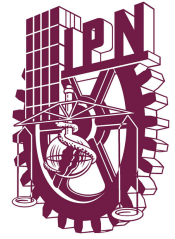




Instituto Politécnico Nacional
Escuela Superior de Cómputo



PROYECTO FINAL: PROTECTING LEGAL DOCUMENTS.

Cryptography

HERNANDEZ VELASCO HECTOR JAIR
MIRANDA MOJICA ERICK
TORRES OLIVERA KARLA PAOLA
VARELA AGUILAR LUIS PAVEL

PROFESORA: DIAZ SANTIAGO SANDRA

30/06/2020

Contents

1	Introducción	4
2	Servicios criptográficos	4
2.1	Autenticación de personas y datos	4
2.2	Confidencialidad	5
2.3	No repudio	5
2.4	Integridad de los datos	5
3	Primitivas criptográficas	5
3.1	Cifrador por bloque	5
3.1.1	Advanced Encryption Standard 128 bits (AES)	6
3.1.2	Modo de operación EAX	6
3.1.3	Secret Sharing Scheme	8
3.2	RSA	8
3.2.1	PKCS#1 OAEP (RSA)	8
3.3	Certificado digital	9
3.3.1	Estándar x.509	9
3.3.2	PyOpenSSL y X.509	10
3.4	Firma digital	10
3.4.1	Función Hash SHA-256	10
3.4.2	PKCS#1 v1.5 (RSA)	10
4	Arquitectura del sistema	11
4.1	Generar los certificados digitales	11
4.2	Generar casos	11
4.3	Modificar documentos	12
4.4	Descargar documentos	13
5	Partes importantes	14
5.1	Generar certificado y claves privadas	14
5.2	Agregar casos	15
5.3	Ver casos	17
5.4	Modificar casos	18
5.4.1	Caso no cifrado	19
5.4.2	Caso cifrado	20
6	Manual de usuario	21
6.1	Hardware, software, requerimientos de instalación e instrucciones	21
6.2	Instalación de Python	21
6.3	Instalación db browser for sqlite	22
6.4	Instalación de librerías para Python	22
6.4.1	Pycryptodome	22
6.4.2	pyOpenSSL	22
6.4.3	Django	22
6.5	Pantallas del sistema	22
6.5.1	Agregar un abogado al sistema	22
6.5.2	Agregar un caso	23
6.5.3	Ingresar a la plataforma	25
6.5.4	Descargar casos	26
6.5.5	Modificar caso	27

7 Conclusiones**30****Referencias****31**

1 Introducción

A lo largo del curso pudimos ver que existen diversos algoritmos criptográficos que son de gran utilidad para resolver ciertos problemas, sin embargo, es importante conocer las características de cada uno para utilizar el adecuado.

Uno de los propósitos del proyecto final es poner en práctica todo lo aprendido, así como justificar cada una de las decisiones que tomamos al momento de diseñar la solución final.

Criptología es el estudio de códigos o el arte de escribir y resolverlos. Esta se divide en dos grandes áreas: criptografía y criptoanálisis. Nosotros únicamente haremos uso de la primera para la solución de la problemática planteada en nuestro proyecto.

Dentro de la criptografía encontramos: criptografía de clave secreta y criptografía de clave pública. En la segunda se utiliza mucho más la aritmética modular y se trabaja con números muy grandes, por lo que el cifrado puede llegar a ser más lento. En consecuencia, no se suele utilizar para cifrar una gran cantidad de información, para esto están los primeros (criptografía de clave secreta).

En este trabajo se podrán encontrar los servicios criptográficos que identificamos para la elaboración de la solución, así como las primitivas criptográficas que decidimos utilizar para cubrirlos. Adicional a esto, se explicarán de manera breve las partes más importantes del código fuente, la arquitectura del sistema y, finalmente un manual de usuario para que cualquiera desde su computadora pueda hacer uso de este.

2 Servicios criptográficos

Los algoritmos criptográficos pueden proporcionar diferentes y determinados servicios, y deben combinarse para cumplir varios objetivos de seguridad de la información. El mal uso de los servicios criptográficos puede conducir a fallas de protocolo.

A continuación se explicarán de manera breve los servicios criptográficos identificados para una posible solución a la problemática presentada en nuestro proyecto.

2.1 Autenticación de personas y datos

La autenticación es un servicio relacionado con la identificación de un mensaje o una entidad. Generalmente se subdivide en dos clases principales: autenticación de origen de datos y autenticación de entidad. El primero se refiere a la validación de una propiedad reclamada de un mensaje, como origen, fecha de origen, contenido de datos, etc. Mientras que el segundo asegura una identidad reclamada de un remitente del mensaje, es decir, que la identidad de la parte que inicia una comunicación no es falsa [1].

La diferencia entre la autenticación de la entidad y la autenticación del origen de datos es que la primera adquiere una comunicación real para mostrar la posesión de un secreto asociado con la parte genuina reclamada, mientras que la segunda no necesita adquirir una comunicación real sino una evidencia para mostrar la fuente (original) de los datos especificados creados en algún momento en el pasado [1].

Considerando la breve explicación dada, estos servicios criptográficos son requeridos para la solución debido a que la información sensible debe ser confidencial, es decir, solo los asociados autorizados pueden leerla o modificarla. Por lo que, es necesario verificar que realmente son quién dicen ser. Por otro lado, cada vez que se suba alguna modificación en los documentos debemos verificar que la fuente origen tenga permiso de realizar esta tarea.

2.2 Confidencialidad

La confidencialidad (o el secreto, o la privacidad) es un servicio utilizado para mantener el contenido de la información de todos menos aquellos autorizados para tenerla. Los métodos para proporcionarla van desde protecciones físicas hasta algoritmos matemáticos que hacen que los datos sean ininteligibles [1].

Tomando en cuenta lo mencionado anteriormente, el servicio de confidencialidad nos servirá para garantizar que únicamente los abogados asociados a un caso puedan tener acceso a la información sensible sobre este. Para esto, se cifrará la información haciéndola no entendible para todos aquellos que no estén autorizados. Adicional a esto, se implementará **Secret Sharing Scheme** con la finalidad de que estén presentes al menos tres abogados relacionados al mismo caso para eliminar la protección del documento y luego proceder a visualizar o modificar la información.

2.3 No repudio

El no repudio es el proceso de probar que un usuario realizó una acción, como enviar un mensaje de correo electrónico. Es un servicio que evita que una entidad niegue compromisos o acciones anteriores [1].

Un requisito de la problemática a resolver es que debe existir una manera de saber quiénes fueron los asociados que modificaron la información sensible por última vez, de tal manera que no puedan negarla. Por lo que, el no repudio es el servicio criptográfico que nos ayudará a cubrir este punto.

2.4 Integridad de los datos

La integridad de los datos es un servicio que nos ayuda a detectar alteraciones no autorizadas en los datos. Dichas alteraciones incluyen crear, escribir, eliminar o cambiar los mensajes transmitidos [1].

Es importante tener en cuenta que el cifrado proporciona protección solo contra ataques pasivos, mientras que un atacante activo puede reproducir el mensaje o construir una representación fraudulenta donde se necesita la seguridad de la integridad de los datos [1].

Con la breve explicación dada anteriormente, pudimos concluir que necesitamos de este servicio para que cada vez que un asociado autorizado acceda a información confidencial, podamos verificar que esta no haya sido alterada por entidades no autorizadas, cubriendo así otro punto de la problemática planteada.

3 Primitivas criptográficas

Las primitivas criptográficas son algoritmos criptográficos que se utilizan como base para construir protocolos criptográficos. A continuación se explicarán brevemente las que se utilizaron para la elaboración del proyecto, indicando los servicios que ofrece cada una.

3.1 Cifrador por bloque

Como su nombre lo indica, los cifradores por bloque operan en “bloques” de datos. Estos son típicamente de 64 o 128 bits de longitud, los cuales se transforman en bloques del mismo tamaño bajo la acción de una clave secreta [2].

Las operaciones básicas de sustitución y permutación son particularmente importantes para este tipo de cifrado. La mayoría, si no todos, los cifradores por bloque contendrán

alguna combinación de ambas, aunque la forma exacta de la sustitución y la permutación puede variar mucho [2].

3.1.1 Advanced Encryption Standard 128 bits (AES)

El cifrado AES es casi idéntico al cifrado de bloque Rijndael. El tamaño del bloque y la clave de Rijndael varían entre 128, 192 y 256 bits. Sin embargo, el estándar AES solo requiere un tamaño de bloque de 128 bits. El número de rondas internas del cifrado depende de la longitud de la clave de acuerdo con la Tabla 1.

Tamaño de la llave	# rounds
128 bit	10
192 bit	12
256 bit	14

Tabla 1: Longitudes de clave y número de rondas para AES

AES consta de las llamadas capas. Cada capa manipula los 128 bits de la ruta de datos. A esta también se le conoce como el estado del algoritmo. Solo hay tres tipos diferentes de capas, las cuales se repiten en cada una de las rondas a excepción de la primera. En AES, la aritmética de campo de Galois se usa en la mayoría de las capas, especialmente durante las *S - BOX* y *MixColumn* [3].

Para la elaboración de nuestro proyecto decidimos hacer uso de AES con un tamaño de llave de 128 bits. Este nos servirá para cifrar los documentos con información sensible, debido a que como se recordará, los cifradores por bloque se utilizan para cifrar una gran cantidad de información. Por lo que, el servicio criptográfico que proporciona es **confidencialidad**.

3.1.2 Modo de operación EAX

EAX es un modo de operación para cifrados por bloques criptográficos. Es un algoritmo AEAD (Authenticated Encryption with Associated Data) para proporcionar tanto autenticación y privacidad al mensaje (cifrado autenticado) con un esquema de dos pasos, uno para lograr la privacidad y otro para la autenticidad de cada bloque. EAX fue presentado el 3 de octubre de 2003 a la necesidad del NIST para reemplazar CCM como modo de operación AEAD estándar, ya que el modo CCM carece de algunos atributos deseables de EAX y es más complejo [4].

Algunas de las características del modo EAX es que hace uso de un valor de nonce único, además de que no tiene ninguna restricción en cuanto a la primitiva de cifrado de bloque que se use, ni en el tamaño del bloque y admite mensajes de longitud arbitraria.

El algoritmo EAX. Definir un cifrador de bloque $E : Key \times (0, 1)^n \rightarrow (0, 1)^n$ y una longitud de tag $\tau \in [0...n]$.

Por lo general, los parámetros se acordarían de manera autenticada entre el remitente y el receptor, o se fijarían de forma permanente para alguna aplicación en particular. Dados estos parámetros, EAX proporciona un valor de nonce EAX basado en el esquema AEAD $[E, \tau]$ cuyo algoritmo de cifrado tiene firma $Key \times Nonce \times Header \times Plaintext \rightarrow Ciphertext \cap INVALID$ donde Nonce, Header, Plaintext, Ciphertext son todos $(0, 1)^*$ [4].

El algoritmo EAX se especifica en la figura 1 mientras que en la figura 2 se ilustra el cifrado usando EAX.

Algorithm EAX.Encrypt $_{K}^{N,H}(M)$	Algorithm EAX.Decrypt $_{K}^{N,H}(CT)$
10 $\mathcal{N} \leftarrow \text{OMAC}_K^0(N)$	20 if $ CT < \tau$ then return INVALID
11 $\mathcal{H} \leftarrow \text{OMAC}_K^1(H)$	21 Let $C \parallel T \leftarrow CT$ where $ T = \tau$
12 $C \leftarrow \text{CTR}_K^N(M)$	22 $\mathcal{N} \leftarrow \text{OMAC}_K^0(N)$
13 $\mathcal{C} \leftarrow \text{OMAC}_K^2(C)$	23 $\mathcal{H} \leftarrow \text{OMAC}_K^1(H)$
14 $\text{Tag} \leftarrow \mathcal{N} \oplus \mathcal{C} \oplus \mathcal{H}$	24 $\mathcal{C} \leftarrow \text{OMAC}_K^2(C)$
15 $T \leftarrow \text{Tag}$ [first τ bits]	25 $\text{Tag}' \leftarrow \mathcal{N} \oplus \mathcal{C} \oplus \mathcal{H}$
16 return $CT \leftarrow C \parallel T$	26 $T' \leftarrow \text{Tag}'$ [first τ bits]
	27 if $T \neq T'$ then return INVALID
	28 $M \leftarrow \text{CTR}_K^N(C)$
	29 return M

Figura 1: Cifrado y descifrado en modo EAX. El texto sin formato es M , el texto cifrado es CT , la clave es K , el nonce es N y el encabezado es H . El modo depende de un cifrado de bloque E (que CTR y OMAC usan implícitamente) y una longitud de etiqueta τ

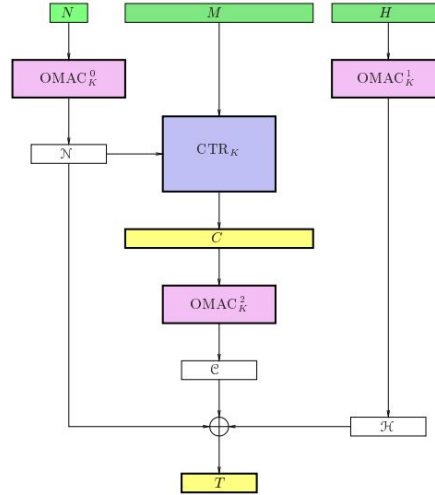


Figura 2: Cifrado bajo EAX. El mensaje es M , la clave es K y el encabezado es H . El texto cifrado es $CT = C \parallel T$.

Atributos de EAX. Al ser un esquema de dos pasos, el modo EAX es más lento que un esquema de un paso basado en las mismas primitivas. El modo EAX tiene varios atributos deseables, en particular:

1. Seguridad demostrable (dependiente de la seguridad del cifrado primitivo subyacente).
2. La expansión del mensaje es mínima y se limita a la sobrecarga de la longitud de la etiqueta.
3. El uso del modo CTR significa que el cifrador solo debe implementarse para el cifrado, para simplificar la implementación de algunos cifrados (atributo especialmente deseable para la implementación de hardware).
4. El algoritmo es "en línea", lo que significa que puede procesar un flujo de datos, usando memoria constante, sin conocer de antemano la longitud total de los datos [4].

Conociendo las características del modo EAX, lo elegimos porque permite que el receptor detecte cualquier modificación no autorizada (de manera similar, podríamos haber usado otro modo de cifrado autenticados como CCM) pero como vimos, CCM fue remplazado por EAX y además, la documentación de la librería criptográfica que estamos utilizando nos recomienda utilizar EAX como modo de cifrado autenticado.

3.1.3 Secret Sharing Scheme

Un esquema para compartir secretos es un protocolo criptográfico en el que, como su nombre indica, se divide un determinado secreto en fragmentos que se reparten entre los participantes del esquema [5].

Se divide un secreto S en n partes S_1, \dots, S_n tal que:

1. El conocimiento de m o más piezas S_i hace a S eficiente.
2. El conocimiento de $m - 1$ o menos S_i piezas deja a S completamente indeterminado. Esta combinación se denomina umbral (m, n) .

El esquema de Secreto compartido perfecto fue inventado de forma independiente por Adi Shamir en 1979 y George Blakley [5].

Por ejemplo, es posible que se desee otorgar a 16 personas la posibilidad de acceder a un sistema con un código de acceso, a condición de que al menos 3 de ellos estén presentes al mismo tiempo. A medida que se unen a sus acciones, se revela el código de acceso. En ese caso, $n = 16$ y $m = 3$.

En el esquema de intercambio secreto de Shamir, las n acciones se crean definiendo primero un polinomio de grado $m - 1$:

$$q(x) = a_0 + a_1x + a_2x^2 + \dots + a_{m-1}x^{m-1}$$

El coeficiente a_0 se fija con el valor secreto. Los coeficientes $a_1 \dots a_{m-1}$ son aleatorios y se descartan tan pronto como se crean los recursos compartidos.

Cada llave compartida es un par (x_i, y_i) , donde x_i es un número arbitrario pero único asignado al destinatario del recurso compartido y $y_i = q(x_i)$.

3.2 RSA

El esquema criptográfico RSA, a veces denominado algoritmo Rivest-Shamir-Adleman, es actualmente el esquema criptográfico asimétrico más utilizado, a pesar de que las curvas elípticas y los esquemas de logaritmo discreto están ganando terreno. RSA fue patentado en los EE.UU. (Pero no en el resto del mundo) hasta el año 2000 [3].

El criptosistema RSA permite no sólo garantizar la **confidencialidad** de la comunicación entre dos partes, cifrando en origen el mensaje que se va a transmitir por un canal inseguro y descifrándolo en recepción como se hace en el proyecto al cifrar las sub-llaves de AES, sino que también proporciona otros servicios, como son la **autenticación**, la **integridad** o el **no-repudio** (mediante la firma digital).

3.2.1 PKCS#1 OAEP (RSA)

PKCS#1 OAEP es un cifrado asimétrico basado en RSA y el relleno OAEP. Se describe en **RFC8017** donde se llama RSAES-OAEP.

RSAES-OAEP combina las primitivas RSAEP y RSADP con el método de codificación EME-OAEP. Este último se basa en el esquema de cifrado asimétrico óptimo de Bellare y Rogaway (OAEP) [6].

RSAES-OAEP es semánticamente seguro contra ataques adaptativos de texto cifrado elegido. Esta garantía es demostrable en el sentido de que la dificultad de romper RSAES-OAEP puede estar directamente relacionada con la dificultad de invertir la función RSA, siempre que la función de generación de máscara se vea como una caja negra [6].

Advertencia:

PKCS# 1 OAEP no garantiza la autenticidad del mensaje que descifra. Como la clave pública no es secreta, todos podrían haber creado el mensaje cifrado. El cifrado asimétrico generalmente se combina con una firma digital [6].

3.3 Certificado digital

En general, un certificado digital no es solo una firma en un solo par (Alice, la clave pública de Alice); uno puede colocar todo tipo de otra información, posiblemente propia de la aplicación, en el certificado. Se trata de un documento digital que permite identificar a las personas en Internet. Contiene nuestros datos identificativos que están autenticados por un organismo oficial [7].

La principal ventaja del certificado digital es que nos ahorrará tiempo y dinero al permitir realizar trámites administrativos a través de Internet, a cualquier hora y en cualquier lugar. Además, vamos a necesitarlo si queremos cumplir con nuestras obligaciones tributarias ya que la mayoría de los modelos debemos presentarlos electrónicamente. El emitir un certificado digital es delicado ya que como se explico, sirve para identificar a una persona. Dependiendo del tipo de certificado, si es para persona física o moral (representantes de sociedades), entonces se requerirán diferentes datos para generar el certificado.

Como en cada sistema, es importante identificar si realmente una persona es quien dice ser y el nuestro no es la excepción. Para ello, investigamos los tipos de formatos de certificados. Por lo mismo que generar certificados es delicado y falsificarlos es ilegal, no se tiene tanta información sobre como generarlos. A partir de esta investigación, elegimos el estándar x.509 para la generación de certificados digitales para el proyecto.

3.3.1 Estándar x.509

X.509 es un estándar ampliamente utilizado para definir certificados digitales. El X.509 es un estándar de la ITU (Unión Internacional de Telecomunicaciones) para PKI (Infraestructura de clave pública) que define formatos específicos para los certificados de clave pública (PKC) y el algoritmo de validación de ruta de certificación [8].

El protocolo X.509 fue publicado como una recomendación de la ITU llamada ITU-T X.509 y ISO/IEC/ITU 9594-8. Ambas publicaciones proveen una descripción casi idéntica para la definición de llaves publicas y los atributos que deben poseer certificados digitales [8].

1. X.509 (Versión 1): Expuesto en 1988 como parte de la recomendación ITU X.500 para servicios de directorio y se base en un sistema jerárquico donde los principales entes son autoridades certificadoras que emiten certificados digitales de identificación.
2. X.509 (Versión 2): Aparece en 1993 cuando X.509 fue revisado. Se mejoro la versión de formato de datos, incluyendo campos adicionales para proveer soporte y acceder a controles del directorio.
3. X.509 (Versión 2): Define el formato para extensiones de los certificados, utilizados para almacenar información adicional referente al propietario y define el uso del mismo. Incluye compatibilidad con otras topologías como puentes y permite la opción de usarlo punto a punto (peer-to-peer) similarmente al método utilizado por OpenPGP.

4. X.509: Se refiere a la ultima versión publicada en el estándar a menos que se indique otra cosa. Actualmente el nombre X.509 se utiliza ampliamente para referirse al IETF's PKI (Public Key Infrastructure) [8].

3.3.2 PyOpenSSL y X.509

Para generar los certificados digitales, se utilizó la librería pyOpenSSL la cual nos proporciona las funcionalidades de OpenSSL. Este último es un juego de herramientas robusto, de calidad comercial y con todas las funciones para los protocolos de Seguridad de la capa de transporte (TLS) y la Capa de sockets seguros (SSL). También es una biblioteca de criptografía de uso general para cifrar los datos enviados a otra computadora dentro de una red y a su vez descifrarlos adecuadamente por el receptor, evitando así, el acceso a la información por intrusos [9].

pyOpenSSL fue creado originalmente por Martin Sjögren porque el soporte SSL en la biblioteca estándar de Python 2.1 (la versión contemporánea de Python cuando se inició el proyecto pyOpenSSL) fue muy limitado al igual que otros contenedores de OpenSSL para Python.

El protocolo SSL (Secure Sockets Layer) permite un intercambio de información entre el servidor web y el navegador del usuario (cliente) de forma segura. El objetivo de esta tecnología es permitir que sólo el usuario autorizado pueda efectuar las operaciones. Mientras el protocolo HTTP utiliza un formato de dirección que empieza por http://, las direcciones en el protocolo SSL empiezan por https:// [9].

3.4 Firma digital

Las firmas digitales son una de las herramientas criptográficas más importantes que hoy en día se utilizan ampliamente. Las aplicaciones para firmas digitales van desde certificados digitales para comercio electrónico seguro hasta firma legal de contratos para actualizaciones de software seguras. Junto con el establecimiento de claves sobre canales inseguros, forman la instancia más importante para la criptografía de clave pública [3].

Las firmas digitales comparten algunas funcionalidades con las firmas manuscritas. En particular, proporcionan un método para asegurar que un mensaje sea auténtico para un usuario, es decir, de hecho se origina en la persona que afirma haber generado el mensaje. Por lo que, la firma nos servirá para proporcionar los siguientes servicios criptográficos: **autenticación de origen de datos**, la **integridad** y **no-repudio** [3].

3.4.1 Función Hash SHA-256

Las funciones hash más ampliamente implementadas son MD-5, RIPEMD-160, SHA-1 y SHA-2, todas estas se basan en la construcción Merkle-Damgard utilizando una función de compresión fija (es decir, sin clave) f [10].

En SHA-256 la función f tiene 64 rondas de pasos individuales y una longitud de bits de salida de 256 bits. Procesa el mensaje de entrada bloque por bloque, donde cada aplicación de la función $f_k(m)$ es una función de 64 iteraciones de un solo paso. La función de paso utiliza funciones f y g ligeramente diferentes a las utilizadas en MD-4 y SHA-1 [10].

Las funciones hash proveen **integridad** y la relación es en un solo sentido, es decir, podemos ir pero no regresar o dicho en otras palabras no podemos encontrar H^{-1} .

3.4.2 PKCS#1 v1.5 (RSA)

Para generar las firmas digitales en el proyecto se hizo uso del esquema PKCS#1 v1.5 basado en RSA. Se llama más formalmente RSASSA-PKCS1-v1_5 en la Sección 8.2 de RFC8017.

RSASSA-PKCS1-v1_5 combina las primitivas RSASP1 y RSAVP1 con el método de codificación EMSA-PKCS1-v1_5. La longitud de los mensajes en los que puede operar no está restringida o está restringida por un número muy grande, dependiendo de la función hash subyacente al método EMSA-PKCS1-v1_5 [6].

4 Arquitectura del sistema

4.1 Generar los certificados digitales

Como se puede observar en la Figura 3 la generación de los certificados digitales y las llaves privadas se hace a través de un archivo json el cual contiene los nombres y correos de cada abogado para posteriormente enviarlos mediante el uso del correo que se creó en gmail.

Para esto, primero se lee el archivo json y se manda a llamar a la función para generar llaves, certificados y envío de correos tantas veces como abogados aparezcan en el archivo json, la función `genera_envia_claves_RSA()` donde se generan las llaves RSA (pública y privada).

Al haber generado el par de llaves RSA de cada abogado, se genera un certificado digital utilizando el estándar X.509 para cada uno de los abogados. Cada certificado contiene información de acuerdo al abogado como su país, ciudad, nombre de la empresa, correo de la empresa y la llave publica del abogado. Para firmar el certificado, utilizamos la función hash sha256 pasándole como texto la llave privada de la empresa para asegurar que nuestro documento esta firmado por la empresa. Al final tenemos dos archivos resultantes, el certificado digital y la llave privada RSA, ambas por cada abogado y se adjuntan en un correo que es enviado a cada uno de los abogados.

Diagrama 1 : Generar los certificados

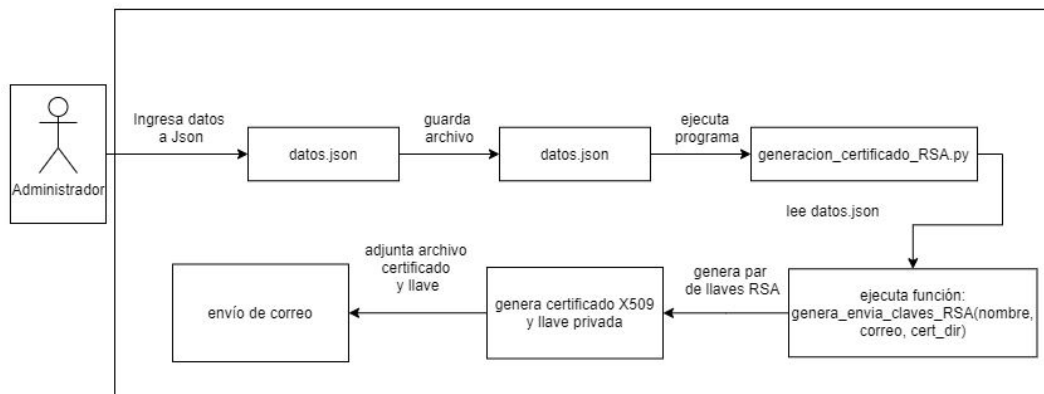


Figura 3: Generar certificados

4.2 Generar casos

En la Figura 4 se muestra el proceso para la generación de un caso, para esto se requiere que el abogado líder a cargo de este haga el proceso, el cual debe llenar un formulario y especificar si se trata de un caso cifrado o no, de serlo, debe de ingresar los valores de N y K para posteriormente hacer el cifrado del documento mediante AES de 128 bits y utilizando la llave original de AES hacer Secret Sharing Scheme. Posteriormente hacer el cifrado de las sub llaves de AES a través de RSA, utilizando un bot de Telegram que envía su Id

del abogado y su llave, finalmente se carga el archivo y se guarda. Cabe mencionar que el campo de K hace referencia a los abogados necesarios que se requerirán para poder acceder al archivo una vez que se haya guardado en el servidor, cuyas llaves deben de ser correctas para hacerlo. En caso de no ser cifrado el documento, simplemente se carga y se guarda en el servidor.

Diagrama 2: generar caso

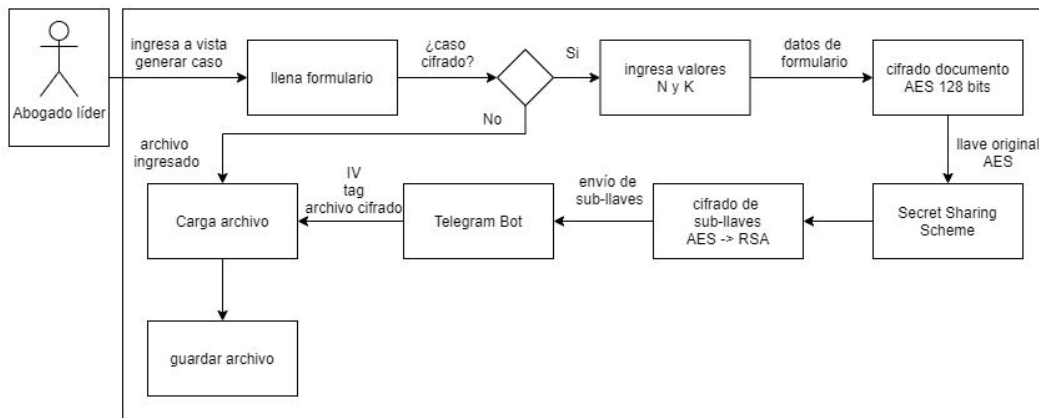


Figura 4: Generar casos

4.3 Modificar documentos

Como se puede observar en la Figura 5 en caso de que el archivo este cifrado se requiere que estén presentes los K abogados pero el que va a modificar el documento necesita haber iniciado su sesión para poder elegir si el archivo a modificar esta cifrado o no, dicha sesión muestra los casos en los que el abogado esta involucrado y puede seleccionar alguno de ellos. Si el archivo esta cifrado el abogado que realizará los cambios debe ingresar los datos solicitados: el archivo donde se encuentra su certificado digital, el archivo con las modificaciones realizadas y el archivo donde tiene su llave privada de RSA, adicional a esto debe ingresar su índice y sub llave de AES asignada. Por otro lado, los demás abogados deben ingresar únicamente los últimos tres datos mencionados anteriormente. Las validaciones importantes son que el nombre del documento concuerde con el original, que la llaves sean realmente privadas y la verificación del certificado digital del abogado, en caso de que no pase alguna validación muestra un mensaje de error.

Posteriormente se hace el descifrado de las sub llaves de AES con RSA y se unen, para hacer el cifrado del documento modificado junto con los datos de la modificación como son: nombre del abogado, fecha y hora del cambio y, la firma digital del abogado en cuestión. Finalmente, se sube el archivo cifrado junto con el archivo que contiene el vector de inicialización y tag el generados al servidor.

Si el documento no está cifrado se ingresa el certificado digital del abogado, así como su llave privada de RSA y el documento con las modificaciones, válida los datos necesarios como son la firma digital, que el nombre del documento concuerde, entre otras. Si pasa la verificación se sube y guarda el archivo en el servidor, en caso contrario muestra mensajes de error.

Diagrama 3: modificar documentos

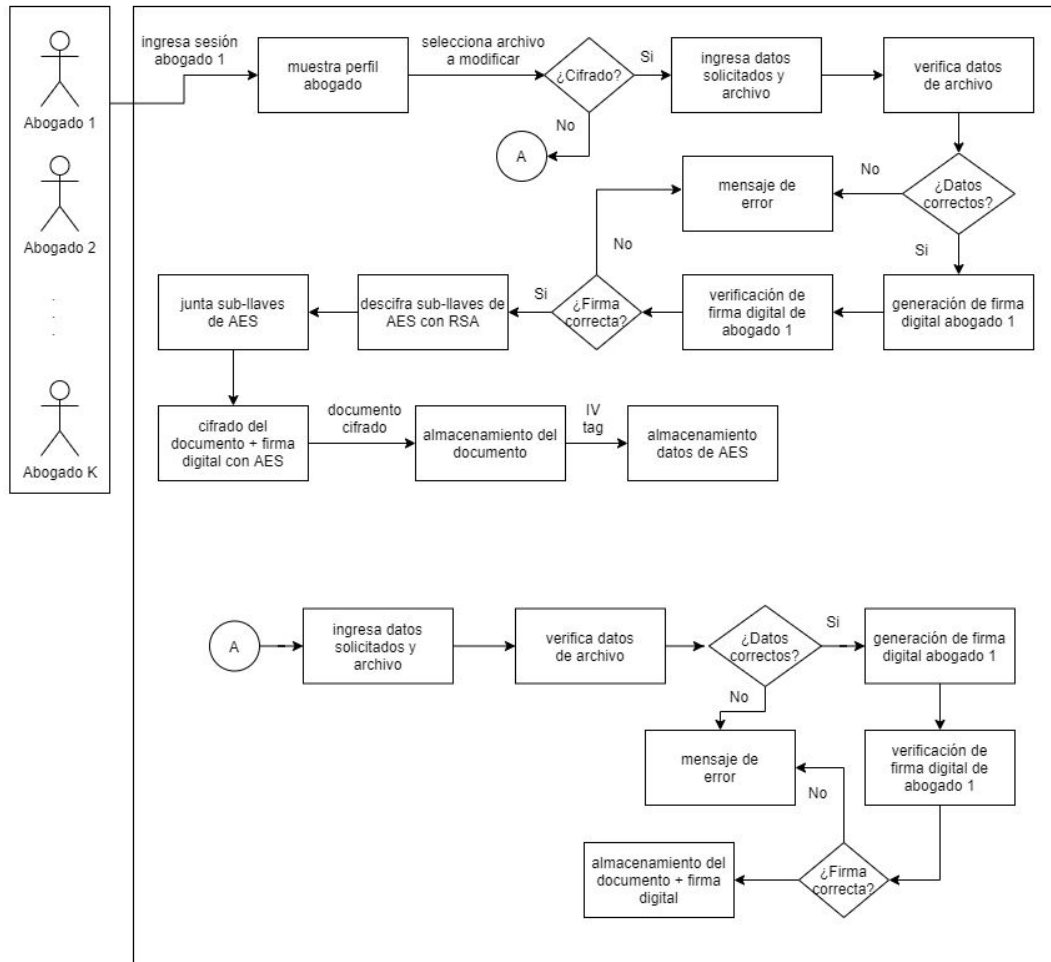


Figura 5: Modificar documentos

4.4 Descargar documentos

Como se puede observar en la Figura 6 el abogado que quiera descargar el caso debe iniciar sesión previamente, en caso de que el documento este cifrado se pide que estén presentes los K abogados para que cada uno ingrese el índice de la sub llave de AES que le tocó, así como la misma y el documento que contiene su llave privada de RSA para poder llevar a cabo el proceso de descifrado. Si todos los datos son correctos, en el backend se obtienen las llaves de AES originales descifrándolas mediante RSA para realizar el proceso correspondiente y obtener la original, posteriormente, se lee el archivo que contiene el vector de inicialización y el tag. Finalmente, se descarga el archivo de manera automática, en caso de que algún dato o este mal y no pase la validación manda un mensaje de error.

Si el documento no está cifrado simplemente se descarga de manera automática el archivo.

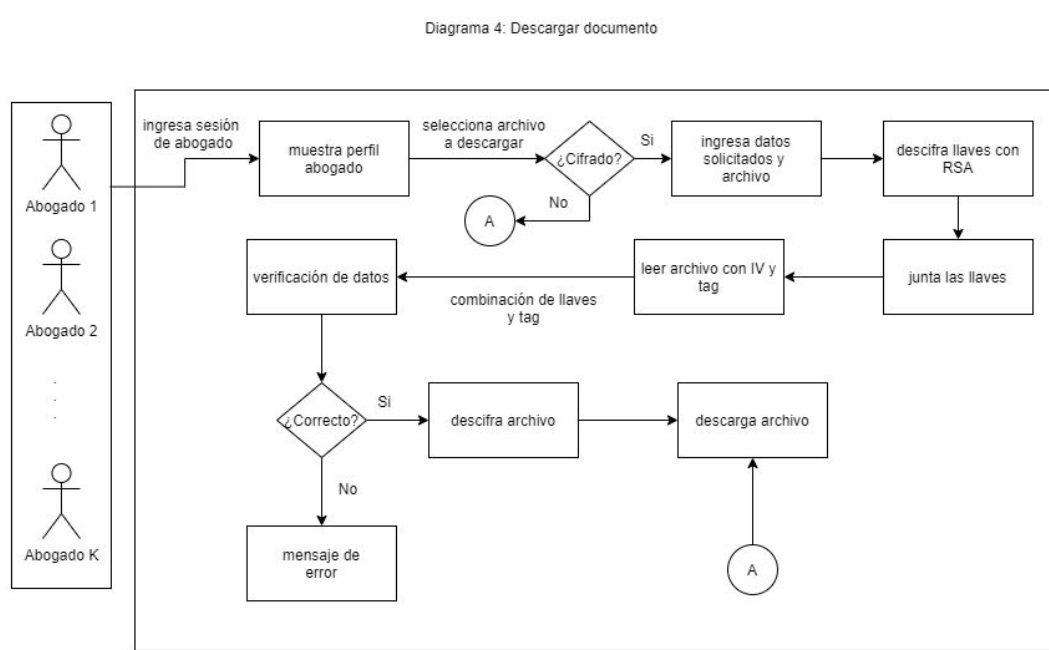


Figura 6: Descargar documentos

5 Partes importantes

Nuestro sistema debe garantizar los servicios criptográficos mencionados anteriormente, para ello primero que nada un administrador genera las claves públicas y certificados para cada abogado registrado en nuestro sistema. Un abogado líder será quien agregue nuevos casos, los cuales pueden estar o no cifrados haciendo uso de AES-128 bits; a los abogados que forman parte de ese caso les llegará un mensaje de confirmación por Telegram junto con una parte de clave de AES cifrado con un algoritmo basado en RSA. Para poder observar los archivos es necesario reconstruir la llave original de AES, por lo que, será necesario juntar las partes necesarias, lo mismo ocurre cuando se quieran modificar, además para modificarlos habrá que adjuntar la clave privada de RSA únicamente del abogado que realizó los cambios para firmar el archivo.

En la siguiente sección se presentarán los bloques de código más importantes para entender el funcionamiento de todo el sistema. Es así que nos enfocaremos sobretodo en las partes correspondientes al cifrado y descifrado de archivo; y exceptuaremos partes de la interfaz e implementación. Para la implementación de los aspectos que presentaremos se utilizó la librería **PyCryptodome** para el lenguaje Python.

5.1 Generar certificado y claves privadas

A diferencia de las demás partes que se describirán en esta sección, este módulo no será accesible por el usuario, solo el administrador del sistema podrá tener acceso a este módulo. El fragmento de código 1 es el encargado de generar los certificados y claves privada para cada usuario del sistema.

```

1  # creamos el objeto PKey para las claves RSA
2  k = crypto.PKey()
3  # Generamos las llaves RSA con longitud de 2048 bytes

```

```

4 k.generate_key(crypto.TYPE_RSA, 2048)
5
6 # leemos la llave privada Master Key que es la llave privada de la empresa
7 contenido = open( PRIVATE_FILE, 'rb' ).read()
8 # cargamos la llave privada Master Key como un tipo de llave privada
9 keyMaster = crypto.load_privatekey(crypto.FILETYPE_PEM, contenido)
10
11 # creamos un objeto de tipo certificado x.509 Req ya que este tipo de
12 # y certificado nos permite verificar la identidad de los abogados
13 cert = crypto.X509Req()
14 # Pais
15 cert.get_subject().C = "MX"
16 # Estado
17 cert.get_subject().ST = "CDMX"
18 # Localidad
19 cert.get_subject().L = "GAM"
20 # Organizacion de la entidad
21 cert.get_subject().O = "Smith&Jones"
22 # Unidad de organizacion de la entidad
23 cert.get_subject().OU = "IPN"
24 # Alias de la entidad
25 cert.get_subject().CN = "SandyLovers"
26 # Correo
27 cert.get_subject().emailAddress = "smithjones20201@gmail.com"
28 # Ingresamos al certificado la llave publica RSA que se genero al abogado
29 cert.set_pubkey(k)
30 # Usamos la funcion hash sha256 usando la llave privada de la organizacion
31 # para generar
32 # el certificado
33 cert.sign(keyMaster, 'sha256')
34
35 # Imprimimos que el certificado del abogado se genero y lo guardamos en un
36 # archivo
37 print ('\nEl certificado del abogado ' + nombre + ' se guardo en el
38 # archivo ' + nombre + ' certificado.pem')
39 open(join(cert_dir, CERT_FILE), "wb").write(crypto.
40 dump_certificate_request(crypto.FILETYPE_PEM, cert))
41
42 # Imprimimos que la llave privada RSA del abogado se genero y la guardamos
43 # en un archivo
44 print ('\nLa clave privada del abogado ' + nombre + ' se guardo en el
45 # archivo ' + nombre + ' private.pem')
46 open(join(cert_dir, KEY_FILE), "wb").write(crypto.dump_privatekey(crypto.
47 FILETYPE_PEM, k))

```

Código 1: Generación de certificado y clave privada

Los datos necesarios para generar cada una de las claves, se encuentra en archivo con el nombre "data.json", con dichos datos se genera un certificado y se le hace llegar al usuario por medio de un correo electrónico a nombre de la empresa.

5.2 Agregar casos

Como se mencionó anteriormente, cuando se quiere agregar un nuevo caso al sistema se tienen dos opciones: subir el archivo sin cifrar o subirlo cifrado haciendo uso de AES-128 bits. En el primer caso las tareas a realizar son muy sencillas, se guarda el archivo proporcionado así como la relación de los abogados con el caso en la base de datos. En caso contrario, es decir, que se requiera cifrar el archivo con AES-128 se sigue el siguiente proceso:

1. Cifrar el archivo.

Para cifrar el archivo se utiliza la función que se puede observar en el fragmento de código 2. En el cual lo primero que se hace es la generación de la clave de 128 bits (16 bytes) de manera pseudoaleatoria. En seguida se utiliza la llave recién creada para cifrar el contenido del archivo, indicando que el modo de operación a utilizar será EAX. Esto con la finalidad de generar un tag que más adelante nos servirá para verificar la autenticidad del mensaje.

```

1 def encryptAES( plainText ):
2     # Generacio pseudoaleatoria de la clave (16 bytes)
3     key = get_random_bytes( KeySizeByte )
4     # Creamos una instancia de un cifrador AES con modo EAX
5     cipher = AES.new( key, AES.MODE_EAX )
6     # Ciframos el texto plano, retorna texto cifrado y tag
7     cipherText, tag = cipher.encrypt_and_digest( plainText )
8     return cipher.nonce, key, tag, cipherText

```

Código 2: Cifrado con AES

Lo último que se puede observar es que la función retorna el vector de inicialización, la clave, authentication tag y el texto cifrado; respectivamente. Tanto el vector de inicialización, authentication tag y el texto cifrado serán guardados para futuras peticiones, sin embargo, la clave se somete al siguiente proceso.

2. Separar la llave de AES en N partes.

Es necesario repartir la llave entre N abogados y poder reconstruirla a partir de K abogados, para ello se implementó el protocolo de secreto compartido de Shamir (véase sección 3.1.3). En el fragmento de código 3 se puede observar la implementación de este. El primer paso consiste en dividir la clave original proporcionada en *valorN* partes, indicando que solo serán necesarias *valorK* partes para reconstruirla, haciendo uso de la función `split`, las partes son asignadas a los N abogados. Cada uno de los abogados asignados al caso recibirán una confirmación con su parte de la clave cifrada con RSA, así como un ID correspondiente a la parte que le pertenece de la llave.

```

1 def separateKey( key, valorN, valorK, nombreCaso, nameUserList, chatID,
2     nombreArchivo ):
3     # Se divide la llave en N partes, se puede reconstruir usando K
4     partes
5     shares = Shamir.split( valorK, valorN, key )
6     users = []
7     cont = 0
8     objectCaso = obtenObjectCaso( nombreCaso, nombreArchivo )
9     # Asignacion de partes de la llave a los abogados
10    for idx, share in shares:
11        nombreAbogado = nameUserList[cont]
12        idAbogado = chatID[cont]
13        # Se cifra su parte de llave con RSA (Public Key RSA)
14        RSAKeyCaso, excepcion = cifrarKeyRSA( nombreAbogado, share )
15        bot = telegram.Bot( token = '1225575856:
16        AAElahpfpWVks_1B1w5lnW187un80SRkRS4' )
17        # Se le envia un mensaje de erro al usuario si lo hubiera
18        if excepcion is False:
19            bot.send_message( chat_id = idAbogado, text = "Hola abogado %
20            s \n Tu certificado digital proporcionado es erroneo. \n https://lh3.
21            googleusercontent.com/proxy/9Q4EyDWry4aDbJdZIB--
22            e0eNpYDCL0YZ_8L7l4gQNNt9yYJojoP8qzs60YSMnenMrte0YVaiBNhrqYLkbPhX3EGPUcHHdSLPEdc_acPJ7v7j0
23            -iIB9KTS1ZsUGQ" % nombreAbogado )
24            return "", False
25        # Se le envia un mensaje con su llave AES cifrada con RSA
26        msg = "Hola abogado %s \n El ID de tu llave es: %d \n Tu llave es
27        : %s \n \nPor favor no la compartas, la necesitaras para tener acceso

```



```

20         a los archivos del caso '%s' \n Debes ingresar nicamente lo que
21         esta entre comillas simples" % ( nombreAbogado, idx, base64.b64encode
(RSAKeyCaso), nombreCaso )
22         bot.send_message( chat_id = idAbogado, text = msg )
23         users.append( nameUserList[cont] )
24         cont = cont + 1
25         # Se guardan las relaciones de los abogados con los casos
26         objectAbogado = almacenaAbogado( nombreAbogado, idAbogado )
27         almacenaRelacion( objectCaso, objectAbogado )

return users, True

```

Código 3: Separación de clave

La función utilizada para el cifrado con RSA, se puede observar en el fragmento de código 4. En este caso, la función cifra la parte de clave original que le corresponde al abogado con RSA, para esto se utiliza la clave pública contenida en su certificado digital.

```

1 def cifrarKeyRSA( nombreAbogado, message ):
2     try:
3         # Obtenemos el certificado del abogado con la clave publica del
4         abogado
5         contenidoCertificado = open( PATH + nombreAbogado + " certificado
.pem", 'rb' ).read()
6         certificado = crypto.load_certificate_request( crypto.FILETYPE_PEM
, contenidoCertificado)
7         # Obtenemos la clave publica de nuestro sistema
8         contenidoPublica = open( PUBLIC_FILE_MASTER, 'rb' ).read()
9         llavePublica = crypto.load_publickey( crypto.FILETYPE_PEM,
contenidoPublica)
10        #Verificamos el certificado con clave publica de nuestro sistema
11        certificado.verify( llavePublica )
12        # Obtenemos la clave publica del abogado a partir del certificado
13        publicKeyAbogado = crypto.dump_publickey( crypto.FILETYPE_PEM,
certificado.get_pubkey() )
14        archKey = RSA.import_key( publicKeyAbogado )
15        # Ciframos el texto con la clave publica del abogado
16        cipher = PKCS1_OAEP.new( archKey )
17        cipherText = cipher.encrypt( message )
18        return cipherText, True
19    except:
20        return "", False

```

Código 4: Cifrado con RSA

Finalmente, se almacenan las relaciones de abogados con el caso. Después de este último paso, se procede a guardar el archivo ya cifrado en nuestro servidor.

5.3 Ver casos

Al igual que en el caso anterior se presentan dos opciones: descargar un caso no cifrado o descargar uno que sí este cifrado. En el caso de que el archivo no este cifrado, no se proporciona ningún recurso, simplemente se descarga el documento que se requiere. En caso contrario, se necesitará proporcionar los ID's y las K partes para volver a formar la clave de AES, además, se necesitarán las K claves privadas de RSA para descifrar su correspondiente parte de la clave. Dicho proceso se especifica a continuación:

1. Reconstruir la clave de AES.

En el fragmento de código 5 se puede observar la función utilizada para combinar las K partes de clave. Cada parte de la clave es descifrada con su correspondiente clave

privada de RSA proporcionada por el abogado. Una vez que tenemos todas las partes descifradas, procedemos a combinarlas para reconstruir la clave original de AES.

```

1 def juntarLlaves( numeroAES, clavePrivadasAES, clavePrivadasRSA ):
2     combine = []
3     try:
4         # Se combinan las K partes para volver a formar la llave de AES
5         for idx, parteKey, clavePrivada in zip( numeroAES,
6         clavePrivadasAES, clavePrivadasRSA ):
7             # Desciframos las partes con RSA (Private Key RSA)
8             keyRSA = RSA.importKey( clavePrivada )
9             # Creamos una instancia de un cifrado PKCS#1 OAEP
10            cipherRSA = PKCS1_OAEP.new( keyRSA )
11            # Desciframos las partes de llave
12            share = cipherRSA.decrypt( base64.b64decode( parteKey ) )
13            combine.append( ( int(idx), share) )
14            # Reconstruimos la llave a partir de las K partes
15            key = Shamir.combine( combine )
16            return key, True
17    except:
18        return "", False

```

Código 5: Reconstruir la clave

Por último, la función retorna el estado de la petición, en caso de haber un problema se retornará un mensaje de error, en caso contrario se retorna la clave reconstruida. Una vez que hemos obtenido la clave original de AES se procede a el siguiente paso.

2. Descifrar el archivo cifrado.

Como se había mencionado antes, tanto el vector de inicialización como el authentication tag se guardan en el sistema, en este paso del proceso serán necesarios para poder ver el documento original. La función que se puede observar en el fragmento de código 6 recibe el vector de inicialización, authentication tag y archivo cifrado almacenados en el propio servidor; además de recibir la clave de AES recién reconstruida en el paso anterior.

```

1 def decryptAES( IV, key, tag, cipherText ):
2     # Cargamos la llave proporcionada por el usuario y el IV guardado
3     cipher = AES.new(key, AES.MODE_EAX, nonce = IV)
4     # Desciframos el archivo, retorna el texto plano
5     plainText = cipher.decrypt( cipherText )
6     try:
7         # Verificamos la authentication tag
8         cipher.verify(tag)
9         return plainText, True
10    except ValueError:
11        return plainText, False

```

Código 6: Descifrado con AES

La función descifra el archivo cifrado utilizando la clave y vector de inicialización pasados en la cabecera de la misma. En seguida se verifica la integridad del mensaje, verificando el mensaje recién descifrado con el authentication tag original que tenemos almacenado en el sistema. La función retorna el estado de la petición así como el texto original, dicho contenido es puesto a descargarse directamente al equipo del usuario.

5.4 Modificar casos

Al igual que los anteriores casos se vuelven a presentar las mismas dos opciones, modificar un archivo no cifrado o uno que si esta cifrado. Para ambos casos se presenta a continuación sus respectivos procesos que va siguiendo el sistema.

5.4.1 Caso no cifrado

Para el caso que se suba un archivo sin cifrado, el usuario pasara los siguientes datos:

- Llave privada de RSA. Para firmar el documento modificado.
- Certificado digital. Para la autenticación y verificación de la firma.
- Archivo modificado. Archivo con las modificaciones hechas por el usuario.

A continuación, se describe el proceso que sigue el sistema para cargar modificaciones de un archivo sin cifrado:

1. Generación firma digital.

En el fragmento de código 7 se puede observar una función que recibe la clave privada de RSA y el mensaje a firmar. Lo primero sera crear el digesto del archivo con la función hash SHA-2, en seguida se firma el digesto con la clave privada proporcionada por el usuario.

```

1 def generarFirmaDigital( clavePrivadaRSA, message ):
2     try:
3         # Obtenemos la clave privada del usuario
4         key = RSA.importKey( clavePrivadaRSA )
5         # Se crea un digesto del archivo con SHA-2
6         resultHash = SHA256.new( message )
7         # Se firma con un esquema basado en RSA (PKCS#1 v1.5)
8         signature = pkcs1_15.new( key ).sign( resultHash )
9         return signature, True
10    except:
11        return "", False

```

Código 7: Firma digital

Por último, la función retorna el estado de la petición, en caso de que surja algún problema también retorna un mensaje de error, en caso contrario retorna la firma digital.

2. Verificación firma digital.

En el fragmento de código 8 se puede observar una función que recibe el certificado digital, mensaje original y la firma obtenida en el anterior paso. Antes que nada, verificamos la integridad del certificado proporcionado por el usuario usando la llave pública de nuestro sistema. Proseguimos a verificar la firma anteriormente generada, para ello creamos un nuevo digesto con el mensaje original y con ayuda de la clave pública del usuario verificamos la integridad de la firma.

```

1 def verificarFirma( contenidoCertificado, message, signature ):
2     try:
3         # Obtenemos el certificado
4         certificado = crypto.load_certificate_request( crypto.FILETYPE_PEM
5         , contenidoCertificado )
6         # Obtenemos la clave publica de nuestro sistema
7         contenidoPublica = open( PUBLIC_FILE_MASTER, 'rb' ).read()
8         llavePublica = crypto.load_publickey( crypto.FILETYPE_PEM,
9         contenidoPublica )
10        # Verificamos el certificado del usuario con la clave publica de
11        nuestro sistema
12        certificado.verify( llavePublica )
13        # Obtenemos la clave publica del usuario del certificado
14        publicKeyAbogado = crypto.dump_publickey( crypto.FILETYPE_PEM,
15        certificado.get_pubkey() )

```

```

12     key = RSA.import_key( publicKeyAbogado )
13     # Se crea el digesto del mensaje
14     resultHash = SHA256.new(message)
15     # Se verifica la firma pasada
16     pkcs1_15.new(key).verify( resultHash, signature )
17     return True
18 except:
19     return False

```

Código 8: Verificación firma digital

Una vez hecha la verificación, la función solo retorna si es o no es correcta la firma digital que se generó. Por último, el sistema concatena el mensaje original con la fecha de modificación, usuario que realizó la última modificación y la firma digital recién generada.

5.4.2 Caso cifrado

Para el caso que se suba un archivo con cifrado, el usuario pasará los siguientes datos:

- Claves privadas de RSA de cada abogado. Para descifrar sus partes correspondientes de la clave de AES, además, la clave privada del primer abogado será utilizada para generar la firma digital.
- Las K partes de la clave de AES junto con su ID correspondiente a cada abogado. Para volver a formar la clave original de AES y poder cifrar el archivo con esa misma clave.
- Certificado digital del primer abogado. Para la autenticación y verificación de la firma.
- Archivo modificado. Archivo con las modificaciones hechas por el usuario.

A continuación, se describe el proceso que sigue el sistema para cargar modificaciones de un archivo con cifrado:

1. Generar y verificar firma digital.

Igual al caso de modificar un archivo sin cifrado, utilizamos las funciones para generar y verificar la firma digital; que se pueden observar en los fragmentos de código 7 y 8 respectivamente.

2. Reconstruir clave de AES.

Como lo pudimos observar en el fragmento de código 5 en el proceso de ver los casos, se muestra la función de la que también hacemos uso en este proceso para poder reconstruir la clave original de AES a partir de las K partes que cada abogado tiene en su posesión.

3. Cifrar el documento con AES.

En el fragmento de código 9 se puede observar la función utilizada para cifrar con AES documentos modificados por el usuario. Dicha función toma como parámetro la clave original de AES, texto plano, nombre del primer abogado y la firma digital que se creó en un paso anterior a este. Al texto plano se le concatena la firma digital junto con la fecha e identificación de la última modificación, esta combinación es cifrada utilizando la clave original de AES.

```

1 def modificarAES( key, plainText, nombreAbogado, firmaDigital ):
2     formatoFecha = '%b %d %Y %I:%M%p'
3     dateTime = datetime.now()
4     dateTimeStr = dateTime.strftime( formatoFecha )
5     # Se crea instancia de cifrador AES con modo EAX
6     # Se usa la llave proporcionada por el usuario
7     cipher = AES.new( key, AES.MODE_EAX )
8     # Se cifra el texto plano concatenado con la fecha, nombre y firma de
9     # la ultima modificacion
10    cipherText, tag = cipher.encrypt_and_digest( plainText + b'\n\n' + b'
11    Ultima modificacion: ' + nombreAbogado.encode() +
                                     b' ' + dateTimeStr.encode
                                     () + b'\n\n' + base64.b32encode( firmaDigital ) )
12    return cipher.nonce, tag, cipherText

```

Código 9: Cifrado con AES, archivo modificado.

Al terminar la función, esta retorna los nuevos valores del vector de inicialización y authentication tag; además, de retornar el mensaje cifrado. Por último, el sistema almacena los datos que fueron arrojados por la función anteriormente mencionada.

6 Manual de usuario

6.1 Hardware, software, requerimientos de instalación e instrucciones

Para la parte de hardware se mencionan los requisitos mínimos recomendados para que la ejecución del proyecto sea de manera correcta

- Procesador a 1.6 GHz o superior.
- Al menos 4 GB de memoria RAM.

Para la parte de software se en listan los requerimientos recomendados, así como las librerías necesarias para el correcto funcionamiento del proyecto

- Sistema operativo Windows 10 Home o superior.
- Django versión 3.0.7.
- Python 3.7 o superior.
- Librería Pycryptodome versión 3.9.8.
- Librería Path
- Librería OpenSSL

Para utilizar de manera correcta el proyecto se requieren pasos previos antes de poder utilizarlo, los cuales se mostraran a continuación

6.2 Instalación de Python

Descargar la última versión de python desde la pagina oficial [Python](#) y descargar el archivo ejecutable , posteriormente instalarlo como cualquier programa.

6.3 Instalación db browser for sqlite

Se recomienda instalar el gestor **DB Browser for SQLite** el cual se puede descargar desde su sitio oficial [DB Browser for SQLite](#) con la finalidad de poder verificar el contenido de la base de datos y sus tablas.

6.4 Instalación de librerías para Python

Para instalar las librerías necesarias se recomienda usar el comando `pip` en la terminal de Windows, para la versión 3.4 en adelante ya tiene por defecto el comando por lo que se recalca que es adecuado instalar la última versión de Python disponible en la página oficial.

6.4.1 Pycryptodome

Para instalar la librería con los métodos de criptografía necesarios se hace mediante el siguiente comando.

```
1 pip install pycryptodome
```

Código 10: Instalar pycryptodome con pip

6.4.2 pyOpenSSL

Para instalar la librería OpenSSL se debe ingresar el siguiente comando.

```
1 pip install pyOpenSSL
```

Código 11: Instalar OpenSSL

6.4.3 Django

Para instalar Django se tiene que ingresar el siguiente comando.

```
1 pip install Django
```


Código 12: Instalar Django

6.5 Pantallas del sistema

En la siguiente sección presentaremos una guía de usuario para poder utilizar de manera correcta nuestro sistema. Antes de poder utilizar nuestro sistema tendrás que asegurarte de estar registrado en el mismo, de ser así, te habrán llegado las claves necesarias para realizar los pasos que se presentaran a continuación.

6.5.1 Agregar un abogado al sistema

En la figura 7 se presenta la pantalla para poder agregar un abogado al sistema, una vez agregado a nuestra base de datos será posible tomarlo en cuenta para añadirlo a los casos que se den de alta.



Agrega a un abogado al caso!

Enviar datos

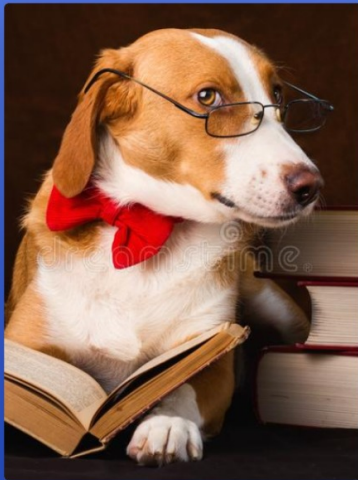
Figura 7: Agregar abogado

Como se puede observar para agregar un abogado al sistema solo se necesita su nombre y ID de Telegram; será necesario que el abogado cuente con una cuenta de Telegram, con la finalidad de poder enviarle por ese medio claves de acceso o mensajes de error que puedan generarse durante el uso del sistema.


6.5.2 Agregar un caso

Nuestro sistema cuenta con dos opciones para almacenar archivos, los abogados pueden guardar sus archivos con o sin cifrado. Para cada caso se presentará la pantalla correspondiente para poder utilizar dicha opción en nuestro sistema.

La primera opción será subir un archivo sin cifrado, en la figura 8 se puede observar la pantalla para hacer uso de esta opción. En dicha pantalla el usuario deberá ingresar el nombre del caso, el archivo que desea subir y el número de abogados implicados en el caso.



Bienvenido al sistema, guarda tu caso!

 Caso_Bitcoins.docx

Ingresa el total de abogados en el caso:


Guardar Caso
 Ir a Google
Cifrar documento

Figura 8: Agregar caso

Una vez que se hayan llenado los campos que se piden, se procede a guardar el caso pulsando el correspondiente botón, si la operación se realiza correctamente arrojará la siguiente pantalla:

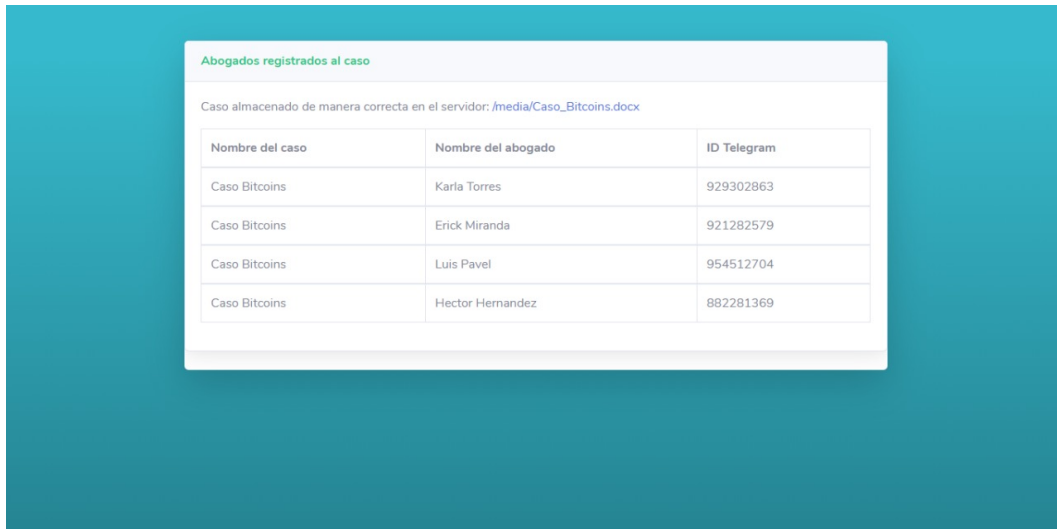


Figura 9: Confirmación al agregar caso

En la figura 9 se muestra el mensaje de confirmación que el sistema arroja cuando un caso se sube de manera correcta al servidor. En el mensaje se especifica los abogados que pertenecen al caso, así como sus IDs de Telegram para con los que son identificados.

En caso contrario de que se desee realizar la segunda opción, se tendrá que acceder a ella a través del botón "Cifrar documento" que se puede observar en la figura 8. Al pulsar dicho botón nos llevará a la pantalla siguiente:

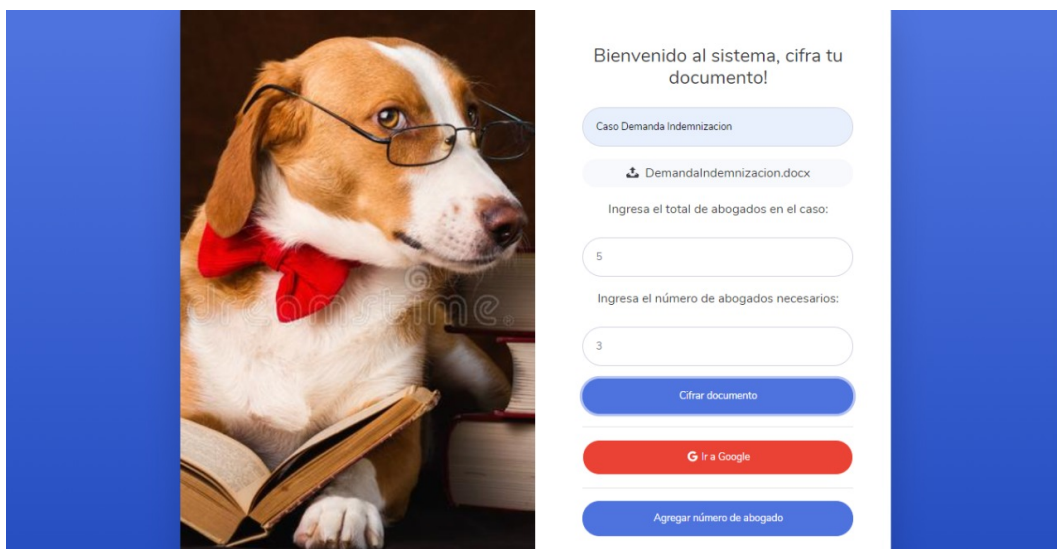
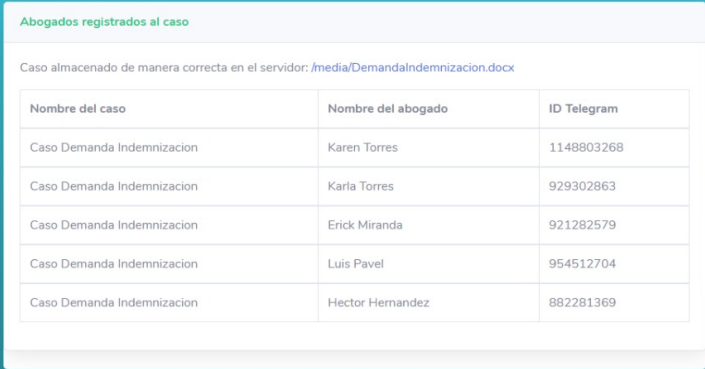


Figura 10: Agregar caso con cifrado

En la figura 10 se puede observar la pantalla donde el usuario podrá un guardar un

archivo con cifrado dentro de nuestro sistema. El proceso es muy similar al que pudimos observar al subir un archivo sin cifrado, lo único que se agrega en este caso, es un campo para introducir la cantidad K de abogados necesarios para reconstruir la clave de AES. En caso de que el proceso se realice de forma correcta se arrojará el mensaje que puede observar en la figura 11.



Nombre del caso	Nombre del abogado	ID Telegram
Caso Demanda Indemnizacion	Karen Torres	1148803268
Caso Demanda Indemnizacion	Karla Torres	929302863
Caso Demanda Indemnizacion	Erick Miranda	921282579
Caso Demanda Indemnizacion	Luis Pavel	954512704
Caso Demanda Indemnizacion	Hector Hernandez	882281369

Figura 11: Agregar caso con cifrado

En dicho mensaje se puede observar la asignación de abogados al caso, además, el sistema mandará mensaje de confirmación a cada abogado a través de Telegram, el mensaje será acompañado por la parte de la clave de AES que le corresponde al abogado. En caso de que haya un problema con algún certificado de los abogados, le llegara específicamente a dicho abogado un mensaje de error.

6.5.3 Ingresar a la plataforma

Para el ingreso a la plataforma se accede desde la pantalla que se puede observar en la figura 12a, para acceder solo será necesario que el abogado proporcione su ID de Telegram. En caso de que no te encuentres registrado en la plataforma, el sistema arrojará el mensaje que se puede observar en la figura 12b. Una vez que hemos entrado a la plataforma, se puede

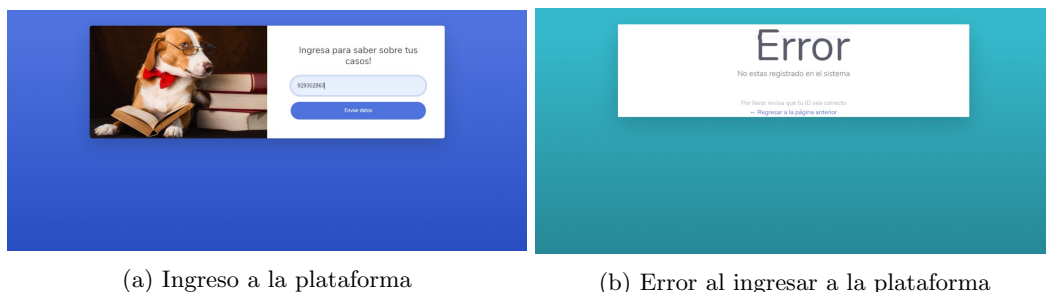


Figura 12: Ingreso a la plataforma

observar en la figura 13a los casos en los que el abogado se encuentra dado de alta, así mismo del lado izquierdo se observan los datos pertenecientes al abogado. Mientras que en

la figura 13b se observa los abogados con los que se encuentra trabajando actualmente el usuario.

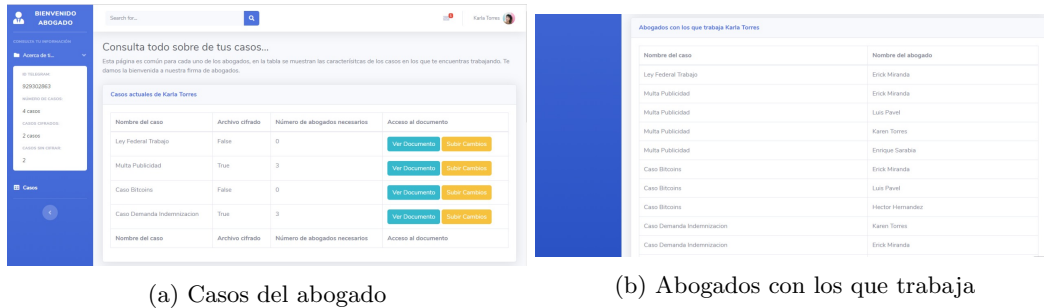


Figura 13: Plataforma

6.5.4 Descargar casos

Dentro de la plataforma dentro de la sección de los caso con lo que el abogado esta ligado, se encuentra un botón por caso para descargar dicho caso. A continuación, se muestra como descargar tanto archivos con cifrado, como archivos sin cifrado junto con su respectiva pantalla si fuera el caso.

Para el caso que se requiera descargar un archivo sin cifrado, solo se tendrá que pulsar el botón "Ver caso" que se encuentra a un lado del caso que se desea descargar. Sin embargo, si se quisiera descargar un archivo con cifrado aparecería la siguiente pantalla:

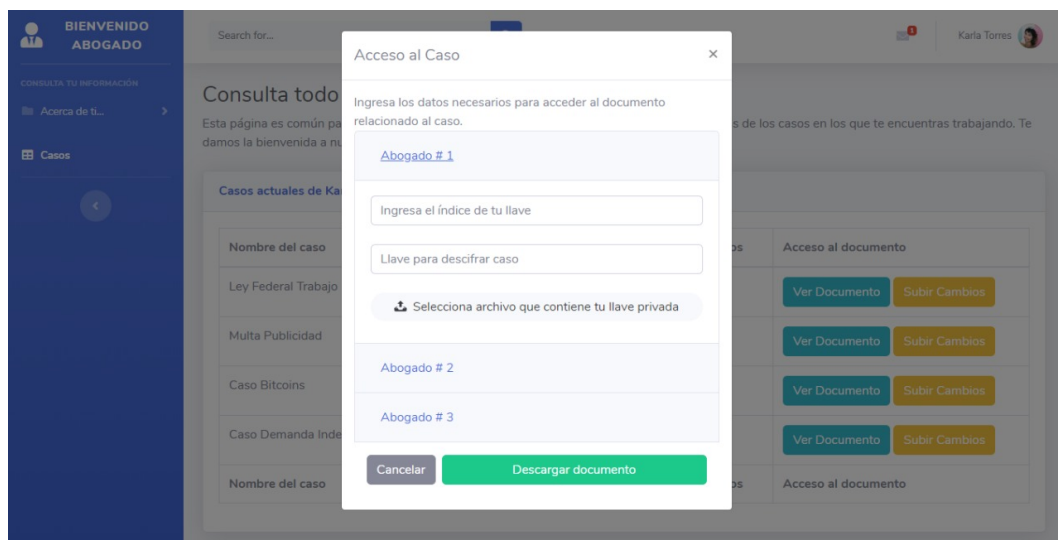


Figura 14: Descargar caso cifrado

Como se puede observar en la figura 14, el usuario introduce las K partes de cada abogado para volver armar la clave original de AES, así como el ID que le corresponde, además, dichas partes se encuentra cifrada con RSA, para poder descifrarla también deberán proporcionar la claves privada correspondiente a cada abogado. En caso de que la operación se lleve de forma correcta se descargar automáticamente el archivo como se muestra en la siguiente figura:

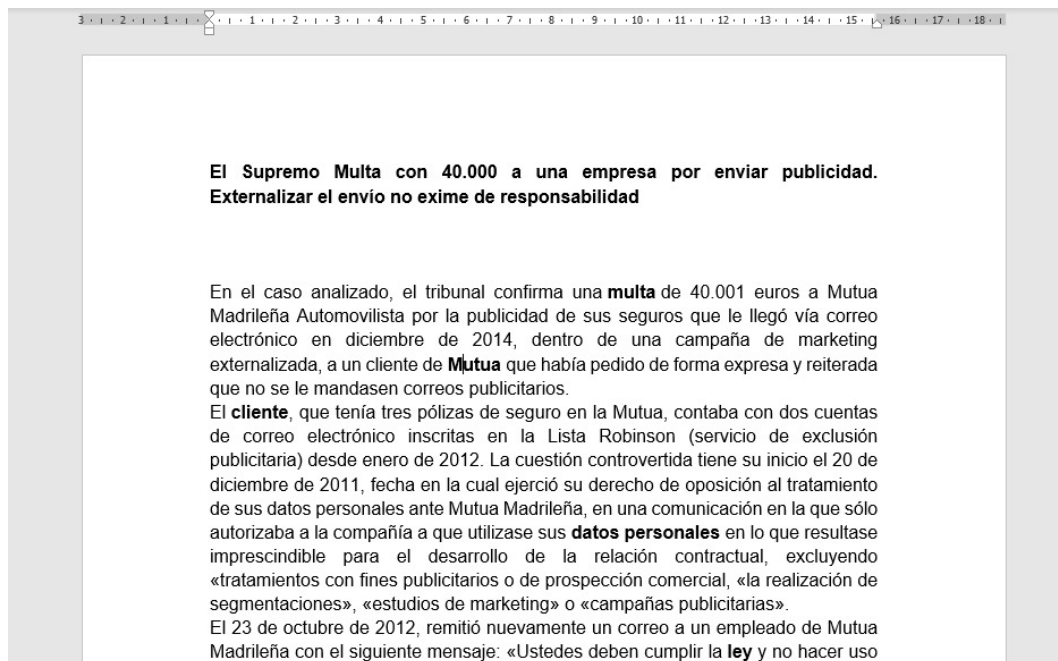


Figura 15: Archivo descargado

6.5.5 Modificar caso

Al igual que en las funciones anteriores se tiene la opción de modificar un archivo con cifrado o la modificar un archivo sin cifrado. En cada caso se encuentra un botón "Subir cambios" para que el usuario pueda modificar dicho archivo. Para ambos caso es un proceso similar, que se describirá a continuación con las pantallas correspondiente a cada opción.

Para la opción de que el usuario modifique un archivo sin cifrado, tendrá que introducir certificado digital para su autenticación, clave privada para firmar el documento y el archivo modificado con el mismo nombre que el original. En la figura 16 se puede observar el formulario para ingresar los datos anteriormente mencionados.

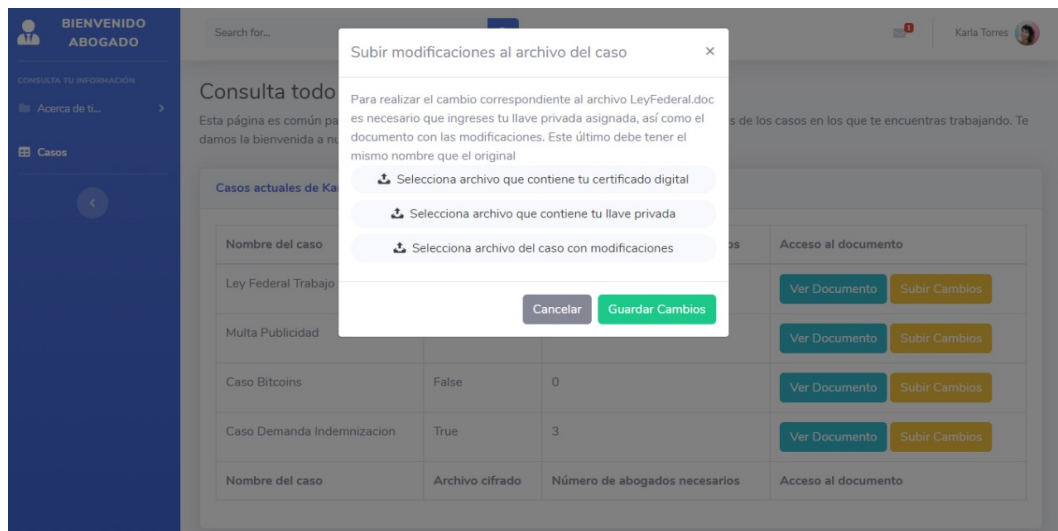


Figura 16: Modificar archivo no cifrado

Durante el proceso se puede presentar algún error, en la figura 17 se muestran los posibles errores que podrían surgir durante el proceso de estar modificando algún caso. En caso de

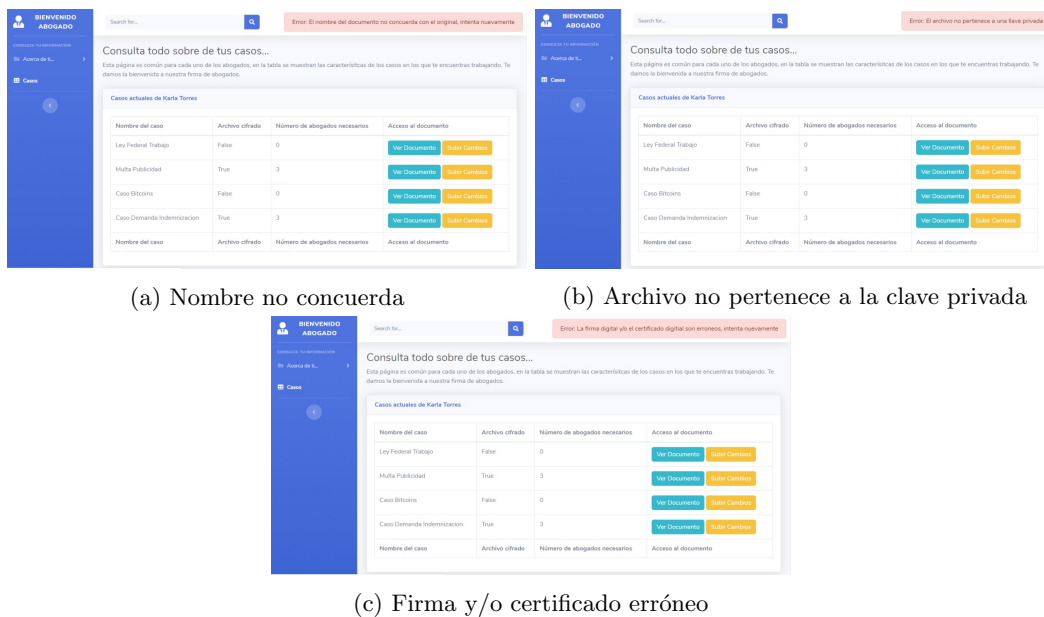


Figura 17: Errores al modificar archivo no cifrado

que se quiera utilizar la opción de modificar un archivo con cifrado, aparecerá la siguiente pantalla:

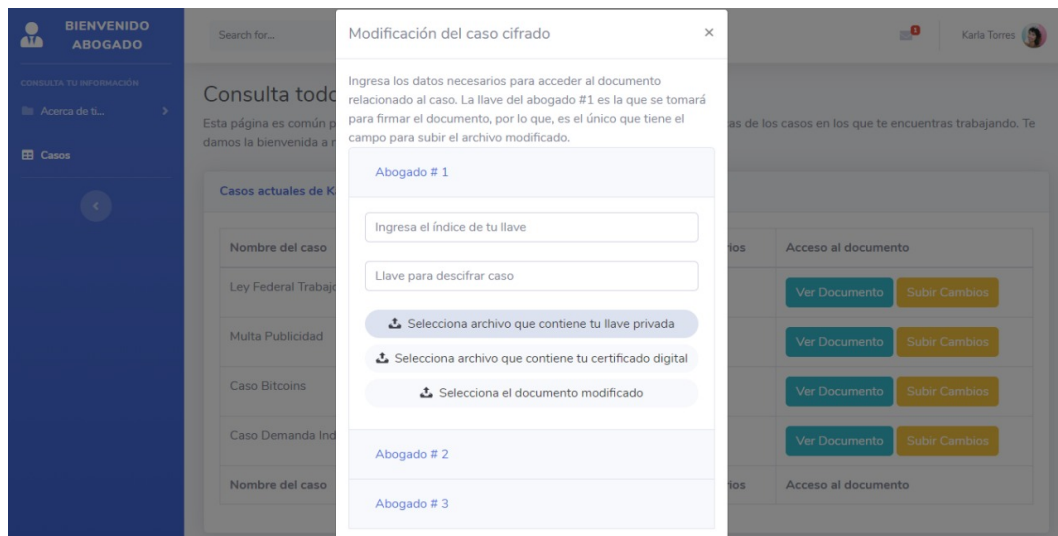


Figura 18: Modificar archivo con cifrado

En la figura 18 se puede observar el formulario para que el usuario pueda modificar el archivo con cifrado. El usuario deberá proporcionar las K partes de la clave original de AES, así como sus respectivos IDs y claves privadas para descifrar dicha parte. Además, el abogado #1 que corresponde al usuario que se encuentra en ese momento en la sesión, es quien introducirá su certificado digital, archivo modificado con el mismo nombre que el original y de quien se hará uso su clave privada para firmar el documento. En este proceso pueden ocurrir los mismos errores que se presentan en el proceso anterior. Ya sea la opción que haya elegido el usuario, en el servidor habrá una versión de su archivo firmado con o sin cifrado según sea el caso. En cuanto se haga una modificación, la siguiente visualización que se haga se vera de la siguiente forma:

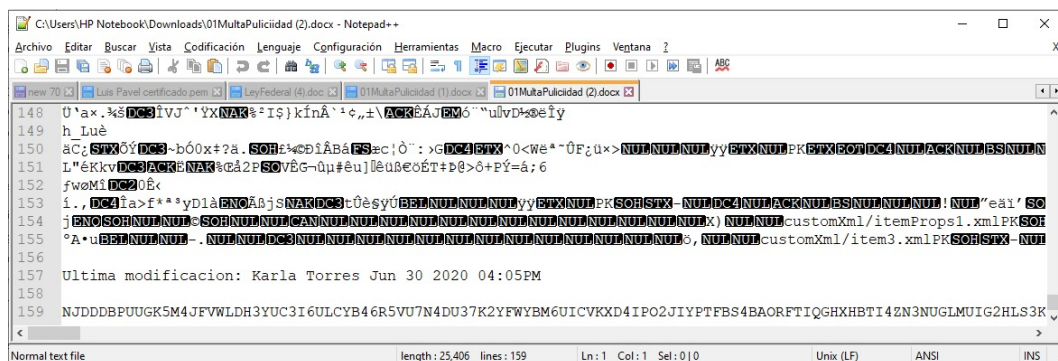


Figura 19: Modificar archivo con cifrado

En la figura 19 se muestra un archivo firmado después de ser modificado por un usuario de nuestro sistema. Consta del archivo modificado, fecha de modificación, responsable de la modificación y la firma digital.

7 Conclusiones

En conclusión podemos afirmar que se resolvió la problemática del proyecto de una manera correcta debido a que se cubrieron los puntos requeridos, esto es, dejando los módulos funcionales y probados de diferentes formas para evitar un error que se pudiera presentar. Para esto, se hizo uso de los conocimientos adquiridos durante el curso acerca de los algoritmos criptográficos, cuidando desde los servicios criptográficos y primitivas criptográficas utilizadas.

Un reto que se nos presentó durante la elaboración del proyecto es el hecho de trabajar a distancia, debido a que no siempre nos podíamos conectar todos al mismo tiempo para intercambiar ideas, sin embargo, organizándonos y determinando un horario pudimos avanzar y con la ayuda de las tecnologías actuales poder concretarlo de manera que cada quien estuviera EN CASA respetando las normativas del gobierno y evitando ponernos en riesgo a nuestros familiares y nosotros mismos.

Adicional a lo anterior, para la implementación de la firma digital y el certificado digital nos metimos a investigar un poco en libros y sitios de internet, debido a que no entendíamos en su totalidad cómo funcionaban. Pero al investigar y ver cómo se puede poner información para identificar la entidad que está usando dicho certificado es de gran importancia para tener un nivel de seguridad que permita confiar en la fuente, con la ventaja de poder ahorrar tiempo y dinero para hacer trámites a través de internet utilizándolo en cualquier momento que lo necesitemos y proporcionemos seguridad a quien enviamos datos, en este caso un documento que posiblemente contenga información sensible que requiere de verificarse para no ocasionar algún tipo de fraude o práctica ilícita.

Creemos que con experiencia e investigación podemos llegar a tener la capacidad de extender los usos de la criptografía ya que es claro que es una rama muy importante dentro del campo del mundo digital, principalmente por los nuevos retos que cada día se presentan y no sólo eso, sino también como se perfeccionan y mejoran las técnicas para las aplicaciones que lo requieran, optimizando los tiempos para dar una mejor calidad a una aplicación o servicio.

Por último, agradecemos a la maestra Sandra Díaz Santiago quien nos proporcionó la ayuda necesaria durante todo el curso tanto de manera presencial como de manera virtual durante el horario de clases y fuera de este en la resolución de dudas.

References

- [1] L. Dong and K. Chen, *Cryptographic Protocol: Security Analysis Based on Trusted Freshness*. Berlín: Springer, 2012.
- [2] L. R. Knudsen and M. J. B. Robshaw, *The Block Cipher Companion*. New York: Springer-Verlag, 2011.
- [3] C. P. J. Pelzl, *Understanding Cryptography*, 2nd ed. Berlin Heidelberg: Springer-Verlag, 2010.
- [4] D. W. Mihir Bellare, Phillip Rogaway. The eax mode of operation. [Online]. Available: <https://www.iacr.org/archive/fse2004/30170391/30170391.pdf>
- [5] “Sistema de almacenamiento seguro de archivos basado en secreto compartido,” Trabajo terminal No. 2016-B009.
- [6] B. K. Verisign, J. Jonsson, and et al., “Pkcs 1: Rsa cryptography specifications version 2.2,” November 2016.
- [7] Holded. El certificado digital, qué es, para qué sirve y cómo obtenerlo. [Online]. Available: <https://www.holded.com/es/blog/certificado-digital-sirve-obtenerlo>
- [8] N.D. X509. [Online]. Available: <https://certificadora.com/x509.htm>
- [9] T. H. Eric Young. Openssl. [Online]. Available: <https://www.openssl.org/>
- [10] N. P. Smart, *Cryptography Made Simple*. Bristol, UK: Springer-Verlag, 2016.