Documento de Diseño - Codefest Ad Astra 2024 AIr Reinforcement

Sergio Oliveros *, Daniel Álvarez †, Haider Fonseca ‡, Sebastian Urrea §, y Daniel Perea ¶
Universidad de los Andes. Bogotá, Colombia
*s.oliverosb@uniandes.edu.co, †da.alavarezv@uniandes.edu.co, ‡h.fonseca@uniandes.edu.co,
§js.urrea@uniandes.edu.co, ¶dfpereag@emavi.edu.co

I. Introducción

En el marco del *Codefest Ad Astra 2024* se propone el reto de crear una aplicación en los lenguajes de programación C/C++ que:

- Dada una ruta de de una imagen de entrada y una ruta de un archivo salida, sea capaz de cifrar imágenes satelitales y almacenar dichas imágenes cifradas en directorios específicos.
- Dada una ruta de un archivo de entrada y una ruta de de una imagen de salida, sea capaz de descifrar imágenes cifradas y almacenar dichas imágenes en directorios específicos.
- Sea capaz de cifrar y descifrar las imágenes de los satélites mediante el uso de llaves dinámicas.

En este documento se presenta la implementación de una aplicación que cumple con los requerimientos previamente mencionados.

II. Arquitectura

II-A. Solución implementada

La solución propuesta tiene como base el siguiente diagrama UML:

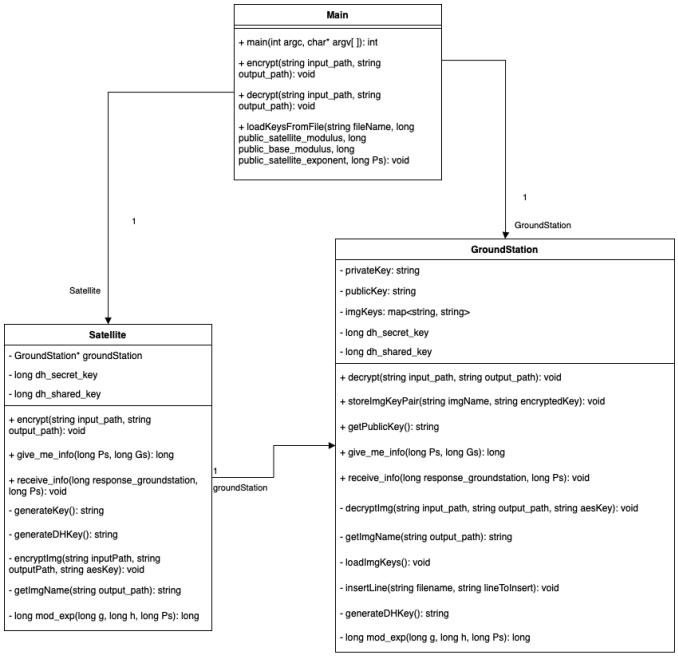

Figura 1. UML solución propuesta

La solución consta de 3 clases: **Main**, **Satellite**, y **GroundStation**. Cada una responsable de cumplir con una parte de la solución de tal forma que el conjunto de ellas es capaz de cumplir con los requerimientos especificados, pero también de contar con bajo acoplamiento para permitir la extensión y adaptación de esta solución a diferentes contextos.

La clase **Main** es la plantilla brindada por los organizadores del Codefest, esta tiene como responsabilidad recibir los argumentos esperados del programa (`<operation><input_path><output_path>`) y almacenarlos en variables para utilizar en el programa. Estos son utilizados por *main* para llamar las funciones correspondientes de las clases con las cuales tiene relación de asociación (*Satellite* y *GroundStation*). En específico, dependiendo de los argumentos, la plantilla ejecuta la función en *Satellite.encrypt* o la función *GroundStation.decrypt*.

La clase **Satellite** tiene como responsabilidad principal el cifrado de las imágenes. Para ello, cuenta con la función *encrypt* (que es llamada por la clase *Main*) y es la encargada gestionar las demás funciones de la clase *Satellite* para realizar este proceso. A grandes rázagos el proceso consta en generar una llave de cifrado simétrico aleatoria. El proceso de generación de una llave simétrica aleatoria toma como semilla un número compartido a través del método Diffie-Hellman entre la clase *Satellite* y *GroundStation*, de manera que, si ambos conocen la semilla, ambos son capaces de generar la misma llave simétrica. Una vez con la llave simétrica se cifra la imagen y se guarda en el directorio correspondiente. Posteriormente, la clase *Satellite* le comparte a *GroundStation* el directorio de la imagen cifrada, de manera que *GroundStation* conoce la llave simétrica (porque la recrea con el número compartido por Diffie-Hellman) y el archivo asiciado a dicha llave. Con estos datos es posible descifrar los datos en un futuro.

La clase **GroundStation** tiene como responsabilidad principal el descifrado de las imágenes cifradas por la clase *Satellite*. La forma en que realiza esto es a través de un proceso que consta de 2 partes. En un inicio, después de que la clase *Satellite* transmite el nombre de la imagen cifrada y *GroundStation* recrea la llave simétrica, almacena los datos en

un mapa y, también, en un archivo que es de acceso exclusivo para *GroundStation*. Este archivo de acceso exclusivo contiene los nombres de las imágenes cifradas y las llaves con que se cifraron, y es cargado a memoria dentro de una mapa cada vez que la clase es instanciada. Esto permite descifrar un archivo, ya que si se ingresa un archivo previamente cifrado, se conoce la llave y por tanto cómo descifrarlo.

III. Estrategia de cifrado

III-A. Cifrado de imágenes

Para el cifrado de las imágenes satelitales se usa una estrategia de cifrado por bloques con llaves simétricas. En particular se está utilizando el algoritmo de *Advanced Encryption Standard* (AES) con llaves de tamaño 256 bits y en modo *Counter* (CTR). La razón por la cual se decidió utilizar este algoritmo es debido a que ha demostrado ser considerablemente robusto contra ataques de fuerza bruta y por tanto se ha vuelto un estándar de cifrado con llaves simétricas. Adicionalmente, al tratarse de una llave simétrica esto hace que el proceso de cifrado sea computacional-mente menos costoso a la hora de cifrar, aspecto que es a tener en cuenta al manipular imágenes de gran tamaño. Finalmente, este es el algoritmo recomendado por los estándares de seguridad espacial.

La implementación que se usa de este algoritmo es la provista por la libreria OpensSSL para C/C++, la cual viene instalada por defecto en la distribución de Linux Ubuntu Desktop.

Sin embargo, la forma en la que se hace uso de esta librería es a través de una interfaz implementada en un repositorio público de GitHub *openssl-aes-cipher*:

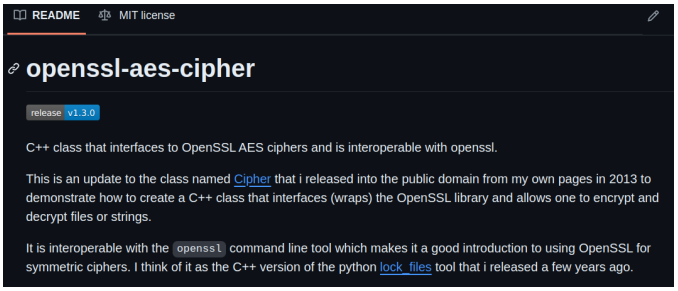

Figura 2. Repositorio de implementación de interfaz para algoritmos AES OpensSSL

Interfaz que cuenta con una licencia de uso MIT, la cual permite el uso de la interfaz sin restricciones, siempre y cuando, sea utilizada en software que también cumpla con la licencia de uso MIT. El enlace al repositorio se encuentra disponible en la referencia 1 de este documento.

La interfaz provee un repertorio de funciones que hacen más accesible el uso de las funciones de OpenSSL (las cuales tiene poca documentación). La forma en que se usaron las funciones fue para para codificar el binario de la imágenes en base64 (caracteres de texto) y para cifrar dichas secuencias de caracteres con el algoritmo AES y con las llaves correspondientes.

De forma que las funciones provenientes de esta interfaz se usan en el programa para leer y cargar en memoria la secuencia de bytes de las imágenes, después se codifican dichas secuencias en base64 para generar secuencias de caracteres, y las secuencias de caracteres se cifran mediante el algoritmo AES con llaves simétricas dinámicas. Las llaves simétricas dinámicas se obtienen al generar cadenas de caracteres alfanuméricos aleatorios y de tamaño variable.

III-B. Comunicación cifrada de llaves simétricas

La manera en que se logra la comunicación de las llaves simétricas es mediante el uso del algoritmo Diffie-Hellman, el cual es un algoritmo criptográfico publicado en 1976 por *Whitfield Diffie* y *Martin Hellman* cuyo propósito es compartir llaves a través de un medio inseguro. Para ello, los 2 agentes que vayan a intercambiar llaves definen de manera pública 2 números. El primero es g, que se conoce como generador, este número puede ser un número cualquiera. El segundo número es p, que se conoce como primo, este es un número primo, el cual entre más grande sea, más robusto será la seguridad del cifrado. Posteriormente el agente 1 necesita definir un número a de forma secreta y el agente 2 necesita definir un número b de forma secreta, estos son un números positivos cualquiera, idealmente grandes.

Una vez con estos números definidos el agente 1 realiza la siguiente operación $A = g^a modp$ y el agente 2 genera la siguiente operación $B = g^b modp$. Posteriormente el agente 1 le comunica el número A al agente 2, y el agente 2 le comunica el número B al agente 1. Finalmente, el agente 1 realiza esta última operación: $key = B^a modp$ y, el agente 2 realizar esta última operación $key = A^b modp$.

El resultado de esto es que ambos agentes generan el mismo valor pues $A^b modp = g^{ab} modp = g^{ba} modp = B^a modp$. Logrando así comunicar una llave secreta.

Teniendo en cuenta este principio, en la solución implementada lo que se hace es comunicar esta llave secreta entre *Satellite* y *GroundStation*, de manera que ambos generan la misma llave compartida. Y esta llave compartida sirve como semilla generadora para crear una llave de cifrado AES. Dado que ambos tienen la misma semilla. Ambos tienen la misma llave AES. Garantizando la seguridad en la comunicación.

IV. Estrategia de descifrado

IV-A. Descifrado de imágenes

Como se explicó en la sección anterior el cifrado de las imágenes se hace mediante el uso del algoritmo de cifrado por bloques AES con llaves simétricas de 256 bits. Por lo que el descifrado de las imágenes se hace también con este algoritmo, y con la misma llave utilizada para cifrar la imagen originalmente, la cual se obtiene al compartir la semilla generadora a través del algortimo Diffie-Hellman.

Este procedimiento se logra mediante el uso de la interfaz propuesta en *openssl-aes-cipher* y consta de cargar el archivo cifrado en base64 que representa a una imagen en memoria, para luego descifrar la secuencia de caracteres, y finalizar convirtiendo la secuencia de caracteres en una secuencia de bytes que son almacenados en el directorio pasado por parámetro al inicio del programa.

IV-B. Comunicación cifrada de llaves simétricas

Al igual que en el proceso de cifrado, en la proceso de descifrado utilizamos la llave AES generada con la semilla compartida por Diffie-Hellman. Esta llave es almacenada en un archivo secreto de *GroundStation* y es cargada en memoria para utilizar al momento de descifrar.

V. Estrategia para gestión de memoria

Para la gestión de memoria se utiliza una estrategia de *streamming*. En donde se lee la imagen que se desea cifrar mediante chunks. Para ello lo que se hace crear un buffer de $3GB$, posteriormente se leen las primeras $3GB$ de la imagen y se cargan en memoria RAM. Seguido se cifran estas $3GB$ y se guardan en la ruta indicada por parámetro. Al repetir este proceso las veces que sea necesario, se generan múltiples archivos, cada uno de máximo $3GB$. Que en conjunto contienen la información necesaria para reconstruir la imagen. De esta manera se garantiza que en ningún momento se supera la restricción de $4GB$ de imagen en memoria.

Posteriormente, para la reconstrucción de la imagen lo que se hace es que se toma el conjunto de archivos que representan a la imagen. Se descifra cada una de estas imágenes cifradas y posteriormente se escriben los datos en la ruta especificada para la imagen resultante. De forma que a la hora de realizar el descifrado también se garantiza que no se superan las $4GB$ de memoria RAM, pues cada uno de los archivos se lee de forma individual y, dado que cada uno tiene un tamaño no superior a $3GB$.

VI. Validación solución dentro de Jetson Nano

La solución fue validada al ejecutar el algoritmo de cifrado y de descifrado dentro de la Jetson Nano.

Para empezar se cifra la imagen dentro de la Jetson Nano. El resultado que se obtiene es el siguiente:

```
claves.txt  image3.tif  main  out.enc
codefest16@codefest16-desktop:~/Codefest2024/Reto1/repo$  rm -r out.enc/
codefest16@codefest16-desktop:~/Codefest2024/Reto1/repo$ time ./main encrypt ima
ge3.tif out.enc
input_path=image3.tif
output_path=out.enc
Encrypted image

real    0m3,822s
user    0m2,532s
sys     0m1,260s
codefest16@codefest16-desktop:~/Codefest2024/Reto1/repo$
```

Figura 3. Cifrado en Jetson Nano

Allí es posible observar que el proceso de cifrado tarda un total de aproximadamente $4s$ para una imagen de $133MB$.

Al realizar el proceso de descifrado se obtiene el siguiente resultado:

```
codefest16@codefest16-desktop:~/Codefest2024/Reto1/repo$ time ./main decrypt out
.enc newImg3.tif
input_path=out.enc
output_path=newImg3.tif
Decrypted image

real    0m41,967s
user    0m41,012s
sys     0m0,836s
codefest16@codefest16-desktop:~/Codefest2024/Reto1/repo$
```

Figura 4. Descifrado en Jetson Nano

De este proceso es posible observar que para una imagen de $133MB$, el descifrado tarda aproximadamente $42s$. Adicionalmente, cabe resaltar que el tamaño del archivo se conserva para el proceso de cifrado y descifrado.

Finalmente, se copia la imagen descifrada y se visualiza esta para evaluar su resultado cualitativo.

Figura 5. Resultado cualitativo de descifrado en Jetson Nano

Al visualizar la imagen resultante es posible comprobar que se conservan los detalles finos de la imagen. Por lo que podemos afirmar que el proceso de cifrado y descifrado no destruye ni altera la información.

VII. Conclusiones

Se espera que haya sido posible dar cuenta cómo la idea general de la solución propuesta en este documento es capaz de cumplir con los requerimientos especificados para el Reto 1 del Codefest Ad Astra 2024, como también que la solución implementada sirva como prueba para validar la solución realizada.

Referencias

[1] Linoff, J. “openssl-aes-cipher“. GitHub. https://github.com/jlinoff/openssl-aes-cipher?tab=readme-ov-file#openssl-aes-cipher

[2] Tyson, m. “Understand Diffie-Hellman key exchange“. InfoWorld https://www.infoworld.com/article/2334365/understand-diffie-hellman-key-exchange.html