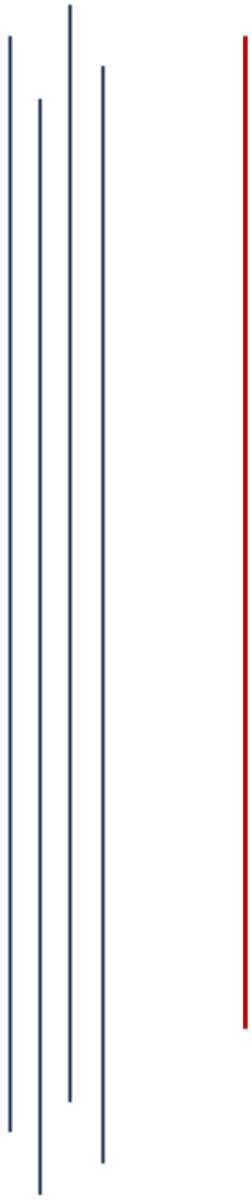
TEC | Tecnológico de Costa Rica

Proyecto Individual

The Imitation Game: Diseño e Implementación de un ASIP de desencriptación mediante RSA

CE4301 Arquitectura de Computadores I
Prof. Luis Alberto Chavarría Zamora



José Alejandro Chavarría Madriz
2019067306

Semestre I
2023

Proyecto Individual

The Imitation Game: Diseño e Implementación de un ASIP de desencriptación mediante RSA

José Alejandro Chavarría Madriz

28 de marzo de 2023

1 Introducción

En este documento se desarrolla la documentación de diseño del proyecto individual. Aquí se discute un análisis detallado respecto al proceso de diseño del proyecto enfocado a través de: los requerimientos del sistema desde una perspectiva de ingeniería, una evaluación objetiva de opciones para la resolución del problema, así como una comparación de las mismas y finalmente una explicación a detalle de la forma en que se soluciona el problema planteado por el proyecto. El cual consiste en desarrollar, mediante algún ISA a escoger, el algoritmo de desencriptación RSA. Dicho algoritmo será utilizado para desencriptar una imagen de 640x480 píxeles la cual ha sido encriptada y escrita en un archivo .txt que contiene la información de 640x960 píxeles. (Dado que la encriptación RSA duplica la cantidad de información de la imagen).

2 Listado de requerimientos

El proyecto en general consiste en dos partes principales. El procesamiento de la imagen realizado completamente en algún lenguaje ensamblador (x86, RISC-V, ARM), y una interfaz gráfica (desarrollada en cualquier lenguaje de alto nivel) donde se pueda seleccionar y visualizar la imagen encriptada, indicar la llave privada y ver los resultados de la imagen al ser desencriptada.

A continuación se listan los requerimientos específicos del sistema:

1. Interfaz:
    - Debe mostrar tanto la imagen encriptada como la imagen desencriptada una al lado de la otra.
    - Debe ser capaz de definir el archivo .txt con la imagen desencriptada.
    - Debe poder indicar la llave con la que se va a desencriptar la imagen.
    - Debe poder llamar el proceso de desencriptación escrito en ensamblador.
    - Puede realizar cualquier preprocesamiento necesario al .txt antes de llamar al proceso de desencriptación.
    - Debe notificar al usuario de cualquier error que haya ocurrido.
    - Debe asegurarse que las imágenes y el programa puedan visualizarse correctamente. Ya sea escalando las imágenes o adaptando la interfaz.

2. Procesamiento:

    - Debe poder extraer los datos encriptados de la imagen desde un archivo.
    - Debe poder escribir los datos desencriptados de la imagen de tal forma que la interfaz pueda mostrarlos.
    - Debe aplicar eficientemente exponenciación modular para desencriptar cada píxel.
    - Debe ser capaz de desencriptar imágenes de tamaño 640x960, para producir imágenes de 640x480 píxeles. Ambas en escala de grises.
    - Debe de ser capaz de extraer la información de los 2 bytes que se desencriptarán para formar un único píxel de un byte.
    - Debe recibir los datos de la llave privada proveídos por la interfaz.
    - Debe de implementar algún tipo de optimización, sabiendo que no se desencriptarán más de 255 valores distintos.

3 Evaluación de opciones

3.1 Solución #1

La primera solución planteada hace uso de x86 como ISA de preferencia. La idea de esta solución es hacer uso de las facilidades que brinda un ISA CISC al programador. Por esta razón, esta solución propone la implementación de un algoritmo más complejo para la realización de la exponenciación modular, la exponenciación binaria o exponenciación rápida. Tomando en cuenta la eficiencia del algoritmo, esta solución prescinde de algún tipo de optimización mediante el guardado de los datos desencriptados, ya que lo considera como una ganancia despreciable y por lo tanto un aumento innecesario en la complejidad del código, que de otra forma puede ser sumamente conciso y puntual al aprovechar capacidades del ISA CISC. Otra característica importante a que apuesta esta solución es que la mayoría de los computadores actuales utilizan x86, por lo que lo convierte en un programa fácilmente portable a cualquier computador cotidiano moderno, y evita el uso de emulación que pueda entorpecer el proceso de desarrollo.

Un diagrama de la solución se puede apreciar en la figura 1.

3.2 Solución #2

La segunda solución planteada hace uso de RISC-V como ISA de preferencia (específicamente RV32I). Esta solución busca una implementación del programa para microarquitecturas más básicas aprovechando un ISA RISC. Esta solución no puede depender de instrucciones complejas o de capacidades extendidas del ISA por lo que para realizar la exponenciación modular se vale del algoritmo de exponenciación binaria, pero tomando las consideraciones necesarias. Ya que estas consideraciones provocan la aparición de instrucciones adicionales, esta solución implementa un ”pseudo-caché” un espacio de memoria reservado donde guardar el valor de los datos que han sido desencriptados, de donde pueden ser obtenidos sin necesidad de realizar el procesamiento de nuevo. Para una mayor optimización este pseudo-caché hace uso de los 4 bytes en cada celda de memoria. Donde los 2 primeros son utilizados para guardar el valor sin desencriptar y el último guarda el valor desencriptado. Esta solución apuesta a que los accesos de memoria vayan a ser más rápidos que la ejecución de la aritmética modular en una microarquitectura y un ISA que no están optimizados para esta clase de operaciones.

Un diagrama de la solución se puede apreciar en la figura 2.

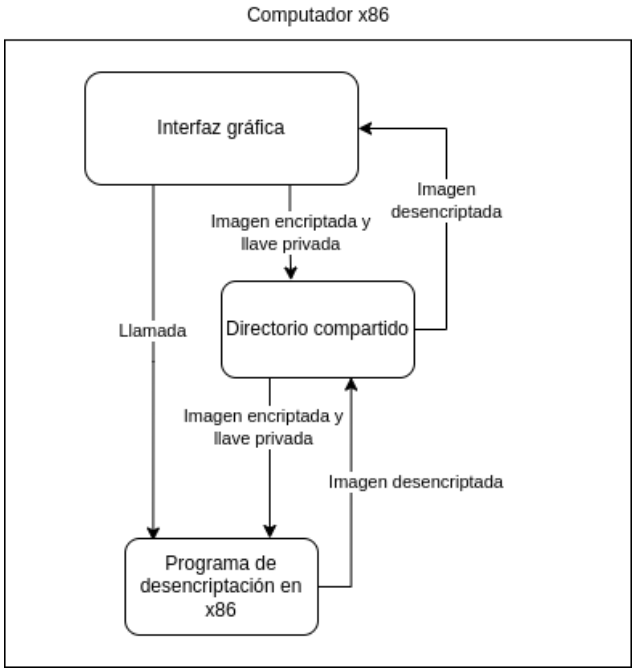


Figura 1: Solución 1. Elaboración propia

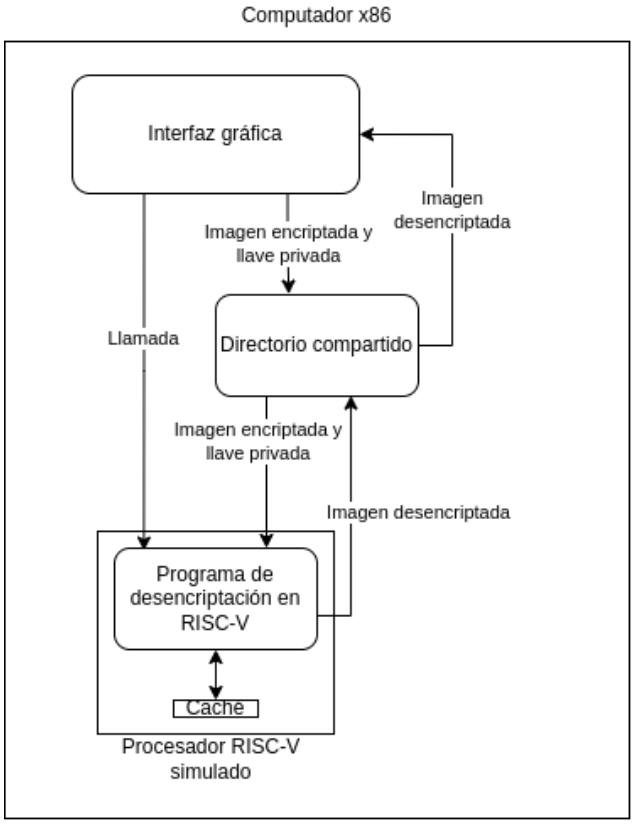


Figura 2: Solución 2. Elaboración propia

4 Comparación de opciones

Ambas opciones se ajustan a un caso de aplicación distinto resolviendo el mismo problema. La solución expuesta en 3.1 está pensada para ejecutarse dentro de computadores de uso común, aprovechando el poder de un ISA CISC y de una microarquitectura potente. Esta solución también puede aprovechar ventajas como una única integración completa del sistema, ya que tanto la desencriptación como la interfaz gráfica pueden ser ejecutadas nativamente como un único framework. Además como se menciona en el planteamiento de la solución, x86 permite un nivel de portabilidad más alto que la solución descrita en 3.2, que utiliza un ISA open-source menos común. Sin embargo, esta segunda solución ataca el problema enfocándose en la resolución del cómputo en sistemas más básicos, implementa soluciones a posibles problemas de eficiencia que microarquitecturas pequeñas que utilicen RISC-V puedan tener, este enfoque le permite ejecutarse en dispositivos especializados. Esta segunda solución encuentra dificultades para integrarse en un único framework ya que el procesamiento de la imagen debe realizarse en hardware simulado o externo al, presumiblemente, computador x86 donde se mostrará la interfaz gráfica, sin embargo ofrece la ventaja de, en caso de ejecutarse en hardware externo, eximir al procesador de la computadora de efectuar el cálculo y más bien delegarlo a un dispositivo externo. Ambas soluciones hacen uso de la exponenciación binaria para realizar el cálculo, pues es una de las maneras más eficientes tanto computacionalmente como en consumo de memoria que hay, sin ser mucho más complicada de implementar que otros métodos. (En la figura 3 se muestra el código en C#). Queda claro que ambas son opciones que pueden resultar mejores o peores dependiendo del caso de aplicación, así como la habilidad del programador para tratar con ISAs RISC o CISC de manera que logre extraer las capacidades de cada uno, así como la elección correcta respecto al uso deseado de sistema.

```
Bignum modpow(Bignum base, Bignum exp, Bignum m) {

   Bignum result = 1;

   while (exp > 0) {
      if ((exp & 1) > 0) result = (result * base) % m;
      exp >>= 1;
      base = (base * base) % m;
   }

   return result;

}
```

Figura 3: Código de exponenciación rápida en C# Schneier 1995

5 Propuesta final

La selección de la solución final debe tomar en cuenta todos los requerimientos listados en la sección 2, así como el caso de aplicación que mejor se ajuste al contexto del problema. Bajo estas consideraciones la solución escogida se basa primariamente en la solución expuesta en 3.1. Esto debido a que el problema está planteado para ser resuelto en una computadora personal, x86 presenta la mejor opción ya que en la mayoría de los casos se evitará el uso de simuladores, lo que permite que el proceso corra nativamente en la microarquitectura del procesador, generando así ganancias en el desempeño del sistema en general. Además, esto también permite una mejor integración entre el sistema de interfaz y el proceso de desencriptación. Dicho sea de paso que

Tamaño de imagen	Tiempo promedio sin caché [s]	Tiempo promedio con caché [s]	% de mejora
320x320	1.11897	1.0445	6.655
640x480	3.4714	3.20208	7.758

Cuadro 1: Comparación entre ejecuciones con y sin caché implementado. Elaboración propia. Ver anexo A

para la interfaz se ha utilizado Python con la biblioteca TKinter, debido a su simplicidad, de uso y su portabilidad a diferentes sistemas. Sin embargo con el afán de exprimir la mayor cantidad de eficiencia posible la solución final también adopta un aspecto de la solución de 3.2, el uso de una ”pseudo-caché”. Se analiza que aunque el nivel de eficiencia de la primera solución puede ser alto, esta si puede beneficiarse de una optimización como esa. En el cuadro 1 se pueden observar datos recopilados utilizando o no la optimización. A pesar de que la ganancia es pequeña, prueba que existe una oportunidad de mejora. Se cree que el aspecto a mejorar principal en el pseudo-caché es la indexación secuencial. Optimizaciones posteriores aprovecharían el dato encriptado para buscar inmediatamente el valor de desencriptación, en vez de recorrer uno por uno los datos que se han almacenado.

Referencias

Schneier, B (1995). *Applied Cryptography*.

A Tiempos de ejecución

320x320	
Tiempo sin caché [s]	Tiempo con caché [s]
1.0978	1.0821
1.0251	0.9991
1.0179	1.1401
1.0066	1.0315
1.1674	1.1433
1.1541	1.0965
1.1181	1.0134
1.2245	0.9879
1.1841	0.9498
1.1941	1.0013

640x480	
Tiempo sin caché [s]	Tiempo con caché [s]
3.2649	2.9531
3.7279	3.178
3.8061	3.2266
3.066	3.2189
3.4921	3.4338
3.4714	3.20208