TEC | Tecnológico de Costa Rica

INSTITUTO TECNOLÓGICO DE COSTA RICA

Área Académica de Ingeniería en Computadores

Arquitectura de Computadores I

The Imitation Game

Diseño e Implementación de un ASIP de desencriptación mediante RSA

Estudiante:
José Julián Camacho Hernández

Profesor:
Luis Alberto Chavarría Zamora

24 de marzo de 2023

Tabla de contenidos

1. Documentación de diseño

A continuación, se presenta la documentación relacionada con el diseño e implementación de la aplicación de desencriptación de imágenes por medio de RSA.

1.1. Listado de requerimientos del sistema

A partir de la lectura y el análisis de la especificación del problema, se obtienen la siguiente lista de requerimientos:

- Se debe diseñar y desarrollar en ensamblador, ya sea ARM, x86, RISC-V, entre otros ISA, un programa que desencripte información mediante el sistema criptográfico RSA.
- El programa debe recibir únicamente la llave privada para la desencriptación del mensaje, es decir los parámetros d y n, y esta puede ser introducida por medio de consola o de un archivo llamado **llaves.txt**.
- Como entrada, el programa debe tomar una imagen encriptada para ser procesada, en formato de texto donde los datos están en decimal de un *byte* cada uno.
- El programa debe procesar una imagen de 640×480 para desencriptarla con una llave privada. La imagen es en escala de grises con píxeles con valores entre [0, 255] y la representación de cada píxel está dada por 2 *bytes* sucesivos, por lo que la imagen encriptada se debe observar como una de 640×960.
- En la actualidad, es importante que los valores de llaves utilizados en el algoritmo RSA sean elevados con el fin de garantizar una mejor seguridad [1]. Se recomienda que sean de 1024 o 2048 bits a lo sumo. Por esta razón, para la desencriptación el programa debe seguir un algoritmo para realizar de manera óptima cálculos de resultados altos. Para esto, es necesario aplicar la aritmética modular, en específico la exponenciación modular, con el fin de llevar a cabo operaciones como la que se presenta a continuación de forma eficiente:

  $$m = c^d mod(n)$$

  Donde c, d y n se encuentran en el rango [0, 65535].
- El programa debe desplegar al usuario final una cuadrícula de 1×2, es decir, una fila y dos columnas, donde la izquierda sea la imagen de entrada que se encuentra encriptada y la derecha contenga la imagen una vez fue procesada o desencriptada. Para esta visualización final, se debe utilizar algún software de alto nivel como lo puede ser Python, Matlab, Octave, entre otros.
- Una de las deventajas que posee el algoritmo RSA es que puede ser lento respecto a otros de su tipo [1], por lo que es necesario implementar una optimización del código de desencriptación, por medio de alguna estructura que almacene resultados anteriores, con el fin de no volver a calcularlos.

- Automáticamente, el programa debe generar un archivo de salida que consiste en los valores de los pixeles en escala de grises que conforman la imagen desencriptada.
- Los programas que toman la imagen en formato de texto, la procesan en ASM y permiten la visualización al usuario deben estar integrados en un solo *framework*.

1.2. Elaboración de opciones de solución al problema

Seguidamente, se presentan dos de las opciones de solución que fueron planteadas para la resolución del problema.

1.2.1. Primera opción de solución

En la figura 1 se presenta el diagrama de la primera opción de solución.

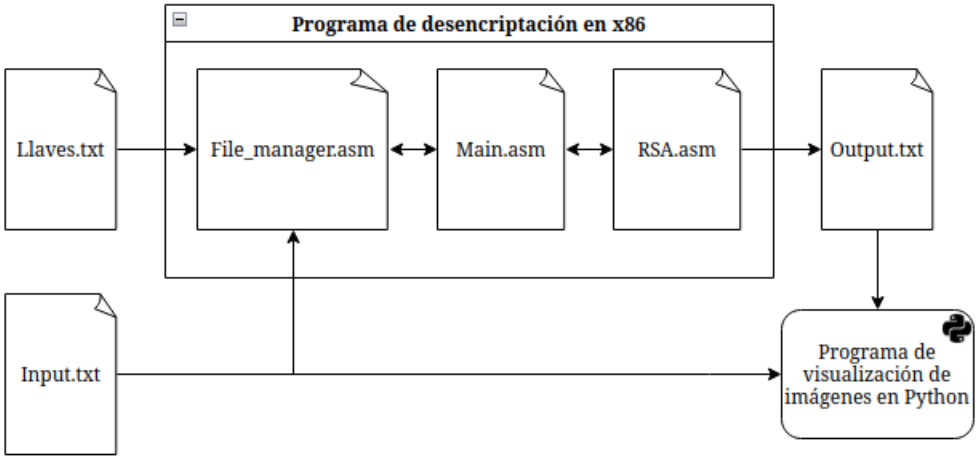

Figura 1: Diagrama de la primera opción de solución.

Como se observa, esta solución está compuesta por los siguientes bloques generales:

- **Programa de desencriptación en x86:** Este programa es el encargado de realizar el procesamiento de la imagen. Para ello, debe tomar los archivos `llaves.txt` e `Input.txt` y manejarlos por medio del programa `File_manager.asm`. Esto para obtener los pixeles y operarlos con las llaves mediante el algoritmo RSA en el archivo de ese mismo nombre. El resultado de esto, se escribe en el archivo de salida `output.txt`.

  Todo el procesamiento descrito se realiza en el lenguaje ensamblador elegido para esta etapa, que es x86 de 32 bits. Este ISA es de tipo CISC, por lo que presentará ciertas ventajas como la facilidad de programación al contar con gran variedad de instrucciones que son complejas y poderosas, que pueden simplificar la legibilidad del código al tener menos cantidad de líneas y por ende el desarrollo del algoritmo.

  Otro de los aspectos que se tomaron en cuenta al seleccionar este ISA, es que es de tipo *Register/Memory*, por lo que tiene la capacidad de realizar operaciones directamente

en memoria. Esto permitirá la definición de variables y su operación sin tener que cargarlas previamente a registros de propósito general.

Relacionado con el punto anterior, otro de los factores positivos de x86 es que cuenta con múltiples modos de direccionamiento, por lo que acceder a espacios de memoria como variables puede ser más conveniente. De igual manera, la facilidad en el uso del *stack* mediante operaciones como *push* y *pop*, es importante para el almacenamiento de resultados anteriores.

- **Programa de visualización de imágenes:** Este *script* es el encargado de manejar los archivos de entrada y de salida para procesar los valores de píxeles que contienen y así desplegar las imágenes tanto encriptada como desencriptada en una cuadrícula de 1×2.

  Se elige este lenguaje y en específico la biblioteca *TKinter* debido a que ofrecen las herramientas y funciones necesarias para la lectura de los archivos y visualización de las imágenes en una nueva ventana.

1.2.2. Segunda opción de solución

Por otro lado, la segunda opción de solución se muestra en el siguiente diagrama de la figura 2.

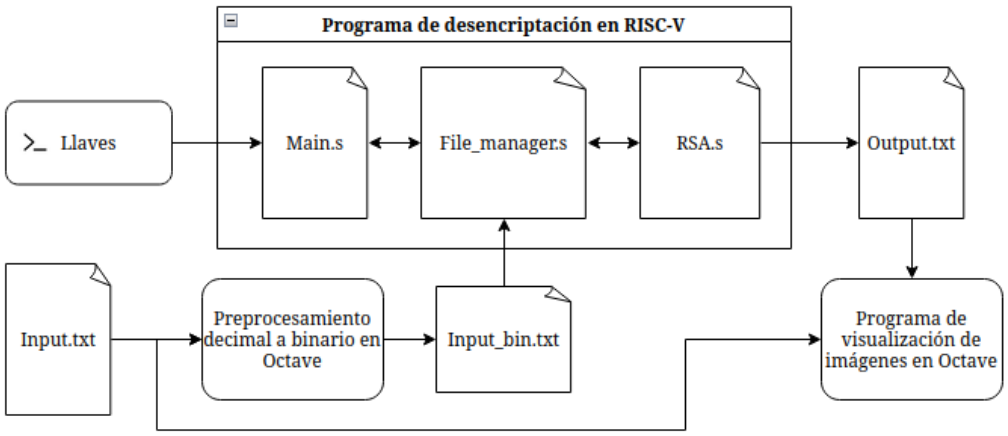

Figura 2: Diagrama de la segunda opción de solución.

Las partes del diagrama se detallan a continuación:

- **Programa de conversión decimal a binario:** Este programa de Octave tiene la función de llevar a cabo un pre-procesamiento del archivo de entrada. Consiste en tomar cada número separado por espacio en el archivo `input.txt` y convertir los valores a binario por medio de la función `dec2bin()` de Octave, para así generar el archivo `input_bin.txt`. Esto se propone con el objetivo de facilitar la lectura del archivo en ensamblador, ya que se sabrá de antemano que cada número estará compuesto de exactamente 8 bits.

- **Programa de desencriptación en RISC-V:** Este es el programa en ensamblador RISC-V que será el encargado de procesar la imagen encriptada. Para ello, se propone que tome los valores de la llave privada por medio de consola. Posteriormente, obtendrá los valores de píxeles en binario del archivo `input_bin.txt` para así iniciar el procesamiento y las operaciones por medio del algoritmo de RSA. Los valores de píxeles desencriptados serán guardados en el archivo `output.txt` según se detalla en los requerimientos.

  La selección de este ISA se basa en las características que ofrece por ser de tipo RISC. Entre ellas, se destaca la simplicidad que puede tener el uso de una gran cantidad de registros de propósito general. A pesar de que por ser de tipo RISC el tamaño y la dificultad de código puede ser mayor, se puede sacar provecho del set reducido de instrucciones y programar funcionalidades complejas al unirlas de manera adecuada.

- **Programa de visualización de imágenes:** La visualización de las imágenes se realizará por medio de un *script* de Octave. Se elige porque presenta gran facilidad de cargar los archivos y generar una imagen de salida a partir de píxeles.

1.3. Comparación de opciones de solución

Las soluciones planteadas están compuestas por diferentes aspectos, entre ellos el lenguaje ensamblador en el cual se llevará a cabo la desencriptación de la imagen. La diferencia fundamental entre ambos lenguajes propuestos radica en que el primero, x86, es de tipo CISC, mientras que RISC-V, como su nombre lo indica, es de tipo RISC. Esto genera a su vez gran cantidad de diferencias, ventajas y desventajas entre ellos.

Entre estas, se toma en consideración que x86 proporciona un gran y poderoso conjunto de instrucciones, aspecto que puede resultar ventajoso para simplificar la implementación del código. Esto a diferencia de RISC-V, ya que este solo ofrece un set reducido de instrucciones, por lo que programar una funcionalidad conllevará más lógica e instrucciones en caso de elegir la segunda solución.

Por ejemplo, se identifica que para el algoritmo RSA será necesario contar los bits significativos de la llave *d*. Dicho objetivo se puede realizar en x86 con solo la instrucción `bsr`, mientras que en RISC-V se debe utilizar `clz` y posteriormente realizar una resta para obtener el mismo resultado. De igual forma, x86 proporciona instrucciones como *push* y *pop*, mientras que en RISC-V se deben realizar sumas o restas al puntero del *stack* para acceder a la pila.

Adicionalmente, otro de los aspectos que diferencian estos dos tipos de arquitecturas es la cantidad de registros de propósito general. Para este caso, se analiza que x86 cuenta con pocos, mientras que RISC-V ofrece gran cantidad, por lo que sería una mejor opción para realizar operaciones y guardar resultados temporales de manera más eficiente. Pero por su lado, x86 tendría la ventaja de poder realizar operaciones directamente a memoria, aspecto que simplifica el hecho de no tener que realizar *load/store*, pero puede ser un poco menos eficiente debido a que las operaciones en registros son más rápidas que consultar la memoria.

Una de las diferencias entre las opciones de solución es la manera de obtener los valores de la llave privada. En la primera propuesta, se plantea realizarlo por medio de la lectura de un archivo. x86 posee la ventaja de que proporciona las herramientas para hacer este

tipo de operación de manera sencilla. Por otro lado, en la segunda se propone llevarlo a cabo mediante la introducción de estos valores en consola. Ambas opciones son viables, no obstante, se podría sacar provecho de las funciones de lectura en x86 y simplemente adaptar el código de lectura del archivo de entrada para hacerlo también con el archivo de llaves.

Otro de los aspectos diferenciadores entre ambas soluciones, es que la segunda conlleva un procesamiento previo del archivo de entrada. Esto puede generar ventajas al momento de leerlo, ya que se puede definir un tamaño de lectura fijo, sin tener que verificar la cantidad de caracteres por leer o el espacio como delimitador. En la primera solución se puede prescidir de este pre-procesamiento, ya que x86 proporciona la posibilidad de utilizar funciones de C como `atoi` para convertir números a decimal directamente, pero tendrá la parte negativa de tener que programar más lógica para la lectura.

Respecto a los lenguajes propuestos para el desarrollo de los programas de visualización de las imágenes, se puede apuntar que tanto Octave como Python pueden cumplir satisfactoriamente esta tarea. Ambos proporcionan funciones de bibliotecas que son sencillas de utilizar por lo que el desarrollo del programa de alto nivel no será de gran complejidad.

1.4. Selección de la propuesta final

Al analizar las comparaciones planteadas respecto al ISA con el cual se realizará el procesamiento, se elige x86 debido a que puede simplificar el desarrollo y la legibilidad del código con su amplio set de instrucciones. Esto tomando en consideración que no se cuenta con gran experiencia en programación en ensamblador.

También, como la eficiencia del programa no es un factor crítico, se considera que se puede trabajar con variables desde memoria y se intentará trabajar con los pocos registros que se tiene a disposición para acceder a ellas la menor cantidad de veces posible.

Otra de las desventajas de x86 es el consumo de potencia mayor respecto a RISC-V. Sin embargo, como este tampoco es un factor determinante, se considera la utilización del primero.

Otro de los aspectos importantes a tomar en consideración es el *hardware* de destino para el que se desarrolla la aplicación. Como el proyecto será implementado para ejecutarse sobre una computadora portátil con procesador Intel x86, se considera que el desarrollo en este lenguaje será más óptimo, ya que si se elige RISC-V será necesario utilizar algún emulador o recurrir a algún *toolchain* para realizar compilación cruzada.

Una de las grandes ventajas que proporciona RISC-V es que su comercialización es gratis, no obstante, como el proyecto es de carácter académico se selecciona x86.

Respecto al método de obtención de la llave privada, como se expuso en la comparación, se concluye que como de igual manera se debe implementar la lectura de archivos, esta se puede reutilizar, por lo que no es necesario programar funcionalidad extra para la lectura de consola.

Sobre el tema del pre-procesamiento, al elegir x86 se descarta este paso y se selecciona la opción de aumentar la lógica de lectura y la utilización de funciones de conversión de ASCII a decimal o viceversa.

Finalmente, respecto a los lenguajes de alto nivel, en vista de que ambas opciones ofrecen capacidades y complejidad similares, se considera tomar Python ya que es un lenguaje conocido por el desarrollador.

Por lo tanto, una vez se presentaron y se compararon las opciones de solución, y se analizaron considerando diversos aspectos, se selecciona la primera opción como la solución final.

Referencias

[1] GeeksforGeeks. *RSA Algorithm in Cryptography*. URL: `https://www.geeksforgeeks.org/rsa-algorithm-cryptography/`.