Instituto Tecnológico de Costa Rica

Escuela de Ingeniería en Computadores

Programa de Licenciatura en Ingeniería en Computadores

Curso: CE-4301 Arquitectura de Computadores I

Especificación Proyecto Grupal #1

Profesor:

Dr.-Ing. Jeferson González Gómez

Fecha de entrega: 10 de octubre de 2025

Proyecto Grupal #1: Arquitectura del Set de Instrucciones (ISA) Específica tipo RISC para Aplicaciones de Seguridad de la Información

Grupo: 4 estudiantes

1. Objetivo

Mediante el desarrollo de este proyecto, el estudiante aplicará los conceptos de arquitectura de computadores en el diseño, y posterior implementación en un modelo de software, de una arquitectura del set de instrucciones (ISA) tipo RISC específica y propia de cada grupo de trabajo para un conjunto de aplicaciones relacionadas con la seguridad de la información, específicamente en el dominio de la firma digital y verificación de integridad de datos.

2. Descripción General

En el transcurso de las dos décadas pasadas, las arquitecturas RISC han crecido significativamente en popularidad debido principalmente a su eficiencia. El mercado de los sistemas empotrados, en general, y en específico los dispositivos móviles actuales, se encuentra dominado por procesadores RISC y computadores basados en Sistemas en Chip (SoC) o microcontroladores (MCU), cuyo núcleo, es también un procesador RISC. Es por esta razón que, en el marco de la arquitectura de computadores moderna, es fundamental conocer las características de este tipo de arquitectura, aplicable directamente a tecnologías y tendencias de punta como lo son el Internet de las cosas (IoT) y los sistemas empotrados en general.

Con el auge en la popularidad de dichos sistemas, así como el incremento en la complejidad de los mismos, la seguridad de las aplicaciones y la información dentro de estos sistemas emergentes se ha convertido en un tema de interés global. Con el objetivo de promover ambientes seguros de ejecución para aplicaciones, los fabricantes de procesadores han optado por incluir extensiones específicas de seguridad en sus arquitecturas (e.g., ARM TrustZone, Intel SGX), que permiten la inclusión de diferentes primitivas de seguridad para un rango variado de aplicaciones de manera eficiente y con soporte de hardware.

En este proyecto, se dará una introducción a este tipo de sistemas, por medio del diseño de una ISA con soporte nativo para intrínsecas de seguridad relacionadas con funciones hash eficientes, almacenamiento seguro de llaves privadas y operaciones de firma digital.

3. Especificación

3.1. Aplicaciones de Seguridad

En este proyecto se deberá dar soporte de hardware a instrucciones específicas para el manejo eficiente de operaciones relacionadas con la firma digital y verificación de integridad. Específica-

mente, se deberá dar soporte a dos áreas: cómputo eficiente de funciones hash y almacenamiento seguro de llaves privadas para firma digital. En esta sección se detallan dichas áreas. Los requisitos de arquitectura que se derivan de estas áreas se detallan más adelante (Sec. 3.2)

3.1.1. Función Hash Eficiente (ToyMD)

Las funciones hash son herramientas fundamentales en ciencias de la computación y ciberseguridad que proporcionan representaciones compactas de tamaño fijo de datos arbitrarios. Sus principales propiedades —resistencia a colisiones, resistencia a pre-imagen y cómputo eficiente —las hacen esenciales en campos como firmas digitales, protección de contraseñas, verificación de integridad de datos y tecnologías blockchain.

En este proyecto se deberá utilizar una función hash simplificada basada en el paradigma Merkle–Damgård [1] llamada *ToyMDMA* (Toy Merkle-Damgård with Mixing + Arithmetic + Multiplication). Este algoritmo procesa datos de entrada en bloques de 64 bits, manteniendo un estado interno de cuatro valores de 64 bits (A, B, C, D) que se actualizan iterativamente usando operaciones no lineales, multiplicación y aritmética modular. En la Fig. 1 se presenta la implementación de referencia de la función hash. Para efectos de implementación en este proyecto, el algoritmo deberá procesar **bloques de 64 bits**, mantener **estado de 256 bits** (cuatro valores de 64 bits), y producir un **hash final de 256 bits**. Para efectos de validación, se deberá utilizar la constante de proporción áurea **0x9e3779b97f4a7c15**, el primo **0xFFFFFFFB** para operaciones modulo, y valores iniciales estándar para A, B, C, D.

3.1.2. Almacenamiento seguro de llaves privadas (bóveda)

Como se mencionó anteriormente, los algoritmos de firma digital requieren llaves privadas que deben mantenerse en secreto absoluto para garantizar la seguridad del sistema. Dado que la eficacia y fortaleza de la firma digital dependen completamente de la protección de la llave privada, es fundamental que su manejo y almacenamiento sea seguro.

Uno de los mecanismos empleados para asegurarse de que las llaves privadas no sean atacadas (por ejemplo, si un atacante logra tener acceso al sistema) es no mantenerlas en archivos o memoria general, sino **almacenarlas directamente en el hardware** utilizando una *raíz de confianza* (*Root of Trust - RoT*), inaccesible para un usuario común del sistema, pero accesible para las aplicaciones seguras del sistema (e.g., firma digital).
Para este proyecto se deberá modelar una raíz de confianza de forma de una bóveda de llaves (memoria segura) que permita almacenar **al menos 4 llaves privadas de 64 bits cada una y 4 valores iniciales de hash de 64 bits cada uno** (correspondientes a los valores A, B, C, D del estado inicial). Dicha bóveda deberá ser accesible directamente/solamente por el CPU, por medio de instrucciones del set específicas para escribir u operar con los datos de la misma. La bóveda NO debe poder ser accedida como una memoria tradicional.

3.1.3. Firma digital

La firma digital es un mecanismo criptográfico que permite autenticar el origen de un documento y verificar que no ha sido modificado. El proceso utiliza funciones hash y llaves privadas para crear una huella digital única que vincula un documento específico con su autor.

```c

// Rotate left (64-bit)
static inline uint64_t rol64(uint64_t x, unsigned r) {
    return (x << r) | (x >> (64 - r));
}

// Toy Merkle-Damgard with Mixing + Arithmetic + Multiplication
// Processes one 64-bit block and updates 256-bit hash state
void toy_mdma_hash_block(uint64_t block,
                         uint64_t *a, uint64_t *b,
                         uint64_t *c, uint64_t *d) {
    // Load current state
    uint64_t A = *a, B = *b, C = *c, D = *d;

    // Non-linear mixes
    uint64_t f = (A & B) | (~A & C);
    uint64_t g = (B & C) | (~B & D);
    uint64_t h = A ^ B ^ C ^ D;

    // Multiply-mix step
    uint64_t mul = (block * 0x9e3779b97f4a7c15ULL);
    mul &= 0xFFFFFFFFFFFFFFFFULL; // mask to 64 bits

    // Round updates with multiplication and mod
    A = rol64(A + f + mul, 7) + B;
    B = rol64(B + g + block, 11) + (C * 3);
    C = rol64(C + h + mul, 17) + (D % 0xFFFFFFFBULL);
    D = rol64(D + A + block, 19) ^ (f * 5);

    // Store updated state
    *a = A; *b = B; *c = C; *d = D;
}
```

Figura 1: Función hash de referencia ToyMDMA basada en Merkle–Damgård.

Para efectos de este proyecto, el proceso de firma digital consiste en: (1) calcular un hash del documento original usando ToyMDMA, (2) firmar este hash aplicando una operación con una llave privada almacenada en la bóveda, y (3) almacenar la firma junto con el documento. Para verificar, se recalcula el hash del documento y se compara con el hash obtenido al procesar la firma usando la misma clave.

Para este proyecto, los estudiantes deberán implementar un sistema completo de firma digital que incluya:

1. **Carga de archivos**: El sistema debe cargar un archivo en memoria.
2. **Cómputo de hash**: Implementar la función ToyMDMA en ensamblador para procesar el archivo en bloques de 64 bits y almacenar el hash resultante (256 bits) en memoria.
3. **Generación de firma**: Usar una llave privada K de 64 bits desde la bóveda para calcular la firma como `S = (A XOR K, B XOR K, C XOR K, D XOR K)` y almacenarla en memoria.
4. **Almacenamiento de archivo firmado**: Guardar el archivo original y la firma en un nuevo archivo. La firma deberá ser agregada al final del archivo (últimos 256 bits).
5. **Verificación de firma**: Recalcular el hash, cargar la firma almacenada, aplicar la operación inversa con la clave (descrifrado), y comparar los resultados para determinar la validez.

3.2. Requisitos Específicos de las Aplicaciones de Seguridad para la ISA

Para la implementación eficiente de las aplicaciones de firma digital mencionadas arriba, se deberán considerar los siguientes requerimientos para el diseño del set de instrucciones:

3.2.1. Bóveda de llaves privadas

- Para la bóveda de llaves se deberá contar una o más instrucciones específicas para almacenar llaves privadas de 64 bits y valores iniciales de hash de 64 bits. Ya que sólo se permite almacenar 4 llaves privadas y 4 valores iniciales (A, B, C, D), se deberá considerar esta limitante en el diseño de la(s) instrucciones.
- Toda operación de firma digital y cómputo de hash que involucre llaves privadas o valores iniciales deberá utilizarlos desde la bóveda.
- Los datos almacenados en la bóveda NO podrán ser escritos en registros del CPU o memoria general del sistema.
- Solamente las instrucciones que operen directamente sea de manera parcial o total con las llaves privadas o valores iniciales pueden acceder a la bóveda como operando fuente. Tome en cuenta la aplicación de firma digital como referencia para este requerimiento.

3.2.2. Función Hash ToyMDMA y Firma Digital

- El cómputo de hash debe realizarse de manera eficiente. Deberá incluir en el set **al menos tres** instrucciones específicas para acelerar el algoritmo de hash, considerando las operaciones de multiplicación, aritmética modular, y funciones no lineales.
- La firma digital debe implementarse como una operación que combine el hash del mensaje con la llave privada. Deberá incluir **al menos una** instrucción específica para generar firmas.
- Deberá considerarse un balance entre eficiencia y posible costo (área, complejidad) en el diseño de las instrucciones relacionadas con el hash y firma digital. Las operaciones de multiplicación por constantes, aritmética modular, y funciones no lineales pueden requerir unidades funcionales especializadas. Una sola instrucción para la compresión completa resultaría en un módulo muy complejo y costoso.

3.3. Requisitos Generales de la Arquitectura del Set de Instrucciones

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). Las/los estudiantes deben realizar un análisis en un software de alto nivel para encontrar los valores idóneos.

2. Las instrucciones a desarrollar son libres así como el tipo de datos. Aunque no hay un límite en cuanto ala cantidad de instrucciones, es importante que provea al menos instrucciones para control de flujo, operaciones aritméticas-lógicas, acceso a memoria.

3. La arquitectura debe ser personalizada para las aplicaciones descritas y debe ser diseñada completamente por los estudiantes, **no se aceptarán ISAs basados en arquitecturas existentes (e.g., ARM, x86, RISC-V, otros)**. Debe justificar cada característica del mismo.

4. Los productos finales de esta etapa son los siguientes:

   a) Hoja de referencia rápida de instrucciones (*Instruction reference sheet* o *green card*).

   b) Documentación de aspectos de diseño de set de instrucciones incluyendo argumentación y resultados de scripts usados para justificar las características de la ISA, como parte del **Documento de diseño**, en la sección correspondiente a la ISA.

3.4. Requisitos de Implementación y Validación de la Arquitectura

3.4.1. Implementación

La implementación de la arquitectura del conjunto de instrucciones se debe realizar por medio de un modelo de software. Dicho modelo deberá ser ejecutable y funcionalmente equivalente a un procesador. Para esto, deberán modelarse adecuadamente los componentes internos del CPU (banco de registros, unidades funcionales, unidad de control, etc.), así como las interconexiones de los mismos. Además, se deberá implementar un modelo de memoria(s) que permita la carga y escritura de datos e instrucciones.

El modelo de software del procesador que implementa el set de instrucciones propio deberá brindar métricas de la ejecución de los programas (como mínimo, cantidad de ciclos). Adicionalmente, la simulación del procesador deberá permitir en todo momento la visualización de elementos claves, como los registros internos del CPU, así como la bóveda de llaves criptográficas.

A pesar de tratarse de una solución de software, la eficiencia de la arquitectura y microarquitectura implementada deberá ser considerada en el diseño.
Cada grupo deberá realizar una aplicación tipo compilador/ensamblador que permita traducir las instrucciones del ISA a binario, con la finalidad de ser cargada al modelo de la memoria y ejecutarlo en el procesador.

El lenguaje de programación a elegir queda a criterio de cada grupo de trabajo. Los productos finales de esta etapa son:

1. El código fuente (repo) y comandos/scripts de compilación y ejecución del modelo funcional de software del CPU, así como el compilador/ensamblador.

2. Un diagrama de bloques de la microarquitectura y descripción de las interacciones entre ellos, incluido en el **Documento de diseño**, en la sección correspondiente a la organización.

3. Programa (código fuente en ensamblador) que permita evaluar todas las instrucciones incorporadas en el set, a diseñar por cada grupo de trabajo.

4. Documentación de diseño de todo el software del sistema. Incluido en el **Documento de diseño**, en la sección correspondiente al modelado de la organización/microarquitectura.

3.4.2. Validación en Aplicación de Seguridad

Para validar el diseño en cuanto a la inclusión de las instrucciones para el almacenamiento seguro de llaves privadas, así como el cómputo eficiente de hash y generación de firmas digitales, a cada grupo se le suministrarán dos archivos de prueba para firmar y verificar según se describe a continuación:

- Para la primera prueba, el profesor brindará uno más archivos regulares y una llave privada única de 64 bits. Cada grupo deberá crear un programa que calcule el hash ToyMDMA del archivo y genere una firma digital utilizando su arquitectura/modelo de procesador. La firma debe agregarse al final del archivo original (últimos 256 bits del archivo resultante).

Para verificar el proceso, el profesor comparará la firma obtenida con el valor esperado calculado con la implementación de referencia.

- Para la segunda prueba, el profesor brindará uno o más archivos firmados (archivo original + firma de 256 bits al final) y una llave privada correspondiente. Cada grupo deberá crear un programa que verifique la autenticidad de la firma separando el archivo original de la firma, recalculando el hash del archivo original y comparando con la firma utilizando su arquitectura/modelo de procesador. El grupo deberá reportar si la firma es válida o inválida.

Nota: Todo proceso de cómputo de hash y firma digital con llaves privadas deberá utilizar la bóveda.
Salidas esperadas de esta etapa: código(s) fuente(s) en repositorio correspondiente.

4. Evaluación del proyecto

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

- Presentación proyecto funcional (75 %): La defensa se realizará de **forma presencial** en un horario a definir (preferiblemente en horas de clase). La defensa se debe realizar en un espacio de 25 minutos, aproximadamente. Deberá mostrarse el sistema en funcionamiento, así como la ejecución de todas las instrucciones y de la validación con la aplicación de firma digital.

  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.
  2. Microarquitectura:
     a) Diagrama de bloques de la microarquitectura.
     b) Estado de los registros y bóveda durante la ejecución
  3. Otro Software:
     a) Compilador/Ensamblador usado.
     b) Código fuente en lenguaje ensamblador para la arquitectura diseñada.
  4. Validación Firma Digital
     a) Validación generación de firma
     b) Validación verificación de firma

- Repositorio y documentación de diseño (25 %): La documentación del diseño deberá contener las siguientes secciones:

  - Gestión de repositorio (10 %):

- Uso adecuado de Git con flujo de trabajo profesional: El proyecto debe mantener un repositorio Git con historial de commits descriptivos, uso de ramas de desarrollo (dev branches) para nuevas funcionalidades. Se evaluará la calidad de los mensajes de commit, la organización de branches, y el uso de pull requests o merge requests. (5 %)
- README completo y claro: El repositorio debe incluir un archivo README detallado que explique el propósito del proyecto, instrucciones de compilación y ejecución, dependencias necesarias, y cualquier otra información relevante para entender y utilizar el código, así como ejemplos de uso con archivos de prueba de ejemplo. (2 %)
- Gestión de issues y milestones: Demostrar el uso de issues para tracking de funcionalidades, bugs y mejoras, organizados en milestones que reflejen el progreso del proyecto. (3 %)

- Documentación de diseño (15 %): La documentación del diseño deberá ser agregada al repositorio, en un directorio *docs*. La documentación deberá presentarse en formato markdown (.md), y deberá contener las siguientes secciones:
  1. Arquitectura del set de instrucciones (5 %): Descripción detallada del set de instrucciones diseñado, incluyendo modos de direccionamiento, tipos y tamaños de datos, sintaxis y codificación de instrucciones, registros disponibles y su propósito. Justificación de cada característica desde el punto de vista de diseño (complejidad, costo, área, recursos disponibles). Incluir hoja de referencia rápida de instrucciones (*Instruction reference sheet* o *green card*).
  2. Organización/Microarquitectura (5 %): Descripción de la organización interna del procesador, incluyendo diagrama de bloques que muestre los componentes principales (banco de registros, unidades funcionales, unidad de control, etc.) y sus interconexiones. Explicación del flujo de datos e instrucciones dentro del procesador.
  3. Modelado del software (5 %): Descripción del modelo de software implementado para simular el procesador, incluyendo detalles sobre la implementación del banco de registros, unidades funcionales, memoria y cualquier otra característica relevante. Explicación de cómo se manejan las instrucciones y el ciclo de ejecución.

Referencias

[1] Ivan Bjerre Damgård. A design principle for hash functions. In *Advances in Cryptology — CRYPTO' 89 Proceedings*, volume 435 of *Lecture Notes in Computer Science*, pages 416–427. Springer, 1990.