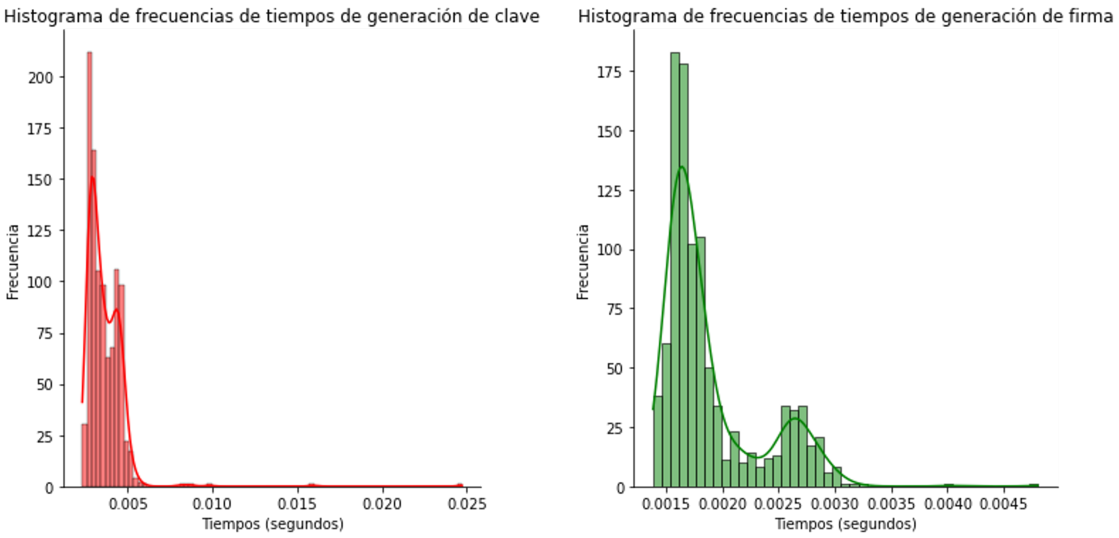

Figura 14: Histogramas de frecuencias del tiempo requerido para el proceso de generación de claves (a la izquierda) y para el proceso de generación de firma (a la derecha).

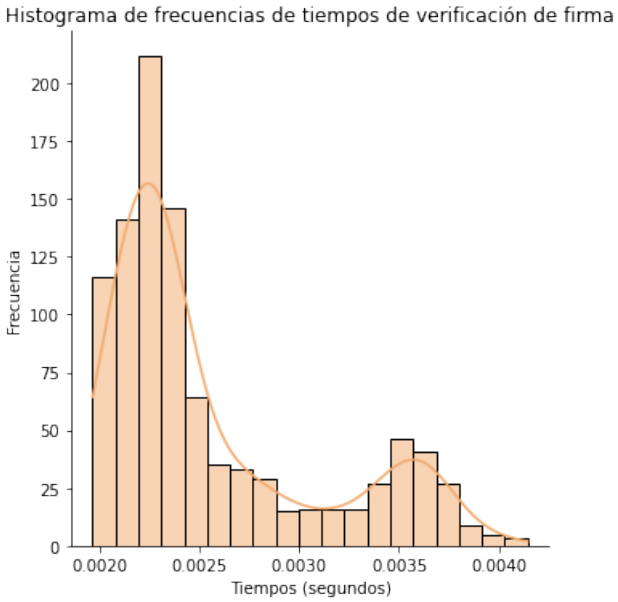

Figura 15: Histograma de frecuencias del tiempo requerido para el procesos de verificación de firma.

4.3. Discusión

Gracias a los resultados generados anteriormente y que se observan en el cuadro 7, se puede observar de forma general que la media de los tiempos es muy baja, ya que estos ni siquiera superar el segundo para completar los procesos. Esto habla de un buen funcionamiento del esquema de firma digital, ya que al tener tiempos muy bajos se pueden llegar a evitar los conocidos ataques de tiempo que analizan a profundidad el tiempo que le toma al algoritmo completar las etapas. Este tiempo promedio tendrá un impacto positivo, ya que uno de los objetivos del socio formador era agilizar los procesos de compras, y la solución implementada lo logra. Claro que los tiempos obtenidos no toman en cuenta cuánto tiempo se tarda el usuario en buscar los archivos en su propio equipo y subirlos a la página web, ya que se

automatizó el proceso, por lo que es algo que se debe tener en cuenta, al igual que los promedios pueden variar dependiendo del equipo que se utilice para firmar y verificar documentos. De igual forma, se debe destacar que el esquema logra obtener las firmas electrónicas de manera correcta, ya que como se indicó con anterioridad, se utilizaron vectores de prueba obtenidos de RFCs.

5. Conclusiones y Recomendaciones a Futuro

En conclusión, se logró generar una solución que se trata de un esquema de firma digital que implementa el algoritmo ECDSA. Tras la construcción y prueba del esquema, junto con la aplicación web, se le puede brindar a Fundación Teletón una solución la cual cumple con los objetivos planteados por ésta, como lo es la agilización de los procesos de compras, a través de la migración de un esquema de firma autógrafas a uno electrónico. De igual forma, gracias al esquema, se puede garantizar la autenticidad e integridad de todos los documentos que se firman de una forma eficiente, ya que se utilizan curvas elípticas, con las cuales se pueden generar claves más pequeñas, pero con una mayor seguridad. También, se logró ofrecer a la fundación una aplicación sencilla y fácil de utilizar por el usuario, para que de forma rápida pueda realizar las firmas y verificaciones requeridas para autorizar procesos, además de tener un control en el estado de los certificados.

Sobre las recomendaciones a futuro, se espera que primeramente dicho esquema de firma digital pueda implementarse de manera efectiva en los procesos del corporativo de la fundación, y después de un tiempo, expandirse a los centros de rehabilitación de la Fundación Teletón, para así poder optimizar de igual forma los procesos relacionados con atenciones médicas, y ofrecer a los clientes una mayor comodidad, pero sin perder seguridad y confiabilidad. De igual modo, se recuerda que aunque se hayan hecho varias pruebas de la aplicación, hay muchos aspectos a mejorar. Una de ellas sería la experiencia que se le ofrece al usuario, además de un posible guardado de la información a través de alguna plataforma en la nube como 'Amazon web Services' para que no se use almacenamiento interno del equipo de los usuarios. De igual forma, se recomienda darle mantenimiento a la base de datos para así asegurar que el nivel de seguridad del usuario sigue siendo elevado. Finalmente, se recomienda estar al pendiente de bug, o fallos de la aplicación, ya que aunque realizando las pruebas no apareció ninguno, siempre se puede haber pasado algo por alto.

6. Anexos

En esta sección se anexan algunos de los vectores de prueba utilizados en la sección de resultados (sección 4). Dichos vectores fueron obtenidos del RFC 6969 del Internet Engineering Task Force (IETF), los cuales se pueden encontrar en [8]. De igual forma se anexan imágenes del funcionamiento de la aplicación web construida.

6.1. Vectores de prueba

ECDSA, 256 Bits (Campo finito)

q = FFFFFFFF00000000FFFFFFFFFFFFFFFFBCE6FAADA7179E84F3B9CAC2FC632551

x = C9AFA9D845BA75166B5C215767B1D6934E50C3DB36E89B127B8A622B120F6721

U_x = 60FED4BA255A9D31C961EB74C6356D68C049B8923B61FA6CE669622E60F29FB6

U_y = 7903FE1008B8BC99A41AE9E95628BC64F2F1B20C2D7E9F5177A3C294D4462299

Con SHA-256, mensaje = 'sample':

k = A6E3C57DD01ABE90086538398355DD4C3B17AA873382B0F24D6129493D8AAD60

r = EFD48B2AACB6A8FD1140DD9CD45E81D69D2C877B56AAF991C34D0EA84EAF3716

s = F7CB1C942D657C41D436C7A1B6E29F65F3E900DBB9AFF4064DC4AB2F843ACDA8

Con SHA-256, mensaje = 'test':

k = D16B6AE827F17175E040871A1C7EC3500192C4C92677336EC2537ACAEE0008E0

r = F1ABB023518351CD71D881567B1EA663ED3EFCF6C5132B354F28D3B0B7D38367

s = 019F4113742A2B14BD25926B49C649155F267E60D3814B4C0CC84250E46F0083

Con SHA-512, mensaje = 'sample':

k = 5FA81C63109BADB88C1F367B47DA606DA28CAD69AA22C4FE6AD7DF73A7173AA5

r = 8496A60B5E9B47C825488827E0495B0E3FA109EC4568FD3F8D1097678EB97F00

s = 2362AB1ADBE2B8ADF9CB9EDAB740EA6049C028114F2460F96554F61FAE3302FE

6.2. Imágenes de aplicación web

Figura 16: Pestaña de la aplicación web en donde se sube el archivo `csv` para generar las claves y certificados.

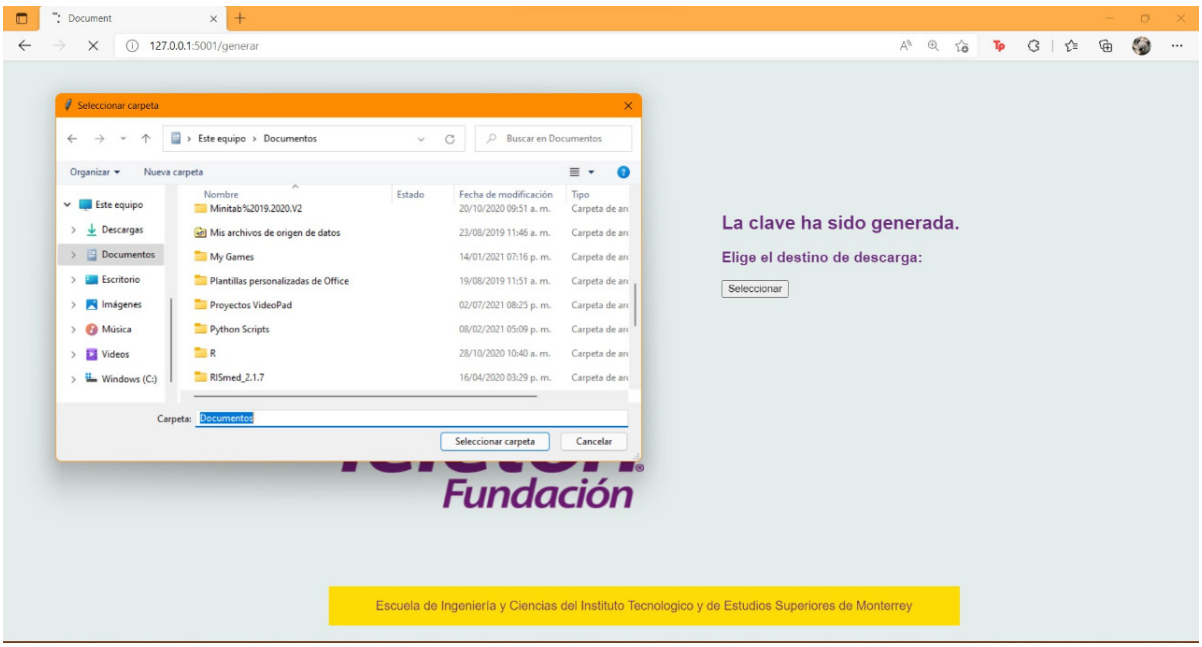

Figura 17: Pestaña de la aplicación web en donde se indica que la clave ha sido generada. Adicionalmente se le permite seleccionar al usuario la ruta en donde quiere descargar el documento generado.

Figura 18: Pestaña de la aplicación web en donde se carga un archivo `pem` indicando así el certificado a revocar

Figura 19: Pestaña de la aplicación web en donde se muestra el mensaje de que el certificado fue revocado exitosamente

Figura 20: Pestaña de la aplicación web en donde se carga el archivo `pem` y `pdf` para generar la firma digital

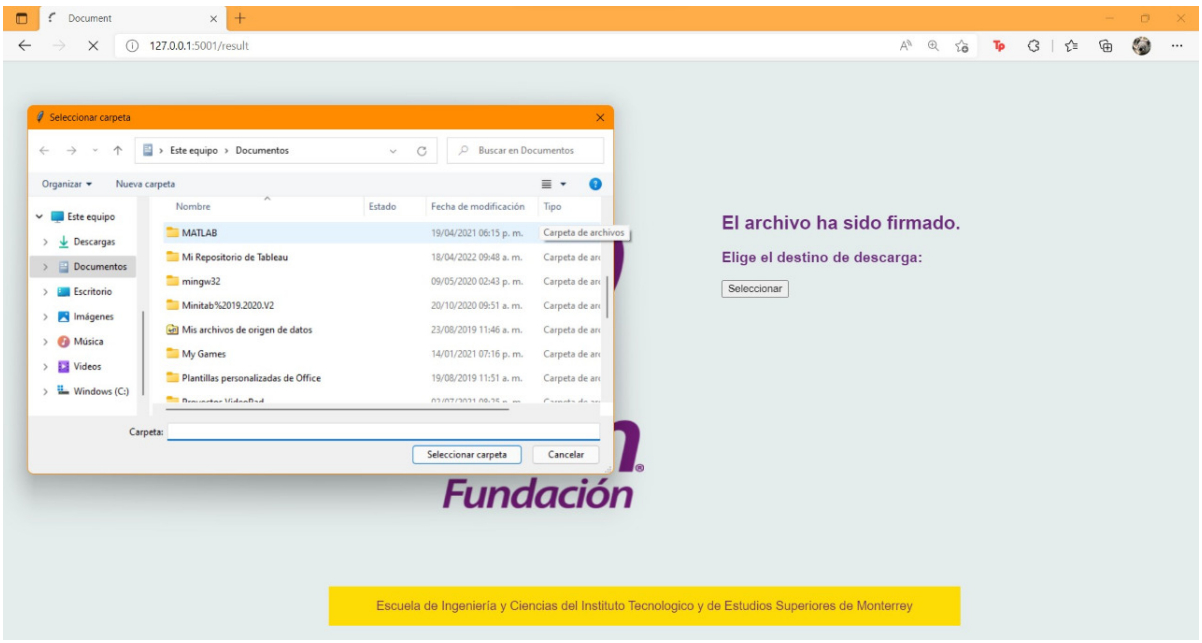

Figura 21: Pestaña de la aplicación web en donde se indica que el archivo fue firmado correctamente y se permite la opción de guardar el archivo con la firma en una ruta específica.

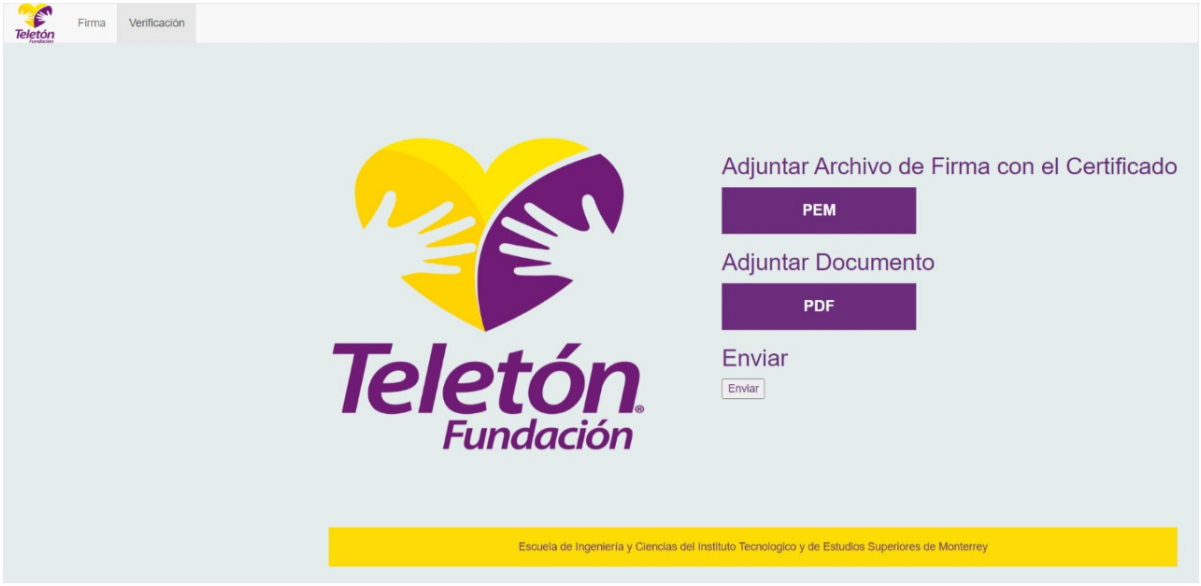

Figura 22: Pestaña de la aplicación web en donde se carga el archivo `pem` y `pdf` con la firma digital y documento a validar.

Figura 23: Pestaña de la aplicación web en donde se muestra que la firma fue verificada con éxito.

Referencias

[1] F. Teletón. Nosotros. [Online]. Available: https://teleton.org/nosotros/

[2] O. de las Naciones Unidas. Objetivo 9: Construir infraestructuras resilientes, promover la industrialización sostenible y fomentar la innovación. [Online]. Available: https://www.un.org/sustainabledevelopment/es/infrastructure/

[3] S. A. V. Alfred J. Menezes, Paul C. van Oorschot, *Handbook of Applied Cryptography*, 1996.

[4] S. Navkov, *Practical Cryptography for Developers*, 2018.

[5] M. I. Mihailescu and S. L. Nita, "Cryptography libraries in c/c++ 20," in *Pro Cryptography and Cryptanalysis with C++ 20.* Springer, 2021, pp. 151–186.

[6] T. O. P. Authors. (2022) ec. [Online]. Available: openssl.org/docs/man1.1.1/man1/ec.html

[7] Welcome to pyca/cryptography. [Online]. Available: https://github.com/pyca/cryptography

[8] T. Pornin, "Deterministic usage of the digital signature algorithm (dsa) and elliptic curve digital signature algorithm (ecdsa)," Internet Engineering Task Force, Tech. Rep., August 2013.

[9] I. T. Laboratory, "Digital signature standard (dss)," National Institute of Standards and Technology, Tech. Rep., July 2013.

[10] SECTIGO. (2021) Differences between rsa, dsa, and ecc encryption algorithms. [Online]. Available: https://sectigo.com/resource-library/rsa-vs-dsa-vs-ecc-encryption

[11] S. Limited. (2022) Rsa, dsa y ecc. [Online]. Available: https://www.ssl247.es/certificats-ssl/rsa-dsa-ecc

[12] U. G. S. Administration. (2022) What is pki (public key infrastructure) and why do i need it? [Online]. Available: https://www.fedidcard.gov/faq/what-pki-public-key-infrastructure-and-why-do-i-need-it

[13] P. S. Foundation. (2022) hashlib - secure hashes and message digests. [Online]. Available: https://docs.python.org/3/library/hashlib.html