Diseño e implementación de una aplicación de desencriptación RSA

Arturo Córdoba-Villalobos
email: arturocv16@gmail.com
Área Académica de Ingeniería en Computadores
Instituto Tecnológico de Costa Rica

***Abstract*—This document presents the desing of an assembler program which decrypts an image that was encoded using the RSA system, using the modular exponentiation algorithmn to handle the mathematic operations.**

***Palabras clave*—Encriptación, decodificación, RSA, ensamblador, x86-64.**

I. Introducción

En este documento se detalla el diseño realizado para una aplicación que desencripta imágenes que fueron codificadas con el sistema RSA, el cual encripta un mensaje a partir de una llave pública utilizando su representación numérica, en conjunto con las operaciones matemáticas de exponenciación y módulo, realiza una codificación que solo puede ser revertida con el uso de una llave privada [1]. El programa fue desarrollado en ensamblador x86-64, y se utiliza Octave para mostrar la imagen codificada y decodificada. Primero se explica el algoritmo desarrollado, donde se detallan los pasos que realiza el programa, cómo se obtiene un pixel codificado, y cómo se decodifica. Posteriormente, se muestran los resultados, donde se muestra una imagen codificada y la decodificación obtenida con el programa. Finalmente, se concluye en base a los resultados.

II. Algoritmo desarrollado

En esta sección se explicará el algoritmo que se desarrolló para dar solución al problema. La imagen encriptada debe estar en un archivo de texto, y cada pixel codificado se debe representar con dos números, donde el primero corresponde a los 8 bits más significativos, y el segundo a los 8 bits menos significativos. El usuario debe ingresar por consola la ruta de la imagen codificada, el exponente y el módulo, utilizando Octave se forma el comando que contiene el nombre del ejecutable y los argumentos, y se llama al sistema para empezar la ejecución del programa en ensamblador. Cuando inicia el programa de decodificación, se abre la imagen codificada y se empieza a leer un caracter a la vez. Para todas las lecturas realizadas, se verifica si se ha llegado al final del archivo, ya que esta es la condición de parada. Posteriormente, se verifica si el caracter leído es un espacio en blanco, ya que este es el delimitador entre los números codificados. Si no se trata de un espacio en blanco, se multiplica por diez el número que se haya construido hasta el momento, se suma el nuevo número y se vuelve a leer otro caracter para repetir el proceso. Si se trata de un espacio, se aumenta el contador de números el cual se utiliza para detectar que se han leído los dos números pertenecientes a un pixel codificado. Si este contador ha llegado a dos, se empieza la decodificación de un pixel, en caso contrario se lee otro caracter para repetir el proceso hasta obtener toda la información necesaria.

La decodificación de un pixel necesita de los dos números que lo representan, para armar el valor codificado se aplica un corrimiento lógico hacia la izquierda de ocho posiciones al número que representa los bits más significativos, y se le aplica un OR con el número que representa a los bits menos significativos. En este punto, se verifica en una tabla de decodificaciones si el valor ha sido calculado anteriormente. La distribución de memoria de esta tabla se muestra en la figura 1, el par de valores codificado y decodificado se almacenan en tres bytes, donde los primeros dos corresponden al número codificado y el último corresponde al número decodificado. Se necesitan 768 bytes para almacenar los 256 valores que puede tomar un pixel. Cuando se obtiene el valor de un pixel codificado, se verifica si su valor se encuentra en la tabla de decodificaciones, si es así se obtiene el respectivo valor decodificado y se escribe en el archivo de salida, si no, se aplica el algoritmo de exponenciación modular para decodificar el pixel.

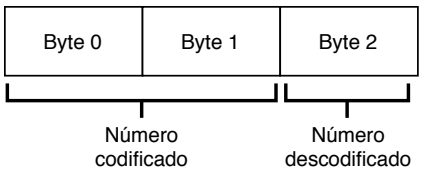

Fig. 1. Distribución de memoria en la tabla de decodificaciones

Para realizar la decodificación de un pixel se utiliza el algoritmo de exponenciación modular, el cual permite aplicar las operaciones que le dan el nombre de una manera rápida y eficiente. En este algoritmo se calculan las potencias de dos de la base aplicándole el módulo, hasta obtener todas las necesarias para obtener el exponente de la llave privada [2]. Utilizando el valor binario del exponente de la llave privada se determinan cuáles potencias deben ser utilizadas, que son aquellas donde se tiene un valor de uno. Los valores que se deben utilizar deben ser multiplicados entre sí para obtener el resultado final [3]. Este algoritmo se implementó en un ciclo, se tiene un registro donde se va almacenando el resultado,

otro donde se almacena el exponente de la iteración, otro donde se almacena el exponente de la llave privada, y otro donde se almacena el módulo. En cada iteración se verifica si el resultado del módulo debe ser utilizado para obtener el resultado final, si es así entonces se multiplica su valor con el registro que va almacenando el resultado final. Al finalizar cada ciclo se multiplica el exponente por dos y el resultado del módulo por sí mismo, para obtener la siguiente potencia. La condición de parada es cuando el exponente de la iteración es mayor al exponente de la llave privada. Una vez decodificado el pixel, se escribe en el archivo de salida y se guarda el par codificado-decodificado en la tabla.

Una vez que se han decodificado todos los pixeles y se ha llegado al final del archivo de la imagen encriptada, Octave lee el archivo de salida generado por el programa en ensamblador y le muestra el resultado final al usuario.

III. Resultados

En la figura 2 se puede observar la imagen codificada, tiene una dimensión de 640 x 960 pixeles. El resultado obtenido luego de desencriptar utilizando el programa escrito en ensamblador se muestra en la figura 3, se puede apreciar que la decodificación fue exitosa.

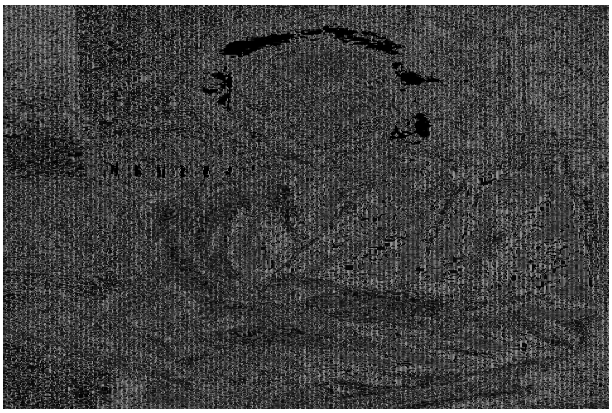

Fig. 2. Imagen codificada

En la tabla I se puede ver una comparación del tiempo que le tomó al programa en ensamblador decodificar la imagen, utilizando la tabla de decodificaciones y sin utilizar esta tabla. Se obtuvo un promedio de 4.156 segundos para las ejecuciones sin utilizar la tabla, y un promedio de 3.974 segundos para las ejecuciones con la tabla. Esto representa una mejora del 4.38%, la cual puede llegar a ser significativa si se utiliza un volumen de datos mucho mayor.

N.	**Tiempo [s]**	
	Sin tabla	**Con tabla**
1	3.982	3.991
2	4.171	3.923
3	4.187	3.901
4	4.196	3.920
5	4.244	4.134
Promedio	4.156	3.974

Tabla I
COMPARACIÓN DE TIEMPO CON Y SIN TABLA DE DECODIFICACIONES

Fig. 3. Imagen decodificada

Con base en los resultados obtenidos, se puede asegurar que el diseño cumple con la especificación del problema, ya que logra decodificar de manera exitosa una imagen de 640 x 960 pixeles, utilizando RSA y el algoritmo de exponenciación modular, en un tiempo aceptable. Además, el propósito de la tabla de decodificaciones se cumplió, ya que logró reducir el tiempo que le toma al programa obtener el resultado.

IV. Conclusiones

El diseño propuesto para dar solución al problema fue exitoso, ya que se consiguió decodificar la imagen consiguiendo el resultado esperado. Se logró implementar el algoritmo de exponenciación modular y la búsqueda de valores procesados en la tabla de decodificaciones, consiguiendo que el programa se ejecute de manera óptima sin realizar un desperdicio de memoria o de tiempo.

La aplicación de algoritmos como el de exponenciación modular en un programa computacional es fundamental cuando se quiere obtener un rendimiento aceptable, en el caso de esta aplicación, si no se hubiera aplicado este algoritmo la decodificación hubiera sido mucho más compleja, ya que los números no podrían ser representados en los 64 bits que tienen los registros de x86-64, por lo que hubiera sido necesario utilizar otras técnicas que provocarían que el programa fuera mucho más lento con un mayor uso de la memoria.

Es fundamental estar consciente de las restricciones y estándares que tienen las herramientas utilizadas en el diseño e implementación de soluciones, en el caso de utilizar herramientas como x86-64 donde la escritura de código se vuelve

más compleja debido al bajo nivel, es muy importante mantener un código limpio que facilite la identificación de errores que provoquen un comportamiento no esperado.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, “A method for obtaining digital signatures and public-key cryptosystems,” vol. 21, no. 2, pp. 120–126, 1978.

[2] GVSUmath. ”Modular Exponentiation,” *YouTube*, Oct. 28, 2014 [Video file]. Available: https://www.youtube.com/watch?v=EHUgNLN8F1Y. [Accessed: Oct. 15, 2020].

[3] Heyman, R. ”Modular Exponentiation Made Easy,” *YouTube*, Aug. 2, 2015 [Videl file]. Available: https://www.youtube.com/watch?v=tTuWmcikE0Q. [Accessed: Oct. 15, 2020].