TEC | Tecnológico de Costa Rica

Instituto Tecnológico de Costa Rica
Área Académica de Ingeniería en Computadores
CE 4301 — Arquitectura de Computadores I

Proyecto Grupal 1

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

Fecha de asignación:	14 de septiembre de 2022	Fecha de entrega:	21 de octubre de 2022
Grupos:	3-4 personas	Profesor:	Luis Chavarría Zamora

Mediante el desarrollo de este proyecto, la/el estudiante aplicará los conceptos de arquitectura de computadores en el diseño e implementación en hardware de un *Application Specific Instruction Set Processor* (ASIP) para desencriptación de RSA. Atributos relacionados: **A**nálisis de **P**roblemas (AP), el cual se encuentra en **A**vanzado (A).

1. Descripción General

El RSA (Rivest-Shamir-Adleman) es un sistema criptográfico de clave pública desarrollado en 1979. Es asimétrico, lo que significa que cuenta con una llave pública (Public Key) y una llave privada (Private Key). El funcionamiento se muestra a continuación:

1. Un cliente (e.g., buscador) envía su llave pública a un servidor y solicita información. El cliente también tiene una llave privada que solo tiene él (esta se debe guardar celosamente).
2. El servidor encripta la información usando la llave pública y se la envía al cliente.
3. El cliente recibe la información y la desencripta usando la llave privada.

El funcionamiento a nivel de esquema de las llaves públicas y privadas se muestra en la Figura 1. Como se observa, cualquier ente que tenga acceso a la llave pública puede encriptar un mensaje, pero no cualquiera puede desencriptarlo pues necesita la llave pública. En este vídeo se encuentra una explicación interactiva del algoritmo RSA.

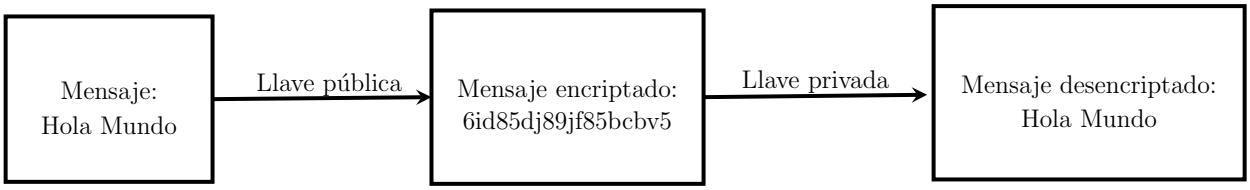

Figura 1: Esquema de encripción y desencripción

Para encriptar un mensaje se tiene que realizar la operación

$$c = m^e \mod n, \tag{1}$$

donde m es el mensaje, c es el mensaje encriptado y (e y n) son los parámetros de la llave pública.

Para desencriptar el mensaje se tiene que realizar la operación

$$m = c^d \mod n, \tag{2}$$

donde c es el mensaje encriptado, m es el mensaje desencriptado o recuperado y (n[1] y d) son los parámetros de la llave privada.

Para la generación de llaves se usa el siguiente algoritmo:

1. Seleccione dos números primos aleatorios (p y q). Por ejemplo 7 y 2.
2. Calcule $n = pq$. Este es el módulo de la llave pública y privada. Por ejemplo, en este caso $n = 7 \times 2 = 14$.
3. Calcule la función de Euler $\phi = (p-1)(q-1)$. Por ejemplo, en este caso $\phi = (7-1)(2-1) = 6$.
4. Se selecciona un entero positivo e menor que ϕ (no puede ser igual), este debe ser coprimo a ϕ. Este es el exponente de la llave pública. Un número es coprimo de otro si el máximo común divisor es igual a 1. Dos números coprimos no tienen que ser primos, por ejemplo 6 y 19 son coprimos (19 es primo, 6 no es primo). Si e es muy pequeño podría suponer una vulnerabilidad importante para el sistema. Por ejemplo, en este caso $e = 5 < \phi = 6$.
5. Se selecciona un entero positivo d que satisfaga: $de \mod \phi = 1$. Por ejemplo, en este caso $de = 11 \times 5 = 55 \rightarrow 55 \mod 6 = 1$. Entonces $d = 11$ cumple el requerimiento.

De esta manera se genera el siguiente par de llaves:

- Llave pública o de encripción: (5,14).
- Llave privada o de desencripción: (11,14).

Se pueden encriptar todos los números mayores a 1 [2]. Por ejemplo se encriptará el número 3 con las llaves públicas y privadas expresadas anteriormente

$$c = m^e \mod n = 3^5 \mod 14 = 243 \mod 14 = 5.$$

Para desencriptar se muestra la operación

$$m = c^d \mod n = 5^{11} \mod 14 = 48828125 \mod 14 = 3,$$

[1] note que este parámetro n es el mismo que el de la llave pública.
[2] dado (1) y (2): $1^x = 1 \forall x \in R$

se verifica como se obtiene $m = 3$ tanto en el mensaje original como en la desencripción. También resulta bastante evidente que como la cantidad de números a encriptar es relativo al módulo.

Es evidente como la dificultad del algoritmo RSA radica en el manejo de números extensos esto se hace evidente viendo las operaciones involucradas en (1) y (2). No es un secreto que entre más extensos sean los números más difícil es encontrar la combinación llave pública y privada. Por esta razón se usan llaves públicas y privadas de 1024 bits ($\approx 1{,}8 \times 10^{308}$ combinaciones, necesitando 309 dígitos en base decimal), 2048 bits ($\approx 3{,}2 \times 10^{616}$ combinaciones, necesitando 617 dígitos en base decimal) o incluso más. En este enlace hay un generador de llaves RSA.

Se vuelve bastante claro como no se pueden procesar las operaciones exponente y modular de forma tradicional, sino el sistema requeriría mucha memoria o sería muy lento. Por esta razón se requiere el uso de una ASIP especializada como la que se realizará en este proyecto.

2. Especificación

Se le solicita desarrollar la arquitectura y microarquitectura que realice la desencripción de un mensaje de n caracteres que ya fueron cargados previamente en el diseño (el profesor dará los datos minutos antes de la defensa). La salida del sistema será interpretada por un elemento interprete (por ejemplo un Arduino) que recibirá el mensaje ya desencriptado en 8 bits y un reloj, como se observa en la Figura 2. El elemento interprete se conecta con el computador para ir mostrando el mensaje en pantalla.

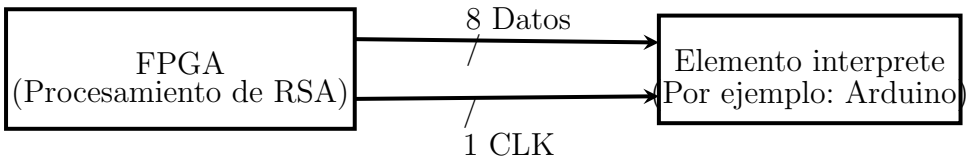

Figura 2: Esquema de encripción y desencripción

Como el problema usa números bastantes extensos no se permite el uso de operaciones tradicionales, sino módulos de aritmética modular. El profesor le dará los siguientes archivos:

- Una mensaje encriptado de 100 caracteres con representación de 2 bytes cada caracter o dato [3] con una llave pública,

[3] el dato encriptado es de 8 bits, pero como el n de la llave pública y privada determina el rango de números a encriptar, se procuró un $256 < n < 65536$ o n representable con a lo sumo 16 bits, esto también para guardar el alineamiento de la memoria

- La llave pública.
- La llave privada.

De esta manera cada grupo recibirá una serie de datos incomprensibles que luego de ser procesados generan un mensaje comprensible.

1. El diseño completo debe poder ser sintetizable en una tarjeta de desarrollo Terasic DE10-Nano.
2. Las datos de entrada y la salida debe ser almacenado en memoria.
3. El ISA debe ser eficiente y congruente, con criterios de diseño definidos. Indiquen a la mayor brevedad si necesitan reuniones con el profesor para guía.
4. La representación es en ASCII.
5. El sistema (FPGA) debe permitir la interacción del usuario para indicar cuando ejecutar el desencriptado de la imagen.
6. El sistema (FPGA) debe indicar cuando terminó de desencriptar el mensaje.

Requisitos de Arquitectura ISA:

1. Debe diseñar un conjunto de instrucciones y arquitectura que permita solucionar el problema planteado, considerando detalles como:

   a) Modos de direccionamiento.
   b) Tamaño y tipo de datos.
   c) Tipo y sintaxis de las instrucciones.
   d) Registros disponibles y sus nombres.
   e) Codificación y descripción funcional de las instrucciones

   Tome en cuenta que estos detalles deben ser justificados desde el punto de vista de diseño (complejidad, costo, área, recursos disponibles).

2. Las instrucciones a desarrollar son libres así como el tipo de datos. Aunque no hay un límite en cuanto la cantidad de instrucciones, es importante que provea al menos instrucciones para control de flujo, operaciones aritméticas-lógicas, acceso a memoria.
3. Los productos finales de esta etapa son el *instruction reference sheet* o *green sheet*.

4. El ISA debe ser personalizado y realizado por los estudiantes, no se aceptarán ISAs ya diseñados (e.g., ARM, x86, RISC-V, otros). Debe justificar cada característica del mismo.

Requisitos de Microarquitectura

1. La implementación diseñada debe ser correcta respecto a las reglas definidas por la arquitectura, esto quiere decir que el procesador debe ser capaz de ejecutar todas las instrucciones definidas y su especificación respecto a errores y excepciones.

2. El procesador diseñado debe emplear pipelining. Tenga en cuenta las implicaciones respecto a riesgos de dicha técnica, el uso de registros y unidades de ejecución. **No se revisará si no tiene pipeline.**

3. Debe ser implementado usando SystemVerilog.

4. No se permite realizar módulos especializados de hardware. Es un curso de Arquitectura de Computadores, no de Diseño de Sistemas Digitales.

5. El procesador debe tener capacidad de segmentación de memoria en datos e instrucciones además debe ser capaz de acceder los dispositivos de entrada y salida del sistema (GPIO, volcado de memoria, switches, etc).

6. Cada unidad funcional del sistema debe ser debidamente probada en simulación, para verificar su funcionamiento correcto (unit tests). Además debe incluir pruebas de integración y sistema. Se le solicita un plan de pruebas donde especifique los objetivos y descripción de las pruebas junto con sus resultados.

7. Los resultados finales de esta etapa son:

   a) El código fuente (SystemVerilog) y el bitstream para programar la tarjeta de desarrollo.

   b) Un diagrama de bloques de la microarquitectura y descripción de las interacciones entre ellos.

   c) Simulaciones de las pruebas unitarias y de integración.

8. Reporte de consumo de recursos del FPGA para el modelo.

Requisitos de Software:

1. Crear una aplicación (software) empleando la arquitectura diseñada, con el fin de implementar la aplicación descrita.

2. Debe realizar un programa ('compilador') que permita traducir las instrucciones del ISA a binario, con la finalidad de ejecutarlo en el procesador. No es necesario que realice análisis léxico, sintáctico y semántico (este curso no es de Compiladores).

El proceso de diseño debe incluir propuestas y comparación de viabilidad de las mismas.

3. Metodología de trabajo

El proyecto debe seguir los siguientes aspectos de desarrollo colaborativo en git, sino, la parte funcional no será calificada y obtendrá nota de cero:

1. Utilice una cuenta de repositorio gratuita.
2. Cree un repositorio con el siguiente nombre: `<user_id>_computer_architecture_1_2022`. El `user_id` estará compuesto por la primera letra del nombre y el apellido. Por ejemplo, para el estudiante Luis Chavarría, el nombre del repositorio será:

   `lchavarria_computer_architecture_1_2022`.
3. Si el repositorio es privado, proporcione acceso a `luchazam` (bitbucket) o `luchazam` (GitHub).
4. El repositorio de Git contendrá dos ramas principales: `master` y `development`.
5. Inicialmente, la rama de `development` se crea a partir del `master`.
6. Al trabajar en un proyecto, el estudiante debe crear una nueva rama de trabajo desde `develop` y cuando la función esté lista, la rama debe fusionarse para `develop`. Cualquier corrección o modificación adicional después de `merge` debería requerir que se repita el proceso (es decir, crear la rama desde `develop` y fusionar los cambios más tarde). Una vez que el código de desarrollo esté listo, se fusionará con `master` y se debe crear una `tag`. El proceso se describe en la siguiente Figura 3.

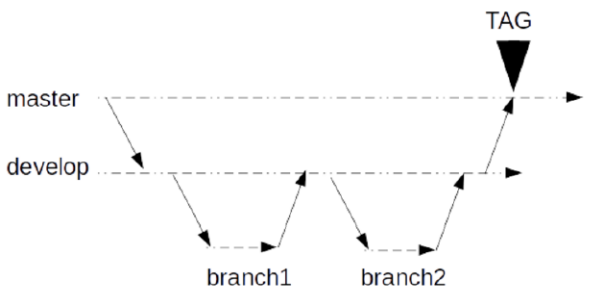

Figura 3: *Git workflow*

Adicionalmente se coloca este enlace recomendado.

7. Después de haber realizado algunos proyectos la rama `master` debe verse así:

   - `master/`
     - `proyecto_1`
     - `proyecto_2`
     - ...
   - ...

   Donde cada directorio de `proyecto_x` contiene todos los entregables para cada proyecto.

   No es permitido realizar todo el trabajo en un solo commit, es decir, que realice el trabajo de forma local y solo suba el último entregable en el repositorio. Si no, obtendrá nota de cero. Debe mostrar avance incremental durante todas las semanas (se revisarán estadísticas). Se revisará el trabajo en equipo.

4. Evaluación y entregables

La defensa será el mismo día de la entrega y todos los archivos (incluyendo código fuente) serán entregados a las 11:59 pm ese mismo día (realícenlo progresivamente y no lo dejen para el final). **Si algo no queda claro o no sabe, no lo asuma, pregúntele al profesor.** Como recomendación se presenta el cronograma de trabajo de la Tabla 1.

Tabla 1: Cronograma sugerido para el proyecto

Semana	Actividades sugeridas
1	Estudio del problema y modelado del programa en alto nivel.
2	Análisis y generación de documentación y scripts para justificar el ISA. En esta parte se va a ir definiendo el *Green Card*.
3	Desarrollo de microarquitectura y pruebas unitarias sobre la misma. Desarrollo del compilador. Se pueden ir desarrollando los diagramas del sistema.
4	Desarrollo final de la microarquitectura. Validación de la arquitectura usando pruebas de integración (arquitectura sobre microarquitectura).
5	Validaciones de integración finales en las tres etapas.

La evaluación del proyecto se da bajos los siguientes rubros contra rúbrica correspondiente:

- Presentación proyecto 100 % funcional (75 %): La defensa se realizará de la siguiente manera: Debido a la situación actual (COVID-19) no se puede tener acceso a hardware u otros

instrumentos y medios que requieran presencia y contacto físico tanto entre el profesor como l@s estudiantes. Por esta razón, para la defensa se debe presentar en un espacio de media hora el diseño sintetizable y ejecutado en la tarjeta Terasic DE10-Nano. Adicionalmente se deben mostrar simulaciones y el diseño que está siendo ejecutado.

Los entregables adicionales que se revisarán durante la defensa son los siguientes:

1. Arquitectura:
   a) *Instruction reference sheet* o *green sheet*.
   b) Scripts usados para justificar las características del ISA. **No cuenta como entregable el mismo script proveído por el profesor en este enunciado.**
2. Microarquitectura:
   a) Diagrama de bloques de la microarquitectura.
   b) Reporte de consumo de recursos del FPGA.
3. Software:
   a) Compilador usado.
   b) Código ensamblador para la arquitectura diseñada.
   c) Programa de alto nivel para representar de la imagen de salida generada por el sistema.

- Documentación de diseño (25 %): Este documento se encuentra directamente ligado con el atributo AP. La documentación del diseño deberá contener las siguientes secciones:

1. Listado de requerimientos del sistema: Cada estudiante deberá determinar los requerimientos de ingeniería del problema planteado, considerando partes involucradas, estado del arte, estándares, normas, entre otros.
2. Elaboración de opciones de solución al problema: Para el problema planteado deberán documentarse al menos dos opciones de solución. Cada solución deberá ser acompañada de algún tipo de diagrama. **Estas opciones de solución no deben ser fácilmente descartables y deben llevar un análisis objetivo con base en criterios técnicos o teóricos.** Al ser un curso de Arquitectura de Computadores, las opciones deben ser esencialmente basadas en el ISA.
3. Comparación de opciones de solución: Se deberán comparar explícitamente las opciones de solución, de acuerdo con los requerimientos y otros aspectos aplicables de salud, seguridad, ambientales, económicos, culturales, sociales y de estándares.
4. Selección de la propuesta final: Se deberá evaluar de forma objetiva, válida y precisa las soluciones planteadas al problema y escoger una solución final.

Se seguirán los siguientes lineamientos:

1. Los documentos serán sometidos a control de plagios para eliminar cualquier intento de plagio con trabajos de semestres anteriores, actual o copias textuales, tendrán nota de cero los datos detectados. Se prohíbe el uso de referencias hacia sitios no confiables.

2. No coloque código fuente en los documentos, quita espacio y aporta poco. Mejor explique el código, páselo a pseudocódigo o use un diagrama.