Instituto Tecnológico de Costa Rica

Área Académica de Ingeniería en Computadores

Arquitectura de Computadores I

Diseño e implementación de una aplicación de desencriptación RSA

Documentación de Diseño

Arturo Córdoba Villalobos

II Semestre 2020

15 de octubre de 2020

1. Listado de requerimientos del sistema

En esta sección se detallan los requerimientos que debe cumplir la aplicación de desencriptación RSA, los cuales se presentan en la tabla 1, donde se presentan el identificador y su especificación.

ID	Requerimiento
1	El programa debe desencriptar información que fue codificada en RSA.
2	El programa debe procesar completamente en ensamblador una imagen codificada de 640 x 960 pixeles, para obtener una imagen decodificada de 640 x 480 pixeles.
3	El usuario debe tener la capacidad de ingresar la ruta de la imagen codificada.
4	El usuario debe tener la capacidad de ingresar los valores de la llave privada.
5	El programa debe ser capaz de reconstruir un pixel codificado tomando dos numeros sucesivos del archivo de la imagen codificada, donde el primero corresponde al byte más signicativo, y el segundo al byte menos significativo.
6	El programa debe utilizar el algoritmo de exponenciación modular para manejar las operaciones de exponenciación y módulo con números de hasta 16 bits.
7	Se debe implementar una tabla de coincidencias, donde se almacenen los valores codificados y su respectivo valor decodificado.
8	El programa debe generar un archivo de salida donde se almacene la imagen decodificada.
9	Se debe proveer un programa escrito en un lenguaje de alto nivel, el cual muestre las imágenes encriptada y desencriptada en una cuadrícula 2x1.
10	Se debe integrar el visualizador de imágenes y el procesamiento en ensamblador en un solo framework.

Tabla 1: Requerimientos del sistema

2. Opciones de solución al problema

En esta sección se detallarán dos soluciones planteadas al problema, las cuales fueron ideadas con el objetivo de cumplir con todos los requerimientos. Cada una de las soluciones es acompañada de dos diagramas, esto con el fin de explicar de una manera más clara el funcionamiento de cada una de ellas. El primer diagrama explica los pasos que debe seguir el programa, y el segundo explica el proceso de decodificación.

2.1. Solución A

La primera propuesta se denominó Solución A, aquí el usuario ingresa los valores de la llave privada a través de la consola, al igual que la ruta de la imagen encriptada. Se utiliza una tabla para ahorrar procesamiento en la decodificación de pixeles, de esta manera si un valor ya ha sido calculado se puede cargar sin la necesidad de procesar nuevamente. Se utiliza el algoritmo de exponenciación modular para realizar el proceso de decodificación con números considerablemente grandes, consiguiendo una optimización de tiempo y memoria para estos cálculos. A continuación se detallan el flujo de ejecución del programa planteado y el proceso de decodificación de la imagen.

2.1.1. Flujo de ejecución del programa

En la figura 1 se puede apreciar el diagrama de flujo del programa para la solución A. Se parte del inicio, donde se lee un caracter del archivo de la imagen codificada, posteriormente se verifica si se ha llegado al final del archivo. Si no se ha llegado al final del archivo se entra en una etapa de lectura y almacenamiento de los números codificados, y se utiliza un espacio en blanco como divisor entre ellos, no se sale de esta etapa hasta leer todos los números de la imagen codificada. Si se llegó al final del archivo se entra en una etapa de decodificación, en la cual se arma el valor de un pixel, se obtiene su correspondiente valor decodificado y se almacena el resultado en la salida, este proceso se repite hasta decodificar todos los pixeles de la imagen.

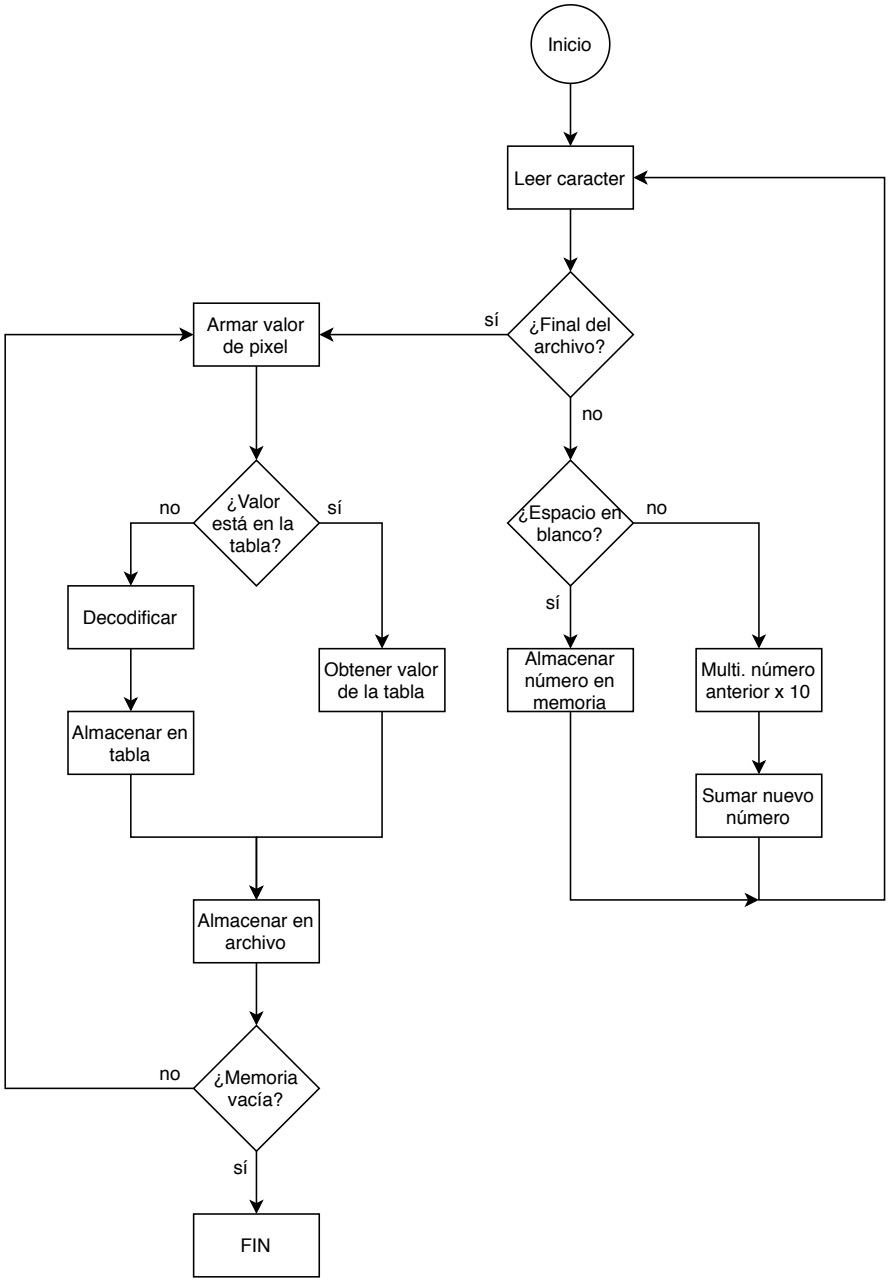

Figura 1: Diagrama de flujo del control del programa

2.1.2. Proceso de decodificación

En la figura 2 se muestra un diagrama de flujo que describe el proceso de decodificación para la solución A. En primer lugar, se calcula el exponente modular según el número de iteración, si el resultado obtenido es necesario para construir el resultado final entonces se multiplica con los resultados anteriores, se aplica un módulo para reducir la magnitud del número obtenido. Posteriormente, se multiplica por dos el exponente utilizado para calcular el de la siguiente iteración. Finalmente, se verifica si el exponente obtenido es mayor al exponente de la llave, si esta condición se cumple entonces el número se ha decodificado, si no, se vuelve a repetir el proceso.

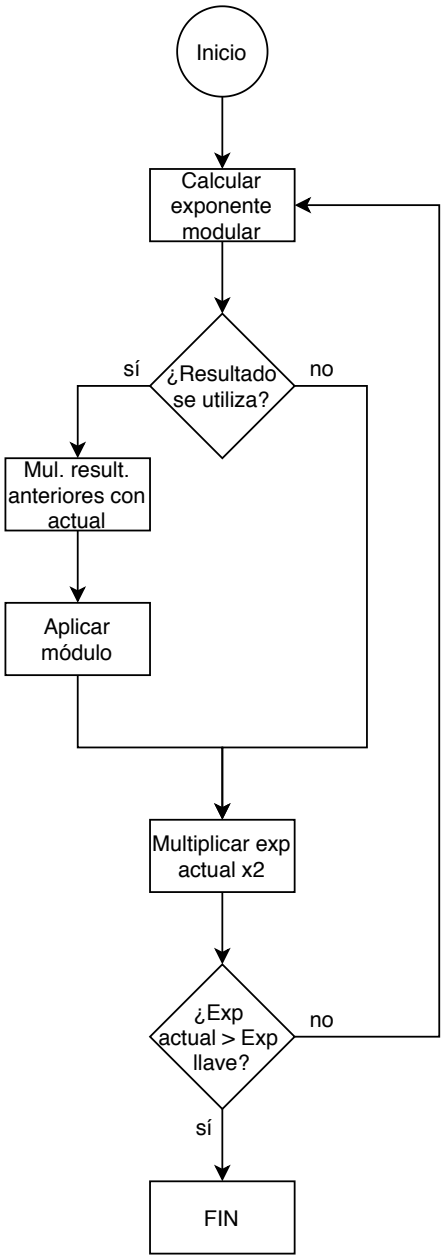

Figura 2: Proceso de decodificación de un pixel

2.2. Solución B

La segunda propuesta se denominó Solución B, en esta el usuario también ingresa los parámetros de entrada por medio de consola, los cuales serían la ruta de la imagen encriptada, el valor del exponente y el valor del módulo. Se utiliza una tabla para ahorrar procesamiento en la decodificación de los pixeles, se almacena el valor codificado y su respectivo valor decodificado para aquellos pixeles que no han sido procesados con anterioridad. Se propone utilizar el algoritmo de exponenciación modular para el manejo de las operaciones de decodificación. A continuación se detallan el flujo de ejecución del programa y el proceso de decodificación.

2.2.1. Flujo de ejecución del programa

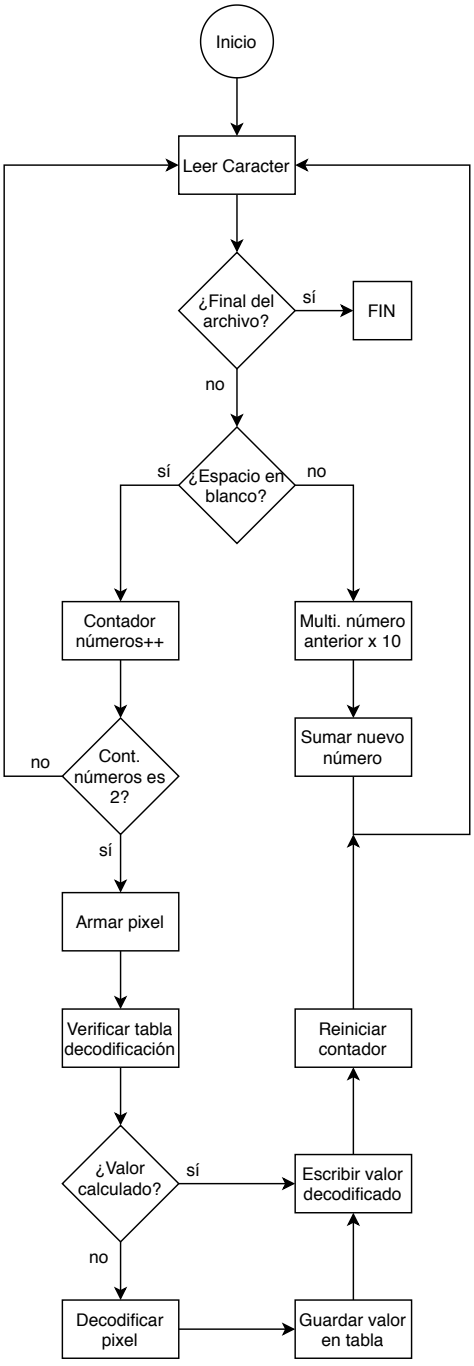

Figura 3: Diagrama de flujo del control del programa

En la figura 3 se puede observar el flujo del programa para la solución B. Se inicia leyendo un caracter del archivo de la imagen codificada, se verifica si se ha llegado al final del archivo, en este caso se finaliza el programa. Si no se ha llegado al final del archivo se procede con la lectura del número hasta encontrar un espacio en blanco, momento en el cual se aumenta un contador de números, si el contador de números es igual a dos, entonces se arma el valor del pixel codificado y se obtiene el valor decodificado correspondiente, este se escribe en el archivo de salida, se reinicia el contador de números, y se procede a leer el siguiente caracter para comenzar el proceso nuevamente.

2.2.2. Proceso de decodificación

En la figura 4 se puede observar un diagrama general del proceso de decodificación para la solución B. En este se plantea calcular y guardar en memoria todos los exponentes de dos hasta llegar a obtener cada posición de la representación binaria del exponente de la llave privada, para posteriormente cargar cada valor y verificar si es necesario utilizarlo para obtener el resultado final.

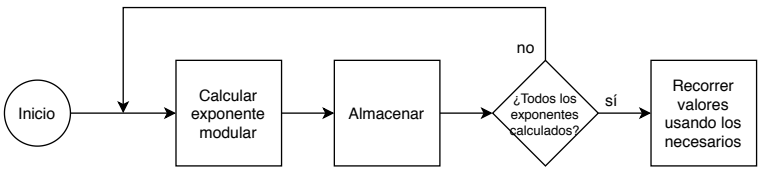

Figura 4: Proceso de decodificación de un pixel

3. Comparación de opciones de solución

En esta sección se realizará una comparación entre los planteamientos propuestos por la solución A y la solución B, tanto en la ejecución del flujo del programa como en el proceso de decodificación de los pixeles de la imagen encriptada. En general, ambas soluciones cumplen con los requerimientos que debe cumplir el proyecto, los cuales se pueden observar en la tabla 1.

3.1. Flujo de ejecución del programa

El flujo del programa de la solución A ofrece una división clara de etapas de lectura y decodificación, lo que facilita la modularización de estas dos secciones, sin embargo, tiene el inconveniente de almacenar en memoria todos los números que se encuentren en el archivo codificado, que en total son 614 400 valores de 1 byte cada uno, por lo que se ocuparía mucho espacio en memoria para guardarlos todos. Por otra parte, en el flujo del programa de la solución B, la etapa de lectura y decodificación se encuentran juntas, es decir, se realiza la lectura, decodificación y escritura del pixel en el momento que es obtenido, por lo que no es necesario almacenar su valor, lo que significa un uso más óptimo de la memoria con respecto a la solución A.

3.2. Proceso de decodificación

El proceso de decodificación de la solución B almacena los resultados intermedios que se obtienen en el algoritmo de exponenciación modular, el resultado final se empieza a procesar hasta completar el cálculo de todos los exponentes. Si se compara esta perspectiva con el proceso de decodificación de la solución A se puede apreciar que no es necesario guardarlos, si no que al momento de obtenerlos pueden utilizarse de una vez si así es requerido, optimizando el uso de la memoria.

4. Selección de propuesta final

Para la solución final se escogió el flujo de ejecución del programa propuesto en la solución B, ya que este tiene un uso más eficiente de la memoria del programa. Se seleccionó el proceso de decodificación de la solución A, ya que este es más eficiente en términos de memoria que el propuesto en la solución B.