Sobre la confidencialidad en el almacenamiento de pruebas de diagnóstico por la imagen de

[GRAN ENTIDAD HOSPITALARIA]

Fecha de realización: 13/03/2024

Consultoría realizada por:

Juan Luis Ruano Muriedas

José Joaquín Rojas Romero

Antonio José Suárez García

Índice

Resumen ejecutivo

Tras un exhaustivo estudio de su problema, nos gustaría proponerle las distintas **soluciones** a las que hemos llegado para cada una de sus consultas.

Para la ***Consulta 1*** hemos definido las herramientas ***KeyStore*** y ***KeyTrust*** para el **almacenamiento y manejo de claves seguras** en cualquier entorno así como un caso de uso de ejemplo. Nuestros profesionales cuentan con experiencia en el uso de Java KeyStore en caso de que tenga problemas con la implementación de la solución.

Para la ***Consulta 2*** hemos contrastado tres modelos de cifrado de tipo AES para elegir aquel que mejor cumpla con las especificaciones descritas. Como resultado final establecemos el modelo **AES-256-GCM** como modelo de cifrado/descifrado de imágenes empleando el lenguaje de programación *Python*. Cualquier duda en la prueba del código proporcionado puede ser respondida con el mayor grado posible de detalle por nuestros profesionales.

Para la ***Consulta 3*** hemos analizado las distintas opciones que barajaban y le recomendamos **mantener su *NAS* secundario**, aunque modificando su comportamiento y su seguridad. En caso de que tenga problemas realizando dichas modificaciones al ***NAS***, por lo que no dude en contactarnos para realizarlas o resolver cualquier duda que tenga al realizar las mismas.

Metodología y resultados

Consulta 1

Para el problema propuesto le recomendamos la implementación de ***KeyStores*** y ***TrustStore***. Las *KeyStores* son herramientas permiten **almacenar claves de cifrado simétrico** en distintos dispositivos de manera segura y las *TrustStore* guarda un registro

del certificados digitales de dispositivos que se pueden confiar, implementando ambos en cliente y servidor se puede **garantizar la seguridad de las claves** incluso si se encuentran en **servidores públicos**. El *KeyStore* almacena claves privadas, certificados con claves públicas o claves secretas para fines criptográficos, donde se coloca un alias para identificar a cada elemento almacenado. Se utiliza para desarrollos e **implementaciones del lado de servidor**.

Usualmente, nosotros usamos el *KeyStore* cuando queremos que nuestro servidor use el protocolo HTTPS, donde precisamente el servidor busca la clave privada desde el almacén y presenta su correspondiente clave pública y certificado al cliente.

De manera similar, si el cliente también necesita autenticarse, entonces el cliente también tiene un *KeyStore* y también presenta su clave pública y su certificado. Aparte de eso, las claves públicas pueden verificar o cifrar datos, mientras que las claves privadas pueden firmar o descifrar datos.

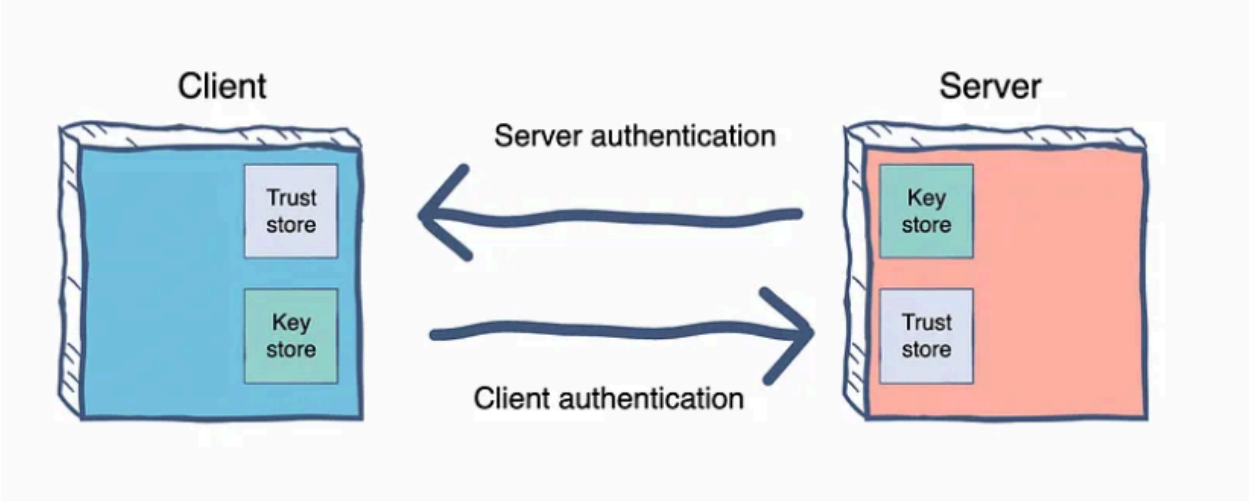

Figura 1. Esquema de la conexión cliente-servidor con servicios KeyStore y TrustStore.

Un *TrustStore* es todo lo contrario. Mientras que un *KeyStore* normalmente conserva certificados que nos identifican, un ***TrustStore* conserva certificados que identifican a otros**.

En el proceso de comunicación propuesto, la *KeyStore* se encargará de almacenar las claves de descifrado de las pruebas de diagnóstico y el médico podrá acceder a ellas tras verificar su certificado digital con la *TrustStore* del servidor y la contraseña personal del profesional de tal manera que **ningún agente externo pueda ver dichas consultas**.

Por último, queremos recomendarle un sistema de *KeyStore/TrustStore* de Java el cual utiliza la librería *Keytool* para la **administración de las herramientas**.

A continuación le enseñamos **un ejemplo práctico de uso** de *KeyStore* usando *Keytool* y que en caso de que tuviera más problemas, no dude en contactar con nuestros profesionales.

- Generar una Java keystore y una clave

```
keytool -genkey -alias mydomain -keyalg RSA -keystore keystore.jks  -keysize 2048
```

- Importe un certificado de CA de la raíz o intermedio a una KeyStore Java existente

```
keytool -import -trustcacerts -alias root -file Thawte.crt -keystore keystore.jks
```

- Importar un certificado primario firmado a un existente Java KeyStore

```
keytool -import -trustcacerts -alias mydomain -file mydomain.crt -keystore keystore.jks
```

- Generar un KeyStore y un certificado autofirmado

```
keytool -genkey -keyalg RSA -alias selfsigned -keystore keystore.jks -storepass password
-validi
```

- Exportar un certificado desde un keystore

```
keytool -export -alias mydomain -file mydomain.crt -keystore keystore.jks
```

- Listar certificados de CA confiables

```
keytool -list -v -keystore $JAVA_HOME/jre/lib/security/cacerts
```

- Eliminar un certificado de un KeyStore Java

```
keytool -delete -alias mydomain -keystore keystore.jks
```

En el caso que desee **usar código python con una *KeyStore***, puede usar la librerías **pyjks** de python donde simplemente tendrá que instalar las librerías en su compilador y después simplemente importarlas usando la siguiente línea de comando:

```
import jks
keystore = jks.KeyStore.load('keystore.jks', 'passphrase')
```

Consulta 2

En primer lugar, para resolver el problema que se plantea en esta consulta hemos decidido elaborar una comparativa entre tres algoritmos del modelo AES, debido a su robustez contra ataques externos y facilidad de implementación que los sitúa por encima de las otras alternativas así como al ser cifradores de bloque simétricos ideales para el tratamiento de imágenes, para reflejar la eficiencia en materia de tiempo y esfuerzo para conseguir obtener una conclusión lo suficientemente óptima a dicho problema. Los tres algoritmos elegidos son:

- AES-128-CBC
- AES-256-ECB
- AES-256-GCM

A pesar de que la otra consultora no recomienda el algoritmo AES-128-CBC por sus fallas de seguridad descubiertas en el pasado, nosotros hemos querido incluirlo en la comparativa para extraer más información al respecto y poder aportar mejores soluciones.

Dos de los tres algoritmos cumplen con los cuatro puntos relacionados con la eficiencia y seguridad del cifrado/descifrado, que son:

1. Alta Garantía que la información cifrada no sea inteligible por terceros.
2. Alto mantenimiento de la integridad de la información al aplicar los algoritmos de cifrado/descifrado considerados.
3. Bajos tiempos medio de procesamiento del cifrado en los algoritmos considerados.
4. Bajos tiempos medio de procesamiento del descifrado en los algoritmos considerado

También se ha tenido en cuenta el volumen de ficheros y tamaño de los mismos a la hora de elaborar los *scripts* para el cifrado/descifrado. De esta manera, se ha elaborado la tabla siguiente con los datos experimentales obtenidos tras las pruebas de los tres algoritmos de cifrado/descifrado implementados a través de dichos *scripts*:

Modelo del algoritmo	Tiempos de cifrado/descifrado	Información íntegra
AES-128-CBC	0.507394 segundos	No (poco seguro)
AES-256-ECB	0.019948 segundos	Sí (más seguro)
AES-256-GCM	0.045265 segundos	Sí (mucho más seguro)

Todas las pruebas han sido realizadas con varios formatos de imagen, entre los que encontramos: PNG, JPG y DICOM. En la tabla se muestran los resultados tras probar la misma imagen en todos los scripts desarrollados bajo condiciones similares de cifrado/descifrado a excepción del método que implementa cada distinto modelo internamente. Se muestra una captura de la salida por terminal de las pruebas para corroborar los datos mostrados (los cuales se han recortado para facilitar su legibilidad)

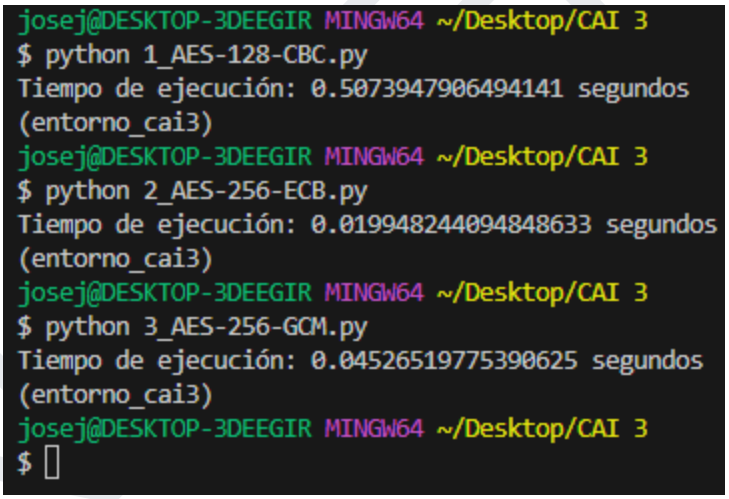

Figura 2. Resultados obtenidos tras ejecutar cada script de cifrado/descifrado mostrado en segundos.

En segundo lugar y contrastando las pruebas obtenidas, nuestro equipo ha decidido emplear el modelo **AES-256-GCM** y la justificación es la siguiente:

1. AES-256-GCM tarda mucho menos en completar el cifrado/descifrado que el modelo AES-128-CBC y solamente un poco más que el modelo AES-256-ECB a pesar de ser mucho más seguro que este último.

2. El modelo AES-256-ECB es un poco más complejo de usar ya que requiere de mayor conocimiento para emplear padding/relleno, y cuya mala implementación puede dar origen a vulnerabilidades conocidas donde a partir del relleno se obtiene la imagen original.

3. El modelo AES-256-ECB es rápido, pero hemos comprobado que existe una pequeña diferencia de 4 bytes entre la imagen original y la descifrada debido a que el relleno tiene que ser múltiplo de 16 bytes por la manera en la que está implementado el modelo. Se muestra una captura de ello en la ***Figura 3.***

4. El modelo AES-128-CBC necesita redimensionar las imágenes, por ejemplo a 512x512 píxeles, para evitar corrupciones de la imagen en el cifrado/descifrado debido a cómo está desarrollada la librería pertinente. Esto puede originar un diminuto efecto *zoom* apenas distinguible en la imagen descifrada resultante.

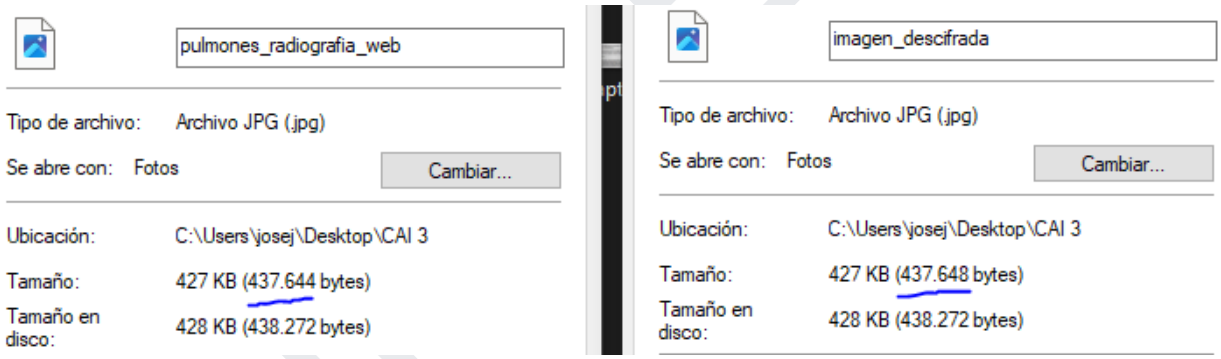

Figura 3. Se muestran subrayados los bytes de las imágenes original y descifrada respectivamente, donde existe una diferencia de 4 bytes más en la descifrada.

Finalmente, se ha establecido el modelo **AES-256-GCM** como la solución final al problema de cifrado/descifrado debido a que no se pierde/gana información durante el proceso, manteniendo la máxima calidad de la imagen una vez descifrada y el mismo hash que la imagen original. Además es robusto y no tiene tantas vulnerabilidades conocidas en comparación a los anteriores modelos debido a que no se usan tanto tal y como muestra la Fuente: *The 2021 TLS Telemetry Report de F5 Labs*.

El *script* realizado en el lenguaje de programación *Python* llamado ***solucion_propuesta.py***, una vez ejecutado nos pedirá en la terminal que introduzcamos el nombre completo de la imagen con su formato incluído, la cual se debe encontrar en el mismo directorio que el *script* de prueba, y la clave segura que el médico o trabajador debe conocer previamente, tal y como se muestra en la siguiente figura:

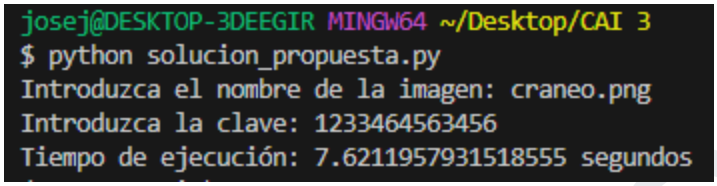

Figura 4. Ejecución del programa de cifrado/descifrado donde pide nombre de la imagen y clave.

Si todo se realiza correctamente, obtendremos en la misma ruta (para facilitar la comprensión) tanto la imagen cifrada como la imagen descifrada para comprobar que efectivamente funciona y realiza la tarea que pide el cliente, tal y como se muestra en la siguiente figura:

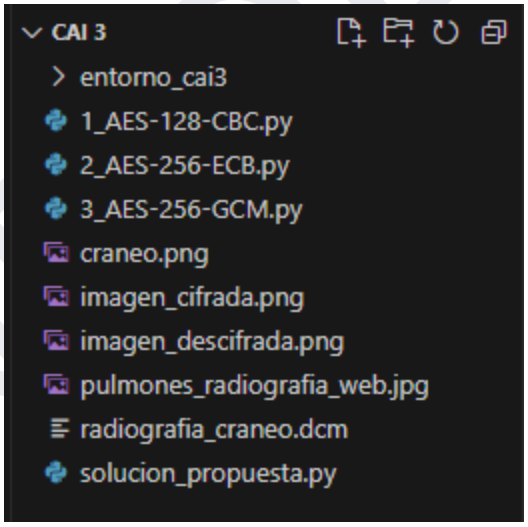

Figura 5. Archivos "imagen_cifrada.png" e "imagen_descifrada.png" generados por el programa llamado "solución_propuesta.py" tras cifrar y descifrar una imagen dada (*craneo.png*) en un formato de imagen dado (*png*).

Consulta 3

Para empezar, nos gustaría exponer una gestión de ciclo de vida de los datos **(DLM)** de su entidad. Tratándose sus datos únicamente de **imágenes de sus pacientes** y estudiando la forma en la que se procesan dentro de su red una vez se realizan las imágenes, entendemos que los datos son procesados de la siguiente forma:

Para empezar, después de obtener la imagen, ésta **pasa directamente al *NAS***, sin almacenarse en los dispositivos que las generan (ej: máquina de rayos x).

Tras esto, se realizan **copias de seguridad en el segundo *NAS***, el cual está designado para eso mismo. Sin embargo, estas copias **no son protegidas** de ningún modo, por lo que se encuentran visibles para todo el mundo.

Nosotros recomendamos que **mantenga el uso del *NAS* secundario**, solo que añadiendo varias capas de seguridad. Para empezar, los datos que se escriban en el mismo deberán **cifrarse** con una **clave distinta** a la usada para cifrar en el *NAS* primario. Además, recomendamos configurar el servidor para realizar **copias en espejo**, reduciendo así el tiempo de recuperación de los datos al **mínimo posible**.

Tras todos estos cambios, el DLM de una imagen en su sistema sería tal que: **Es creada y enviada a ambos *NAS* → Se cifra con cada una de las claves y se almacena → Cuando una imagen sea modificada, se leerá desde el *NAS* primario y se guardará en el mismo.**

El *NAS* secundario solamente será actualizado cuando se **añada un nuevo archivo** o **cada cierto intervalo de tiempo** que considere oportuno (dado el volumen de datos que maneja, no recomendamos que el intervalo sea mayor a **una semana**).