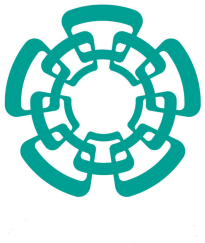


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional

Unidad Cinvestav Tamaulipas

Aceleración en CPU-GPU de un esquema de cifrado basado en sobre digital para seguridad de datos en la nube

Tesis que presenta:

Luis Fernando Guerrero Carrizales

Para obtener el grado de:

**Maestro en Ciencias
en Ingeniería y Tecnologías
Computacionales**

Dr. Arturo Díaz Pérez, Director
Dr. Miguel Morales Sandoval, Co-Director

Cd. Victoria, Tamaulipas, México. Octubre, 2017

La tesis presentada por Luis Fernando Guerrero Carrizales fue aprobada por:

Dr. José Juan García Hernández

Dr. Victor Sosa Sosa

Dr. Arturo Díaz Pérez, Director

Dr. Miguel Morales Sandoval, Co-Director

Cd. Victoria, Tamaulipas, México., 9 de Octubre de 2017

Agradecimientos

Índice General

Índice de Figuras

Índice de Tablas

Índice de Algoritmos

Resumen

Aceleración en CPU-GPU de un esquema de cifrado basado en sobre digital para seguridad de datos en la nube

por

Luis Fernando Guerrero Carrizales

Unidad Cinvestav Tamaulipas

Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional, 2017

Dr. Arturo Díaz Pérez, Director

Dr. Miguel Morales Sandoval, Co-Director

La alta disponibilidad de recursos de almacenamiento en la nube a bajo costo y las ventajas que ésta ofrece como lo son la flexibilidad y la fiabilidad ha motivado a que los propietarios de datos usen servidores externos para resguardar su información. Sin embargo, el uso de estos servidores trae consigo riesgos de seguridad de información, principalmente debido a que los datos son accesibles por el proveedor. Esto ha llevado a la creación de esquemas de seguridad que no solo sean efectivos para garantizar la seguridad de los datos que se almacenan y se acceden en un entorno de nube, sino que también realicen las operaciones de seguridad de forma eficiente para reducir el costo computacional asociado. En esta tesis se propone una estrategia para la paralelización de un esquema de seguridad informática basado en la técnica de sobre digital, esto a través del particionamiento de las tareas que conforman dicho esquema de seguridad para su ejecución en arquitecturas CPU-GPU. Dicho esquema de seguridad provee los servicios de confidencialidad y control de acceso a los datos, ideal para asegurar la información que reside en la nube. El esquema de seguridad en cuestión es DET-ABE, el cual usa el cifrador simétrico AES para garantizar la confidencialidad de los datos y el cifrador CP-ABE para proteger la llave de cifrado AES a través de una política de acceso, la cual sirve como mecanismo de control de acceso sobre ella. En la evaluación del esquema DET-ABE paralelo se realizaron seis experimentos, donde cada uno evalúa los tiempos de procesamiento, trabajo adicional o aceleración bajo diferentes condiciones. La aceleración que se obtiene en las distintas operaciones de DET-ABE (cifrar, descifrar, generar la llave de descifrado) varía dependiendo de la configuración de los parámetros de entrada del esquema. Los resultados indican que la propuesta de paralelización es eficiente, reduciendo los tiempos de procesamiento del esquema con una aceleración de hasta $4.5\times$ con respecto al esquema secuencial.

Abstract

CPU-GPU acceleration of a digital envelope-based encryption scheme for cloud data security

by

Luis Fernando Guerrero Carrizales
CINVESTAV Tamaulipas
Research Center for Advanced Study of the National Polytechnic Institute, 2017
Dr. Arturo Díaz Pérez, Advisor
Dr. Miguel Morales Sandoval, Co-Advisor

The high availability of storage resources at low cost and the advantges offered by the cloud such as flexibility and reliability has motivated data owners to use external servers to safeguard their information. However, the use of these servers bring security risks, mainly because the data is accessible by the storage service provider. Bacause of that, it is necesary to create security schemes that are not only effective but also efficient for ensuring the security services over the data that is stored and accessed in a cloud environment. Efficiency is required to reduce the computational cost and overhead associated to the security operations applied over large amounts of data. This thesis proposes a strategy for the parallelization of a data security scheme based on the digital envelope technique. This strategy consists in the partitioning of the tasks that conform the security scheme in order to implement them in CPU-GPU hybrid architetures. The security scheme under study is DET-ABE, which provides the services of confidentiality and access control, ideal for securing information that resides in the cloud. DET-ABE uses the symmetric cipher AES to ensure data confidentiality and the asymmetric CP-ABE cipher to protect the AES encryption key through an access policy, which serves as access control mechanism over it. In the evaluation of the parallel DET-ABE scheme, six experiments were performed, each evaluating processing time, additional work or acceleration under different conditions. The acceleration obtained in the different DET-ABE operations (encryption, decryption and key generation) varies depending on the configuration of the input parameters of the scheme. The results showed that the proposed parallelization is efficient, reducing the processing time of the scheme with an acceleration up to 4.5$\times$ compared to the sequential scheme.

Nomenclatura

3DES	Triple Data Encryption Algorithm
AES	Advanced Encryption Standard
BDHP	Bilinear Diffie-Hellman Problem
CBC	Cipher Block Chaining
CDHP	Computational Diffie–Hellman Problem
CP-ABE	Ciphertext-Policy Attribute-Based Encryption
CPU	Central Processing Unit
CPU-GPU	Central Processing Unit - Graphics Processor Unit
CTR	Counter Mode
CUDA	Compute Unified Device Architecture
CWC	Carter–Wegman + CTR Mode
DES	Data Encryption Standard
DET	Digital Envelope Technique
DET-ABE	Digital Envelope Technique Attribute-Based Encryption
EC	Elliptic Curve
ECB	Electronic Codebook
ECC	Elliptic Curve Cryptography
GPGPU	General-Purpose Computing on Graphics Processing Units
GPU	Graphics Processor Unit
IBE	Identity Base Encryption
KP-ABE	Key-Policy Attribute-Based Encryption
NIST	National Institute of Standards and Technology
PBC	Pairing-Based Cryptography
PCIe	Peripheral Component Interconnect Express
PKI	Public Key Infrastructure
RC5	Rivest Cipher 5
RBAC	Role-Based Access Control
SM	Streaming Multiprocessors
TCP	Transmission Control Protocol

1

Introducción

1.1 Antecedentes

El cómputo en la nube es un modelo para permitir el acceso a través de Internet a un conjunto compartido de recursos computacionales configurables (por ejemplo, redes, servidores, almacenamiento, aplicaciones y servicios) que se pueden aprovisionar y liberar rápidamente con un esfuerzo mínimo de administración o interacción de provisión de servicios. Este modelo de nube se compone de cinco características esenciales, tres modelos de servicio y cuatro modelos de implementación [42]. Dentro de los servicios del cómputo en la nube se encuentra el de *almacenamiento* o *cloud storage* [28], el cual se enfoca en almacenar y recuperar datos de un servidor de almacenamiento a través de Internet [26].

Algunas de las ventajas del almacenamiento en la nube son la alta disponibilidad de la información a través de Internet, la reducción de costos, generalmente el espacio de almacenamiento se ajusta a las necesidades del usuario y es posible recuperar información de forma mas rápida que en un equipo

convencional, entre otros [22, 28, 53].

Un sistema de almacenamiento en la nube es un sistema de servicio cooperativo entre múltiples dispositivos, dominios de aplicaciones y formas de servicios. Un sistema de almacenamiento en la nube se beneficia de las redes de banda ancha (web 2.0), virtualización de almacenamiento, redes de almacenamiento, tecnología de clúster, sistemas de archivos distribuidos, redes de distribución de contenidos, compresión de datos y cifrado de datos [26]. El almacenamiento en la nube se realiza a partir de la distribución del trabajo en tres entidades y un módulo de comunicación, los cuales corresponden a [53]:

1. **Dispositivo emisor.** A través de esta entidad, el **propietario de datos** o **cliente**, transfiere la información a un servidor de almacenamiento externo a través de un canal de comunicación (proceso "Upload"). El propietario de datos específica algunas restricciones sobre el acceso de este recurso y posiblemente algún servicio de seguridad bajo su control, independientemente del que implemente el proveedor del servicio.

2. **Internet (módulo de comunicación).** Este componente está a cargo de las operaciones correspondientes a la transmisión de los datos hacia y desde los servidores externos. Este componente incluye sistemas operativos, software de aplicación y almacenamiento temporal.

3. **Servidores de almacenamiento (nube).** Esta entidad resguarda la información del propietario de los datos y define algunas configuraciones con base en el perfil del usuario. Este módulo posee los siguientes componentes:

   - **Sistema operativo.** Se encarga de la ejecución paralela o distribuida de las peticiones de los clientes.
   - **Software de hospedaje.** Define alguna configuración particular en el servidor, basándose en las características de usuario y solicitudes particulares.

- **Dispositivos de almacenamiento.** Son dispositivos de almacenamiento de altas prestaciones, los cuales están disponibles en todo momento.

4. **Dispositivo receptor.** A través de esta entidad el **consumidor de datos**, puede acceder a la información resguardada en los servidores en la nube (proceso "Download"). Esta entidad puede ser el mismo propietario de los datos u otro.

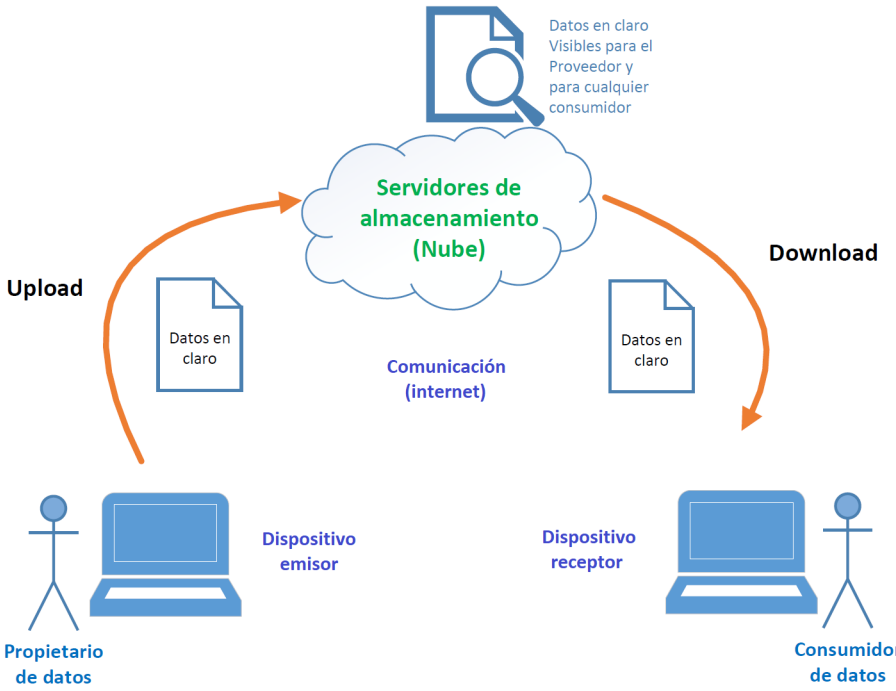


Figura 1.1: Esquema del modelo de almacenamiento en la nube.

En la Figura 1.1 se muestra un diagrama del modelo de almacenamiento en la nube descrito anteriormente. En este modelo, el propietario de los datos transfiere la información a un servidor de almacenamiento externo (proceso "Upload") a través de los siguientes pasos [29]:

1. Realiza una solicitud de inicio de sesión en el servidor de almacenamiento en la nube.
2. El servidor de almacenamiento en la nube permite el acceso al cliente o propietario de datos.
3. El propietario de datos realiza una solicitud de almacenamiento al servidor de almacenamiento externo.

4. El servidor de almacenamiento recibe la petición del cliente, realiza los preparativos para el almacenamiento de la información y responde al cliente autorizando el proceso de transferencia.

5. Se realiza la transferencia mediante el módulo de comunicación.

El consumidor de datos accede a la información almacenada en la nube a través de los siguientes pasos [29]:

1. Realiza una solicitud de inicio de sesión al servidor de almacenamiento en la nube (si es necesario).

2. El servidor de almacenamiento en la nube permite el acceso al consumidor de datos.

3. El consumidor de datos realiza una solicitud de información al servidor de almacenamiento definiendo el nombre del documento al que se desea acceder.

4. Se realiza la transferencia mediante el módulo de comunicación para transferir los datos del servidor al consumidor de datos.

1.2 Seguridad en el almacenamiento en la nube

El almacenamiento en la nube es demandado en la actualidad por el crecimiento de la información con el fenómeno del BigData [72]. Es una solución práctica y atractiva para salvaguardar la información de empresas o entidades, ya que ofrece un mecanismo fiable, flexible y accesible para el almacenamiento de datos [26, 72]. De esta manera, el propietario de la información delega al proveedor del servicio de almacenamiento la responsabilidad de mantener disponibles sus datos en todo momento. Sin embargo, el uso de estos servidores trae consigo riesgos de seguridad de información. Los principales servicios de seguridad que deben garantizarse en el servicio de almacenamiento en la nube son [28]:

1. **Confidencialidad.** Los datos del propietario deben mantenerse en secreto, impidiendo que tanto el servidor de almacenamiento en la nube como otros usuarios no autorizados accedan a ellos. Debido a que los datos se almacenan en servidores en la nube, los cuales son controlados y gestionados por proveedores honestos pero curiosos [66] (como por ejemplo el caso de la exposición pública del padrón electoral en México [1]), la certeza de que la información almacenada en la nube sea confidencial se pone en duda.

2. **Control de acceso.** El término de control de acceso bajo un entorno de nube se centra en que un propietario de datos debe tener la capacidad de definir quien puede o no acceder a la información que almacena en servidores externos. Aunque existen características de control de acceso en los servidores de almacenamiento para procurar dar seguridad a la información de un cliente, no obstante el proveedor de estos servicios siempre tendrá el privilegio de acceder a la información del cliente.

Algunas propuestas para proveer los dos servicios de seguridad descritos previamente hacen uso de *algoritmos criptográficos* (cifradores). En términos generales, un cifrador (simétrico) usa una llave (generalmente secreta) para transformar texto claro (inteligible) en texto cifrado (ininteligible). A esta operación se le llama *cifrar*. El mismo cifrador, mediante la operación inversa (*descifrar*) y con el uso de la misma llave transforma el texto cifrado nuevamente en el texto claro original. Los *cifradores simétricos* son ideales para garantizar el servicio de confidencialidad, pero generan un problema de distribución de llaves de descifrado [3]. Debi do a esto, se han desarrollado nuevos esquemas criptográficos basados en la *técnica de sobre digital* (DET, por sus siglas en inglés) [56], los cuales son efectivos para garantizar la seguridad de los datos que se almacenan y se acceden en un entorno de nube [29, 71]. El sobre digital se implementa usando tanto cifradores simétricos y *asimétricos* conjuntamente, ya que se aprovecha el alto desempeño de los cifradores simétricos para cifrar cantidades grandes de información mientras que con los cifradores asimétricos se consigue

[1] http://expansion.mx/politica/2016/04/27/movimiento-ciudadano-admite-que-subieron-el-padron-electoral-a-amazon

reducir la complejidad asociada a la distribución de las llaves.

El esquema **DET-ABE** [45] es una técnica de sobre digital aplicada, en la cual se utiliza el cifrador simétrico AES [52] para garantizar la confidencialidad de los datos y la llave de cifrado se protege usando el cifrador asimétrico CP-ABE [16], el cual usa una política para cifrar la llave AES y sirve como mecanismo de control de acceso sobre ella. En el siguiente capítulo se detallan más las características de los cifradores simétricos y asimétricos, principalmente los cifradores AES y CP-ABE. Este esquema de cifrado en particular resulta atractivo debido al uso de una estrategia de cifrado basado en el uso de políticas de acceso (uso de atributos de usuario para su validación) y una entidad de confianza para la generación de las llaves de descifrado.

La Figura 1.2 muestra el modelo simple de almacenamiento en la nube, indicando los algoritmos de DET-ABE para cifrar los datos antes de transferirlos a los servidores de almacenamiento. En este modelo, los datos cifrados (con la función `DET-ABE.Cifrar`) se almacenan en servidores externos en un formato ininteligible. Junto con los datos cifrados se encuentra la llave de descifrado, pero también en formato cifrado. Cuando el consumidor de los datos descarga la información cifrada (junto con la llave de descifrado), debe primero descifrar la llave y después usar esa llave para descifrar los datos (con la función `DET-ABE.Descifrar`).

Para poder descifrar la llave, el consumidor necesita de una llave especial, generada específicamente para él (con la función `DET-ABE.genLlaveUsuario`) a partir de una serie de atributos que lo identifican dentro del sistema.

En DET-ABE, la capacidad para descifrar la llave AES es lo que permite el descifrado de los datos a todo aquel consumidor que satisfaga la política de acceso que se haya usado para cifrar dicha llave. Así, DET-ABE garantiza tanto la confidencialidad de los datos (al cifrarlos) y el control de acceso (al restringir el acceso a la llave de descifrado de los datos).

DET-ABE requiere de una entidad de confianza, la cual se encarga de generar las llaves de descifrado de los consumidores y tiene bajo su resguardo parámetros de seguridad del esquema.

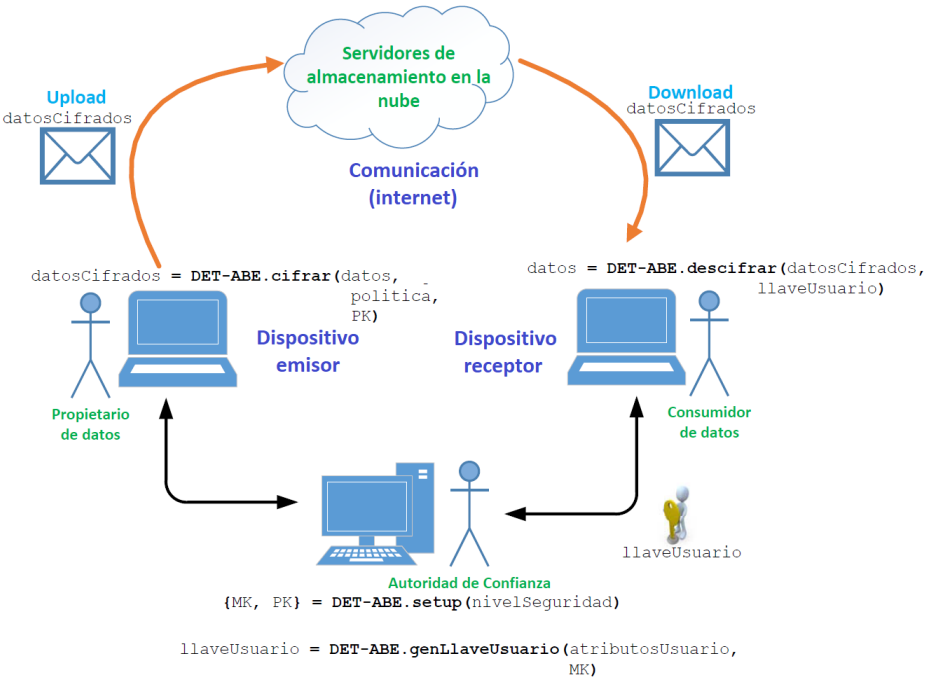


Figura 1.2: Esquema de transferencia de información a la nube utilizando el esquema DET-ABE, donde *MK* es una llave maestra que permite generar llaves de usuario a partir de los atributos que posea el consumidor de los datos, y *PK* es una llave pública usada en la operación de cifrado.

1.3 Planteamiento del problema

Aunque los esquemas de seguridad del tipo DET, como DET-ABE, ofrecen una solución al problema de control de acceso y confidencialidad de datos almacenados en servidores en la nube, resultan ineficientes cuando la cantidad de datos es muy grande o cuando se requieren políticas de acceso elaboradas (por ejemplo en organizaciones) y altos niveles de seguridad. El esquema DET-ABE [45] consume tiempo considerable para realizar las operaciones de cifrado y descifrado de la información, ya que por un lado, el cifrado de los datos es bloque por bloque, requiriendo varias rondas de procesamiento. Por otra parte, también realiza una gran cantidad de operaciones aritméticas en estructuras algebraicas abstractas, cuyos elementos requieren varios centenares o incluso miles de bits. Así, aunque el esquema DET-ABE puede resultar efectivo para resolver los problemas de

seguridad en el almacenamiento en la nube, tiene una desventaja debido a los elevados tiempos de procesamiento, ya que al asignar seguridad a un conjunto grande de datos o utilizar políticas de acceso complejas con niveles de seguridad altos, los tiempos de procesamiento pueden ser llegar a ser considerablemente altos, lo cual vuelve poco atractivo su uso para fines prácticos. Por ejemplo, en el esquema DET-ABE, el cifrado de un documento de 182 Kbytes requiere aproximadamente 9 segundos, mientras que el descifrado del mismo contenido se realiza en aproximadamente 17 segundos [45], esto para un nivel de seguridad de 256 bits (el más alto). Cuando el tamaño de los datos a cifrar es pequeño, estos tiempos de procesamiento pudieran ser aceptables, pero si la cantidad de datos se incrementa, el esquema DET-ABE puede resultar muy lento y poco atractivo. Debido a lo anterior, para que el esquema DET-ABE sea eficiente y cumpla con los niveles de seguridad requeridos, es necesario desarrollar, implementar y evaluar una estrategia alterna que permita reducir sus tiempos de procesamiento actuales.

En esta tesis, se proponen estrategias para acelerar el esquema DET-ABE mediante el uso de arquitecturas híbridas, dividiendo el trabajo tanto en CPUs como aceleradores basados en GPUs, de tal manera que las operaciones secuenciales o poco demandantes computacionalmente se ejecuten en los CPUs, mientras que el procesamiento de información con paralelismo inherente (no dependiente de otros resultados) y más costoso computacionalmente se lleve a cabo en arquitecturas GPU o procesamiento multicore.

1.4 Motivación

La necesidad de mantener de forma confidencial datos importantes almacenados en la nube usando un mecanismo de seguridad eficiente es la principal motivación de este trabajo. Los requerimientos de seguridad para almacenamiento de datos bajo un entorno de nube pueden ser cubiertos mediante el uso de técnicas del tipo DET. Sin embargo, esquemas de seguridad como DET-ABE suelen ser costosas cuando la cantidad de datos es muy grande, cuando se requieren altos

niveles de seguridad, o cuando las políticas de control de acceso son más elaboradas.

El avance de la tecnología ha permitido el desarrollo de nuevos dispositivos de cómputo capaces de realizar procesos de forma más eficiente, contando con características multicore modernas, así como con procesadores gráficos (GPUs) para la aceleración de procesos bajo diversos lenguajes de programación. En la actualidad es más factible contar con equipos de cómputo con estas características, ya que la nueva tecnología multicore se centra en el uso de más procesadores secundarios, los cuales poseen altas capacidades de procesamiento. Sin embargo, la estrategia seguida por los dispositivos GPU es utilizar procesadores de menores prestaciones de forma masiva, pasando de utilizar *n* núcleos eficientes en CPU a quitar trabajo de estos núcleos y repartirlo a miles de núcleos de bajas prestaciones en GPUs. Actualmente se está incorporando esta tecnología en la mayoría de los dispositivos de cómputo asequibles para usuarios promedio, por lo cual resulta menos complicado acceder a éstos.

La disponibilidad de dispositivos con varios CPUs, características multinúcleo, y disponibilidad de GPUs, hace necesario desarrollar técnicas que exploten las ventajas de estas arquitecturas híbridas para acelerar aplicaciones que poseen alto grado de paralelismo, aprovechando al máximo el poder de cómputo disponible.

Modificar un programa secuencial para que se ejecute en paralelo con GPUs tiene dos atractivas razones: acelerar la velocidad de ejecución y/o mejorar la cantidad de memoria utilizable al disponer de las características de almacenamiento de la tarjeta gráfica [25]. Considerando las características de los equipos de cómputo en la actualidad, pueden ser empleados los núcleos de los CPU y de los GPU para conseguir mayores rendimientos al procesar cantidades de datos mucho mayores a las que son procesadas únicamente por los CPU de forma secuencial.

El paradigma de programación en plataformas híbridas ha mostrado que es posible obtener una aceleración notable de algoritmos [9, 10, 35, 50]. Razón por la cual parece posible definir métodos de implementación sobre estas arquitecturas que permitan reducir los tiempos de procesamiento demandados por un esquema de seguridad complejo como DET-ABE. Dado lo anterior, el tomar

provecho de este tipo de arquitecturas resulta atractivo, ya que una mejora en los tiempos de procesamiento del esquema de seguridad DET-ABE puede reducir el *overhead* de los algoritmos de seguridad informática (en tiempo de procesamiento) para hacer viable su aplicación en escenarios reales para envío y recuperación de datos hacia y desde la nube. Esto es más notorio cuando la cantidad de datos a manejar es muy alta, tal como ocurre con el fenómeno de BigData [72].

1.5 Pregunta de investigación

En esta tesis se da respuesta a las siguientes preguntas de investigación:

1. *¿De qué manera puede ser paralelizado el esquema DET-ABE, considerando todos sus niveles de procesamiento?*
2. *¿Es factible la implementación del esquema de seguridad DET-ABE en plataformas híbridas CPU-GPU?*
3. *¿Cuánta aceleración puede ser obtenida al paralelizar el esquema DET-ABE?*

1.6 Hipótesis

La hipótesis que se demuestra en esta tesis es la siguiente:

El esquema de cifrado DET-ABE puede ser acelerado a través del paradigma del cómputo híbrido, explotando las capacidades de cómputo CPU-GPU que actualmente están disponibles en dispositivos de cómputo y con ello se puede obtener una aceleración de por lo menos $2x$ respecto a la versión secuencial.

Un estudio preliminar reveló que el esquema DET-ABE puede ser paralelizable en sus distintos niveles de procesamiento, y que su aceleración puede ser viable, explotando las características multihilo y multinúcleo de los dispositivos de cómputo actuales, que cuentan con las capacidades

de cómputo híbridas CPU-GPU. Bajo un enfoque de cómputo híbrido, las partes inherentemente secuenciales se ejecutan en el CPU mientras que las paralelizables se implementan en GPUs.

1.7 Objetivos

1.7.1 Objetivo general

Mejorar los tiempos de procesamiento del esquema de seguridad DET-ABE haciendo un particionamiento adecuado de sus tareas para ejecutarlas en arquitecturas híbridas CPU-GPU, logrando así contar con un mecanismo más eficiente para proveer servicios de seguridad de datos en la nube.

1.7.2 Objetivos específicos

1. Definir estrategias de paralelización del esquema DET-ABE y realizar su implementación en arquitecturas de cómputo híbridas CPU-GPU.

2. Definir la arquitectura de un prototipo experimental que permita evaluar la estrategia de paralelización propuesta.

3. Conseguir una aceleración significativa de DET-ABE a través de la estrategia de paralelización propuesta, de al menos el doble respecto a su implementación secuencial.

1.8 Metodología de investigación

La metodología seguida para alcanzar los objetivos de la tesis está basada en tres etapas, las cuales son descritas a continuación:

1. Etapa 1: Particionamiento de DET-ABE para su implementación en plataformas híbridas.

- Estudio de los algoritmo a implementar, experimentando con la implementación secuencial del esquema, comprendiendo así el funcionamiento del esquema, sus componentes y detalles de implementación.
- Identificar las secciones paralelizables y las inherentemente secuenciales del esquema DET-ABE, a partir de un análisis de las tareas que se realizan en el esquema y la relación entre ellas.
- Planificación de las estrategias necesarias para la implementación paralela del esquema DET-ABE, bajo un enfoque modular y por capas.
- Particionamiento de los algoritmos de cifrado, descifrado y generación de llaves del esquema DET-ABE, definiendo las secciones paralelizables y secuenciales.
- Implementación del algoritmo DET-ABE utilizando las estrategias propuestas. Los procesos `AES.Encrypt` y `AES.Decrypt` del cifrador simétrico son acelerados con GPUs en el lenguaje de programación CUDA-C y los procesos `CP-ABE.Encrypt`, `CP-ABE.Decrypt` y `CP-ABE.KeyGen` del cifrador asimétrico son acelerados con una estrategia multicore en el lenguaje de programación C utilizando la biblioteca OpenMP [11].

2. Etapa 2: Pruebas y evaluación de las estrategias de aceleración de DET-ABE (módulos independientes).
   - Evaluación de funcionalidad y desempeño del cifrador AES paralelo, variando los tamaños de archivos a cifrar, el número de bloques de procesamiento en CUDA y el nivel de seguridad.
   - Realizar un ciclo de identificación de mejoras, adecuación de la implementación y evaluación del cifrador AES.
   - Evaluación de funcionalidad y desempeño del cifrador CP-ABE paralelo, variando la cantidad de atributos utilizados en las políticas de acceso, la cantidad de atributos de

usuario utilizados, los niveles de seguridad, tipos de curva elíptica y el número de hilos en los que se divide el trabajo de los procesos.

- Realizar un ciclo de identificación de mejoras, adecuación de la implementación y evaluación del cifrador CP-ABE.
- Evaluación del desempeño del esquema DET-ABE paralelo variando los tamaños de archivo a cifrar, las políticas de acceso y la serie de atributos de usuario utilizadas.
- Documentación de resultados a partir del enfoque de la mejor implementación.

3. Etapa 3: Construcción de un prototipo experimental (evaluación de la integración de módulos).

- Diseño de la plataforma de experimentación para subir y descargar datos cifrados a la nube con DET-ABE acelerado.
- Implementación de un protocolo de subida y descarga de datos en la nube a través de TCP con sockets.
- Determinación de la aceleración conseguida y el impacto de ésta usando la plataforma de experimentación propuesta. Para ello, se evalúa bajo las mismas condiciones (software, hardware y datos de prueba) el desempeño de la versión secuencial de DET-ABE y la versión acelerada propuesta en esta tesis.
- Verificación y validación de desempeño del esquema DET-ABE paralelo.

De forma gráfica, en la Figura 1.3 se muestra un diagrama que define la metodología propuesta únicamente para las etapas 1 y 2, enfocadas a la definición de las estrategias de aceleración del esquema DET-ABE. En esta figura también se muestra al esquema DET-ABE seccionado en tres capas, las cuales corresponden a los niveles de procesamiento que posee el esquema. En la capa alta se encuentran los proceso de cifrado y descifrado a nivel de esquema, en la capa media están los procesos de cifrado y descifrado de AES y CP-ABE de forma independiente y en la capa más baja se encuentran las operaciones en campos finitos y grupos (CP-ABE) y operaciones lógicas (AES).

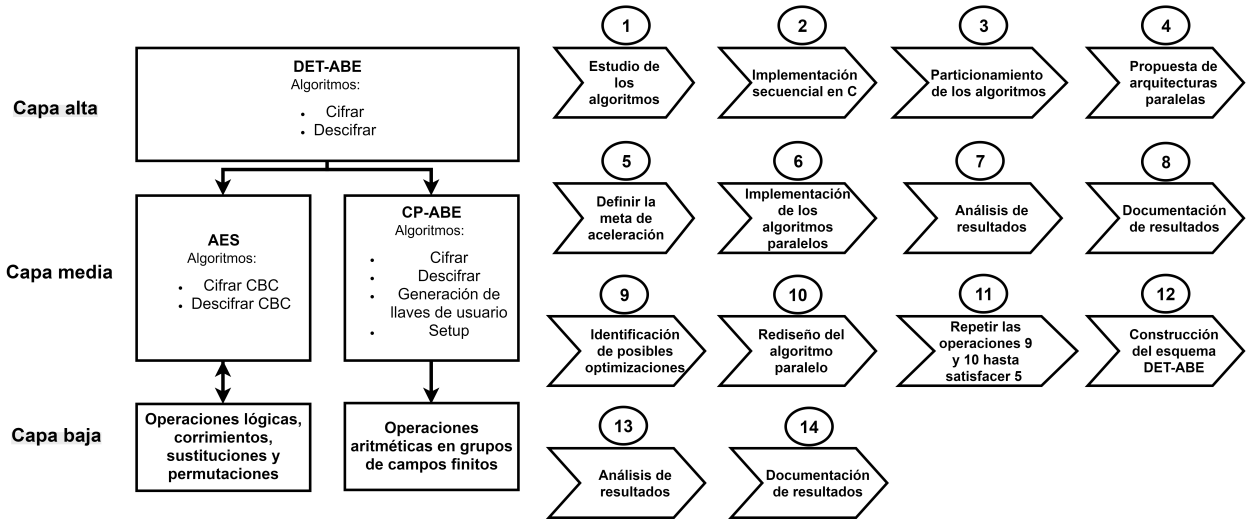


Figura 1.3: Metodología para la aceleración del esquema DET-ABE.

1.9 Organización del trabajo de tesis

La tesis se encuentra organizada de la siguiente manera:

- **Capítulo 2. Marco teórico:** En el segundo capítulo se describen los conceptos más relevantes sobre los que se desarrolla esta tesis, tanto de seguridad informática como de cómputo paralelo.
- **Capítulo 3. Estado del arte.** En este capítulo se describen los trabajos relacionados identificados en la literatura, organizados de a cuerdo a las diversas capas del esquema DET-ABE:
  - **Capa alta**. En esta capa se describen trabajos sobre aceleración de esquemas criptográficos basados en la técnica de sobre digital.
  - **Capa media**. En esta capa se describen trabajos sobre aceleraciones de AES y de CP-ABE.
  - **Capa baja**. En esta capa se describen trabajos donde se aceleran operaciones lógicas en AES y operaciones de álgebra abstracta que realiza el cifrador CP-ABE.

- **Capítulo 4. Diseño e implementación de DET-ABE secuencial**. Se describen las implementaciones de los cifradores AES, CP-ABE y el esquema DET-ABE en sus versiones secuenciales.

- **Capítulo 5. Diseño y desarrollo del método propuesto**. En este capítulo se describen las estrategias de paralelización de DET-ABE propuestas, en sus distintos niveles de procesamiento. De igual forma, se proveen los detalles de implementación de dicha estrategias.

- **Capítulo 6. Experimentación y resultados**. Se describen los distintos experimentos que se realizan para evaluar las estrategias de paralelización de DET-ABE. Se describen las métricas de interés, los datos de prueba y de configuración. En cada experimento, se discuten los resultados obtenidos, resaltando principalmente la aceleración obtenida en cada caso.

- **Capítulo 7. Conclusiones**. Se presentan las conclusiones obtenidas, se describen las principales contribuciones y se listan los aspectos que pueden abordarse como trabajo futuro.

2

Marco teórico

En este capítulo se aborda el marco conceptual en el que se sustenta el trabajo de tesis. Se describen conceptos como criptografía, servicios de seguridad informática, niveles de seguridad, cifradores simétricos, cifradores asimétricos y la técnica de sobre digital. Además, se describen herramientas y estrategias de cómputo paralelo para la aceleración de algoritmos. Se describe brevemente la evolución de la tecnología, que ha hecho posible la programación híbrida CPU-GPU.

2.1 Servicios de seguridad y criptografía

En esta sección se describen algunos conceptos y teoría acerca de la construcción de esquemas criptográficos a través de cifradores, los cuales se usan para proveer servicios de seguridad informática. Los cifradores, como bloques básicos, permiten construir esquemas de seguridad informática más complejos, como DET-ABE, el cuál permite implementar los servicios de confidencialidad y control de acceso sobre los datos que se almacenan y comparten en un entorno de nube.

2.1.1 Servicios de seguridad informática

La seguridad informática consiste en proveer servicios a la información cuando ésta se almacena o transmite por medios inseguros. Los servicios de seguridad que generalmente se demandan son [33]:

1. **Confidencialidad**. Impide la divulgación de datos a individuos, entidades o procesos no autorizados. En este servicio se asegura que el acceso a los datos únicamente es permitido a las entidades autorizadas.

2. **Autenticación**. Permite a una entidad garantizar su identidad y evitar la suplantación.

3. **Integridad**. Permite verificar si los datos han sido manipulados y alterados accidental o intencionalmente.

4. **No repudio**. Permite verificar que alguna entidad realizó alguna acción particular, evitando que ésta pueda negar haberla realizado.

5. **Control de acceso**. Permite restringir las acciones que una entidad puede realizar o los recursos a los que ésta puede acceder y usar dentro de un sistema informático.

La mayoría de los anteriores servicios de seguridad se proveen a través de la criptografía.

2.1.2 Criptografía

La palabra criprografía proviene de las palabras griegas *kryptos* (ocultar) y *grafos* (información). La criptografía estudia el diseño e implementación de cifradores, los cuales son algoritmos que transforman información legible en ilegible y viceversa, mediante el uso de una llave la cual es generalmente secreta y disponible solo para aquellas entidades autorizadas para acceder a la información en su forma legible [40]. Los cifradores ejecutan un par de operaciones principales:

- **Cifrar:** Es una operación que transforma texto plano en texto cifrado ilegible a través del uso de una llave, lo cual permite garantizar que el texto cifrado puede ser únicamente visualizado por el poseedor de la llave (únicamente entidades autorizadas).

- **Descifrar:** Es la operación inversa del cifrado, es decir, permite recuperar el texto plano de un texto cifrado haciendo uso de una llave, la cual no necesariamente tiene que ser la misma que se usó para cifrar el texto.

Los sistemas criptográficos pueden ser clasificados de forma genérica en tres dimensiones independientes:

1. **Mecanismo para transformar texto plano a texto cifrado**. La mayoría de los algoritmos de cifrado se basan en dos principios generales: de sustitución, en el que cada elemento en el texto plano se asigna a otro elemento en el texto cifrado y, la transposición, en los que elementos en el texto plano son reorganizados. El requisito fundamental es que no se pierda o destruya la información [63].

2. **Cantidad y longitud de las llaves utilizadas**. De manera general se suelen identificar dos categorías principales: criptografía simétrica, en la cual se utiliza una llave la cual tanto el emisor como el receptor conocen; y criptografía asimétrica, la cual hace uso de dos llaves, una llave pública que todos pueden conocer y una llave privada que sólo el propietario conoce [27].

3. **Mecanismo para el procesamiento de la información**. Para el procesamiento de la información se utilizan principalmente el cifrado de bloque y el de flujo. En el cifrado por bloques la información de entrada es fragmentada en bloques de tamaño fijo, produciendo un bloque cifrado en la salida por cada bloque de entrada. Por el contrario, en un cifrado de flujo, se trata la información como un flujo de bits o bytes, en el cual se llevan a cabo funciones matemáticas de forma individual, produciendo en la salida un flujo de datos cifrados [58].

La criptografía es el elemento fundamental para desarrollar aplicaciones en donde se requiere almacenar y transmitir información de forma tal que sólo aquellos a quienes se destina puedan leerla y procesarla. Está compuesta por técnicas orientadas a proteger la información sensible que necesite ser almacenada o transmitida en medios que sean considerados inseguros.

Un sistema que utiliza algoritmos de cifrado y descifrado para enviar mensajes de una manera segura es conocido como un criptosistema [58]. En el contexto de este trabajo de tesis, los datos se consideran como el mensaje que debe ser enviado y almacenado de una forma segura. La criptografía de llave privada y la de llave pública juegan un papel vital en las aplicaciones criptográficas modernas. Para varias aplicaciones, la combinación de éstos o más métodos se vuelve crucial al momento de proveer alguno de los servicios de seguridad mencionados en la Sección 2.1.2.

En la criptografía, el nivel de seguridad es el grado de protección sobre la información cifrada; El nivel de seguridad hace más complicado el descifrado de la información, entre más alto el nivel de seguridad, más difícil es realizar el descifrado sin el conocimiento de la llave mediante algún ataque. En el pasado, los cifradores usaban niveles de seguridad pequeños, por ejemplo, 80 bits eran suficientes para garantizar seguridad usando los dispositivos de cómputo existentes hasta el año 2010. El ataque más simple para intentar descifrar información cuando no se cuenta con la llave, es tratar de adivinar la llave mediante fuerza bruta (probando todas las combinaciones posibles). Sin embargo, con el avance de la tecnología el descubrimiento de la llave con un nivel de seguridad de 80 bits puede realizarse en algunos meses a inclusive días usando un ataque de fuerza bruta.

Debido a lo anterior, cada vez son utilizados niveles de seguridad más altos. Además, se crean nuevos esquemas criptográficos más complicados que permiten incrementar considerablemente la complejidad de descifrar la información por fuerza bruta, lo cual también incrementa el tiempo de procesamiento de los mismos cifradores. Actualmente se considera seguro un nivel de seguridad de 128 bits para cifradores simétricos y 256 bits para cifradores asimétricos basados en curvas elípticas hasta el año 2030 según el NIST [4, 46].

2.1.3 Cifradores simétricos

Un cifrador simétrico usa una sola llave, la cual es privada y debe ser salvaguardada por el emisor y receptor del mensaje cifrado. La llave es simétrica porque se usa tanto para cifrar como para descifrar un mensaje. La Figura 2.1 muestra un esquema del funcionamiento del cifrador simétrico.

Figura 2.1: Operaciones de cifrado simétrico.

La descripción formal de un cifrador simétrico es a través de la quíntupla (P, C, φ, E, D), donde: P representa el conjunto de un número finito de textos planos posibles, C representa el conjunto de un número finito de textos cifrados posibles, y φ representa el espacio de llaves, es decir, el conjunto finito de llaves posibles. Teniendo así las siguientes reglas:

- Para toda llave K en un conjunto de llaves φ, existe una función E_K que puede cifrar un conjunto de datos (regla de cifrado).
- Para toda llave K en un conjunto de llaves φ, existe una función D_K que puede descifrar un conjunto de datos (regla de descifrado).

Así, $E_K : P \rightarrow C$ y $D_K : C \rightarrow P$ son funciones bien definidas tal que para todo texto plano T, $T = D_K(E_K(T))$.

En un cifrador simétrico, un conjunto de datos T es cifrado por la función $E_K(K,\ T)$ generando un texto cifrado CT. Usando una función de descifrado $D_K(K,\ CT)$ es posible obtener el conjunto original T. Los algoritmos criptográficos basados en cifrado simétrico son considerados seguros si

calcular K a partir del texto cifrado CT se vuelve un problema computacionalmente intratable. En un cifrador simétrico es necesario que el emisor y el receptor compartan la misma llave antes de la transmisión de los datos cifrados, por lo que el intercambio seguro de la llave privada es un problema.

Debido a que el emisor y el receptor utilizan la misma llave para cifrar y descifrar, un cifrador simétrico puede proporcionar confidencialidad, pero no puede proporcionar autenticación o no repudio, ya que no existe forma de probar cuál de los dos usuarios que comparte la llave cifró los datos.

Otro problema con los cifradores simétricos es el relacionado con la administración y almacenamiento de las llaves necesarias para establecer comunicación entre N usuarios, ya que se necesitan $N(N-1)/2$ llaves, lo cual es un crecimiento cuadrático en la cantidad de llaves [63].

Un cifrador simétrico se utiliza generalmente para cifrar grandes cantidades de información ya que utiliza operaciones relativamente simples (sustituciones, transposiciones, operaciones booleanas, etc). Estos cifradores solo proporcionan el servicio de confidencialidad [3].

2.1.4 Estandar de cifrado avanzado (AES, por sus siglas en inglés)

El cifrador simétrico AES [17, 52] nació en el contexto de un concurso iniciado en 1997 en National Institute of Standards and Technology (NIST) con el fin de descubrir al sucesor para el estándar de cifrado DES, y fue declarado el estándar efectivo actual en el año 2002. El estándar de cifrado avanzado (AES, por sus siglas en inglés), es uno de los algoritmos más seguros y más utilizados hoy en día.

AES basa su funcionamiento en varias sustituciones, permutaciones y transformaciones lineales, ejecutadas en bloques de datos de 16 Bytes - por lo que se le llama cifrado por bloques [13]. Estas operaciones se repiten varias veces, en “rondas de procesamiento". En cada ronda se calcula una llave de ronda, la cual se usa para transformar un bloque de datos. Dadas las características del cifrado por bloque, el cambio de un solo bit dentro de un bloque de texto plano, resulta en un bloque de texto cifrado completamente diferente. Esto es una clara ventaja sobre cifrados de flujo tradicionales.

El cifrado con AES puede ser realizado con tres diferentes niveles de seguridad, los cuales definen la longitud de la llave: 128, 192 o 256 bits.

El cifrado de un bloque de datos en AES se realiza en 11 rondas de procesamiento. Las transformaciones del bloque de datos se realiza sobre una matriz de 4×4 bytes llamada "*state*". Al término de las 11 rondas de procesamiento, la matriz *state* contiene los datos cifrados.

Sea **S** el valor del "*state*" previo a una transformación en AES y **S'** el "*state*" resultante después de aplicar dicha transformación sobre él. A continuación se muestran los procesos que se aplican al bloque de datos (*SubBytes*, *ShiftRows*, *MixColumns* y *AddRoundKey*), de manera visual junto a una breve descripción del proceso:

1. *SubBytes*. Como se muestra en la Figura 2.2, el bloque *state* se transforma remplazando cada byte por otro correspondiente a una tabla de sustitución.

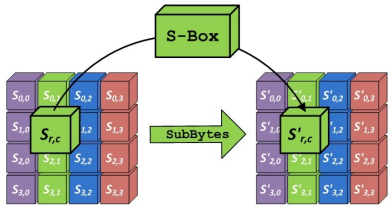


Figura 2.2: Tarea *SubBytes* de AES.

2. *ShiftRows*. Operación sobre las filas del *state*. La Figura 2.3 muestra el proceso de rotación que se realiza de forma cíclica a los bytes en cada fila del *state* con un determinado desplazamiento.

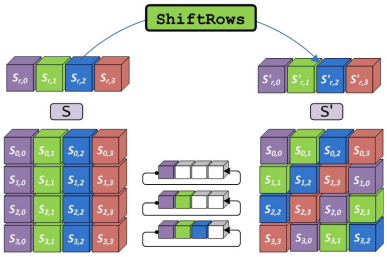


Figura 2.3: Tarea *ShiftRows* de AES.

3. *MixColumns*. Opera sobre las columnas del *state*. Los valores de una columna en el *state* se toman como los coeficientes de un polinomio de grado 4, dicho polinomio se multiplica por otro fijo *C(x)* con reducción módulo el polinomio $x^4 + 1$. La columna original se sustituye por los coeficientes del polinomio resultante. La Figura 2.4 ejemplifica como se modifican los valores de los bytes en las posiciones de la columna.

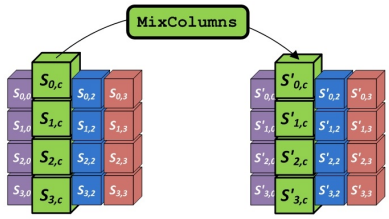


Figura 2.4: Tarea MixColumns de AES.

4. *AddRoundKey*. El *state* se combina con la sub-clave de ronda usando la operación **XOR** elemento por elemento, como se muestra en la Figura 2.5.

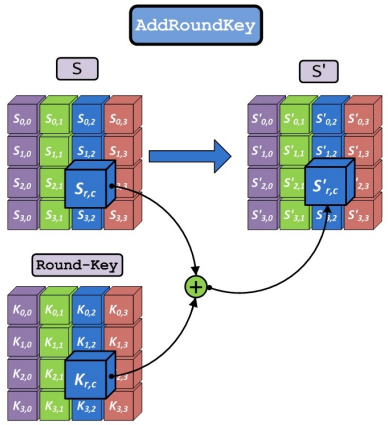


Figura 2.5: Tarea *AddRoundKey* de AES.

Existen diferentes modos de procesamiento de AES, donde el modo CBC [13] es el requerido para este trabajo, ya que es el recomendado para cifrar más de un bloque de datos y cuando es probable que los datos tengan bloques repetidos (imágenes, video o documentos de texto).

En el modo CBC (cipher-block chaining), antes de ser cifrado, a cada bloque de texto se le aplica una operación **XOR** con el bloque cifrado previamente, excepto el primero, el cual es operado con un vector inicial *VI*. De este modo, cada bloque cifrado depende de todos los bloques cifrados hasta ese punto.

Considérese que los datos a cifrar *T* se dividen en *N* bloques P_1, P_2, ... P_N. La fórmula matemática para el cifrado CBC es:

$$C_i = E_k(P_i \oplus C_{i-1}), C_0 = VI$$

Donde C_i es el bloque cifrado, E_k representa al proceso de cifrado (`AES.Encrypt`), P_i es el bloque a cifrar, C_{i-1} es el bloque cifrado previamente y *VI* el vector inicializador. Así, el bloque P_i a cifrar primero se combina mediante la operación *XOR* con el bloque cifrado anterior. A esta tarea se le llama *AddVI*.

En el modo CBC, la fórmula del descifrado es:

$$P_i = D_k(C_i \oplus C_{i-1}), C_0 = VI$$

Donde P_i es el bloque descifrado, D_k representa al proceso de descifrado (`AES.Decrypt`), C_i es el bloque a descifrar, C_{i-1} es el bloque cifrado previo y *VI* el vector inicializador.

El uso de un *VI* generado aleatoriamente evita dos bloques de texto cifrado idéntico en caso de que dos bloques de datos tengan el mismo contenido. No es necesario mantener el valor de *VI* como un parámetro privado, ya que es utilizado únicamente para dar confusión a los datos, el descifrado depende directamente de la llave de cifrado, sin importar que se conozca *VI* [2, 55].

2.1.5 Cifradores asimétricos

Uno de los mayores avances en el campo de la criptografía fue la creación de la criptografía de llave pública, propuesta por Diffie y Hellman en 1976 [12] dando origen a la construcción de los cifradores asimétricos. Este tipo de cifradores utilizan un par de llaves por cada usuario en el sistema, una llave pública y una privada, ambas relacionadas mediante una función matemática. Mientras que la llave pública es conocida y accesible por cualquier entidad, la correspondiente llave privada debe ser salvaguardada por su propietario [62]. La llave pública es empleada para el cifrado de datos, mientras que la llave privada se emplea para la operación de descifrado. Existe una relación entre ambas llaves, dado que a partir de la llave privada puede obtenerse la llave pública, pero la llave privada no puede ser obtenida a partir de la llave pública. La Figura 2.6 muestra de manera gráfica el funcionamiento del cifrador asimétrico, el cual provee los servicios de confidencialidad, integridad y autenticación [67].

Figura 2.6: Operaciones de cifrado asimétrico.

Un criptosistema asimétrico consiste en tres algoritmos principales: generación de par de llaves, algoritmo de cifrado y algoritmo de descifrado. Si $A_{\{K_{pub},K_{priv}\}}$ son la pareja de llaves pública y privada de la entidad A, E y D son las funciones de cifrado y descifrado respectivamente, entonces, el texto plano $T = D_{K_{priv}}(E_{K_{pub}}(T))$. Las llaves $\{K_{pub}, K_{priv}\}$ se relacionan mediante una función de un solo sentido, generalmente derivadas de un problema computacional difícil en el cuál se basan la seguridad del cifrador asimétrico [12].

Los algoritmos de cifrado asimétrico se dividen en tres familias de acuerdo al problema en que

se fundamentan: factorización de enteros, logaritmo discreto en grupos multiplicativos y logaritmo discreto en curvas elípticas [58].

El cifrado asimétrico permite resolver el problema de la distribución de llaves entre entidades, proporcionando las herramientas necesarias para implementar los servicios de confidencialidad, autenticación y no repudio de forma efectiva. Los algoritmos basados en cifrado asimétrico son computacionalmente más costosos que los algoritmos basados en cifrado simétrico. Un uso común de estos cifradores es para proveer el servicio de autenticación y en protocolos de establecimiento de llaves de sesión, donde se cifran cantidades pequeñas de información. Una explicación detallada sobre este tema se puede encontrar en [54, 63].

2.1.6 Criptografía de curva elíptica (ECC, por sus siglas en inglés)

Este tipo de criptografía fue propuesta en 1985 por Victor Miller y Neil Koblitz [44]. Es un tipo de criptografía de llave pública, la cual está fundamentada en el problema de logaritmo discreto en curvas elípticas.

Las curvas elípticas (**EC**, por sus siglas en inglés) son ecuaciones cúbicas definidas sobre campos finitos $\mathbb{F}_q$, generalmente un campo primo o binario denotados de la forma $\mathbb{F}_p$ o $\mathbb{F}_{2^m}$, donde p y 2^m representan el orden del campo finito respectivamente, p es un número primo y m un entero positivo. El orden es el número de elementos del campo finito.

Se dice que la curva elíptica $E(\mathbb{F}_q)$ está bien definida si su discriminante δ es diferente de 0. Esto último asegura que la curva elíptica no contiene puntos singulares para los cuales la adición no puede ser definida. Este grupo abeliano tiene que cumplir con cuatro propiedades, las cuales son asociatividad, conmutavidad, elemento de identidad y la existencia de inversos aditivo y multiplicativo. El elemento de identidad es llamado punto en el infinito P_∞ [20].

2.1.7 Criptografía basada en emparejamientos (PBC, por sus siglas en inglés)

En la teoría de grupos, un grupo es una estructuras algebraicas con número de elementos finito donde se puede definir el problema del logaritmo discreto. Un grupo tiene una operación bien definida sobre sus elementos que satisface distintos axiomas, como cerradura, existencia de inverso, asociatividad, y existencia de elemento neutro. Los puntos de una curva elíptica junto con P_∞ forman un grupo aditivo. Los enteros positivos (sin el cero) con reducción módulo un número primo forman un grupo multiplicativo.

Sea r el orden de un grupo. Un grupo cíclico $\mathbb{G}$ es aquel donde existe un elemento *P* en el grupo, tal que para cualquier elemento Q en el grupo, existe n un entero positivo, $1 \leq n \leq r-1$, que satisface $P^n = Q$, donde P^n indica la aplicación de la operación de grupo sobre P, $n-1$ veces. Además, se cumple que $P^r = P_\infty$. El grupo $\mathbb{G}$ con generador P se denota como $\mathbb{G} = \langle P \rangle$.

Un *emparejamiento bilineal* [6] es una operación definida sobre grupos cíclicos de la siguiente forma: sea $\mathbb{G}_1 = \langle g_1 \rangle$, $\mathbb{G}_2 = \langle g_2 \rangle$, y $\mathbb{G}_T$ grupos cíclicos de orden primo *r*. Un emparejamiento bilineal o mapa bilineal es una función eficiente $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, de tal manera que:

1. $\forall a, b \in \mathbb{Z}_r, e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$
2. $e(g_1, g_2) \neq 1$

Cuando $\mathbb{G}_1 = \mathbb{G}_2 = \mathbb{G}$, el emparejamiento se llama simétrico, de lo contrario se denomina asimétrico. Computacionalmente, $e(g, h)$ también se conoce como "emparejamiento" *g* con *h*.

Suponemos que las operaciones de grupo en $\mathbb{G}_1$, $\mathbb{G}_2$ y $\mathbb{G}_T$, así como el mapa bilineal *e*, son computables en tiempo polinomial con respecto al tamaño λ del grupo, y que los grupos $\mathbb{G}_1$, $\mathbb{G}_2$ y $\mathbb{G}_T$ contienen generadores.

La criptografía basada en emparejamientos es una familia de algoritmos y protocolos que basan su seguridad en el problema bilineal Diffie-Helman (**BDHP**, por sus siglas en inglés). El problema

BDHP consiste en dados g, g^a, g^b, g^c en un grupo $\mathbb{G}$, calcular $e(g,g)^{abc}$ [43].

Este problema es intratable, ya que para resolverlo se debe resolver el problema computacional Diffie-Helman (**CDHP**), el cual es intratable y consiste en que dados g^a, g^b, se puede calcular g^{ab}. Si se conociera g^{ab}, el valor $e(g,g)^{abc}$ se podría calcular fácilmente mediante el emparejamiento $e(g^{ab}, g^c)$, que por propiedades de los emparejamientos es igual a $e(g,g)^{abc}$. En [12, 43] se pueden consultar más detalles sobre **CDHP** y **BDHP**.

2.1.8 Criptografía basada en atributos (ABE, por sus siglas en inglés)

Los orígenes de los sistemas criptográficos ABE (Attribute Based Encryption) se remontan a 1984, cuando Shamir planteó la idea de tener un esquema de cifrado de llave pública, donde la llave pública fuera una cadena arbitraria (por ejemplo, una dirección de correo electrónico). Ese tipo de cifrado, conocido como "*cifrado basado en identidad*" (IBE, por sus siglas en inglés) [59], no fue realizado en práctica hasta el año 2001, cuando Boneh y Franklin propusieron un esquema IBE funcional construido sobre la teoría y las propiedades de emparejamientos bilineales en curvas elípticas especiales [7]. Más tarde, Shai y Waters [57] propusieron la utilización de un conjunto de atributos en lugar de un único identificador para el cifrado y descifrado, lo cual guió a un nuevo enfoque de cifrado de llave pública, conocido como "*cifrado basado en atributos*" (ABE, por sus siglas en ingles).

2.1.9 Implementación de esquemas ABE

Como en otros algoritmos criptográficos de llave pública [12, 62], la seguridad en ABE se basa en problemas complejos sobre grupos algebraicos. Los esquemas de cifrado basado en atributos se han construido principalmente sobre la criptografía basada en emparejamientos bilineales.

Existen dos tipos diferentes de sistemas criptográficos ABE: KP-ABE (Key-Policy Attribute Based Encryption) [16] y CP-ABE (Ciphertext-Policy Attribute Based Encryption) [5]. En ambos

esquemas, se utiliza una *estructura de acceso* $\mathbb{A}$ permitiendo el descifrado de los datos solo a usuarios autorizados, los cuales están implícitamente especificados por una política de control de acceso que se relaciona de forma directa con $\mathbb{A}$. Dicha política es generalmente una expresión booleana definida sobre un conjunto de *atributos* asignados a los usuarios del sistema, los cuales usualmente se representan como cadenas de caracteres. Las llaves de descifrado se generan para cada usuario, a partir de los atributos que éste posee. El número de atributos en la política de acceso y en la llave determina mayormente la complejidad de las operaciones de cifrado y descifrado en los esquemas ABE, pero es independiente del número de usuarios en el sistema. Esta es una característica realmente importante en los esquemas de cifrado ABE en términos de escalabilidad.

Debido a que CP-ABE es considerado más apropiado para implementar el concepto de control de acceso tal como en RBAC (Role Based Access Control) [65], este tipo de criptografía ABE es más atractiva para proveer confidencialidad y control de acceso de grano fino en datos cifrados almacenados en la nube.

2.1.10 Cifrador CP-ABE

CP-ABE se define por medio de cuatro algoritmos principales [5]:

- *Setup*, crea una llave pública *PK* y una llave maestra privada *MK* para un nivel de seguridad específico ζ. *PK* es usada para el cifrado y *MK* es usada para la generación de la llave privada de descifrado.
- *Encrypt*, toma un mensaje *M* como entrada, una política de control de acceso *P* y *PK* para producir un texto cifrado *CT*.
- *KeyGen*, toma como entrada *MK* y un conjunto de atributos *S*. Entonces produce *SK*, la cual es una llave secreta relacionada con *S* necesaria para descifrar el texto cifrado *CT*.
- *Decrypt*, toma como entrada *SK* y *CT*. Recupera el mensaje *M* a partir de *CT* solo si *SK* fue derivada de la serie de atributos que satisface a la política de acceso *P* usada en el cifrado.

2.1.10.1. Políticas y estructuras de acceso

En esquemas ABE, una política de control de acceso es usualmente una expresión booleana (con operaciones AND, OR o de tipo umbral de la forma *k-de-n*) de atributos tomados de un universo *U*, dichos atributos se expresan usualmente como cadenas de caracteres. Por ejemplo, sea el conjunto de atributos *U* = {*medico, enferemera, clinica_81, profersor, estudiante, cs_computacion*}. Las siguientes son ejemplos de políticas de control de acceso en ABE:

$$P_1 = (\textit{medico}\ \textbf{OR}\ \textit{enfermera})\ \textbf{AND}\ \textit{clinica\_81}$$
$$P_2 = \textbf{2-de-2}(\textbf{1-de-2}(\textit{medico, enfermera}),\ \textit{clinica\_81})$$
$$P_3 = \textbf{2-de-3}(\textit{profesor, estudiante, cs\_computacion})$$

En el ejemplo anterior, tanto P_1 como P_2 representan la misma política, pero se expresan en notación diferente, la primera usa operaciones lógicas y la segunda usa la notación de tipo umbral *k-de-n*.

En un esquema de seguridad basado en ABE, a cada usuario se le asigna un conjunto de atributos A_u del universo *U*. Un usuario satisface una política particular si la expresión booleana que representa a la política se evalua como verdadera con el conjunto de atributos A_u. Por ejemplo. Considere a los usuarios u_1, u_2, u_3 cada uno con los atributos {*recepcionista, clinica_81*}, {*medico, clinica_86}*, {*enfermera, clinica_81*}, respectivamente. De estos tres usuarios, solo u_3 satisface la política P_1, la cual es la misma que la política P_2.

A partir de la política de acceso, se crea una estructura de acceso que es utilizada por los algoritmos de cifrado y descifrado en CP-ABE. Formalmente, una estructura de acceso se define de la siguiente forma [5]: Sea $\mathbb{U} = \{p_1, p_2, ..., p_n\}$ un conjunto de partes. Una estructura de acceso es un conjunto $\mathbb{A}$ de subconjuntos no vacíos de $\mathbb{U}$, es decir, $\mathbb{A} \subseteq \mathbb{P}(\mathbb{U})$. Donde los conjuntos en $\mathbb{A}$ se definen como partes autorizadas, mientas que los conjuntos en $\mathbb{P}(\mathbb{U})$ pero que no están en $\mathbb{A}$ son no autorizadas. En el contexto de ABE, las partes son los atributos. Por lo tanto, la estructura de acceso contendrá los conjuntos de atributos autorizados, atributos que hacen que la expresión booleana asociada a la política de control de acceso sea evaluada como verdadera. Diversas implementaciones

de CP-ABE en la literatura han usado estructuras de acceso basadas en árbol [15, 16] o en matrices [38, 57].

En la implementación de CP-ABE descrita en este trabajo, las políticas de acceso se expresan como fórmulas booleanas en términos de operaciones del tipo umbral *k-de-n*. Las estructuras de acceso utilizadas son de tipo árbol, por lo que para facilitar su construcción, las políticas se expresan en notación post-fija. Un ejemplo de una política y su estructura de acceso se muestra en la Figura 2.7.

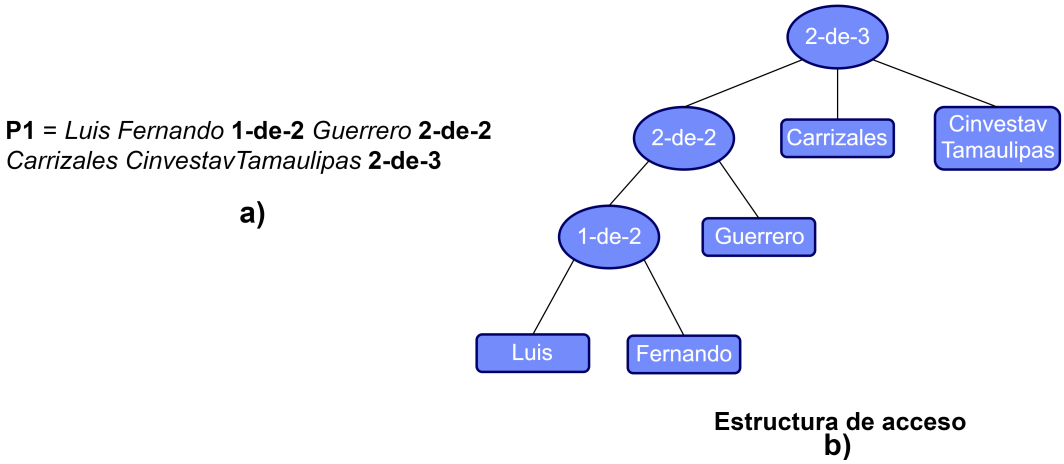


Figura 2.7: Política de control de acceso (a) y su estructura de acceso de tipo árbol asociado (b).

En la Figura 2.7 los nodos internos (en forma de ovalo) representan operaciones de tipo umbral *k-de-n*, los nodos hoja (en forma de cuadro) representan los atributos en la política de acceso.

2.1.10.2. Esquema de Shamir para la distribución de un secreto

Generalmente, un esquema CP-ABE con estructura de acceso basada en árbol usa el esquema de Shamir [5, 16] para implementar la función `CP-ABE.Encrypt` asignando un valor secreto a cada hoja de la estructura de acceso.

En general, el esquema de Shamir distribuye un secreto s a n entidades, asignando sub-secretos a cada uno de ellos. El secreto puede ser reconstruido con al menos k de los n (k-*de*-n) sub-secretos.

Para ello, el esquema de Shamir usa un polinomio de grado *k-1* con coeficientes aleatorios a_i y término independiente *s*, como se muestra a continuación:

$$P(x) = a_{k-1}x^{k-1} + a_{k-2}x^{k-2} + ... + a_1x^1 + s$$
$$P(x = 0) = s$$

Por ejemplo, considerando *s = 1000*, *k = 3*, *n = 4* y coeficientes aleatorios $a_2 = 3$, y $a_1 = 7$, se tiene el siguiente polinomio:

$$P(x) = 3x^2 + 7x + 1000$$

Con el cual se generan los 4 sub-secretos a repartir a las 4 entidades.

$$[1, P(1)], P(1) = 1010$$
$$[2, P(2)], P(2) = 1026$$
$$[3, P(3)], P(3) = 1048$$
$$[4, P(4)], P(4) = 1076$$

2.1.10.3. Esquema de Shamir para la reconstrucción de un secreto

Para poder reconstruir el secreto *s*, se necesita reconstruir el polinomio *P(x)* usando tres de los cuatro sub-secretos previamente distribuidos. Una vez obtenido el polinomio, se evalúa *P(0)* lo cual debe calcular *s*. Suponiendo que se tienen k pares de la forma *[x_i, y_i]* el esquema de Shamir reconstruye el secreto *s* usando la siguiente fórmula [5]:

$$s = \sum_{j=1}^{k-1} y_j * L_j$$

Donde, L_j es el coeficiente de interpolación de Lagrange, el cual se calcula como:

$$L_j = \prod_{i=1}^{k-1} \frac{-x_i}{x_j - x_i}, i \neq j$$

En el ejemplo anterior, dados los pares *[1,P(1)], [2,P(2)], [3,P(3)]*, los coeficientes de Lagrange son:

$$L_1 = \frac{-2}{1-2} * \frac{-3}{1-3} = \frac{-2}{-1} * \frac{-3}{-2} = 2 * 1{,}5 = 3$$

$$L_2 = \frac{-1}{2-1} * \frac{-3}{2-3} = \frac{-1}{1} * \frac{-3}{-1} = (-1) * 3 = -3$$

$$L_3 = \frac{-1}{3-1} * \frac{-2}{3-2} = \frac{-1}{2} * \frac{-2}{1} = (-0{,}5) * (-2) = 1$$

Una vez obtenidos los coeficientes de Lagrange se procede a calcular el secreto empleando la formula establecida:

$$s = (3 * 1010) + (-3 * 1026) + (1 * 1048)$$
$$s = 3030 - 3078 + 1048$$
$$s = 1000$$

2.2 Flujo de diseño e implementaciones en arquitecturas de cómputo híbrido

En esta sección se describen algunos conceptos acerca del cómputo paralelo, el desarrollo de algoritmos en el lenguaje de programación en C y el avance que se ha conseguido en el diseño de nuevos dispositivos de cómputo. También se describe como el lenguaje de programación C ha sido utilizado para explotar las características de estos dispositivos utilizando bibliotecas y estrategias de cómputo para obtener el mayor desempeño de éstos, consiguiendo así una mayor eficiencia en el desarrollo de tareas especificas como lo son el cifrado de datos u otras tareas que pueden ser identificadas en la literatura.

2.2.1 Aceleración de algoritmos mediante cómputo paralelo

Cómputo GPU o GPGPU (cómputo de propósito general en unidades de procesamiento de gráficos) es la utilización de unidades de procesamiento gráfico (GPU) para realizar cómputo en aplicaciones de ingeniería, medios digitales y científicos. Tradicionalmente, los GPUs son manejados por una unidad de procesamiento central (CPU). Los procesadores gráficos actúan como un co-procesador y poseen un enorme potencial de procesamiento paralelo en comparación con un CPU con múltiples núcleos [31].

En [32] se muestran estudios del paralelismo en arquitecturas híbridas, en donde ejecutan pruebas sobre diversos dispositivos con estas características, comprobando la factibilidad de su uso y la mejora en la eficiencia que puede obtenerse.

2.2.2 Estrategias de cómputo paralelo

La programación en C se basa en la construcción de procesos y sub-procesos, donde los procesos en un programa de ejecución es una imágen binaria en disco o en cualquier otro medio de almacenamiento. Cuando se ejecuta un programa, se lee el área de almacenamiento en el dispositivo periférico. Cada proceso consta de varios segmentos del programa, recursos del sistema e información del estado de ejecución. Estos procesos se encargan de desarrollar una serie de operaciones que satisfacen las necesidades de cada programa en cuestión. Por otra parte, los sub-procesos son secuencias de instrucciones ejecutadas dentro del contexto de un proceso. Los sub-procesos permiten que un proceso realice múltiples operaciones de forma paralela [36].

En el desarrollo de algoritmos, se parte de una serie de operaciones regidas por distintos procesos del programa que se está implementando, estas operaciones permiten establecer una comunicación entre las partes del proceso, permitiendo ejecutar ciertas operaciones en el sistema o realizar suspensiones en los procesos con el fin de administrar las operaciones realizadas por algunos procesos hijo. Así mismo, existen estrategias para la administración de procesos, lo cual puede dar al usuario la

impresión de que son realizadas una serie de procesos de forma simultánea. El objetivo de administrar los procesos es encargarse de que la CPU siempre se encuentre ocupada, realizando cálculos de forma "simultánea" en los diferentes procesos de una operación.

En la actualidad, la mayoría de los dispositivos de cómputo poseen la característica multinúcleo que hace posible contar con una serie de hilos, los cuales son manejados por un núcleo de forma independiente. Estos hilos pueden realizar procesos de forma paralela, implicando que cada uno puede ejecutar una serie de operaciones siguiendo la estrategia multi-procesador, obteniendo una mayor eficiencia al realizar diversas operaciones, estas operaciones son limitadas por la comunicación entre los hilos que realizan las operaciones requeridas por el programa a ejecutar. La comunicación entre estos hilos es realizada a través de primitivas de sincronización como exclusión mutua, uso de semáforos y variables de condición, lo cual reduce la eficiencia de las operaciones que son ejecutadas por los hilos. Sin embargo, al utilizar eficientemente dichas primitivas, es posible obtener un mayor beneficio en función a la cantidad de operaciones que realiza cada hilo previamente al uso de éstas.

Cuando se tiene una serie de hilos trabajando de forma simultánea es posible que varios hilos intenten acceder a la vez a la misma localización de memoria, razón por la cual existe la posibilidad de que una de estas operaciones entorpezca la de otros hilos. La sincronización de sob-procesos es una técnica para garantizar la integridad de los recursos compartidos entre hilos de un proceso. Existen algunas estrategias de sincronización básicas, las más representativas son [36]:

1. *Mutex*. Puede considerarse una variable especial que sirve para implementar un mecanismo de exclusión mutua, es decir, para evitar que más de un proceso ingrese a una sección critica. La sección critica es la sección de código donde se modifica un recurso compartido entre varios procesos.

   Un mutex se encuentra en uno de dos estados:

   - *Bloqueado*. Se restringe el acceso a una sección crítica al resto de los sub-procesos de los hilos de forma temporal. Sólo un subproceso puede bloquear un mutex en un momento

dado. Si un segundo subproceso trata de bloquear el mutex, el bloqueo fallará.

- *Desbloqueado*. Una vez que se termina de utilizar la sección crítica, el resto de los subprocesos pueden bloquear nuevamente el mutex.

2. *Semáforos*. Es una variable especial que constituye el método clásico para restringir o permitir el acceso a recursos compartidos (por ejemplo, un recurso de almacenamiento del sistema o variables del código fuente) en un entorno de multiprocesamiento.

3. *Variables de condición*. Son los tipos de datos y funciones necesarias para, junto con los mutex, poder construir "monitores". Entendiendo monitor como un conjunto de funciones que operan sobre un conjunto de datos en exclusión mutua.

2.2.3 Paralelismo de datos

La explotación del paralelismo de datos [19] proviene de la constatación natural de que ciertas aplicaciones trabajan con estructuras de datos muy regulares (vectores, matrices) repitiendo una misma acción sobre cada elemento de la estructura. Los recursos de cálculo se asocian entonces a los datos. A menudo existe un gran número de datos idénticos. Si el número de datos supera la cantidad de procesadores en segundo plano, el total de datos se reparte en el total de procesadores (Figura 2.8).

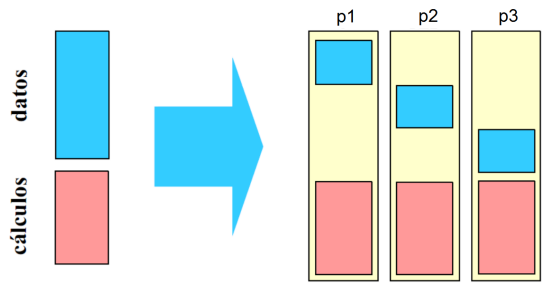


Figura 2.8: Paralelismo de datos.

Las limitaciones de este tipo de paralelismo vienen dadas por la necesidad de dividir los datos

para adecuarlos al tamaño soportado por los procesadores en segundo plano, la existencia de datos escalares que limitan el rendimiento de un dispositivo, además de la existencia de operaciones de difusión y reducciones que no son puramente paralelas. La Figura 2.9 muestra un ejemplo de como son distribuidos los datos a procesar, basado en la cantidades de procesadores en segundo plano disponibles.

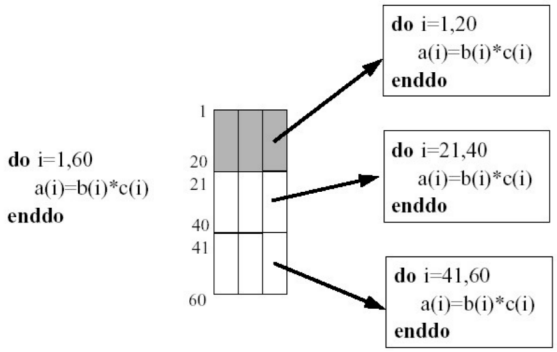


Figura 2.9: Ejemplo de paralelismo de datos.

2.2.4 Arquitectura de dispositivos gráficos de NVIDIA

El progreso técnico en el diseño de microprocesadores ha conseguido mejorar las capacidades de rendimiento de los dispositivos de cómputo actuales a través de algunos avances en [4]:

- El incremento de la densidad de los transistores.
- El incremento en el rendimiento de los transistores.
- Las rutas de acceso a datos más amplias.
- Ejecución superescalar.
- Ejecución especulativa.

La ejecución superescalar define un tipo de microarquitectura de procesador capaz de ejecutar más de una instrucción por ciclo de reloj. Por otro lado, la ejecución especulativa es la ejecución de código por parte del procesador que no tiene por qué ser necesaria *a priori*.

Lo anterior ha encaminado al desarrollo de dispositivos de cómputo con múltiples núcleos, los cuales obtienen un mayor rendimiento al realizar operaciones complejas, permitiendo a su vez el desarrollo de arquitecturas de cómputo paralelas.

Dichas arquitecturas de cómputo son parte crucial en el desarrollo de arquitecturas de cómputo híbridas, ya que se parte de la misma estrategia de contar con diversos núcleos que conforman las unidades de procesamiento gráfico. Estos dispositivos gráficos son co-procesadores dedicados al procesamiento de gráficos u operaciones de punto flotante para aligerar la carga computacional en dispositivos CPU, mejorando el rendimiento de los dispositivos de cómputo actuales [8].

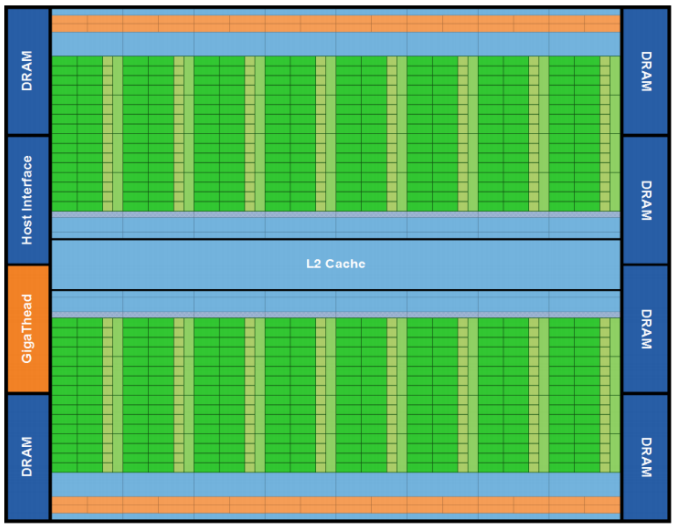


Figura 2.10: Arquitectura de una tarjeta gráfica de NVIDIA (Fermi).

La Figura 2.10 [8, 14] muestra la arquitectura (Fermi) de un dispositivo gráfico NVIDIA, donde los componentes de color verde obscuro son los núcleos de la tarjeta gráfica. Estos núcleos cumplen la función de realizar operaciones de forma paralela, obteniendo una mayor eficiencia al realizar operaciones con la estrategia divide y vencerás, sin embargo, los núcleos de estas tarjetas son menos eficientes que los de un equipo CPU normal, dependiendo así de la cantidad de núcleos

para obtener mejor resultados que los dispositivos multinúcleo convencionales. En la figura, se muestra una composición de múltiples bloques con múltiples núcleos, los cuales son denominados Multiprocesadores de flujo (*SM*, por sus siglas en inglés).

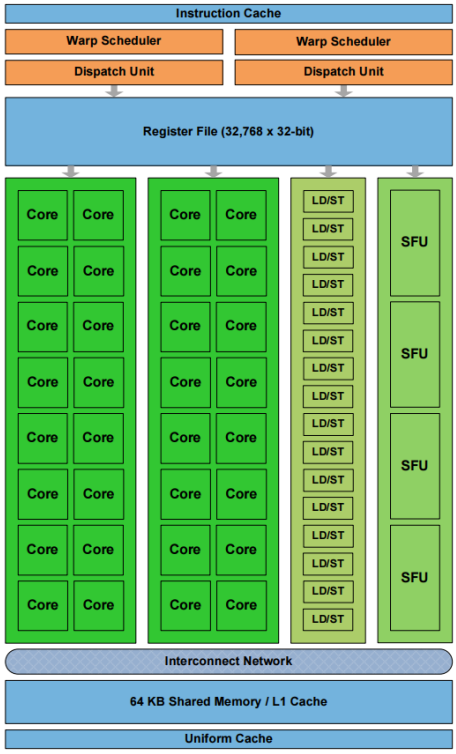


Figura 2.11: Arquitectura de un *SM*.

En la Figura 2.11 se muestra la composición de un SM, donde cada SM puede despachar varias decenas de hilos en paralelo y cada uno de estos hilos puede acceder a la memoria compartida del dispositivo GPU [14]. El total de núcleos de CUDA utilizables esta definido por el modelo y versión del dispositivo GPU disponible, una característica de los *SM* es que al utilizar tantos hilos como núcleos tenga el dispositivo se pueden acceder de forma paralela a los datos en memoria compartida sin necesidad de sincronización, evitando así el problema del cuello de botella. Sin embargo, el limitar el uso de hilos provoca que el aprovechamiento en el procesamiento por *SM* no sea el mejor, debido a que la mejor estrategia para obtener el mayor aprovechamiento de los dispositivos GPU es utilizar todos los recursos la mayor parte del tiempo en que se ejecutan los procesos (como se mencionó

previamente). El total de memoria compartida por bloque es pequeña (por ejemplo, el dispositivo utilizado en la experimentación posee 64KB de memoria compartida), de tal forma que se considera limitada la cantidad de espacios de memoria que pueden ser accedidas por todos los hilos en un *SM* [14].

En arquitecturas híbridas, el método de transferencia de datos del dispositivo anfitrión (CPU) al dispositivo gráfico (GPU) se realiza según la arquitectura mostrada en la Figura 2.12. El CPU se encarga de realizar los procesos secuenciales en el lenguaje CUDA C, mientras que para ejecutar un kernel de procesamiento se necesita transferir los datos a procesar a través de un bus PCI express (PCIe) para conectar dispositivos periféricos directamente a la placa base. Una vez que se transfieren los datos al dispositivo GPU, se ejecuta el kernel en el que se procesan los datos. Una vez que el kernel termina su ejecución, transfiere los datos nuevamente al CPU utilizando la misma estrategia de transferencia.

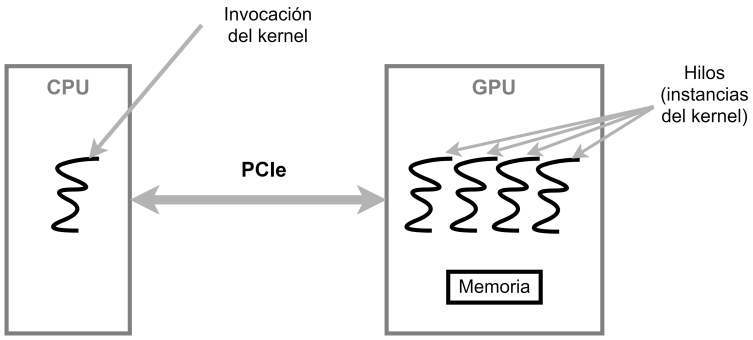


Figura 2.12: Arquitectura de transferencia de datos del CPU al GPU y viceversa.

La estrategia que los dispositivos aceleradores gráficos de NVIDIA siguen para la gestión de datos es la mostrada en la Figura 2.13, donde se muestra el modelo de memoria de CUDA expuesto al programador en términos de asignación, transferencia y utilización de los diferentes tipos de memoria del dispositivo. En la parte inferior de la figura, se encuentran las memorias de tipo global y constante. Estas memorias son aquellas a las que el procesador principal puede transferir datos de manera bidireccional (por bloque de procesamiento). Las memorias locales y compartida se realizan a diferentes niveles de procesamiento. Donde, las memorias locales transfieren datos de manera

bidireccional entre los registros y los hilos de procesamiento (flujo). Por otro lado, la memoria compartida transfiere datos entre los bloques de hilos de procesamiento en el GPU (por bloques de procesamiento).

Desde el punto de vista del dispositivo, se puede acceder a diferentes tipos de memoria con los modos siguientes: [18].

- Acceso de lectura/escritura a la memoria global, por bloque de procesamiento.
- Acceso solo de lectura a la memoria constante, por bloque de procesamiento.
- Acceso de lectura/escritura de los registros, por flujo.
- Acceso de lectura/escritura a la memoria local, por flujo.
- Acceso de lectura/escritura a la memoria compartida, por bloque.

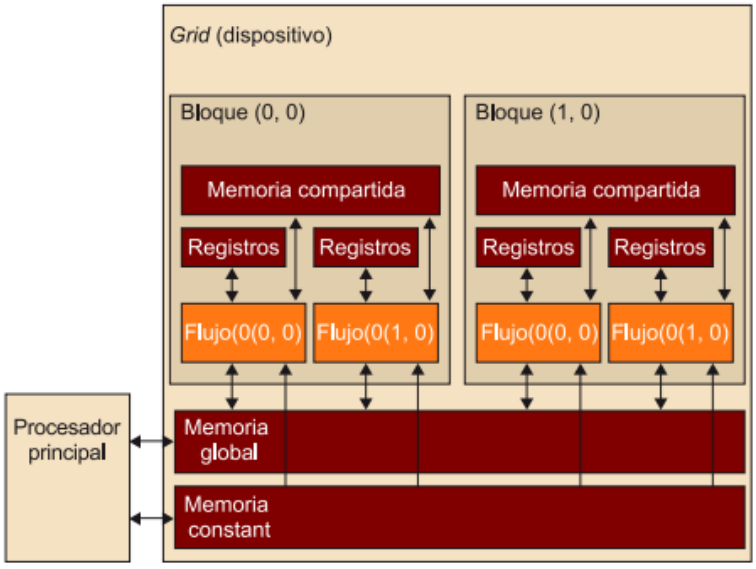


Figura 2.13: Arquitectura de transferencia de datos en CUDA.

2.3 Resumen

En este capítulo se revisaron las primitivas criptográficas principales utilizadas para proveer servicios relacionados con la seguridad de la información. Se analizaron los principales algoritmos criptográficos utilizados en este trabajo de investigación, principalmente el cifrador AES y CP-ABE para construir el esquema de seguridad DET-ABE que permite ofrecer servicios de privacidad, confidencialidad y control de acceso en el almacenamiento de datos en la nube. Los conceptos y esquemas criptográficos descritos en este capítulo son utilizados a lo largo del presente trabajo de tesis para construir las estrategias de aceleración de DET-ABE propuestas.

Así mismo, se revisaron los conceptos sobre el flujo de diseño e implementaciones en arquitecturas híbridas, las cuales son la base de este trabajo de investigación, ya que el propósito de esta tesis es aprovechar las arquitecturas de cómputo modernas para acelerar el esquema DET-ABE mediante una implementación híbrida empleando CPUs y GPUs.

En el siguiente capítulo se discuten trabajos del estado del arte acerca de aceleraciones de los cifradores AES y CP-ABE, así mismo de las aceleraciones de esquemas basados en sobre digital, los cuales sirvieron de referencia para la construcción del esquema DET-ABE híbrido. Por otro lado, estos trabajos sirven de referencia para evaluar y comparar la eficiencia de los algoritmos acelerados en esta tesis.

3

Estado del arte

En la literatura existen trabajos relacionados con el estudio e implementación de esquemas de seguridad de datos en la nube. Algunos se basan en el uso de una Infraestructura de Llave Pública (PKI, por sus siglas en inglés) [1], la cual requiere de una combinación de hardware y software, políticas y procedimientos de seguridad que permite certificar la identidad de un usuario y verificar que dicho usuario es propietario de una llave pública particular, lo cual es necesario por el libre acceso a las llaves públicas de los usuarios. Esquemas como los reportados en [47, 70] hacen uso de este tipo de infraestructuras. El uso del esquema DET-ABE [45] estudiado en esta tesis resulta atractivo, dado que evita el uso de PKI, la cual puede ser costosa e inviable en algunos casos, ya que requiere de una infraestructura para emitir, almacenar, validar y revocar certificados digitales.

En este capítulo se revisan los trabajos mas representativos en la literatura sobre la aceleración de los bloques principales que conforman el esquema DET-ABE, particularmente trabajos reportados sobre aceleración de los cifradores AES y CP-ABE en sus diferentes etapas de procesamiento.

Este es el primer trabajo que estudia la aceleración mediante cómputo híbrido de un esquema de

seguridad basado en sobre digital, el esquema DET-ABE, éste puede ser analizado en sus distintos niveles de procesamiento, de tal forma que es posible reducir sus tiempos de procesamiento a través de una estrategia de paralelización. En la Tabla 3.1 se muestran los algoritmos estudiados de DET-ABE a distintos niveles de porcesamiento.

Tabla 3.1: Algoritmos del esquema DET-ABE a diferentes niveles de procesamiento.

Nivel	Algoritmos involucrados	Descripción
Alto	`DET-ABE.Encrypt(datoslegibles; politica; nivelseguridad)` `DET-ABE.Decrypt(datoscifrados; atributos)`	El esquema DET-ABE en modo cifrar requiere como entrada los datos a cifrar, la política de acceso y el nivel de seguridad, la salida de este proceso es un conjunto de datos cifrados que conforman un sobre digital. En modo descifrar requiere los datos cifrados y un conjunto de atributos de usuario, su salida es el mismo conjunto de datos en claro que sirvió de entrada en el cifrado.

Medio	`AES.KeyGen(nivelseguridad)`
	`AES.Encrypt(datoslegibles; llave)`
	`AES.Decrypt(datoscifrados; llave)`
	`CP-ABE.Encrypt(llave; politica;nivelseguridad)`
	`CP-ABE.KeyGen(atributos; llavepblica)`
	`CP-ABE.Decrypt(llavecifrada; llavedeusuario)`
		El generador de llaves recibe como entrada el nivel de seguridad, su salida es la llave AES. El cifrador AES recibe los datos y la llave previamente generada, su salida es el texto cifrado. El cifrador CP-ABE recibe la llave AES, la política de acceso y el nivel de seguridad, su salida es la llave AES cifrada.El proceso de generación de llaves de cifrado toma como entrada una serie de atributos y una llave pública, la salida de este algoritmo es la llave que permite descifrar los datos. El proceso de descifrado en CP-ABE se lleva a cabo utilizando la llave de usuario y la llave cifrada con `{CP-ABE.Encrypt}, el producto de esta` operación es la llave de descifrado (llave AES). El descifrado en AES toma el conjunto de datos y la llave descifrada, obteniendo así como resultado el conjunto de datos original (datos legibles).

Bajo	g^{β} $e(g,g)^{\alpha} \in \mathbb{F}^*_{q^k}$ textbackslash $A * B$ $A + B$ $A, B \in \mathbb{F}_q$ $g \in \mathbb{G}$ $\beta, \alpha \in \mathbb{Z}_r$	La operaciones aritméticas básicas de CP-ABE incluyen exponenciaciones, multiplicaciones y sumas en campos finitos, así como emparejamientos bilineales.

3.1 Nivel alto

En el nivel alto de DET-ABE se encuentran las operaciones de cifrar y descifrar un conjunto de datos, usando un nivel de seguridad y política de acceso específicos.

No se encontraron trabajos relacionados que implementen esquemas del tipo DET. Sin embargo, en la capa alta, cuando se realiza el cifrado con DET-ABE, una vez generada la llave AES es posible realizar en paralelo el cifrado de los datos con AES y el cifrado de la llave con CP-ABE.

3.2 Nivel medio

En el nivel medio del procesamiento de DET-ABE se encuentran los algoritmos AES y CP-ABE operando independientemente. Para cada uno de ellos se han propuesto versiones aceleradas en cómputo híbrido.

3.2.1 Implementación híbrida de AES

El cifrador AES puede implementarse bajo distintos modos de operación, siendo los más comunes el modo ECB (Electronic Code Book) y el modo EBC (Ciphertext Block Chaining). En el modo ECB, cada bloque de datos se cifra de manera independiente. Por el contrario, en el modo CBC el cifrado se realiza mediante un encadenamiento de bloques, esto es, para cifrar el siguiente bloque se usa el bloque previo cifrado. Una descripción más detallada de estos modos de operación se presenta en el Capítulo 4.

Además de los modos ECB y CBC, existen algunos otros modos de operación, como lo son CTR, CWC y XTS, los cuales son altamente paralelizables, los cuales simulan un cifrado por flujo, no considerado en el contexto de aplicación en esta tesis. Para una especificación más detallada de estos modos de operación, el lector puede referirse a [52].

EL cifrador AES ha sido estudiado para ser implementado de forma paralela en diferentes plataformas que incluyen a los GPUs. AES en modo ECB es altamente paralelizable, sin embargo, el modo ECB no se recomienda para cifrar grandes volúmenes de datos como ocurre en escenarios de cómputo en la Nube y BigData. En el modo CBC, el algoritmo AES puede paralizarse solo en el modo descifrado [69], debido al encadenamiento de los bloques.

En [69] se describen diferentes puntos de paralelización del algoritmo AES, en donde se abordan de manera independiente sus 4 etapas de procesamiento. En ese trabajo se realizan pruebas con una tarjeta gráfica NVIDIA GeForce 8800 GTS, haciendo uso del lenguaje CUDA-C. Se describen también algunas estrategias para una implementación óptima en las diferentes etapas de procesamiento.

Posteriormente al análisis de dichos puntos de paralelización se realizan comparaciones de distintos resultados de la paralelización de AES considerando sus distintos modos de funcionamiento.

El trabajo reportado en [48] describe los resultados de aceleración de AES en cuatro clusters, incluidos en TOP500[1] de Junio del 2012. Dichos clusters fueron empleados para paralelizar los algoritmos de cifrado AES y DES. Se describen las estrategias de paralelización implementadas. En ese trabajo se emplean diferentes métodos y arquitecturas para la paralelización, además de que se analizan los resultados en diversos modos de procesamiento, de donde se concluye que la paralelización en el modo CBC es factible únicamente para la operación de descifrado.

El trabajo reportado en [41] se enfoca también en la paralelización del algoritmo AES. Un aspecto relevante en este trabajo es que se describe especialmente el uso de memoria en la estrategia de paralelización. En ese trabajo se establecen los puntos paralelizables del algoritmo, definiendo con ello las partes paralelizables y las inherentemente secuenciales. Se describe el manejo de memorias del CPU, GPU y de la memoria compartida requerida para el procesamiento, además de aportar ideas sobre cómo se pueden distribuir los datos durante el procesamiento.

El trabajo reportado en [49] realiza una división del trabajo de los componentes de un bloque de cifrado en 16 hilos, lo cual significa una aceleración a bajo nivel, mientras que se realiza la paralelización a nivel de bloques de cifrado de AES. Sin embargo, ese trabajo se basa en el modo CTR (el cual es altamente paralelizable), donde se obtuvieron aceleraciones considerables sin considerar tiempos de lectura y escritura de los datos a memoria. El alto nivel de paralelización obtenido es debido al modo de cifrado utilizado.

En el trabajo reportado en [37] se muestran las aceleraciones obtenidas para diferentes cifradores simétricos (AES, RC5, 3DES y Twofish), utilizando el modo ECB. Twofish resultó ser la implementación paralela con mayor rendimiento, mostrando una mayor cantidad de bytes procesados por segundo (aproximadamente 4100 Mbps) en las pruebas realizadas para un tamaño de archivo de

[1]TOP500 es un ranking de las 500 supercomputadoras con mayor rendimiento en el mundo. https://www.top500.org/

300MB.

En [68] se muestran los tiempos de procesamiento para tres estrategias de cifrado diferentes, donde una de ellas es el cifrador AES en modo XTS (medianamente paralelizable). En ese documento se describen estrategias de aceleración similares a [37, 49], las cuales obtienen menores rendimientos de procesamiento debido al modo de cifrado, lo cual sucede también en este trabajo de investigación al conseguir acelerar un modo de cifrado no paralelizable bajo una nueva propuesta de división por bloques, donde el rendimiento no es mejor que el de otros en la literatura, no obstante, se consigue acelerar tanto el cifrado como el descifrado en AES-CBC.

En el trabajo [21] se describe una estrategia de aceleración en el modo CWC, el cual es un modo de cifrado que proporciona un mayor nivel de seguridad a la información que el modo ECB o CTR y sus características permiten su aceleración. También se muestran diversos enfoques que permiten la implementación de un algoritmo optimizado de AES. Por último, se describen las características del procesamiento en GPUs, como lo son la posibilidad de procesar la información en forma de vectores, matrices o matrices tridimensionales, permitiendo acceder de forma rápida y eficiente a los bytes de cada posición de los bloques de cifrado de AES, los cuales son procesados en paralelo reduciendo los tiempos de traslado de datos de la memoria del CPU a la memoria del GPU.

3.2.2 Implementación híbrida de CP-ABE

En la revisión del estado del arte se identificó un trabajo acerca de la paralelización del cifrador asimétrico CP-ABE, no obstante la generación de las llaves de descifrado, cifrado y descifrado de los datos son paralelizables, ya que tales operaciones procesan secuencialmente un conjunto de atributos, lo cual puede realizarse en paralelo pudiendo conseguir una ganancia significativa en el tiempo de procesamiento.

En el trabajo reportado en [34] se describen algunas estrategias de aceleración para el proceso de cifrado, descifrado y generación de llave de descifrado de CP-ABE. En ese trabajo, el procesamiento de los atributos es repartido en la cantidad de hilos disponibles en el dispositivo (GPU). La

experimentación se realiza sobre dos y cuatro hilos reduciendo el tiempo de procesamiento hasta aproximadamente 5× en los tres algoritmos acelerados.

3.3 Nivel bajo

En el nivel bajo de DET-ABE se encuentran las operaciones demandadas por CP-ABE, las cuales son operaciones aritméticas en curvas elípticas y grupos multiplicativos. Entre las operaciones más demandantes se encuentra el emparejamiento, la exponenciación y operaciones de multiplicación con reducción módulo un número primo grande (de cientos o miles de bits). Los trabajos relacionados en este nivel se refieren a APIs, muy pocas abordan la aceleración en GPU y no están disponibles para su uso, por ejemplo [51].

De acuerdo a la metodología de esta tesis, y dada la complejidad de implementación del conjunto de algoritmos para aritmética en los tres grupos $\mathbb{G}_1$, $\mathbb{G}_2$ y $\mathbb{G}_T$ de CP-ABE, se decidió revisar y usar una API disponible.

En esta sección se revisan las bibliotecas más populares en la literatura sobre aritmética en grupos para criptografía basada en emparejamientos, la cual es la que se requiere para implementar CP-ABE.

1. **PBC [39].** Esta biblioteca realiza operaciones con curvas elípticas y emparejamientos bilineales. Es utilizada en la implementación de DET-ABE secuencial, posee algoritmos complejos y dependencia a una biblioteca externa.

2. **TEPLA [30].** Es una biblioteca muy semejante a PBC, sin embargo los algoritmos que rigen las funciones de las operaciones algebraicas necesarias son mas simples que las de la biblioteca PBC. Esta biblioteca también tiene dependencia de una biblioteca externa.

3. **Pairings in C [64].** En esta biblioteca se realizan las operaciones mas básicas de álgebra abstracta necesarias para la implementación de DET-ABE.

4. **CUMP [61].** Esta biblioteca se basa en la aceleración de las operaciones a bajo nivel necesarias para reemplazar la biblioteca externa utilizada por TEPLA y PBC.

Sin embargo, ninguna de las bibliotecas anteriores permitió realizar una implementación eficiente de CP-ABE en CUDA, ya que las operaciones a bajo nivel no pueden ser aceleradas debido a que el entorno de CUDA no permite el uso de bibliotecas de C y C++ contenidas en dichas APIs. De las bibliotecas revisadas, **PBC** se seleccionó y usó para construir la implementación secuencial de CP-ABE. Los algoritmos de aritmética de esta misma biblioteca se usaron en la construcción de las versiones paralelas de CP-ABE. Aunque estos algoritmos no se aceleran en la capa baja, si se usan en las estrategias de aceleración de capa media, a nivel de algoritmos de CP-ABE.

3.4 Resumen de los trabajos revisados

En la Tabla 3.2 se presenta un resumen de las características más relevantes de los trabajos del estado del arte revisados y descritos anteriormente.

3.5 Discusión

El esquema DET-ABE puede ser paralelizable a distintos niveles como se establece en las Tablas 3.1 y 3.2; De los trabajos revisados, pocos se han enfocado en alternativas para la implementación del paralelismo en esquemas de sobre digital. Además, no fueron identificados casos de estudio en los cuales el paralelismo sea empleado para la aceleración del algoritmo del cifrador asimétrico CP-ABE a pesar de que tres de sus algoritmos principales son altamente paralelizables.

Existen diversos trabajos que reportan aceleraciones del cifrador simétrico AES, los cuales fueron considerados en esta tesis para construir un bloque funcional paralelo de este algoritmo dentro del esquema DET-ABE.

Existen algunos trabajos que han reportado estrategias de paralelismo en la capa baja que da

Tabla 3.2: Tabla comparativa del estado del arte (1)

Ref.	Año	Alg. Imp.	Leng. Prog.	Plataforma(s)	Comentarios	Resultados principales
[69]	2016	AES	OpenGL	NVIDIA GeForce 8800 GTS	Se establecen los puntos de paralelización, se determina que el modo CBC no puede ser paralelizable eficientemente tanto en cifrado como en descifrado.	Se obtiene una ganancia de procesamiento de datos por unidad de segundo de 2× respecto a la versión secuencial.
[48]	2012	AES	OpenCL	NVIDIA Tesla M2050 AMD FirePro V7800 AMD Radeon 6970 Intel Xeon X5650	Se realiza una comparación sobre cuatro diferentes dispositivos GPU.	Se establece que el equipo con la arquitectura híbrida que obtiene el mejor desempeño es por parte de Intel, siendo Xeon X5650 la cual obtuvo el mejor desempeño de aceleración (3.71× respecto a la versión secuencial).
[41]	2010	AES	CUDA-C	NVIDIA GeForce 9200 GS	Se establecen los puntos en los cuales es posible realizar el paralelismo y dónde el algoritmo es inherentemente secuencial. Además, se analiza la reserva de memoria necesaria en la implementación.	Se confirma que el paradigma del cómputo híbrido CPU-GPU aporta una mejora notable para realizar el descifrado de datos a partir del cifrador AES en modo CBC, obteniendo mejores resultados con el uso de 256 hilos.
[49]	2012	AES	CUDA-C	NVIDIA Tesla C2050 NVIDIA GeForce GTX 285	Se describen las estrategias de paralelización de AES, los cuales se retoman en esta tesis.	Además de revisar los resultados de la aceleración de AES en modo CTR, se revisaron otras estrategias de aceleración, utilizando una nueva implementación de AES-NI, obteniendo mejores resultados en througput, utilizando hilos sobre las implementaciones en GPUs de AES-CBC.
[37]	2009	AES	CUDA-C	NVIDIA GeForce 9800 GTX	Se describen las estrategias de aceleración para el modo de cifrado ECB.	Se obtuvieron aceleraciones de entre 4 y 30x con respecto a la versión secuencial. Estas aceleraciones consideran únicamente el procesamiento de los datos, no toman en cuenta operaciones de lectura y escritura de datos.
[68]	2011	AES	CUDA-C	NVIDIA GeForce GTX 285	Se describen estrategias de aceleración para el modo de cifrado XTS.	Se obtuvieron aceleraciones de entre 12 y 23× con respecto a la versión secuencial, considerando únicamente el procesamiento de los datos. En este documento también se mencionan los tiempos de procesamiento en la transferencia de los datos del CPU a los GPUs y viceversa.
[21]	2007	AES	CUDA-C	NVIDIA Geforce 6600 GT AGP8x NVIDIA Geforce 7900 GT PCIe	No se describen aceleraciones precisas, sin embargo el throughput obtenido se puede considerar para evaluar la eficiencia de la implementación paralela de AES de esta tesis, principalmente se consideró utilizar este trabajo para optimizar la estrategia propuesta.	Se describen los enfoques para la construcción de un cifrador AES paralelo, se compara el aprovechamiento contra implementaciones estándar en CPU utilizando operaciones del tipo XOR aceleradas en GPUs. Además se describen otras comparaciones con implementaciones variadas en CPU.
[34]	2016	CP-ABE	C	Intel 13 401OU	Se describen las estrategias de aceleración similares a las seguidas en este trabajo de tesis, las cuales pueden servir de referencia para evaluar el desempeño de la implementación paralela.	Se obtuvieron aceleraciones de entre 3.5 y 5x con respecto a la versión secuencial. En este documento se incrementa el performance de la implementación incrementando considerablemente el throughput obtenido en la versión secuencial.

soporte a la ejecución de los algoritmos de CP-ABE, sin embargo estas operaciones básicas no son directamente compatibles con los tipos de datos utilizados en las bibliotecas que proveen de mecanismos eficientes para la implementación de CP-ABE, por lo que estos trabajos no pueden ser considerados por el momento hasta no contar con una biblioteca propia para las operaciones en

grupos y campos finitos que sea compatible con CUDA.

3.6 Resumen

En este capítulo se revisaron los trabajos del estado del arte acerca de aceleraciones en DET-ABE a distintos niveles de procesamiento, habiendo encontrado una gran cantidad de trabajos enfocados en acelerar el cifrador AES y solamente uno para acelerar el cifrador CP-ABE. Dichos trabajos identificados fueron la base de las implementaciones secuenciales y paralelas del esquema DET-ABE en este trabajo.

Del mismo modo, en los trabajos revisados se identificaron métodos para mejorar las implementaciones, utilizando estrategias de aceleración considerando características de los dispositivos GPU. Un punto crucial en este capítulo es la identificación de trabajos relacionados para comparar la eficiencia de los algoritmos acelerados en esta tesis.

En el siguiente capítulo se describen las implementaciones realizadas para contar con un esquema de cifrado DET-ABE secuencial en C. Después, se describen las propuestas de paralelización de DET-ABE presentando varias versiones en cada nivel de procesamiento. En estas versiones paralelas se describen detalles como el uso de hilos, la necesidad de realizar sincronizaciones y las características de los dispositivos GPU utilizadas para obtener una mayor eficiencia en el cifrado de AES y CP-ABE.

4

Implementación del esquema de seguridad DET-ABE

El esquema de cifrado DET-ABE [45] está basado en la técnica de sobre digital (DET, por sus siglas en inglés) [56]. DET consiste en utilizar un cifrador simétrico para cifrar los datos usando una llave generada dinámicamente. Además, se usa un cifrador asimétrico para proteger la llave dinámica, la cual, al ser cifrada con la llave pública correspondiente a una entidad, permite el acceso a dicha entidad para poder descifrar la llave y por tanto tener acceso a los datos en su forma legible. Los datos cifrados y la llave cifrada son empaquetados y trasmitidos, a este empaquetamiento se le denomina **sobre digital** [56]. DET se emplea generalmente para obtener los beneficios del cifrado simétrico (procesamiento de grandes cantidades de datos) [3, 4] y a su vez del cifrado asimétrico (permite distribuir de forma segura la llave privada del cifrador simétrico a una entidad particular) [4, 62]. En la Figura 4.1 se muestra un esquema del cifrado de datos empleando DET.

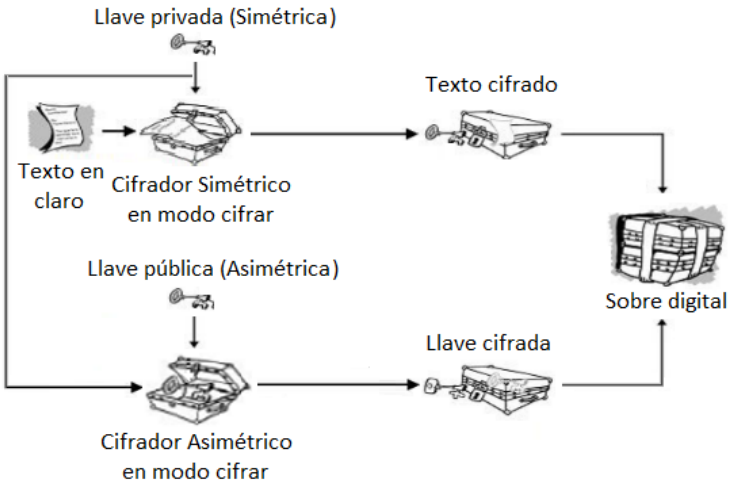


Figura 4.1: Diagrama de cifrado con DET.

El esquema DET-ABE realiza el cifrado de los datos a través del cifrador simétrico AES en modo CBC, mientras que para el cifrado de la llave AES se utiliza el cifrador asimétrico CP-ABE. Un diagrama del esquema DET-ABE se muestra en la Figura 4.2.

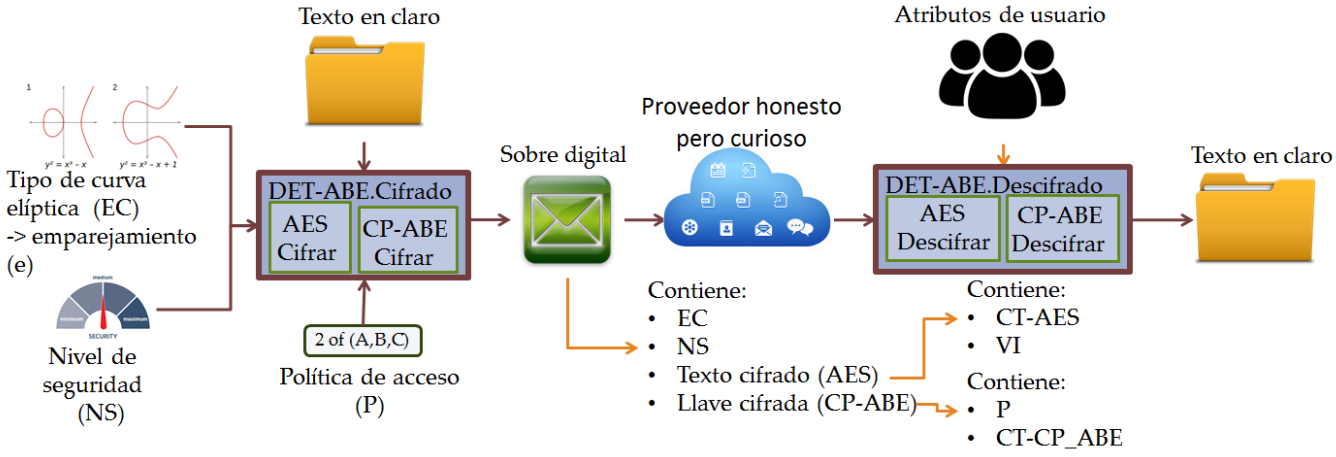


Figura 4.2: Diagrama del esquema DET-ABE.

Dado un tipo de curva elíptica, un nivel de seguridad, una política de acceso y un texto a cifrar (conjunto de datos de entrada) se lleva a cabo la etapa de cifrado con DET-ABE, para el cual el cifrador AES cifra los datos y el cifrador CP-ABE cifra la llave AES. Los datos cifrados y la llave

cifrada son empaquetados en un sobre digital. Adicionalmente también se incluye en el sobre digital el tipo de curva elíptica, el nivel de seguridad usado, el vector de inicialización *VI* de AES y la política de control de acceso.

Para el descifrado con DET-ABE se requiere del sobre digital y una serie de atributos de usuario como parámetros de entrada, donde los atributos son necesarios para validar el acceso de un usuario a los datos descifrados y para generar la llave de descifrado de CP-ABE. Con dicha llave, se descifra la llave AES, una vez establecidos el tipo de curva y el nivel de seguridad. Una vez descifrada la llave AES, se realiza el descifrado de los datos con AES en modo descifrar, la salida del bloque de la etapa de descifrado es el texto en claro.

En este capítulo se detallan las fases del desarrollo e implementación del esquema DET-ABE secuencial. Se describen las estrategias de construcción del esquema y los algoritmos implementados. El proceso de cifrado se realiza con la función `DET-ABE.Encrypt` y el de descifrado con la función `DET-ABE.Decrypt`, las cuales utilizan las funciones de cifrado y descifrado de los cifradores AES y CP-ABE. Por ello se detallan las implementaciones de los algoritmos necesarios para el cifrado, generación de llaves y descifrado en AES y CP-ABE; en el caso del cifrador AES, se definen las operaciones mas básicas, correspondientes a sustituciones, permutaciones y transformaciones lineales. Los procesos `DET-ABE.Setup` y `DET-ABE.KeyGen` son básicamente los mismos utilizados en CP-ABE.

4.1 Implementación del cifrador CP-ABE

A continuación se describirán los algoritmos para construir las estructuras de acceso a partir de políticas de control de acceso, el esquema Shamir para la distribución de un secreto, el cifrado de los datos, la generación de la llave de descifrado, el proceso de reconstrucción del secreto y el descifrado de los datos en CP-ABE, los cuales fueron descritos en el Capítulo 2 y que se usan en los algoritmos subyacentes de cifrado y descifrado de CP-ABE.

4.1.1 Construcción de la estructura de acceso

Dada como entrada una política de control de acceso, expresada con operaciones de tipo *umbral k-de-n* en notación posfija, el Algoritmo 1 construye su correspondiente estructura de acceso de tipo árbol, la cual puede usarse en los algoritmos de cifrado y descifrado de CP-ABE. El Algoritmo 1 usa una pila donde almacena temporalmente los nodos que conformarán la estructura de árbol. Un *nodo* en la estructura de acceso puede ser hoja o padre, donde el *nodo hoja* representa un atributo en la política de acceso, mientras que un *nodo padre* representa a un *umbral*, al cual se puede acceder únicamente si se satisfacen las condiciones que lo rigen. Por ejemplo, para la política {P = *A B C 2-de-3*}, la correspondiente estructura de acceso consiste de un nodo padre y tres nodos hijos (hojas). El nodo padre representa al umbral *2-de-3*, los nodos hoja contienen los atributos A, B Y C. La estructura de acceso se satisface si un usuario tienen dos de los tres atributos indicados, ya sean A y B, B y C o A y C.

Al inicio del Algoritmo 1, la política de acceso se recibe como una cadena de texto en notación posfija. Esta cadena se divide en tokens, obteniendo un arreglo correspondiente de cadenas de caracteres que representan o umbrales del tipo *k-de-n* o atributos.

Iterativamente, cada uno de estos tokens se procesa de acuerdo a las siguientes reglas:

1. Si el *token* es un atributo, (será un nodo hoja en el árbol), se crea un nodo donde se almacena el atributo. El nodo se almacena en la pila.

2. Si el *token* es un umbral *k-de-n* (será un nodo interno en el árbol), se crea un nodo y en él se almacenan los valores de k y n. Se obtienen n nodos de la pila y se asignan como hijos del nodo recién creado. Finalmente, el nodo recién creado, ya con los hijos asignados, se almacena en la pila.

El proceso de construcción de la estructura de acceso termina cuando todos los tokens se han procesado. Al final, la pila debe tener un solo nodo, que corresponde al nodo raíz del árbol. De lo

```
Entrada: P : política de acceso en notación posfija.
Salida: 𝔸 : estructura de acceso.
 1: 𝔸.raíz = NULL
 2: Inicializar(Pila)
 3: tokens ← ObtenerTokens(P)
 4: para cada token en tokens hacer
 5:     si token es "k-de-n" entonces
 6:         nodo ← nuevoNodo(k, n)
 7:         para i ← 0 hasta n hacer
 8:             nodoHijo ← Pila.Extraer()
 9:             nodo.asignarHijo(nodoHijo)
10:         fin para
11:         Apilar(Pila, nodo)
12:     si no
13:         nodo ← nuevoNodo(atributo)
14:         Apilar(Pila, nodo)
15:     fin si
16: fin para
17: 𝔸.raíz ← Pila.Extraer()
18: si !Vacia(Pila) entonces
19:     Escribir "Política errónea"
20:     devolver NULL
21: si no
22:     devolver 𝔸
23: fin si
```

Algoritmo 1: Función ConstruirEstrucura(P), para la construcción de la estructura de acceso $\mathbb{A}$ asociada a una política P.

contrario, el árbol generado será incorrecto y lo más probable es que la política que se ha ingresado no tenía una estructura válida.

Un ejemplo del proceso que se lleva a cabo en el Algoritmo 1 es el que se muestra en la Figura 4.3, donde dada la política de acceso $\{P =$ *Luis Fernando 1-de-2*$\}$ se realiza la lectura de sus componentes de izquierda a derecha (1). Los dos primeros tokens *Luis* y *Fernando* son atributos de usuario (2), por lo que se generan los nodos hoja para cada uno y ambos son agregados a la pila (4). El tercer token "*1-de-2*" corresponde a un nodo interno con componentes $k = 1$ y $n = 2$. A este nodo se le asignan sus *2* nodos hijo en el tope de la pila (atributos *Luis* y *Fernando*) y se agrega a la pila. Dado que no hay más tokens, el proceso termina regresando el único nodo en la pila, el cual corresponde al nodo raíz del árbol mostrado en el punto (3) de la Figura 4.3.

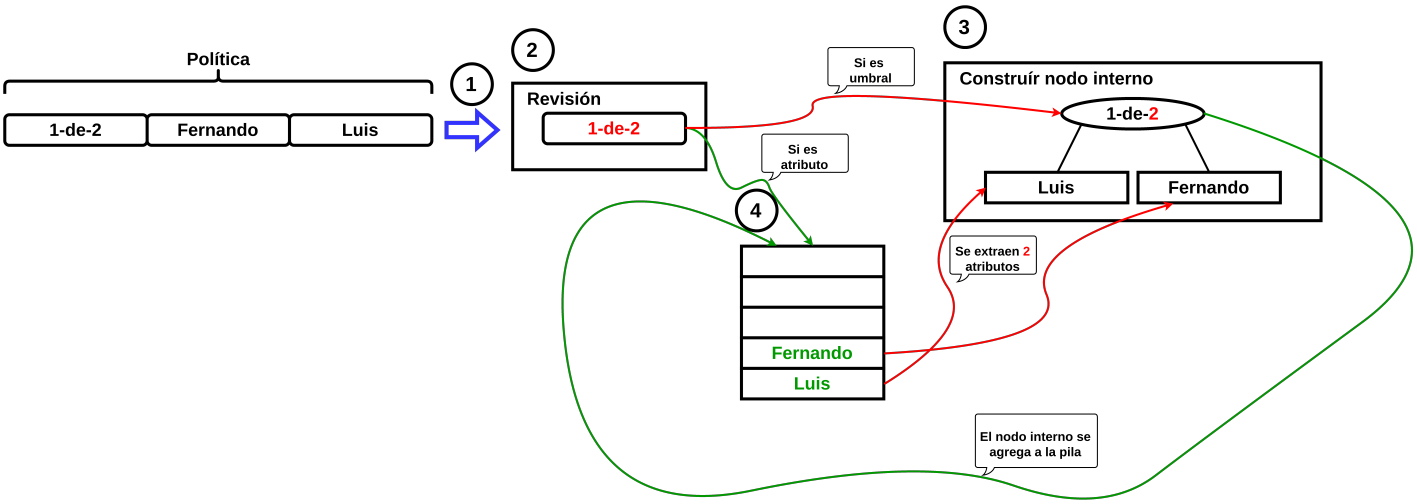


Figura 4.3: Descripción gráfica del Algoritmo 1 para construir la estructura de acceso asociada a la política $P =$ *Luis Fernando 1-de-2*.

4.1.2 Inicialización de parámetros (`CP-ABE.Setup`)

Las operaciones de cifrado y descifrado de CP-ABE se realizan asumiendo que el algoritmo `CP-ABE.Setup` ha sido ejecutado previamente. Este algoritmo genera las llaves pública *PK* y maestra *MK* de CP-ABE, las cuales se implementan como operaciones en los grupos $\mathbb{G}_1$, $\mathbb{G}_2$ y $\mathbb{G}_T$ (los tres de orden r) que definen el emparejamiento $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$.

La llave privada *PK* se define como la tupla $PK = \{g_0, g_1, h = g_0^{\beta}, e(g_0, g_1)^{\alpha}\}$, donde α y β son números aleatorios en $\mathbb{Z}_r$, g_0 y g_1 son generadores en $\mathbb{G}_1$ y $\mathbb{G}_2$ respectivamente. La llave maestra es $MK = \{\beta, {g_1}^{\alpha}\}$.

4.1.3 Cifrado (`CP-ABE.Encrypt`)

El texto cifrado de un mensaje *M* en CP-ABE se define como la tupla $CT = \{C, C', L\}$, donde $C = h^s$, $C' = M \times e(g_0, g_1)^{\alpha s}$ y $L = \{p_y\}^+$ es una lista de uno o más pares $py = [C_y, C'_y]$, que se

obtienen cuando se procesa un nodo hoja de $\mathbb{A}$ que contiene el atributo y. El Algoritmo 2 describe el proceso de cifrado en CP-ABE. En este algoritmo, el mensaje *M* a cifrar se transforma en un número en el grupo $\mathbb{G}_T$. Internamente se genera un número aleatorio *s* en $\mathbb{Z}_r$ y se usa junto con *PK* para calcular C Y C'. El secreto *s* se "oculta" en los atributos de la política $\mathbb{A}$, usando la función `DShamir` descrita en el Algoritmo 3, que implementa el esquema de Shamir para la distribución de un secreto. La función `Mapear(`u`, `$\mathbb{G}$`)`, también mostrada en el Algoritmo 2 convierte una cadena de bits u en un elemento de $\mathbb{G}$. La función `GenerarAleatorio(`$\mathbb{Z}_r$`)` mostrada en el Algoritmo 2 regresa un número aleatorio en $\mathbb{Z}_r$.

Entradas: *PK* = {g_0, g_1, $h = g_0^{\beta}$, $e(g_0, g_1)^{\alpha}$}: Llave pública en CP-ABE, *M*: Mensaje a cifrar, $\mathbb{A}$: Estructura de acceso.
Salida: *CT*: Texto Cifrado en CP-ABE.
1: $M \leftarrow$ `Mapear(`M, $\mathbb{G}_T$`)`
2: $s \leftarrow$ `GenerarAleatorio(`$\mathbb{Z}_r$`)`
3: $C' \leftarrow M \times e(g_0, g_1)^{\alpha s}$
4: $C \leftarrow h^s$
5: $L \leftarrow \{\ \}$
6: `DShamir(`$\mathbb{A}$`.raíz`, *s*, *L*`)`
7: **devolver** $CT \leftarrow \{C, C', L\}$

Algoritmo 2: Función `CP-ABE.Encrypt(`*PK*, *M*, $\mathbb{A}$`)` para cifrado con CP-ABE.

Entradas: *nodo*: Un nodo de la estructura de acceso $\mathbb{A}$, *subs*: Secreto a distribuir en los hijos del nodo de entrada, **L**: Lista asociada al texto cifrado en CP-ABE.
Salida: Modificación de la lista **L** agregando un componente de cifrado cuando *nodo* es una hoja.
1: **si** *nodo* es nodo interno **entonces**
2: $P \leftarrow$ `CrearPolinomio (`*nodo*$.k$, *s*`)`
3: **para** i $\leftarrow$ 1 hasta *nodo*.n **hacer**
4: Calcular([i, $P(i)$])
5: `DShamir(`*nodo*`.hijo[`i`]`, $P(i)$, L`)`
6: **fin para**
7: **si no**
8: $[C_y, C'_y] \leftarrow$ `CifradoParcial(`*nodo*, *subs*`)`
9: `Agregar(`L, $[C_y, C'_y]$`)`
10: **fin si**

Algoritmo 3: Función recursiva `DShamir(`*nodo*, *subs*, L`)` para la distribución de un secreto en una estructura de acceso $\mathbb{A}$ y cifrado en CP-ABE.

En CP-ABE, el esquema de Shamir se usa para distribuir un secreto (valor numérico) asociado a un nodo interno de la estructura de acceso hacia sus nodos hijos. El proceso inicia distribuyendo *s* en los hijos del nodo raíz. El proceso se repite para cada nodo interno, es decir, el subsecreto

asignado al nodo interno se distribuye nuevamente a sus nodos hijos. El proceso de distribución termina cuando se alcanza un nodo hoja, donde el subsecreto se opera matemáticamente y se oculta en una operación de exponenciación en grupos, generando un par $[C_y, C'_y]$. En el Algoritmo 3 se presenta el algoritmo recursivo `DShamir`(*nodo*, *subs*), el cual de forma recursiva distribuye el secreto *s* del nodo raiz a los nodos hoja de la estructura de acceso. Cuando se alcanza un nodo hoja en la estructura de acceso se invoca a la función `CifradoParcial`(*nodo*, *subs*) descrita en el Algoritmo 4, el cual calcula un nuevo par p_y y lo agrega a la lista L. Inicialmente, el esquema de Shamir se invoca como `DShamir`($\mathbb{A}$.*raíz*, *s*, $L = \{\}$). La función `CrearPolinomio`(k, *s*) mostrada en el Algoritmo 3 crea un polinomio de grado $k-1$, término independiente *s* y coeficientes aleatorios. Este polinomio es necesario para el esquema de Shamir (ver Capítulo 2 Sección 2.1.10.2).

Al final del Algoritmo 3, la lista L resultante contiene los pares $[C_y, C'_y]$ para cada atributo y en $\mathbb{A}$, produciendo así el texto cifrado $CT = \{C, C', L\}$.

Entradas: *nodo*: Un nodo hoja de una estructura de acceso $\mathbb{A}$, *subs*: Sub-secreto asignado al nodo.
Salida: p_y: Par $[C_y, C'_y]$ que forman parte del texto cifrado de CP-ABE.

1: $H \leftarrow$ `Mapear`(*nodo*.`atributo`, $\mathbb{G}_1$)
2: $C_y \leftarrow g_0^{val}$
3: $C'_y \leftarrow H^{val}$
4: **devolver** $[C_y, C'_y]$

Algoritmo 4: Función `CifradoParcial`*(nodo, subs)* para generar cada componente de cifrado $[C_y, C'_y]$.

4.1.4 Generación de la llave de descifrado de usuario (`CP-ABE.KeyGen`)

Una llave de descifrado en CP-ABE se define como la tupla $\{D, L\}$, donde $D = g_1^{\frac{\alpha+r}{\beta}}$ y $L = \{q_y\}^+$ es una lista de uno o más pares $q_y = [d_y, d'_y]$, donde $d_y = g_1^r \times H^{r_j}$ y $d'_y = g_0^{r_j}$. Los valores *r* y r_j son números aleatorios en $\mathbb{Z}_r$. El Algoritmo 5 describe el proceso de generación de la llave privada en CP-ABE, asumiendo que se tienen n atributos de usuario. La función `Agregar`*(L,* q_y*)* mostrada en Algoritmo 5 agrega el par q_y a la lista L.

Entradas: $MK = \{\beta, g_1{}^{\alpha}\}$: Llave maestra, A_u: Lista de atributos de usuario.
Salida: $SK = [D, L_2 = \{q_y\}^+]$: Llave de descifrado (Llave secreta).
1: $r \leftarrow$ `GenerarAleatorio`$(\mathbb{Z}_r)$
2: $D \leftarrow g_1^{\frac{\alpha+r}{\beta}}$
3: **para** cada y en A_u **hacer**
4: $r_j \leftarrow$ `GenerarAleatorio`$(\mathbb{Z}_r)$
5: $H \leftarrow$ `MapearAtributo`$(y, \mathbb{G}_2)$
6: $d_y \leftarrow g_1^r \times H^{r_j}$
7: $d'_y \leftarrow g_0^{r_j}$
8: `Agregar`$(L, [d_y, d'_y])$
9: **fin para**
10: **devolver** $SK \leftarrow \{D, L\}$

Algoritmo 5: Función `CP-ABE.KeyGen`(MK, A_u) para generar la llave de descifrado CP-ABE.

4.1.5 Verificación de la política de acceso a partir de la estructura de acceso

Dado un conjunto de atributos de un usuario A_u, se puede validar si dicho conjunto satisface o no una política de acceso usada en el cifrado CP-ABE. Esto se realiza para evitar ejecutar el algoritmo de descifrado en caso de que la política no se cumpla. El Algoritmo 6 describe como realizar este proceso de verificación. Se recorre la estructura de acceso $\mathbb{A}$ de los nodos hoja hacia la raíz, evaluando cada uno de los nodos y asignándoles un valor falso o verdadero. Esta asignación se realiza de la siguiente forma:

1. En el caso de un *nodo* hoja, se le asigna el valor verdadero a dicho nodo si el atributo que éste almacena se encuentra en A_u.

2. En el caso de un *nodo* interno con valores asociados k y n, se le asigna el valor verdadero solo si por lo menos k de sus hijos tienen asignados el valor verdadero.

Cuando se alcanza el nodo raíz, si este resulta tener asignado el valor verdadero, entonces el conjunto de atributos A_u satisface la política de acceso. La Figura 4.4 muestra un ejemplo del proceso de verificación descrito anteriormente (0 representa un valor falso y 1 un valor verdadero). Al realizar el proceso de verificación, dado que con *k-de-n* atributos se puede asignar el valor verdadero

a un nodo padre, los nodos restantes (n-k) se pueden descartar. Al realizar este descarte de nodos se produce una estructura de acceso "podada", la cual se usa para el proceso de descifrado.

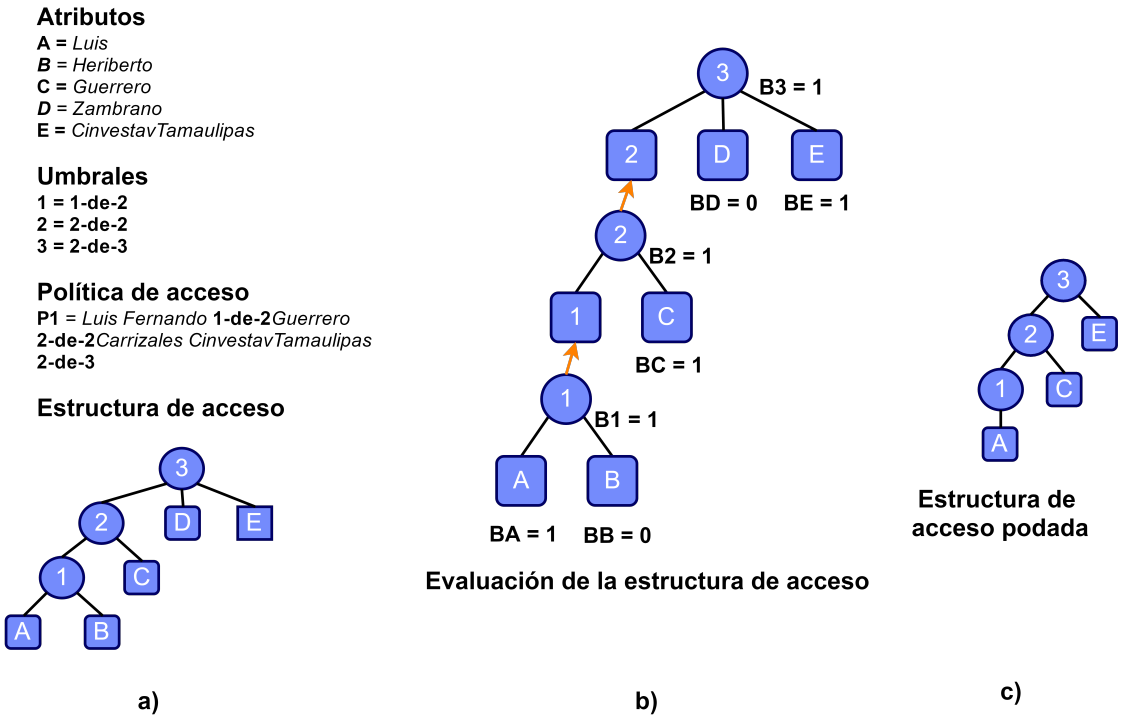


Figura 4.4: Proceso de verificación de atributos en una política, donde **a)** muestra la estructura de acceso original, **b)** indica el proceso de evaluación de la estructura de acceso y **c)** muestra la estructura de acceso "podada".

La verificación de los atributos de usuario en una estructura de acceso $\mathbb{A}$ se realiza con la función `Verificar`($\mathbb{A}$.`raíz`, A_u) descrita en el Algoritmo 6. El procesamiento es *Bottom-Up*, es decir, se procesa de los nodos hoja hacia la raíz. Los nodos a los que el algoritmo asigna un valor falso, o que son innecesarios dado que se ha superado el umbral k, son descartados con la función `Remover`(*nodos*).

Entrada: *nodo*: un nodo (interno u hoja) de la estructura de acceso $\mathbb{A}$, A_u: Lista de atributos de usuario.
Salidas: Verdadero (**1**) si $A_u \in \mathbb{A}$, Falso (**0**) en caso contrario

```
1: si nodo es interno entonces
2:     r ← 0
3:     para i ← 1 hasta nodo.n hacer
4:         r_i ← Verificar(nodo.hijo[i], A_u)
5:         si r_i == 0 entonces
6:             Remover(nodo.hijo[i])
7:         fin si
8:         r ← r + r_i
9:         si r == k entonces
10:            Remover(Todos los hijos restantes)
11:            devolver 1
12:        fin si
13:    fin para
14:    si r < k entonces
15:        devolver 0
16:    fin si
17: si no
18:    si nodo.atributo esta en A_u entonces
19:        devolver 1
20:    si no
21:        devolver 0
22:    fin si
23: fin si
```

Algoritmo 6: Función recursiva `Verificar` *(nodo,* A_u*)* para verificación de política y podado de la estructura de acceso $\mathbb{A}$.

4.1.6 Descifrado (`CP-ABE.Decrypt`)

Dado un texto cifrado $CT = [C, C', L_1 = \{p_y\}^+]$, una llave de descifrado $SK = [D, L_2 = \{q_y\}^+]$, el descifrado de CT para recuperar el texto en claro M consiste en realizar el cálculo que se indica en la Ecuación 4.1:

$$M = \frac{C'}{\left(\frac{e(C, D)}{e(g_0, g_1)^{rs}}\right)} \tag{4.1}$$

El valor $e(g_0, g_1)^{rs}$ se obtiene al realizar el proceso de reconstrucción de secretos de Shamir sobre la estructura de acceso $\mathbb{A}$, procesando cada par p_y y q_y de las listas L_1 y L_2, asociados a cada para el atributo y en A_u.

Así, el proceso de descifrado en CP-ABE es dependiente de los atributos del usuario que desea

acceder a éstos, los cuales pasan por un proceso de validación, a fin de asegurar que dichos atributos satisfacen la política de cifrado y por ende que vale la pena realizar todos los cálculos para recuperar el mensaje *M*.

Sean $X = \{x_1, x_2, ..x_k\}$, $Y = \{y_1, y_2, ..., y_k\}$ los elementos que definen k pares $[x_j, y_j]$ del esquema de Shamir, y que están asociados a un nodo interno en la estructura de acceso podada. Sea `CoeffLagrange`*(j, X)* la función que calcula el coeficiente de Lagrange L_j asociado al j-ésimo par. Entonces, de acuerdo a [5], para cada nodo hoja n_l en la estructura $\mathbb{A}$, se define el *camino* de n_l como la secuencia de nodos $path_{n_l} = \{n_1, n_2, ..., n_i\}$ que se recorren desde el nodo raíz hasta llegar a n_l. Sea la función `Pos(`n_i`)` la función que regresa la posición del nodo n_i respecto a sus nodos hermanos. Para cada $path_{n_l}$ en $\mathbb{A}$, se calcula un exponente, multiplicando todos los valores `CoeffLagrange`*(j, X)* de cada conjunto *X* de cada nodo en $path_{n_l}$.

Así, para cada nodo hoja n_l en A con atributo y, se obtiene un exponente exp_{n_l}. Con este valor, se realiza la operación: $emp_{n_l} = \dfrac{e(d'_y, C'_y)^{exp_{n_l}}}{e(d_y, C_y)}$, donde $\{d_y, d'_y\}$ y $\{C_y, C'_y\}$ son valores asociados a y obtenidos de *SK* y *CT* respectivamente. Cuando todos los nodos hoja en el árbol podado se han alcanzado y procesado, se calcula el producto acumulativo $\prod emp_{n_l}$, lo cual produce como resultado $e(g, g)^{rs}$. El Algoritmo 7 describe el proceso de descifrado en CP-ABE a través de la función `RShamir()`, la cual recorre recursivamente la estructura de acceso del árbol podado por el Algoritmo 6. Inicialmente, este algoritmo se invoca como `RShamir(`$\mathbb{A}$.raíz, *exp*, *CT*, *SK*, *acc*`)`, donde *exp = 1* y *acc = 1*. En cada llamada, el exponente se actualiza al procesar un nuevo nodo n_i de $path_{n_l}$. Si el *nodo* actual es un nodo interno, entonces se calcula su conjunto *X*, se obtiene $j =$ `pos`*(nodo)* y se calcula *newExp* = `CoeffLagrange`*(j, X)*exp*. Entonces, para cada nodo hijo n_i de *nodo*, se realiza la llamada recursiva `RShamir`*(*n_i*, newExp, CT, SK, acc)*. Cuando se alcanza un nodo hoja n_l, se calcula emp_{n_l} usando exp_{n_l} = *newExp*. Entonces, se usa la función `Acumular` para calcular el producto acumulativo $\prod emp_{n_l}$ en la variable *acc*, la cual inicialmente es 1.

En esta implementación de CP-ABE, el proceso de reconstrucción del secreto se realiza de las hojas a la raíz, ya que un proceso recursivo favorece el cálculo de los parámetros a través de la

estructura de acceso, accediendo y calculando de forma acumulativa el componente $e(g_0, g_1)^{rs}$ de la Ecuación 4.1, el cual tiene el parámetro *s* inmerso en él (mensaje distribuido en los nodos hoja).

Entradas: *nodo*: Un nodo de la estructura de acceso, *exp*: Exponente acumulativo, *CT* = $= [C, C', L_1 = \{p_y\}^+]$: Texto cifrado, *SK* $= [D, L_2 = \{q_y\}^+]$: Una llave privada de usuario para el descifrado en CP-ABE, *acc*: Variable acumulativa de componente de descifrado.
Salida: $e(g_0, g_1)^{rs}$

si *nodo* es nodo interno **entonces**
  j = `Pos`*(nodo)*
  X = `Calcular`X_j*(nodo)*
  newExp = *exp* * `CoefLagrange`*(j, X)*
  para $i \leftarrow 1$ hasta *nodo*.k **hacer**
    `RShamir`(*nodo*.hijo[i], *newExp*, *CT*, *SK*, *acc*)
  fin para
si no
  y = *nodo.atributo*
  $[d_y, d'_y] \leftarrow$ `ObtenerPar`*(SK,y)*
  $[C_y, C'_y] \leftarrow$ `ObtenerPar`*(CT,y)*
  $emp = \dfrac{e(d'_y, C'_y)^{exp}}{e(d_y, C_y)}$
  `Acumular`*(acc, emp)*
fin si
devolver {acc = $e(g_0, g_1)^{rs}$}

Algoritmo 7: Función recursiva `RShamir`*(nodo, exp, CT, SK, acc)* para calcular $e(g_0, g_1)^{rs}$.

El proceso de descifrado con CP-ABE se lleva a cabo como se muestra en el Algoritmo 8.

Entradas: $\mathbb{A}$: Estructura de acceso, A_u: Lista de atributos de usuario, *SK* $= [D, L_2 = \{q_y\}^+]$: Llave de descifrado en CP-ABE, *CT* $= [C, C', L_1 = \{p_y\}^+]$: Texto Cifrado en CP-ABE.
Salida: M: Texto plano.

si `Verificar(`$\mathbb{A}$`.raíz,` A_u`) == 0` **entonces**
  Escribir "Error de atributos"
  devolver NULL
si no
  $u \leftarrow$ `RShamir`*(*$\mathbb{A}$*.raíz, CT, SK)*
  $M \leftarrow \frac{u \times C'}{e(C, D)}$
fin si
devolver M

Algoritmo 8: Función `CP-ABE.Decrypt`($\mathbb{A}$, A_u, *SK*, *CT*) para descifrado con CP-ABE.

4.2 Implementación del cifrador AES

En esta sección se describen los algoritmos que son utilizados para la etapa de cifrado y descifrado en AES.

4.2.1 Cifrado (AES.Encrypt)

El Algoritmo 9 muestra la forma en la cual el cifrador AES realiza el cifrado de un conjunto de datos en claro, proporcionando el servicio de confidencialidad. En este algoritmo se genera un conjunto de llaves de ronda a través de la función GenerarLlavesdeRonda, la cual recibe la llave de cifrado AES (de tamaño 128, 192 o 256 bits), dichas llaves de ronda son utilizadas en cada iteración dentro del proceso de cifrado mezclando éstas con el texto cifrado dentro de la tarea *AddRoundKey*. En este algoritmo, el texto en claro T se procesa en bloques de 16 bytes, los cuales se obtienen a partir de la función DividirBloques. Cada bloque se cifra con el Algoritmo 10 (función AES.Cifrar). Por último, la función AgregaraTextoCifrado concatena los bloques cifrados, generando así el texto cifrado una vez procesados todos los bloques.

Entradas: T: Texto en claro, *VI*: Vector de inciialización, AES_{KEY}: Llave de cifrado.
Salida: *CT* Texto cifrado.
1: *Llaves** ← GenerarLlavesdeRonda*(*AES_{KEY} *)*
2: $C_0 \leftarrow$ *VI*
3: $B_1, B_2, ...B_n$ ← DividirBloques*(T)*
4: **para** $i \leftarrow 1$ hasta n **hacer**
5: AddVI(B_i, C_{i-1})
6: C_i ← AES.Cifrar*(*B_i, *Llaves*)*
7: AgregaraTextoCifrado*(CT,* C_i*)*
8: **fin para**
9: **devolver** *CT*

Algoritmo 9: Función AES.Encrypt(T, *VI*, AES_{KEY}) para cifrado de datos.

En el Algoritmo 9 se toman como entrada el texto plano (documento a cifrar), la llave de cifrado y el vector de inicialización (el cual puede ser generado de forma aleatoria como un bloque de 16 bytes). La llave es utilizada para el cifrado de los datos, mientras que el vector de inicialización *VI* es utilizado para aplicar el modo **CBC** sobre el cifrado, es decir, evitar que dos bloques de datos iguales generen la misma salida cifrada. El Algoritmo 10 utiliza las funciones SubBytes, ShiftRows, MixColumns y AddRoundKey para cifrar un solo bloque de datos de 16 bytes, aportando así el servicio de confidencialidad.

Las funciones base del cifrador AES SubBytes, MixColumns y AddRoundKey son iterativas, las

Entradas: *state*: Bloque de datos de entrada, *Llaves**: Llaves de ronda.
Salida: *C* Bloque de datos cifrados.
1: `AddRoundKey(`*state*, *Llaves[1])*
2: **para** $i \leftarrow 2$ hasta 10 **hacer**
3: `SubBytes(`*state*`)`
4: `ShiftRows(`*state*`)`
5: `MixColumns(`*state*`)`
6: `AddRoundKey(`*state*, $Llaves[i]$`)`
7: **fin para**
8: `SubBytes(`*state*`)`
9: `ShiftRows(`*state*`)`
10: `AddRoundKey(`*state*, $Llaves[11]$`)`
11: $C \leftarrow$ *state*
12: **devolver** *C*

Algoritmo 10: Función `AES.Cifrar(`*B*, *Llaves**`)` para cifrar un bloque de datos usando AES.

cuales procesan cada byte del bloque de datos de entrada. El Algoritmo 11 muestra la estructura de la función `SubBytes`, donde se utiliza un ciclo anidado para sustituir cada byte del bloque de datos de entrada por un componente del vector S_{Box}.

Entradas: B: Bloque de datos de entrada, S_{Box}: Vector de sustitución.
Salida: *C* Bloque de datos de salida.
1: **para** i $\leftarrow$ 1 hasta 4 **hacer**
2: **para** j $\leftarrow$ 1 hasta 4 **hacer**
3: $C[i * 4 + j] \leftarrow S_{Box}[B[i * 4 + j]]$
4: **fin para**
5: **fin para**
6: **devolver** *C*

Algoritmo 11: Función `SubBytes(`B, S_{Box}`)` secuencial.

El proceso de mezclado de la llave de ronda tiene un procesamiento similar al de sustitución, siendo que se realiza un recorrido del bloque a cifrar con un ciclo anidado. La operación que permite mezclar la llave de ronda y el bloque a cifrar es la operación lógica XOR. La función `AddRoundKey` se muestra en el Algoritmo 12.

Entradas: B: Bloque de datos de entrada, *Llaves**: Llaves de ronda.
Salida: *C* Bloque de datos de salida.
1: **para** i $\leftarrow$ 1 hasta 4 **hacer**
2: **para** j $\leftarrow$ 1 hasta 4 **hacer**
3: $C[i*4+j] \leftarrow B[i*4+j]$ XOR *llaves*$[i*4+j]$
4: **fin para**
5: **fin para**
6: **devolver** *C*

Algoritmo 12: Función `AddRoundKey`(B, *Llaves**) secuencial.

La función `ShiftRows` produce un corrimiento para cada fila del bloque de datos de entrada. La posición de cada dato se define en un vector llamado *corr*. El Algoritmo 13 muestra el proceso que se realiza en esta función.

Entradas: B: Bloque de datos de entrada, *pos*: Vector de posiciones.
Salida: *C* Bloque de datos de salida.
1: **para** i $\leftarrow$ 1 hasta 4 **hacer**
2: **para** j $\leftarrow$ 1 hasta 4 **hacer**
3: $C[i*4+j] \leftarrow pos[B[i*4+j]]$
4: **fin para**
5: **fin para**
6: **devolver** *C*

Algoritmo 13: Función `ShiftRows`(B, *pos*) secuencial.

La función `MixColumns` mostrada en el Algoritmo 14 toma los valores en cada columna del bloque de datos de entrada y los usa como coeficientes de un polinomio de grado 3. Este polinomio se opera con otro predeterminado y se obtiene la reducción modulo con un polinomio de grado 4, produciendo un polinomio de máximo grado 3. Los coeficientes del polinomio resultante se sustituyen como los nuevos valores en la columna del bloque de datos que se está operando. En este algoritmo `CalcPol` es la función que se encarga de realizar la operación polinomial correspondiente.

Entradas: B: Bloque de datos de entrada.
Salida: *C* Bloque de datos de salida.
1: **para** i $\leftarrow$ 1 hasta 4 **hacer**
2: $C[i*4+1] \leftarrow$ `CalcPol`$(B[i*4+1])$
3: $C[i*4+2] \leftarrow$ `CalcPol`$(B[i*4+2])$
4: $C[i*4+3] \leftarrow$ `CalcPol`$(B[i*4+3])$
5: $C[i*4+4] \leftarrow$ `CalcPol`$(B[i*4+4])$
6: **fin para**
7: **devolver** *C*

Algoritmo 14: Función `MixColumns`(B) secuencial.

4.2.2 Descifrado (`AES.Decrypt`)

El proceso de descifrado de los datos en AES es muy similar al proceso de cifrado. El Algoritmo 15 muestra el proceso de descifrado de los datos a través de la función `AES.Decrypt`, misma que usa la función `AES.Descifrar` para el descifrado de un solo bloque como se describe en el Algoritmo 16. En este algoritmo se realizan las operaciones inversas de los procesos *SubBytes*, *ShiftRows*, *MixColumns* y *AddRoundKey*. La función `AgregaraTextoDescifrado` concatena los bloques de texto descifrado para producir el texto en claro original.

Entradas: *CT* Texto cifrado AES, *VI*: Vector de incialización, AES_{KEY}: Llave de descifrado AES.
Salida: **T**: Texto en claro.
1: *Llaves** ← `GenerarLlavesderonda`*(AES$_{KEY}$)*
2: C_0 ← *VI*
3: $C_1, C_2, ...C_n$ ← `DividirBloques`*(CT)*
4: **para** $i \leftarrow 1$ hasta n **hacer**
5: T_i ← `AES.Descifrar`*(C_i, Llaves*)*
6: `AddVI`*(T_i, C_{i-1})*
7: `AgregaraTextoDescifrado`*(T, T_i)*
8: **fin para**
9: **devolver** *T*

Algoritmo 15: Función `AES.Decrypt`(*CT*, *VI*, AES_{KEY}) para descifrado de datos.

Entradas: *C*: Bloque de datos de entrada, *Llaves**: Llaves de ronda.
Salida: B Bloque de datos de salida.
1: `InvAddRoundKey`*(C, Llaves[11])*
2: **para** $i \leftarrow 10$ hasta 2 **hacer**
3: `InvShiftRows`(C)
4: `InvSubBytes`(C)
5: `InvAddRoundKey`(C, *Llaves[i]*)
6: `InvMixColumns`(C)
7: **fin para**
8: `InvShiftRows`(C)
9: `InvSubBytes`(C)
10: `MezclarLlavederonda`(C, *Llaves[1]*)
11: *B* ← *C*
12: **devolver** *B*

Algoritmo 16: Función `AES.Descifrar`(*C*, *Llaves**) para descifrar un bloque de datos usando AES.

Las funciones base del cifrador AES en modo de descifrado procesan cada byte en el bloque cifrado (de igual manera que en la etapa de cifrado). El Algoritmo 17 muestra la estructura de la función `InvSubBytes`, donde se utiliza un ciclo anidado para realizar un recorrido del bloque de

datos, sustituyendo cada byte por un componente del vector S_{BoxInv}. Este vector posee los caracteres que se tenían originalmente en cada sustitución a la que fue sometido el byte.

Entradas: C: Bloque de datos de entrada, S_{BoxInv}: Vector de sustitución.
Salida: B Bloque de datos de salida.
1: **para** i $\leftarrow$ 1 hasta 4 **hacer**
2: **para** j $\leftarrow$ 1 hasta 4 **hacer**
3: $B[i*4+j] \leftarrow S_{BoxInv}[C[i*4+j]]$
4: **fin para**
5: **fin para**
6: **devolver** B

Algoritmo 17: Función `InvSubBytes`($\underline{C}$, S_{BoxInv}) secuencial.

El proceso de mezclado de la llave de ronda con el bloque que se está descifrando se realiza con la función `InvSubBytes`, como se muestra en el Algoritmo 18.

Entradas: C: Bloque de datos de entrada, *Llaves**: Llaves de ronda.
Salida: B Bloque de datos de salida.
1: **para** i $\leftarrow$ 1 hasta 4 **hacer**
2: **para** j $\leftarrow$ 1 hasta 4 **hacer**
3: $B[i*4+j] \leftarrow C[i*4+j]$ XOR *llaves*$[i*4+j]$
4: **fin para**
5: **fin para**
6: **devolver** B

Algoritmo 18: Función `InvAddRoundKey`(C, *Llaves**) secuencial.

La función `InvShiftRows` descrita en el Algoritmo 19 produce un corrimiento para cada fila de la matriz. La posición de cada dato se define en un vector llamado *corrInv*, y corresponde a la posición original de cada byte en el bloque.

Entradas: C: Bloque de datos de entrada, *posInv*: Vector de posiciones.
Salida: B Bloque de datos de salida.
1: **para** i $\leftarrow$ 1 hasta 4 **hacer**
2: **para** j $\leftarrow$ 1 hasta 4 **hacer**
3: $B[i*4+j] \leftarrow$ *posInv*$[C[i*4+j]]$
4: **fin para**
5: **fin para**
6: **devolver** B

Algoritmo 19: Función `InvShiftRows`(C, *corrInv*) secuencial.

La función `InvMixColumns` realiza un procesamiento de los datos diferente, donde el recorrido que se realiza es a través de un solo ciclo. En esta operación se calcula un coeficiente, el cual corresponde

al byte original que fue evaluado en el polinomio para el cifrado. Esta operación es dependiente de cada resultado obtenido, de tal modo que no puede ser ejecutado cada byte en paralelo. El Algoritmo 20 muestra el proceso que se realiza en la función. En este algoritmo `CalcCoef` es la función que se encarga de calcular el coeficiente correspondiente.

Entradas: C: Bloque de datos de entrada.
Salida: B Bloque de datos de salida.
1: **para** i $\leftarrow$ 1 hasta 4 **hacer**
2: $B[i*4+1] \leftarrow$ `CalcCoef`$(C[i*4+1])$
3: $B[i*4+2] \leftarrow$ `CalcCoef`$(C[i*4+2])$
4: $B[i*4+3] \leftarrow$ `CalcCoef`$(C[i*4+3])$
5: $B[i*4+4] \leftarrow$ `CalcCoef`$(C[i*4+4])$
6: **fin para**
7: **devolver** B

Algoritmo 20: Función `InvMixColumns`(C) secuencial.

4.3 Implementación secuencial del esquema DET-ABE

En esta sección se describen los algoritmos que conforman al esquema DET-ABE, los cuales son dependientes de las implementaciones de los algoritmos de AES y CP-ABE. En el esquema DET-ABE se considera que el algoritmo de generación de las llaves de descifrado es idéntica al de CP-ABE, de tal modo que en este apartado se descarta esta función del esquema.

4.3.1 Inicialización de parámetros (`DET-ABE.Setup`)

La inicialización de DET-ABE consiste en invocar a la función `Setup` de CP-ABE, que toma las especificaciones de nivel de seguridad y el tipo de curva elíptica, con los cuales se definen los grupos $\mathbb{G}_1$, $\mathbb{G}_2$, $\mathbb{G}_T$ y el tipo de emparejamiento. Una vez establecidos estos parámetros, se generan las llaves (PK y MK) de CP-ABE, las cuales son necesarias para cifrar los datos y para generar llaves de descifrado.

La función `DET-ABE.Setup` es ejecutada por una autoridad de confianza, la cual mantiene secreta la llave MK y es quien genera las llaves de descifrado. El nivel de seguridad y tipo de curva deben

ser tales que sean equivalentes a los que provee AES, es decir, se deben utilizar curvas elípticas con tamaños equivalentes para proporcionar niveles de seguridad de 128, 192 y 256 bits.

4.3.2 Cifrado (DET-ABE.Encrypt)

Como se mencionó al inicio de este capítulo, el esquema DET-ABE está conformado por el cifrador AES y el cifrador CP-ABE bajo la realización del concepto de sobre digital, de tal modo que una vez generada la llave AES, CP-ABE la cifra con una política, garantizando el control de acceso a los datos. El esquema DET-ABE provee a los datos de confidencialidad y control de acceso utilizando los algoritmos que conforman a los cifradores AES y CP-ABE, razón por la cual éstas funciones son invocadas con los parámetros de entrada establecidos en la Sección 4.1.

El Algoritmo 21 muestra la forma en que el esquema DET-ABE realiza el cifrado de datos.

Entradas: T: Texto a cifrar, *NS*: Nivel de seguridad, P: política de acceso en notación posfija.
Salida: *SD*: Sobre digital compuesto por CT_1 (Cifrado AES), CT_2 (Cifrado CP-ABE), *NS* (Nivel de seguridad), *CE* (Tipo de curva elíptica), $\mathbb{A}$ (Estructura de acceso), *VI* (Vector inicializador).

1: $AES_{KEY} \leftarrow$ `GenerarLlaveAES`*(NS)*
2: *VI* $\leftarrow$ `GenerarVI`*()*
3: *PK* $\leftarrow$ `CP-ABE.Setup`*(NS)*
4: $\mathbb{A} \leftarrow$ `ConstruirEstrucura`*(P)*
5: $CT_1 \leftarrow$ `AES.Encrypt`*(T, VI, AES*$_{KEY}$*)*
6: $CT_2 \leftarrow$ `CP-ABE.Encrypt`(*PK*, *AES*$_{KEY}$, $\mathbb{A}$)
7: *SD* $\leftarrow CT_1 \,||\, CT_2 \,||$ *NS* $||\, \mathbb{A} \,||$ *VI*
8: **devolver** *SD*

Algoritmo 21: Función `DET-ABE.Encrypt`(*T*, *NS*, *P*) para cifrado con DET-ABE.

4.3.3 Descifrado (DET-ABE.Decrypt)

En DET-ABE, el descifrado de los datos se lleva a cabo por el Algoritmo 22, el cual recibe el sobre digital y como salida se obtiene el texto plano. `DET-ABE.Decrypt` se lleva a cabo a través de cuatro etapas, la primera es verificar si el usuario con el conjunto de atributos A_u satisface la política de acceso. La segunda etapa comprende la generación de la llave de descifrado CP-ABE (*SK*) usando los atributos A_u. La tercera etapa consiste en descifrar la llave AES contenida en el

sobre digital usando *SK*, y finalmente la cuarta etapa consiste en usar la llave AES ya descifrada para descifrar los datos. La llave de descifrado *SK* es generada por la autoridad de confianza con el Algoritmo 5 (`CP-ABE.KeyGen`). Una vez que la autoridad de confianza produce la llave *SK*, el proceso de descifrado de la llave AES comienza al ejecutar el Algoritmo 8 (`CP-ABE.Decrypt`). Al recuperar la llave de descifrado de los datos AES_{KEY}, se ejecuta el Algoritmo 15 (`AES.Decrypt`) para obtener el texto plano.

Entradas: *SD*: Sobre digital compuesto por CT_1 (Cifrado AES), CT_2 (Cifrado CP-ABE), *NS* (Nivel de seguridad), *CE* (Tipo de curva elíptica), $\mathbb{A}$ (Estructura de acceso), *VI* (Vector inicializador), A_u: Lista de atributos de usuario.
Salida: T: Texto en claro.

```
1: si Verificar(𝔸.raíz, A_u) == 0 entonces
2:     Escribir "Error de atributos"
3:     devolver NULL
4: si no
5:     SK ← CP-ABE.KeyGen(MK, A_u)
6:     AES_KEY ← CP-ABE.Decrypt(𝔸.raíz, SK, SD.CT_2)
7:     T ← AESDescifrar( SD.CT_1, AES_KEY, SD.VI)
8: fin si
9: devolver T
```

Algoritmo 22: Función `DET-ABE.Decrypt`(*SD*, A_u) para descifrado con DET-ABE.

4.4 Resumen

En este capítulo se describieron los principales algoritmos que permiten la construcción de los cifradores AES y CP-ABE, mismos que se requieren para implementar las operaciones de cifrado, descifrado y generación de llaves del esquema DET-ABE. El conjunto de algoritmos descritos en este capítulo permitió comprender su funcionamiento a detalle, de tal forma que se identificaron sus puntos paralelizables. Esto sirvió como entrada para la implementación de diversas estrategias de paralelización, de las cuales se describen las mejores implementaciones en el Capítulo 5.

5

Paralelización del esquema de seguridad DET-ABE

El proceso de asegurar información que se transfiere a la nube con DET-ABE puede resultar ser costoso aún utilizando niveles de seguridad bajos y políticas de acceso con pocos atributos. Por un lado, el tiempo de cifrado con AES crece rápidamente cuando el tamaño de datos aumenta; de igual forma, el tiempo de procesamiento de CP-ABE se incrementa al usar niveles de seguridad más altos y políticas de control de acceso más elaboradas. Debido a esto, el cifrado y descifrado de los datos impone una carga de trabajo adicional a la simple transferencia en claro de los mismos.

Una alternativa para compensar el trabajo adicional debido a las funciones necesarias del esquema DET-ABE es el uso de paralelismo para la ejecución de sus algoritmos a diferentes niveles. Si bien las arquitecturas paralelas han existido desde hace varias décadas, desde hace relativamente pocos años las computadoras paralelas están disponibles a bajo costo y para su uso en aplicaciones específicas. Por un lado, la disponibilidad de arquitecturas multicore con cada vez más núcleos de procesamiento ha permitido tener dispositivos de cómputo más eficientes y rápidos (servidores, portátiles, de escritorio, tabletas e incluso teléfonos inteligentes), los cuales son capaces de ejecutar

varias aplicaciones en forma simultánea. La presencia de múltiples núcleos de procesamiento permite el desarrollo de aplicaciones que aprovechen tales recursos mediante soluciones multihilo.

Por otra lado, la demanda de mayores capacidades de visualización y procesamiento gráfico en los dispositivos de cómputo portátiles y de escritorio ha hecho posible tener acceso a aceleradores de procesamiento tipo GPU a bajo costo. Estos dispositivos, además de utilizarse para funciones de procesamiento gráfico, pueden ser usados para acelerar algunas funciones de procesamiento de las aplicaciones en el procesador central.

Algunos dispositivos aceleradores GPU que pueden ser utilizados para paralelizar procesos en un dispositivo de cómputo personal son NVIDIA, ATI, FXF, etc. Estos dispositivos permiten distribuir procesos entre el o los CPUs disponibles y el dispositivo GPU, conformando así un procesamiento híbrido.

Ambas características, procesadores multicore y aceleradores de procesamiento basados en GPUs, son la base para la construcción de las computadoras más rápidas en la actualidad a un costo accesible. Dado que en la actualidad es cada vez más frecuente encontrar tales prestaciones en computadoras convencionales, se propone explotar estas características para la paralelización de los procesos de DET-ABE.

De este modo, aprovechando las características híbridas de los dispositivos de cómputo descritos, se propuso dividir el trabajo de la siguiente forma:

- El uso de varios hilos puede ser explotado para reducir el tiempo de cifrado con CP-ABE, dado que el procesamiento de las hojas (atributos) en la estructura de acceso puede realizarse de manera independiente.

- El uso de aceleradores basados en GPUs puede ser explotado en su máxima capacidad al paralelizar los algoritmos de cifrado y descifrado de AES. Esto se debe al alto grado de paralelización del cifrador, ya que pueden ser dedicados tantos hilos ejecutados en paralelo como datos se procesan.

Las operaciones de cifrado y descifrado de datos con algoritmos simétricos como AES requieren de funciones que se repiten en distintos bloques de datos. Esta característica hace que este procesamiento sea particularmente adecuado para el uso de acelerados tipo GPUs. Sin embargo, el modo de operación original del cifrador AES para el esquema DET-ABE es inheremente secuencial pues el cifrado de un bloque se combina con el cifrado del bloque anterior (modo CBC). Por ello, para el caso del cifrado es necesario hacer modificaciones al modo de operación original del cifrador con el fin de explotar las capacidades de procesamiento paralelo con los aceleradores GPU.

En este capítulo se describen las implementaciones paralelas de los algoritmos que conforman el esquema DET-ABE híbrido utilizando dispositivos con capacidades multinúcleo y un dispositivo acelerador GPU. Así mismo, se detallan las implementaciones paralelas de los algoritmos de cifrado, generación de llaves y descifrado en CP-ABE (utilizando una estrategia multicore) y AES (utilizando GPUs). También se detallan las estrategias de paralelización utilizadas sobre las operaciones más básicas de AES (sustituciones, permutaciones y transformaciones lineales). La implementación de DET-ABE bajo un enfoque de cómputo híbrido consiste en tomar como base la implementación secuencial y reemplazar los módulos AES y CP-ABE por sus versiones paralelas. Además se explora una estrategia de paralelización a nivel de esquema de procesamiento, donde AES y CP-ABE paralelos son ejecutados de forma paralela (`AES.Encrypt` paralelo y `CP-ABE.Encrypt` paralelo se ejecutan al mismo tiempo), con el propósito de aprovechar al máximo los recursos disponibles: núcleos de CPU y de GPU.

En la Tabla 5.1 se presenta un resumen de los algoritmos de DET-ABE que permiten ser paralelizados y la estrategia seguida para ello. Estas estrategias de paralelización serán mostradas a lo largo de este capítulo según su capa de procesamiento, la cual será descrita según la Tabla 5.1 del nivel más bajo al más alto.

Tabla 5.1: Aceleración de DET-ABE a diferentes capas.

DET-ABE				
Capa	**Cifrador**	**Algoritmo**	**Paralelizado**	**Estrategia**
Alta	AES y CP-ABE	DET-ABE.Encrypt	Si	Multi-hilo
		DET-ABE.Decrypt	No	- - - - - - - - -
Media	AES	AES.KeyGen	No	- - - - - - - - -
		AES.Encrypt	Si	GPUs
		AES.Decrypt	Si	GPUs
	CP-ABE	CP-ABE.Setup	No	- - - - - - - - -
		CP-ABE.KeyGen	Si	Multi-hilo
		CP-ABE.Encrypt	Si	Multi-hilo
		CP-ABE.Decrypt	Si	Multi-hilo
Baja	AES	Operaciones básicas	Si	GPUs
	CP-ABE		No	- - - - - - - - -

5.1 Algoritmos paralelos para el cifrador AES

5.1.1 Algoritmos de capa baja

Debido a que el procesamiento de AES es altamente compatible con las características de procesamiento con aceleradores basados en GPU, se construyeron versiones paralelas adecuadas para GPUs de las tareas SubBytes, AddRoundKey, ShiftRows y MixColumns que fundamentalmente procesan en paralelo los 16 bytes del bloque de datos como se muestra en la Figura 5.1. Este nivel de paralelismo se considera bajo, ya que en esta estrategia se realizan a lo más 16 operaciones de forma simultánea.

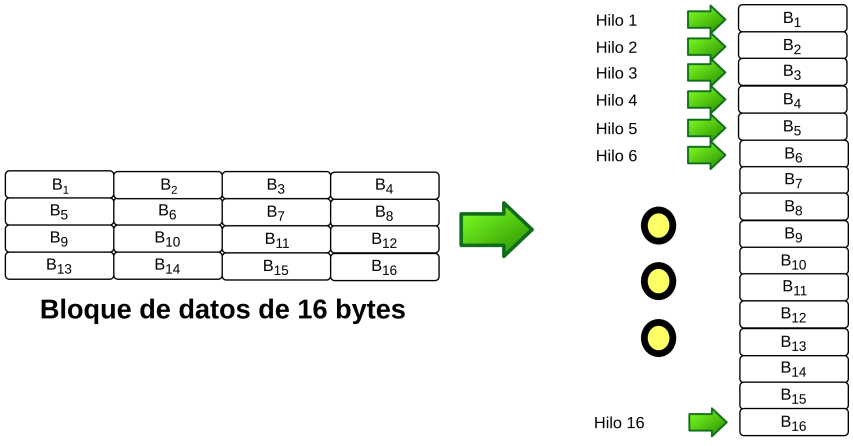


Figura 5.1: División de los bytes de un bloque en 16 hilos en CUDA para su cálculo independiente.

Para la implementación de las estrategias de paralelización en la capa baja, se debe considerar que un kernel es una función global que se ejecuta en los aceleradores basados en GPU, en estos se tiene los siguientes parámetros, los cuales cumplen con las siguientes características de invocación:

```
<<<Blocks, Threads, SharedMem>>>FunctionName(Params)
```

El parámetro `Blocks` define la cantidad de bloques de procesamiento que se desea ejecutar en la operación, `Threads` es la cantidad de hilos que serán ejecutados por cada uno de los bloques y por último `SharedMem` define el tamaño de memoria compartida a asignar por bloque (este parámetro es opcional).

5.1.1.1. Paralelización del proceso de cifrado (un bloque)

Los Algoritmos 23, 24, 25 y 26 muestran el pseudocódigo del núcleo de procesamiento para CUDA de las funciones SubBytes, AddRoundKey, ShiftRows y MixColumns, respectivamente. La invocación de cada uno de los kernels de CUDA se hace como sigue:

- $<<<$ 1, 16 $>>>SubBytesPar(C, B)$
- $<<<$ 1, 16 $>>>AddRoundKeyPar(C, B)$
- $<<<$ 1, 16 $>>>ShiftRowsPar(C, B)$
- $<<<$ 1, 4 $>>>MixColumnsPar(C, B)$

Entrada: B: Bloque de datos de entrada, S_{Box}: Vector de sustitución.
Salida: C Bloque de datos de salida.
Salida: B Bloque de datos de salida.
1: $C[hilo] \leftarrow S_{Box}[B[hilo]]$
2: **devolver** $C[hilo]$

Algoritmo 23: Kernel *SubBytesPar* en CUDA.

Entrada: *B*: Bloque de datos de entrada, *Llaves**: Llaves de ronda.
Salida: *C* Bloque de datos de salida.
1: $C[hilo] \leftarrow B[hilo]$ XOR $llaves[hilo]$
2: **devolver** $C[hilo]$

Algoritmo 24: Kernel *AddRoundKeyPar* en CUDA.

Entrada: *B*: Bloque de datos de entrada, *pos*: Vector de posiciones.
Salida: *C* Bloque de datos de salida.
1: $C[hilo] \leftarrow pos[B[hilo]]$
2: **devolver** $C[hilo]$

Algoritmo 25: Kernel *ShiftRowsPar* en CUDA.

Entrada: *B*: Bloque de datos de entrada.
Salida: *C* Bloque de datos de salida.
1: $C[hilo * 4] \leftarrow$ `CalcPol`$(B[hilo * 4])$
2: $C[hilo * 4 + 1] \leftarrow$ `CalcPol`$(B[hilo * 4 + 1])$
3: $C[hilo * 4 + 2] \leftarrow$ `CalcPol`$(B[hilo * 4 + 2])$
4: $C[hilo * 4 + 3] \leftarrow$ `CalcPol`$(B[hilo * 4 + 3])$
5: **devolver** $C[hilo]$

Algoritmo 26: Kernel *MixColumnsPar* en CUDA.

En este kernel, se asignan tantos hilos como bytes se desean ejecutar en paralelo, generando así en paralelo los bytes cifrados y asignándolos a su respectiva posición en el bloque de salida. De esta forma se satisface la estrategia mostrada en la Figura 5.1, donde para el caso específico de `SubBytes`, el byte que se procesa pasa por la función S_{Box}.

En estos tres primeros kernel se puede ver que la invocación se realiza con un bloque de ejecución y 16 hilos, uno por cada dato. Por otra parte, el kernel de la función `MixColumns` es invoca solamente con 4 hilos, debido a que esta función realiza el cálculo de un polinomio con los valores de los cuatro bytes en cada fila del bloque de datos y para esto existe una dependencia de cada byte calculado.

Sin embargo, ya que la invocación de un kernel resulta costosa, se consideró que los cuatro kernel (`SubBytesPar`, `AddRoundKeyPar`, `ShiftRowsPar` y `MixColumnsPar`) deberían ser ejecutados como operaciones dentro del kernel de cifrado y no como un conjunto de kernels que deben ser llamados en múltiples ocasiones para el cifrado de un solo bloque de datos.

El kernel construido para el cifrado de un bloque de datos de 16 bytes se muestra en el Algoritmo 27, el cual tiene la estructura de su versión secuencial (Algoritmo 9) pero se procesa cada byte del bloque en paralelo y ejecuta las 11 rondas requeridas en el algoritmo de cifrado.

Las 11 rondas de procesamiento de AES se realizan sobre el bloque de datos utilizando 16 hilos, un hilo por byte en el bloque de datos. Es importante notar que dado que en CUDA los hilos se despachan en bloques de 32 (denominados warps) y todos los hilos de un warp funcionan de manera síncrona, no se requieren operaciones de sincronización entre las instrucciones del Algoritmo 27. No obstante, la forma de invocación de este kernel hace que se desperdicie la otra mitad de 16 hilos que potencialmente podrían ser despachado en el mismo bloque de hilos (warp).

Entradas: B: Bloque en claro, *VI*: Vector de inicialización, *Llaves**: Llaves de ronda, S_{Box}: Vector de sustitucion, *pos*: Vector de posiciones.
Salida: **C** Bloque cifrado.

1: $B[hilo] \leftarrow B[hilo]$ `XOR` *llaves*[1]
2: **para** i $\leftarrow$ 2 hasta 10 **hacer**
3: $B[hilo] \leftarrow S_{Box}[hilo]$
4: $B[hilo] \leftarrow B[pos[hilo]]$
5: **si** *hilo* $<$ 4 **entonces** {Dado que se procesa un bloque de 16 bytes en paralelo, los primeros 4 hilos realizan la transformación MixColumns.}
6: $B[hilo*4] \leftarrow$ `CalcPol`$(B[hilo*4])$ {Únicamente se trabaja con hilos}
7: $B[hilo*4+1] \leftarrow$ `CalcPol`$(B[hilo*4+1])$
8: $B[hilo*4+2] \leftarrow$ `CalcPol`$(B[hilo*4+2])$
9: $B[hilo*4+3] \leftarrow$ `CalcPol`$(B[hilo*4+3])$
10: **fin si**
11: $B[hilo] \leftarrow B[hilo]$ `XOR` *llaves*[i]
12: **fin para**
13: $B[hilo] \leftarrow S_{Box}[hilo]$
14: $B[hilo] \leftarrow B[pos[hilo]]$
15: $B[hilo] \leftarrow B[hilo]$ `XOR` *llaves*[11]
16: $C[hilo] \leftarrow B[hilo]$
17: $C[hilo] \leftarrow B[hilo]$ `XOR` *VI*[*hilo*]{Se agrega la combinación del vector inicializador en ronda.}
18: *VI*[*hilo*] $\leftarrow C[hilo]$ {Se actualiza el vector inicializador para la siguiente ronda.}
19: **devolver** C

Algoritmo 27: Kernel para el cifrado paralelo de un solo bloque AES (versión 1).

La invocación del kernel descrito en el Algoritmo 27 se realiza de la siguiente manera:

$<<<$ 1, 16 $>>>AES_{Cifrar}$(*B*, *VI*, *Llaves*, S_{Box}, *pos*)

5.1.1.2. Paralelización del proceso de descifrado (un bloque)

Los Algoritmos 28, 29, 30 y 31 muestran el pseudocódigo del núcleo de procesamiento para CUDA de las funciones InvSubBytes, InvAddRoundKey, InvShiftRows e InvMixColumns, respectivamente. La invocación de cada uno de los kernels de CUDA se hace como sigue:

- $<<<$ 1, 16 $>>>InvSubBytesPar(C, B)$
- $<<<$ 1, 16 $>>>InvAddRoundKeyPar(C, B)$
- $<<<$ 1, 16 $>>>InvShiftRowsPar(C, B)$
- $<<<$ 1, 4 $>>>InvMixColumnsPar(C, B)$

Entrada: C: Bloque de datos de entrada, S_{BoxInv}: Vector de sustitución.
Salida: B Bloque de datos de salida.
1: $B[hilo] \leftarrow S_{BoxInv}[C[hilo]]$
2: **devolver** $B[hilo]$

Algoritmo 28: Kernel *InvSubBytesPar* en CUDA.

Entrada: C: Bloque de datos de entrada, *Llaves**: Llaves de ronda.
Salida: B Bloque de datos de salida.
1: $B[hilo] \leftarrow C[hilo]$ XOR *llaves*[*hilo*]
2: **devolver** $B[hilo]$

Algoritmo 29: Kernel *InvAddRoundKeyPar* en CUDA.

Entrada: C: Bloque de datos de entrada, *posInv*: Vector de posiciones.
Salida: B Bloque de datos de salida.
1: $C[hilo] \leftarrow posInv[B[hilo]]$
2: **devolver** $B[hilo]$

Algoritmo 30: Kernel *InvShiftRowsPar* en CUDA.

Entrada: C: Bloque de datos de entrada.
Salida: B Bloque de datos de salida.
1: $B[hilo*4] \leftarrow$ `CalcCoef`$(C[hilo*4])$
2: $B[hilo*4+1] \leftarrow$ `CalcCoef`$(C[hilo*4+1])$
3: $B[hilo*4+2] \leftarrow$ `CalcCoef`$(C[hilo*4+2])$
4: $B[hilo*4+3] \leftarrow$ `CalcCoef`$(C[hilo*4+3])$
5: **devolver** $B[hilo]$

Algoritmo 31: Kernel *InvMixColumnsPar* en CUDA.

Al igual que en el cifrado, los tres primeros kernels se pueden invocar con un solo bloque de ejecución y 16 hilos, uno por cada dato a descifrar. Por otra parte, el kernel de la función InvMixColumns se invoca solamente con 4 hilos, ya que como en su operación inversa (MixColumns) el cálculo del coeficiente del polinomio que corresponde al byte cifrado se realiza a través de una operación dependiente de los otros demás bytes en la fila del bloque de datos.

Así mismo, los kernels individuales para el descifrado en AES fueron integrados en uno solo (Algoritmo 32), haciendo el proceso de descifrado parcial de los datos dentro de él. En este kernel se cumplen las mismas características del kernel de cifrado, aprovechando la sincronía de los hilos de procesamiento despachados por un warp para procesar 16 bytes, sin embargo también se desperdician 16 hilos que potencialmente podrían ser despachado en el mismo warp.

Entradas: B: Bloque en claro, *VI*: Vector de inicialización, *Llaves**: Llaves de ronda, S_{BoxInv}: Vector de sustitucion, *posInv*: Vector de posiciones.
Salida: **C** Bloque cifrado.
1: $B[hilo] \leftarrow C[hilo]$ `XOR` *VI*$[hilo]$ {Se agrega la combinación del vector inicializador en ronda.}
2: *VI*$[hilo] \leftarrow C[hilo]$ {Se actualiza el vector inicializador para la siguiente ronda.}
3: $B[hilo] \leftarrow B[hilo]$ `XOR` *llaves*[11]
4: **para** $i \leftarrow$ 10 hasta 2 **hacer**
5: $B[hilo] \leftarrow B[$*posInv*$[hilo]]$
6: $B[hilo] \leftarrow S_{BoxInv}[hilo]$
7: $B[hilo] \leftarrow B[hilo]$ `XOR` *llaves*[i]
8: **si** $hilo < 4$ **entonces** {Dado que se procesa un bloque de 16 bytes en paralelo, los primeros 4 hilos realizan la transformación InvMixColumns.}
9: $B[hilo*4] \leftarrow$ `CalcCoef`$(C[hilo*4])$
10: $B[hilo*4+1] \leftarrow$ `CalcCoef`$(C[hilo*4+1])$
11: $B[hilo*4+2] \leftarrow$ `CalcCoef`$(C[hilo*4+2])$
12: $B[hilo*4+3] \leftarrow$ `CalcCoef`$(C[hilo*4+3])$
13: **fin si**
14: **fin para**
15: $B[hilo] \leftarrow B[$*posInv*$[hilo]]$
16: $B[hilo] \leftarrow S_{BoxInv}[hilo]$
17: $B[hilo] \leftarrow B[hilo]$ `XOR` *llaves*[1]
18: **devolver** $B[hilo]$

Algoritmo 32: Kernel de descifrado paralelo de un solo bloque AES (versión 1).

La invocación de este kernel se realiza de la siguiente manera:

$<<<$ 1, 16 $>>>AES_{Descifrar}$(*B*, *VI*, *Llaves*, S_{Box}, *pos*)

5.1.2 Algoritmos de capa media

5.1.2.1. Cifrado en modo CBC

El cifrado en modo CBC de AES establece una secuencia en la cual cada bloque de cifrado sirve de entrada en el procesamiento del siguiente bloque. Lo anterior establece una dependencia inherentemente secuencial entre el cifrado de un bloque en claro y el bloque cifrado anterior. Por el contrario, el descifrado es altamente paralelizable, ya que cada bloque cifrado se puede descifrar simultáneamente y todos los bloques descifrados se combinan con el bloque a descifrar anterior, excepto el primer bloque descifrado que se combina con el vector de inicialización (VI). Lo anterior se ilustra en la Figura 5.2, en la cual, el proceso mostrado en la parte superior corresponde a la etapa de cifrado y la parte inferior corresponde al descifrado.

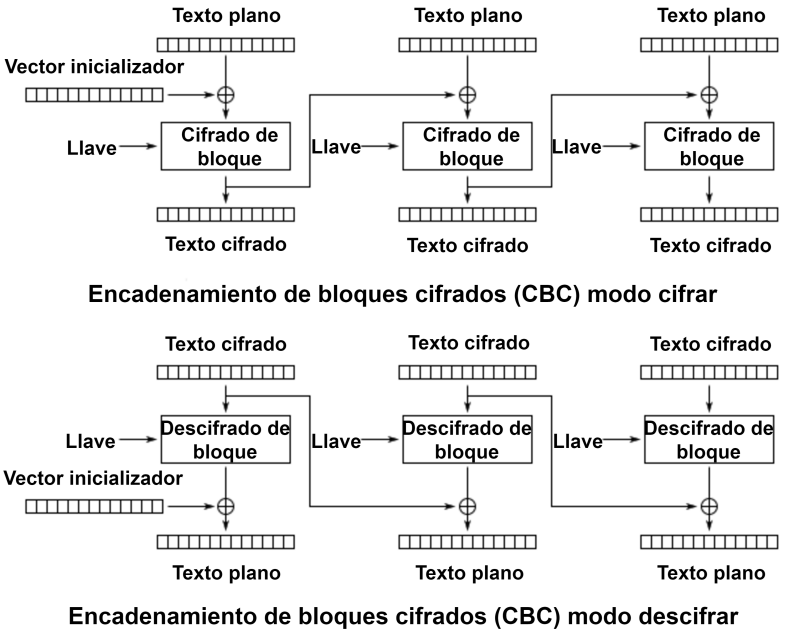


Figura 5.2: Cifrado y descifrado con AES en modo CBC.

Con el propósito de paralelizar el proceso de cifrado de AES en modo CBC, en este trabajo se propone una modificación al algoritmo original mediante el uso de varios vectores de inicialización, uno para cada bloque procesado en paralelo. Estos vectores son obtenidos a partir del *VI* inicial.

5.1.2.2. Cifrado en modo CBC modificado

La estrategia de cifrado y descifrado paralelo propuesta se basa en que estos procesos pueden ser paralelizados al dividir el texto a cifrar en t partes y asignar un vector de inicialización para cada parte del cifrado. Dichos vectores inicializadores son derivados del principal, de tal forma que al tener un vector inicializador *VI* pueden obtenerse VI_0, VI_1, VI_2... VI_{t-1}. El proceso de cifrado se realiza de tal forma que pueden ser cifrados un conjunto de t bloques de datos en paralelo. La Figura 5.3 muestra la división de los datos de entrada en bloques de 128 bits, y donde se pueden procesar t bloques al mismo tiempo usando t módulos AES independientes.

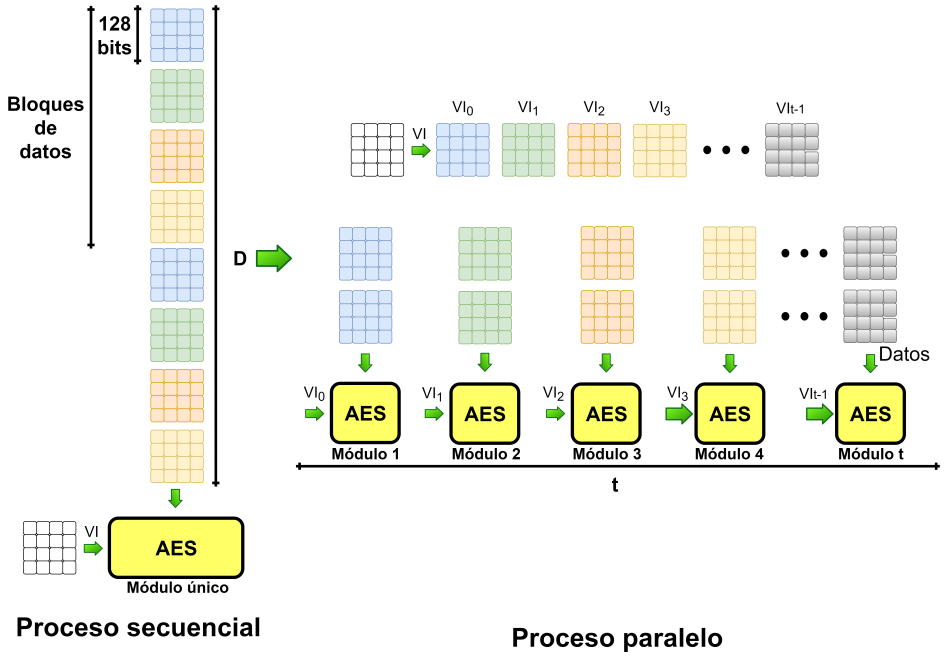


Figura 5.3: Cifrado de datos con AES en modo CBC. En la versión secuencial, se procesa un bloque de datos de 128 bits a la vez. En la versión paralela, se pueden procesar t bloques usando t módulos AES independientes.

5.1.2.3. *Estrategia de paralelización por bloques*

Debido a que la implementación de AES en sus operaciones más básicas son altamente paralelizables, existen diferentes estrategias para aprovechar el máximo de recursos disponibles en los aceleradores GPU para su paralelización. Tomando como base el principio fundamental para el óptimo aprovechamiento de los GPUs, la idea es crear tantos hilos de ejecución como datos se tengan que procesar. Sin embargo, CUDA y los GPUs requieren agrupar los hilos en bloques de procesamiento del mismo tamaño, donde se organiza el procesamiento de los datos dividiendo la cantidad de datos en un determinado número de bloques de ejecución.

La Figura 5.4 muestra la estrategia de paralelización de los datos asignando un hilo a cada dato, donde un determinado número S de bloques de 16 datos se agrupan en un bloque de $S \times 16$ hilos. En el siguiente nivel se crean P bloques de $S \times 16$ hilos, los cuáles procesan $P \times S \times 16$ datos en forma simultánea. CUDA se encarga de distribuir los datos a procesar entre los bloques de hilos construidos. De este modo, se considera que la cantidad de vectores inicializadores es igual al producto de $P \times S$.

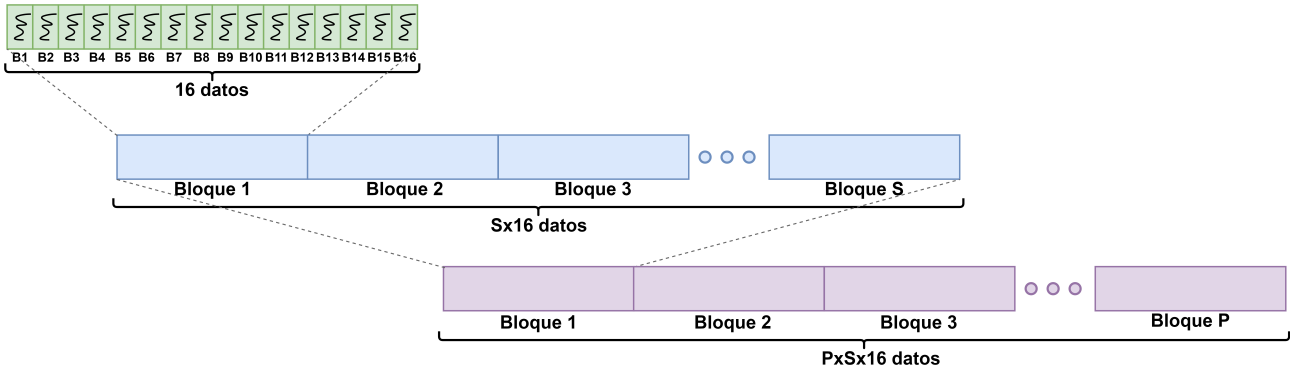


Figura 5.4: Niveles de procesamiento paralelo de AES-CBC en CUDA.

5.1.2.4. *Paralelización del proceso de cifrado (varios bloques)*

Paralelización utilizando bloques de 2x16 hilos

Se utilizó una estrategia capaz de aprovechar los 32 hilos que se despachan por bloque de

procesamiento en cada kernel, de tal modo que en lugar de solo cifrar un bloque de datos, se cifran de forma paralela dos de estos bloques, reduciendo así en un 50 % el tiempo de procesamiento con respecto a la primer versión del cifrador AES paralelo. El Algoritmo 33 muestra el kernel de CUDA que procesa un total de 32 bytes por bloque de hilos (S = 2). En este algoritmo, se realizaron algunas modificaciones correspondientes a la definición de las posiciones de los corrimientos (considerando que se procesa un total de dos bloques de forma paralela) y se agregó un vector de corrimientos *corr*, el cual define la columna sobre la que se calcula el coeficiente del polinomio evaluado (considerando que al procesar dos bloques existe el doble de columnas). La estructura de este kernel permite utilizar múltiples vectores inicializadores, para los cuales se divide el cifrado en V conjuntos de datos cifrados que conforman el documento cifrado.

Entradas: B: Bloque en claro, *VI*: Conjunto de vectores inicializadores, *Llaves**: Llaves de ronda, S_{Box}: Vector de sustitucion, *pos*: Vector de posiciones, *corr*: Vector de corrimientos.
Salida: **C** Bloque cifrado.
1: $B[hilo] \leftarrow B[hilo]$ `XOR` *llaves*[1]
2: **para** i $\leftarrow$ 2 hasta 10 **hacer**
3: $B[hilo] \leftarrow S_{Box}[hilo]$
4: $B[hilo] \leftarrow B[pos[hilo]]$
5: **si** *hilo* < 8 **entonces** {Dado que se procesan dos bloques de 16 bytes en paralelo, los primeros 8 hilos realizan la transformación MixColumns (4 para cada bloque).}
6: $B[corr[hilo*4]] \leftarrow$ `CalcPol`$(B[corr[hilo*4]])$
7: $B[corr[hilo*4+1]] \leftarrow$ `CalcPol`$(B[corr[hilo*4+1]])$
8: $B[corr[hilo*4+2]] \leftarrow$ `CalcPol`$(B[corr[hilo*4+2]])$
9: $B[corr[hilo*4+3]] \leftarrow$ `CalcPol`$(B[corr[hilo*4+3]])$
10: **fin si**
11: $B[hilo] \leftarrow B[hilo]$ `XOR` *llaves*[i]
12: **fin para**
13: $B[hilo] \leftarrow S_{Box}[hilo]$
14: $B[hilo] \leftarrow B[pos[hilo]]$
15: $B[hilo] \leftarrow B[hilo]$ `XOR` *llaves*[11]
16: $C[hilo] \leftarrow B[hilo]$
17: $C[hilo] \leftarrow B[hilo]$ `XOR` *VI*[*hilo*]{Se agrega la combinación del vector inicializador en ronda.}
18: *VI*[*hilo*] $\leftarrow C[hilo]${Se actualiza el vector inicializador para la siguiente ronda.}
19: **devolver** C

Algoritmo 33: Kernel de cifrado paralelo AES de dos bloques de datos (versión 2).

La invocación de este kernel se realiza de la siguiente manera:

$<<<P, 16\times2>>>AES_{Cifrar}(B, VI, Llaves, S_{Box}, pos, corr))$.

Así, $P\times2\times16$ datos son cifrados en paralelo, lo que requiere $P\times2$ vectores inicializadores.

Paralelización utilizando bloques de *S*x16 hilos

Ya que bajo el enfoque propuesto los dispositivos GPU permiten procesar más de dos bloques de datos de forma paralela en cada bloque de procesamiento del kernel, calculando el parámetro S se puede conseguir una mayor reducción del tiempo de procesamiento del cifrado en AES. Sin embargo, si los bloques de procesamiento del kernel contienen más de 32 hilos se requerirán más warps para despacharlos en el el dispositivo GPU. Debido a que los hilos pueden estar en warps diferentes, no se puede garantizar una ejecución síncrona, por lo que, para asegurar el correcto procesamiento paralelo es necesario indicar puntos de sincronización en las secciones donde se pueden presentar condiciones de carrera en los datos.

El valor de S se puede definir tomando como base la cantidad máxima de hilos que se despachan para los núcleos físicos del dispositivo y también el número máximo de bloques que se pueden atender en un solo SM[1]. De esta forma, se generan tantos vectores inicializadores como sea configurado en la entrada (considerando los tamaños de datos como limitante para el cálculo de P).

El Algoritmo 34 describe la estrategia seguida para cifrar los datos con AES, en donde se invocan P bloques de hilos, los cuales procesan $S \times 16$ datos de forma simultánea. En este algoritmo, la función `SincronizarHilos()` actúa como un punto de sincronización global entre todos los hilos del mismo bloque, forzando a que ningún hilo pueda continuar su proceso hasta que todos los hilos del bloque hayan alcanzado el mismo punto.

[1]Por cada SM de un GPU, hay un número máximo de bloques de hilos que se pueden procesar, típicamente un valor de 6 a 8 bloques.

Entradas: B: Bloque en claro, *VI*: Conjunto de vectores inicializadores, *Llaves**: Llaves de ronda, S_{Box}: Vector de sustitucion, *pos*: Vector de posiciones, *corr*: Vector de corrimientos.
Salida: **C** Bloque cifrado.

```
1: B[hilo] ← B[hilo] XOR llaves[1]
2: SincronizarHilos()
3: para i ← 2 hasta 10 hacer
4:     B[hilo] ← S_Box[hilo]
5:     B[hilo] ← B[pos[hilo]]
6:     SincronizarHilos()
7:     si hilo < S×4 entonces {Dado que se procesan S bloques de 16 bytes en paralelo, los primeros S×4 hilos realizan la transformación MixColumns (4 para cada bloque).}
8:         B[corr[hilo*4]] ← CalcPol(B[corr[hilo*4]])
9:         SincronizarHilos()
10:        B[corr[hilo*4+1]] ← CalcPol(B[corr[hilo*4+1]])
11:        SincronizarHilos()
12:        B[corr[hilo*4+2]] ← CalcPol(B[corr[hilo*4+2]])
13:        SincronizarHilos()
14:        B[corr[hilo*4+3]] ← CalcPol(B[corr[hilo*4+3]])
15:        SincronizarHilos()
16:    fin si
17:    B[hilo] ← B[hilo] XOR llaves[i]
18:    SincronizarHilos()
19: fin para
20: B[hilo] ← S_Box[hilo]
21: SincronizarHilos()
22: B[hilo] ← B[pos[hilo]]
23: SincronizarHilos()
24: B[hilo] ← B[hilo] XOR llaves[11]
25: SincronizarHilos()
26: C[hilo] ← B[hilo]
27: SincronizarHilos()
28: C[hilo] ← B[hilo] XOR VI[hilo]{Se agrega la combinación del vector inicializador en ronda.}
29: SincronizarHilos()
30: VI[hilo] ← C[hilo]{Se actualiza el vector inicializador para la siguiente ronda.}
31: devolver C
```

Algoritmo 34: Kernel de cifrado paralelo AES de $P \times S$ bloques de datos (versión 3).

La invocación del kernel descrito en el Algoritmo 34 se realiza de la siguiente manera:

$<<<$ *P*, 16×*S* $>>> AES_{Cifrar}$(*B*, *VI*, *Llaves*, S_{Box}, *pos*, *corr*)

De esta forma, $P \times S \times 16$ datos son cifrados en paralelo. Para esto, se generaron $P \times S$ vectores inicializadores.

5.1.2.5. Paralelización del proceso de descifrado (varios bloques)

Dadas las características del cifrado y descifrado con AES, y la similaridad de sus operaciones básicas, el proceso de descifrado se realiza de igual forma que en el cifrado paralelo. El proceso de

descifrado se ilustra en el diagrama de la Figura 5.3, agrupando por bloques los datos a descifrar y se realizan las operaciones de descifrado con la versión final del *kernel* de descifrado paralelo optimizado que se muestra en el Algoritmo 35, donde se utilizan los mismos vectores inicializadores que en la etapa de cifrado.

Entradas: C_{hilo}: Bloque cifrado, *VI*: Conjunto de vectores inicializadores, *Llaves**: Llaves de ronda, S_{Box}: Vector de sustitucion, *pos*: Vector de posiciones, *corr*: Vector de corrimientos.
Salida: B_i Bloque en claro.

1: $B[hilo] \leftarrow C[hilo]$ `XOR` $VI[hilo]$ {Se agrega la combinación del vector inicializador en ronda.}
2: `SincronizarHilos()`
3: $VI[hilo] \leftarrow C[hilo]$ {Se actualiza el vector inicializador para la siguiente ronda.}
4: `SincronizarHilos()`
5: $B[hilo] \leftarrow B[hilo]$ `XOR` $llaves[11]$
6: `SincronizarHilos()`
7: **para** $i \leftarrow 10$ hasta 2 **hacer**
8: $B[hilo] \leftarrow B[posInv[hilo]]$
9: `SincronizarHilos()`
10: $B[hilo] \leftarrow S_{BoxInv}[hilo]$
11: `SincronizarHilos()`
12: $B[hilo] \leftarrow B[hilo]$ `XOR` $llaves[i]$
13: `SincronizarHilos()`
14: **si** $hilo < S \times 4$ **entonces** {Dado que se procesan S bloques de 16 bytes en paralelo, los primeros $S \times 4$ hilos realizan la transformación MixColumns (4 para cada bloque).}
15: $B[corr[hilo * 4]] \leftarrow$ `CalcCoef`$(B[corr[hilo * 4]])$
16: `SincronizarHilos()`
17: $B[corr[hilo * 4 + 1]] \leftarrow$ `CalcCoef`$(B[corr[hilo * 4 + 1]])$
18: `SincronizarHilos()`
19: $B[corr[hilo * 4 + 2]] \leftarrow$ `CalcCoef`$(B[corr[hilo * 4 + 2]])$
20: `SincronizarHilos()`
21: $B[corr[hilo * 4 + 3]] \leftarrow$ `CalcCoef`$(B[corr[hilo * 4 + 3]])$
22: `SincronizarHilos()`
23: **fin si**
24: **fin para**
25: $B[hilo] \leftarrow B[posInv[hilo]]$
26: `SincronizarHilos()`
27: $B[hilo] \leftarrow S_{BoxInv}[hilo]]$
28: `SincronizarHilos()`
29: $B[hilo] \leftarrow B[hilo]$ `XOR` $llaves[1]$
30: `SincronizarHilos()`
31: **devolver** B_i

Algoritmo 35: Función $AES_Descifrar$ del kernel de descifrado paralelo AES de $P \times S$ bloques de datos (versión 2).

La invocación del *kernel* de descifrado se realiza de la siguiente manera:

$<<<$ *P*, $S \times 16$ $>>> AES_{Descifrar}$(*B*, *VI*, *Llaves*, *pos*, S_{Box}, *corr*)

Similar al caso del cifrado, $P \times S \times 16$ datos son descifrados en paralelo para lo cual se requieren $P \times S$ vectores inicializadores.

5.1.2.6. Análisis para determinar el número de bloques a procesar en paralelo

Para la ejecución de múltiples bloques de cifrado dentro de los bloques de ejecución del kernel, los valores P y S se eligen de forma que el aprovechamiento de los recursos de los dispositivos GPU sea óptimo. Se deben considerar los siguientes aspectos:

- Se debe tomar en cuenta que los hilos se despachan en grupos de 32 hilos que forman un warp y que trabajan de forma síncrona.
- Existe un número límite en la cantidad de hilos que se pueden despachar para tener a todos los núcleos físicos ocupados, lo cual varía de acuerdo con el modelo del dispositivo. Típicamente éste varía entre 1024 y 2048 hilos.
- Existe un número límite de bloques de hilos que se pueden despachar de forma simultánea, típicamente, entre 4 y 8 bloques de hilos.

Para aprovechar la sincronía de los warps, se puede elegir $S = 2$ y eliminar todas las operaciones de sincronización de los Algoritmos 34 y 35. Sin embargo, con el fin de obtener el máximo aprovechamiento del dispositivo se ejecutan más de dos bloques de datos en paralelo, lo cual se define con base en las características del dispositivo GPU que se dispone.

La Ecuación 5.1 muestra la fórmula que permite calcular la cantidad de bloques de datos que puede ser procesada en la ejecución de un solo bloque de ejecución del kernel.

$$S = \frac{MaxHil}{NumSMs \times 16} \tag{5.1}$$

Donde, *MaxHil* es la cantida máxima de hilos que pueden ser despachados entre todos los núcleos físicos del dispositivo GPU y *NumSMs* es la cantidad máxima de SMs que se dispone en el dispositivo. De esta forma, por ejemplo, si se tiene un máximo de 1536 hilos despachados para 8 SM, el total de bloques que pueden ser procesados en paralelo por bloque del kernel es 12 (192 datos).

Por otra parte, P se define en función al tamaño de los datos. Así, para archivos muy grandes se puede usar el límite de 1024 bloques, lo que permitiría procesar 1024x192 (196608 datos) en forma simultánea. Si se tiene un documento de tamaño menor a 19kB, el parámetro P puede ser fijado con base en la cantidad específica de bytes que se desean procesar, para lo cual debe ser definido el valor de P y S.

5.2 Algoritmos paralelos para el cifrador CP-ABE (capa alta)

Ya que el grado de paralelismo de las funciones de CP-ABE está en función de la cantidad de atributos de los políticas de seguridad y, aunque éstas pueden ser suficientemente complejas que requieran decenas o incluso cientos de atributos, no es factible pensar en políticas que tengan miles o millones de atributos. Por lo anterior, es suficiente con usar un enfoque con tantos hilos de ejecución como núcleos físicos tenga una arquitectura de una computadora convencional para acelerar su procesamiento. El paradigma paralelo en datos para GPUs está enfocado a paralelismo masivo en donde se tienen varios miles o millones de datos que procesar. Aunque los GPUs se pueden usar para procesar cientos de datos, su mejor aprovechamiento es cuando se tiene un nivel de paralelismo masivo[2]. En el caso particular de CP-ABE, la estrategia de paralelización se concentra en la capa media, para la cual pueden ser paralelizados sus tres algoritmos principales `CP-ABE.Encrypt`, `CP-ABE.KeyGen` y `CP-ABE.Decrypt`.

[2]Por otra parte, la implementación de las funciones de CP-ABE requiere bibliotecas para realizar emparejamientos con curvas elípticas, las cuales al momento del desarrollo de este trabajo no se encontraban disponibles para los GPUs que se utiizarían en este trabajo.

5.2.1 Paralelización del proceso de generación de la llave de descifrado

El procesamiento que se realiza en la función `CP-ABE.KeyGen` (Algoritmo 4.1.4) comprende principalmente el cálculo de los pares $[d_y, d'_y]$ dentro de un ciclo *for*, donde para cada atributo y se calcula un par diferente. Estos pares conforman la llave de descifrado CP-ABE y se van agregando en el orden en que se procesan. Sin embargo, el orden es irrelevante, ya que para el descifrado en CP-ABE se realiza una búsqueda con base en los atributos del usuario y no en posiciones específicas en un arreglo de atributos. Debido a la característica del cálculo de dichos pares en la implementación secuencial, se puede conseguir una ganancia al repartir este trabajo entre diferentes hilos de procesamiento, los cuales procesarán conjuntos de atributos iguales (en un caso ideal) y a partir de esto, se logrará reducir el tiempo de procesamiento en la función. Ya que no existe dependencia entre pares de diferentes atributos, los cálculos para cada par son altamente paralelizables.

En el Algoritmo 36 se presenta el algoritmo paralelo para la generación de llaves de CP-ABE, realizando primeramente el cálculo del componente *D* de la llave privada de manera secuencial.

Entrada: *MK* = $\{\beta, {g_1}^{\alpha}\}$: Llave maestra, $\mathbf{A_u}$: Lista de atributos de usuario.
Salida: *SK* = $[D, L]$: Una llave privada de usuario para el descifrado en CP-ABE.

1: $r \leftarrow$ `GenerarAleatorio`$(\mathbb{Z}_r)$
2: $D \leftarrow {g_1}^{\frac{\alpha+r}{\beta}}$
3: **para todo** y **en** A_u **hacer** {Cada atributo y se procesa en paralelo.}
4: $\quad r_j \leftarrow$ `GenerarAleatorio`$(\mathbb{Z}_r)$
5: $\quad H \leftarrow$ `Mapear`$(y, \mathbb{G}_2)$
6: $\quad d_y \leftarrow g_1^r \times H^{r_j}$
7: $\quad d'_y \leftarrow g_0^{r_j}$
8: $\quad$ `Agregar`$(L, [d_y, d'_y])$
9: **fin para**
10: **devolver** SK $\leftarrow \{D, L\}$

Algoritmo 36: Función `CP-ABE.KeyGenPar`(MK, A_u) para la generación paralela de la llave de descifrado CP-ABE.

Siguiendo una estrategia de paralelismo de datos, si se tienen en total *m* atributos de usuario en A_u, se pueden repartir entre *h* hilos diferentes, los cuales procesan $\frac{m}{h}$ atributos cada uno. En la Figura 5.5 se muestra como los atributos de un usuario con $m = 8$ se dividen entre $h = 3$ hilos.

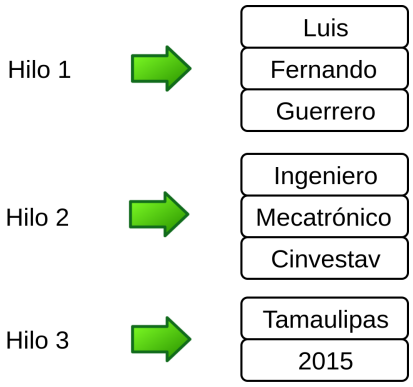


Figura 5.5: Distribución de grupos de atributos en hilos de procesamiento.

Con la estrategia de paralelización descrita, cada hilo se encarga de una cantidad fija de atributos, el último de los hilos se encarga de los atributos restantes. Por otro lado, el cálculo del componente D se puede realizar en forma secuencial antes de procesar los pares $[d_y, d'_y]$, tal como se calcula en la implementación secuencial que se muestra en la Sección 4.1.4.

5.2.2 Paralelización del proceso de cifrado

El esquema de Shamir para la distribución de un secreto es el proceso más costoso que se realiza en el módulo `CP-ABE.Encrypt`, donde una vez que se tienen los secretos correspondientes a cada nodo hoja de la estructura de acceso $\mathbb{A}$, se pueden calcular los pares $[C_y, C'_y]$ de forma independiente. Sin embargo, para obtener esos secretos se debe procesar cada nodo interno hasta llegar a las hojas, donde dicho proceso es secuencial. Durante el procesamiento secuencial, siempre que cada nodo hoja en la estructura de acceso es visitado, se almacenan el atributo *atrr* y el *subsecreto* correspondiente en un par de vectores.

El Algoritmo 37 muestra como en lugar de llamar a la función `CifradoParcial` (Algoritmo 4) para calcular el par $[C_y, C'_y]$, se crea una posición en los arreglos *subsecreto* y *atrr* para que el procesamiento de las hojas se distribuya en paralelo posteriormente. Una vez que se cuenta con los secretos correspondientes a cada atributo de la estructura de acceso se realiza el cálculo de C_y y C'_y para cada *subsecreto* y nodo hoja de forma paralela a través del Algoritmo 38.

Entradas: *nodo*: Un nodo de la estructura de acceso $\mathbb{A}$, *atrr*: Vector de atributos en estructura de acceso $\mathbb{A}$, **s** : Secreto a distribuir en los hijos del nodo de entrada, ***subsecreto***: Vector de sub-secretos asociados a los atributos en la estructura de acceso $\mathbb{A}$.
Salida: L: Lista de pares {atributo, subsecreto}

```
1: si nodo es nodo interno entonces
2:     P ← CrearPolinomio (nodo.k, s)
3:     para i ← 1 hasta nodo.n hacer
4:         Calcular( [i, P(i)] )
5:         DShamirParalelo(nodo.hijo[i], P(i), subsecreto)
6:     fin para
7: si no
8:     Agregar(L, [nodo.atributo, s])
9: fin si
```

Algoritmo 37: Función recursiva `DShamirPar(nodo, atrr, s)` para la distribución de un secreto en una estructura de acceso.

Entradas: *atrr*: Vector de atributos en estructura de acceso $\mathbb{A}$, *subsecreto*: Vector de sub-secretos asociados a los atributos en la estructura de acceso $\mathbb{A}$.
Salidas: **L**: Lista asociada al texto cifrado en CP-ABE.

```
1: para todo y en atrr y s en subs hacer {Cada atributo se procesa en paralelo.}
2:     H ← Mapear(y, G2)
3:     C_y ← g_0^s
4:     C'_y ← H^s
5:     Agregar(L, [C_y, C'_y])
6: fin para
7: devolver L
```

Algoritmo 38: Función `CifradoParcialParalelo`(*atrr*, *subs*) para el cálculo de los pares $[C_y, C'_y]$ en la lista L.

La Figura 5.6 muestra la forma en que la estructura de acceso es recorrida secuencialmente, calculando así el *subsecreto* correspondiente para cada nodo hoja. Después de visitar todos los nodos hoja, se habrá obtenido el vector de sub-secretos necesario para el cálculo de C_y y C'_y. La función *func* representa las operaciones que deben realizarse sobre el secreto asignado a cada nodo para derivar los subsecretos asignados a sus nodos hijo, bajo las reglas de funcionamiento del esquema de Shamir descritas en la Sección 2.1.10.2.

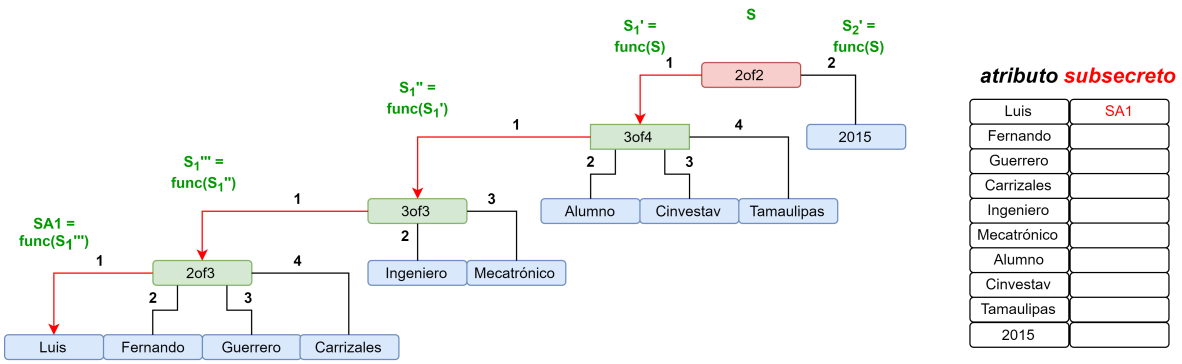


Figura 5.6: Proceso de distribución de Shamir para el cálculo de los *subsecretos* (recorrido del nodo raíz hacia el primer nodo hoja).Los pares [atributo, subsecreto] calculados se almacenan en un vector para su posterior procesamiento en paralelo.

5.2.3 Paralelización del proceso de descifrado

Para cada componente $[d_y, d'_y]$ y $[C_y, C'_y]$ en *SK* y *CT* respectivamente se realiza el cálculo de $\frac{e(d'_y, C'_y)^{exp_{nl}}}{e(d_y, C_y)} = emp_y$ para cada atributo y en la estructura de acceso podada. Por ello, primero se realiza el cálculo de cada valor exp_{nl}, mismo que se obtiene al procesar de forma secuencial cada arista que conforma el camino desde el nodo raíz de la estructura de acceso podada hasta el nodo hoja que contiene el atributo y. De forma similar al cifrado, se genera un par de vectores, uno de los exponentes (*exp*) y el otro de atributos (*atrrdec*). Al tener todos los pares $[y, exp]$ pre-calculados, se calculan los valores de emp_y en paralelo. Después, todos los valores emp_y se multiplican para obtener $e(g_0, g_1)^{rs}$.

El Algoritmo 39 se utiliza para pre-calcular el vector de exponentes asociados a los atributos en la estructura de acceso podada. Este vector *exp* es una entrada de la función junto al vector de atributos *atrrdec*, de tal modo que al visitar cada nodo hoja se agregan tanto el atributo como el valor del exponente.

Entradas: *nodo*: Un nodo de la estructura de acceso, V: Vector de pares [y, *exp*], *exp*: Valor del exponente actual.
Salidas: V: Vector modificado.

```
1: si nodo es interno entonces
2:     j = Pos(nodo)
3:     X = CalcularX_j(nodo)
4:     newExp = exp * CoefLagrange(j, X)
5:     para i ← 1 hasta nodo.k hacer
6:         RShamirParalelo(nodo.hijo[i], newExp)
7:     fin para
8: si no
9:     Agregar(V, [nodo.atributo, exp])
10: fin si
```

Algoritmo 39: Función RShamirParlaleo(*nodo*, V, *exp*) para la reconstrucción de un secreto en una estructura de acceso $\mathbb{A}$ de forma paralela.

En la Figura 5.7 se muestra como se lleva a cabo el cálculo de exponentes correspondientes a los primeros cuatro nodos hoja. En la Figura, la función *func* denota las operaciones que se realizan en cada nodo para derivar el nuevo exponente, y corresponde al proceso de reconstrucción de secretos de Shamir descrito en la Sección 2.1.10.3.

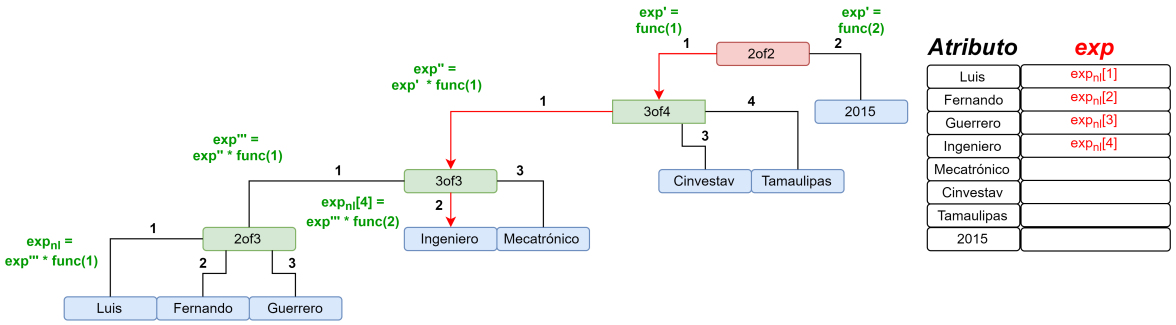


Figura 5.7: Pre-calculo secuencial de los exponentes asociados a cada nodo hoja (atributo y), para su posterior procesamiento paralelo.

Por último, ya que fueron calculados los exponentes correspondientes a cada atributo de la estructura de acceso se realiza el cálculo de emp_y en forma paralela, es decir, procesando cada par (y, *exp*) de forma independiente para generar emp_y. Cada uno de estos valores es acumulativamente multiplicado para obtener $e(g_0, g_1)^{rs}$. Todas estas operaciones son expresadas en el Algoritmo 40.

Entradas: V: Vector de pares [y, *exp*], $CT = [C, C', L_1 = \{p_y\}^+]$: Texto cifrado, $SK = [D, L_2 = \{q_y\}^+]$: Una llave privada de usuario para el descifrado en CP-ABE.
Salida: *SR*: Secreto reconstruido.
1: $SR \leftarrow 1$
2: **para todo** [y, *exp*] **en** V **hacer** {Cada atributo se procesa en paralelo.}
3: $[d_y, d'_y] \leftarrow$ `ObtenerPar`(SK,y)
4: $[C_y, C'_y] \leftarrow$ `ObtenerPar`(CT,y)
5: $emp_y \leftarrow \frac{e(d'_y, C'_y)^{exp}}{e(d_y, C_y)}$
6: $SR \leftarrow SR * emp_y$
7: **fin para**
8: **devolver** $\{SR\}$

Algoritmo 40: Función `CalcularSR`(A_u, *exp*, *CT*, *SK*).

5.3 Paralelización del esquema DET-ABE (capa alta)

El esquema paralelo DET-ABE integra las versiones paralelas de CP-ABE y AES. El Algoritmo 41 muestra la versión paralela de DET-ABE en donde la función `AES.Encrypt` secuencial es reemplazada por la función `AES.EncryptPar`, que contiene el kernel de CUDA para el cifrado de los datos versión 2. Así también, la función `CP-ABE.Encrypt` secuencial es reemplazada por la función `CP-ABE.EncryptPar`.

Es posible construir dos versiones paralelas del Algoritmo 41 (`DET-ABE.EncryptPar`).

1. **Versión 1.** Ejecución secuencial de los algoritmos paralelos `AES.EncryptPar` y `CP-ABE.EncryptPar`, esto es, primero se ejecuta AES paralelo y a continuación CP-ABE paralelo.

2. **Versión 2.** Ejecución paralela de los algoritmos paralelos `AES.EncrypPar` y `CP-ABE.EncryptPar`. Tal y como se muestra en el Algoritmo 41, las líneas 6 y 7 se ejecutan en paralelo creando un hilo de ejecución para cada una. Esta versión es posible debido a que una vez generada la llave AES, los dos procesos de cifrado de AES y CP-ABE pueden realizarse de forma independiente.

Un aspecto a considerar al realizar la paralelización del esquema DET-ABE a este nivel, es que cada cifrador requiere de un hilo de ejecución independiente, es decir, un hilo para AES y otro para

CP-ABE. Para el caso de un dispositivo con cuatro núcleos de procesamiento, lo anterior implica que solo dos hilos quedan disponibles para la paralelización de CP-ABE.

Bajo este ejemplo, con la versión 1 los cuatro núcleos estarían dedicados a AES y depués a CP-ABE. En el segundo caso, aunque ambos procesos de cifrado se realizan en paralelo, los recursos para CP-ABE disminuyen de cuatro a dos hilos.

Entradas: T: Texto a cifrar, *NS*: Nivel de seguridad, *CE*: Tipo de curva elíptica, P: política de acceso en notación posfija.
Salida: *SD*: Sobre digital compuesto por (CT_1 (Cifrado AES), CT_2 (Cifrado CP-ABE), *NS* (Nivel de seguridad), *CE* (Tipo de curva elíptica), $\mathbb{A}$ (Estructura de acceso), *VI* (Vector inicializador), *PK* (Llave pública)).

1: $[AES_{KEY}] \leftarrow$ `GenerarLlaveAES`*(NS)*
2: $[VI] \leftarrow$ `GenerarVI()`
3: $[PK] \leftarrow$ `CP-ABE.Setup`*(NS , CE)*
4: $A \leftarrow$ `ConstruirEstrucura`*(P)*
5: **comienza región paralela**
6: $CT_1 \leftarrow$ `AES.EncryptPar`*(T, VI, AES$_{KEY}$)* $\leftarrow$ hilo1
7: $CT_2 \leftarrow$ `CP-ABE.EncryptPar`*(PK, AES$_{KEY}$,* $\mathbb{A}$*)* $\leftarrow$ hilo2
8: **fin**
9: `Agregar`*(SD, CT$_1$, CT$_2$, NS, CE, A, VI)*
10: **devolver** *SD*

Algoritmo 41: Cifrado paralelo de DET-ABE (`DET-ABE.EncryptPar`).

La implementación del Algoritmo 41 fue realizada usando la biblioteca Iphtread en C [60], ejecutando con un hilo independiente las operaciones de `CP-ABE.EncryptPar` y con otro hilo las de `AES.EncryptPar`. En equipos con un número grande de núcleos de procesamiento disponibles, esta opción de paralelización de DET-ABE podría ser aprovechada.

Para el caso de descifrado con DET-ABE, las operaciones de descifrado de la llave AES con CP-ABE y el descifrado de los datos con AES es inherentemente secuencial. Debido a ello, no es posible construir una implementación paralela de ambas operaciones de descifrado.

5.4 Resumen

En este capítulo se describió el diseño y desarrollo del método propuesto para la implementación del esquema DET-ABE paralelo. Se describieron y detallaron las estrategias de paralelización de los algoritmos de DET-ABE a distintos niveles de procesamiento. Las estrategias de paralelización

propuestas en esta tesis se concentran principalmente en la capa media de DET-ABE, es decir, se realiza la paralelización de las operaciones de los cifradores AES y CP-ABE. Mientras que el enfoque de paralelzicación de AES es para realizar la ejecución del cifrado y descifrado en GPUs, en el caso de CP-ABE se realizó para ejecutar el cifrado, descifrado y generación de llaves bajo un enfoque de cómputo multihilo. Esto se hizo debido a que en la actualidad no se dispone de bibliotecas capaces de realizar las operaciones de emparejamientos en grupos y campos finitos necesarias para la implementación de CP-ABE paralelo. Por otra parte, se describió la propuesta para el cifrado con DET-ABE en paralelo en la capa alta, en la cual se realiza el proceso de cifrado AES y el de CP-ABE en paralelo con la biblioteca Pthreads en C. Esta alternativa de implementación puede ser más atractiva si se cuenta con dispositivos de altas prestaciones, como podrían ser servidores.

6

Experimentación y resultados

En este capítulo se presentan los resultados obtenidos en la experimentación realizada para evaluar las estrategias de paralelización propuestas en esta tesis y descritas en el capítulo 5. Para estos experimentos se proponen tres métricas de evaluación: tiempo de procesamiento, aceleración y trabajo adicional. Además, se utilizan distintas instancias variando los datos de prueba del experimento. A partir de estos experimentos se determinan la carga de procesamiento del esquema DET-ABE secuencial y paralelo, y la aceleración obtenida comparada con la versión secuencial usando distintas configuraciones en los datos de prueba y parámetros de implementación.

El proporcionar seguridad a un conjunto de datos que se transfieren a un servidor de almacenamiento en la nube genera un costo adicional (*overhead*), el cual es muy variable y dependiente principalmente del tamaño de los datos, del nivel de seguridad requerido y de la cantidad de atributos en la política de acceso. El crecimiento del trabajo adicional debido a los algoritmos de seguridad puede ser compensado en la mayoría de los casos mediante el uso de los algoritmos paralelos descritos en el Capítulo 5. Sin embargo, los resultados de implementación también se ven

fuertemente afectados por factores como el número de hilos en ejecución, el número de núcleos de procesamiento físicos disponibles, y el tipo de dispositivo acelerador GPU que se use.

Este capítulo se conforma de cinco secciones. La Sección 6.1 describe el prototipo experimental en el cual se realizan las pruebas del esquema secuencial y paralelo. La Sección 6.2 describe los datos de prueba utilizados para definir el grado de seguridad y los tiempos de procesamiento del esquema DET-ABE. La Sección 6.3 describe las pruebas de funcionamiento realizadas tanto con el cifrador simétrico AES, como para el asimétrico CP-ABE. La Sección 6.4 muestra las métricas de evaluación y las pruebas de rendimiento realizadas para los cifradores AES y CP-ABE, variando los datos de pruebas descritas en la Sección 6.2. La Sección 6.5 muestra los experimentos realizados con el esquema, donde se definen tanto los tiempos de procesamiento, las mediciones de trabajo adicional y las aceleraciones alcanzadas con la versión paralela de DET-ABE.

6.1 Prototipo experimental

En la Figura 6.1 se muestra un diagrama del prototipo experimental donde se desarrollaron las pruebas de funcionamiento y evaluación del esquema DET-ABE en su versión paralela. En esta figura se define al cliente como un dispositivo emisor, el cual cifra los datos y los transfiere a la nube con el proceso "Upload", mientras que el dispositivo receptor solicita los datos cifrados a los servidores en la nube y la llave de descifrado a la autoridad de confianza para descifrarlos.

La Tabla 6.1 describe las características del dispositivo emisor (propietario de los datos) y receptor (consumidor de los datos), así como del servidor de alojamiento de datos en la nube. En la experimentación realizada, el propietario de los datos y el consumidor son representados por el mismo dispositivo de cómputo.

La experimentación se realizó bajo dos sistemas operativos diferentes, el Dispositivo A utiliza Ubuntu 17.04 y el Dispositivo B CentOS 6.8. En ambos se utilizaron los lenguajes de programación C y CUDA C para la implementación secuencial y paralela de DET-ABE respectivamente. Para la

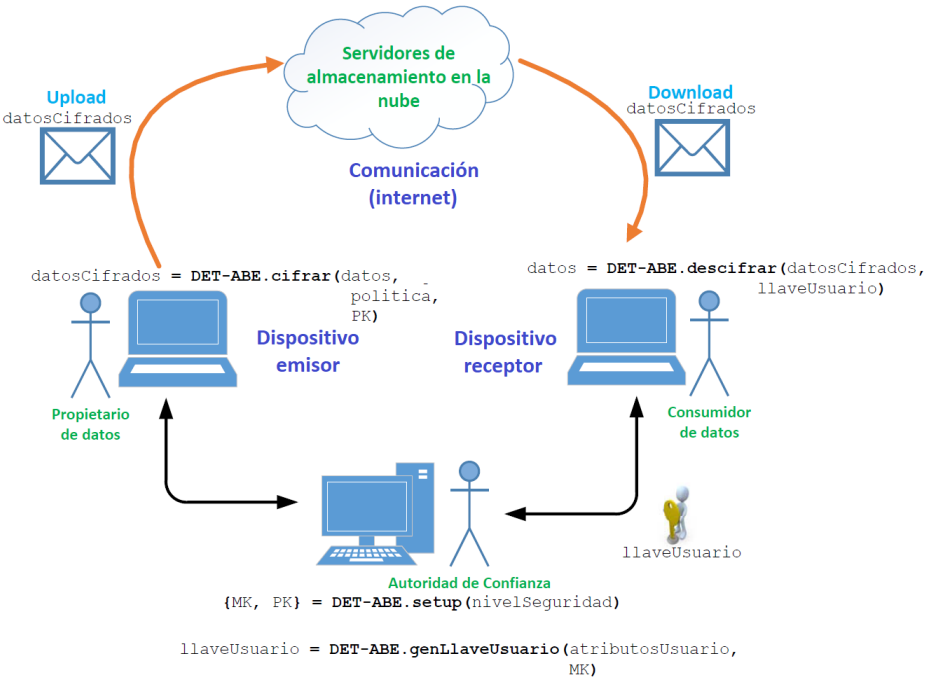


Figura 6.1: Esquema general del prototipo experimental para la transferencia segura de datos a la nube utilizando el esquema DET-ABE.

Tabla 6.1: Características de los dispositivos utilizados.

Dispositivo	Rol en el prototipo experimental	Nombre del equipo	CPU	GPUs
Dispositivo A	Emisor/Receptor	HP Pavilion Gaming 15-AK005LA	Intel Core i5-6300HQ 2.30GHz x 4	NVIDIA GeForce GTX 950M
Dispositivo B	Servidor de almacenamiento en la nube	Servidor Chronos (Cinvestav Tamaulipas)	Intel Xeon X5675 3.07GHz x 12	NVIDIA Tesla C2070

implementación de CP-ABE, se utilizó la biblioteca PBC [39] para cálculos de álgebra abstracta en grupos, campos finitos, y emparejamientos bilineales.

6.2 Datos de prueba y configuración

En la Figura 6.2 se muestran los diferentes datos de prueba usados en la experimentación. A continuación se dan detalles de cada uno de ellos.

- El **documento** es el conjunto de bytes que se cifra con DET-ABE.

- Las **políticas** son cadenas de caracteres que siguen el formato descrito en el Capítulo 2, Sección 2.1.10.1, y que representan la política de control de acceso que se requiere para cifrar un documento con DET-ABE.
- Los **atributos de usuario** son un conjunto de cadenas de caracteres que definen las credenciales de cada usuario que desea descifrar datos con DET-ABE, siempre y cuando dichos atributos satisfagan la política de cifrado usada.
- El **nivel de seguridad** indica el grado de seguridad que se provee durante el proceso de cifrado, entre más grande, mayor seguridad. El nivel de seguridad se expresa en bits y corresponde a uno de los tres niveles de seguridad soportados por el cifrador AES.
- El **número de hilos** es la cantidad de núcleos físicos del CPU, los cuales están disponibles para su uso.
- El **número de bloques AES** es la cantidad de bloques de hilos con que se invocan los kernel de procesamiento en CUDA.

Dichos parámetros afectan de forma diferente el tiempo de procesamiento de los principales componentes de DET-ABE. Por ejemplo, a mayor tamaño de documento, el tiempo de procesamiento AES incrementa, mientras que para CP-ABE el crecimiento se debe a diferentes factores, como lo son el nivel de seguridad y la cantidad de atributos con los que se cifra y descifra la información.

Por otro lado, para DET-ABE los últimos dos parámetros mencionados definen la eficiencia del paralelismo aplicado en el desarrollo de la experimentación, donde a mayor cantidad de hilos y bloques menor es el tiempo de procesamiento invertido en el cifrado, generación de llave de descifrado y descifrado de los datos (datos de configuración).

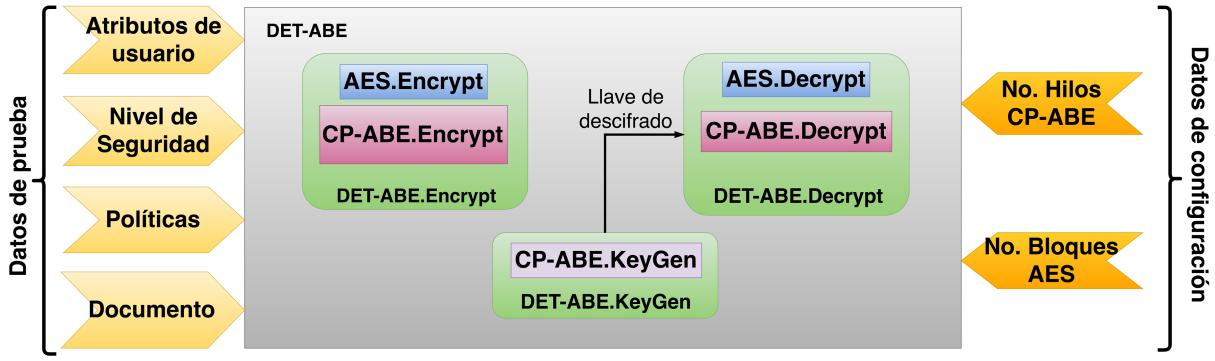


Figura 6.2: Diagrama a bloques de DET-ABE y los distintos datos de prueba y de configuración.

6.2.1 Datos de entrada del esquema

En este apartado se describen los datos de prueba usados en la experimentación y que se usan para evaluar el desempeño de las estrategias de aceleración propuestas para DET-ABE.

6.2.1.1. Niveles de seguridad y tipo de curva elíptica

Se utilizan los tres niveles de seguridad de AES en la experimentación, 128, 192 y 256 bits, utilizando niveles de seguridad equivalentes para CP-ABE, los cuáles se consideran válidos por lo menos hasta el año 2030 (según fue definido por el NIST en [4]). La Tabla 6.2 muestra los tamaños de llave equivalentes para AES y CP-ABE, según su tipo de curva.

Tabla 6.2: Tamaños de llaves recomendadas por el NIST para diferentes niveles de seguridad.

Nivel de seguridad	Tamaño de llave AES (bits)	Tipo de curva (A o F)	Tamaño de llave en curvas elípticas $log_2 q^k$ (bits)	Periodo de protección
Mínimo	128	A	3072	2030-2040
Medio	192	F	7680	>2030
Alto	256	F	15360	>2030

El tipo de curva elíptica es relevante en CP-ABE. En la literatura se han usado ampliamente las curvas tipo *A*, las cuales definen un emparejamiento simétrico. Las curvas *F* se han utilizado

principalmente para niveles de seguridad mayores de 80-bits, pero éstas generan un emparejamiento asimétrico. Como se ha mostrado en [46], el tipo de emparejamiento también afecta el tiempo de procesamiento de cada una de las operaciones de CP-ABE (cifrado, descifrado y generación de la llave de descifrado).

En la implementación secuencial de DET-ABE, la curva tipo *A* resulta más costosa que la curva *F* en términos de tiempo de procesamiento cuando se usan los niveles de seguridad más altos en la etapa de cifrado. Sin embargo, el proceso de descifrado es más rápido.

En la experimentación se utilizarán tanto las curvas tipo *A* como las F, mostrando la diferencia en los tiempos de procesamiento y como estos son afectados. En aquellos experimentos donde no se indique el tipo de curva usada, se asumirá que se utilizaron las curvas tipo *F*.

6.2.1.2. Políticas de control de acceso y atributos de usuario

Existe una estrecha relación entre las políticas de acceso y el conjunto de atributos que pueden generar una llave de descifrado correcta. Los diferentes conjuntos de atributos de usuario se definen con base en la cantidad de atributos en la política de acceso. En esta experimentación se usaron 26, 52, 78, 156, 312 y 624 atributos, aunque se pueden usar conjuntos de cualquier longitud.

En este trabajo, se asume que en un escenario convencional con pocos documentos, 26 atributos pueden ser suficientes para expresar todas las políticas de control de acceso de forma individual. Sin embargo, en un escenario con colecciones grandes de documentos en donde las políticas se pueden generar de manera automática o semiautomática, puede ser necesario experimentar con conjuntos grandes de atributos.

En este trabajo se decidió estudiar el impacto que tiene usar una gran cantidad de atributos para colecciones grandes de documentos.

6.2.1.3. Archivos de prueba de tamaño variable

Un parámetro de entrada importante de DET-ABE es el tamaño de los archivos, los cuáles representan los documentos o información que se desea compartir y proteger a través del esquema DET-ABE. Si bien el nivel de seguridad impacta en los tiempos de cifrado y descifrado de los datos, a partir de ciertos archivos de tamaño moderado, el factor más importante que afecta el tiempo de cifrado y descifrado se debe a la longitud (tamaño) de los datos.

Los tamaños de los datos elegidos para los experimentos son 1MB, 20MB, 80MB, 200MB y 500MB. Estos documentos son generados de forma aleatoria y son considerados documentos de tamaño promedio a los que generalmente se suben y descargan hacia y desde la nube.

6.2.2 Datos de configuración

Las aplicaciones multihilo permiten mejorar el rendimiento al ocultar la latencia de memoria y mantener al procesador ocupado. Sin embargo, su beneficio mayor se observa cuando éstas aplicaciones se ejecutan sobre procesadores multicore, permitiendo que varios hilos se ejecuten simultáneamente. Así que un dato de configuración relevante es el número de hilos que se utilizarán para los procesos de CP-ABE. Otro factor en la configuración de la implementación tiene que ver con el número de núcleos de procesamiento físicos que existen en el procesador. En este trabajo se usa una arquitectura convencional que cuenta con 4 núcleos físicos de procesamiento CPU. Por lo tanto, en las pruebas de funcionamiento se variaron los hilos de 1 a 4.

La cantidad de bloques ejecutables en un kernel de CUDA puede ir desde 1 hasta 1024. En un caso simple, se puede ejecutar un bloque de cifrado por cada bloque del kernel, cifrando en paralelo un total de 1024 bloques de datos (16X1024 bytes). Sin embargo, en la implementación paralela realizada en esta tesis, se ejecutó un conjunto S de bloques de datos por bloque de procesamiento (como se definió en las implementaciones paralelas de AES en la capa media descritas en el Capítulo 5).

Considerando la cantidad de bytes procesados, se decidió mantener constante la cantidad de bloques del kernel en CUDA de 1024, ya que con esta configuración se obtiene la máxima ganancia de aceleración en AES paralelo, además de que los tamaños de prueba son suficientemente grandes para la ejecución paralela de conjuntos de datos de 16kB por kernel de CUDA. Esta configuración será utilizada para todos los experimentos descritos en este capítulo.

6.3 Pruebas funcionales

En primera instancia, se realizaron experimentos para realizar las pruebas funcionales del esquema DET-ABE, tanto en su versión secuencial como paralela. Cada experimento tiene un objetivo particular, debido a ello se fijaron algunos datos de entrada y se varió el resto para las diferentes configuraciones dependiendo del experimento analizado. En la Tabla 6.3 se muestran los experimentos funcionales que se realizaron, donde la columna "*Parámetro analizado*" define qué dato se varía y la columna "*Valor del parámetro*" define las instancias de prueba para cada experimento. Estas pruebas corresponden únicamente a los algoritmos de cifrado, generación de la llave de descifrado y descifrado de los cifradores AES y CP-ABE, evaluados de forma independiente.

Para las pruebas funcionales de AES se utilizaron documentos de distintos tamaños, los cuales fueron cifrados con la función `AES.Encrypt` y posteriormente descifrados con la función `AES.Decrypt`. El documento de salida del descifrado es comparado byte por byte con el documento de entrada del cifrado, solo si todos los bytes son iguales se considera que el cifrador AES funciona correctamente.

Por otro lado, las pruebas de funcionamiento del cifrador CP-ABE se realizan con la misma estrategia, cifrando y descifrando un conjunto de bytes del tamaño de una llave AES de 256 bits. El conjunto de bytes que se cifran con la función `CP-ABE.Encrypt` se compara con el que se obtiene a la salida de la función `CP-ABE.Decrypt`, de tal forma que si ambos conjuntos son idénticos, el funcionamiento del cifrador CP-ABE se considera correcto.

En CP-ABE, las pruebas de funcionalidad se realizaron a partir de la variación de los niveles de seguridad (cifrado, descifrado y generación de la llave de descifrado), la cantidad de atributos en la política de acceso (cifrado) y la cantidad de atributos de usuario (descifrado y generación de la llave de descifrado). En el cifrado, se verificó la correcta construcción de la estructura de acceso, definiendo diferentes cadenas de atributos y verificando al final la existencia de únicamente un nodo en la pila (nodo raíz). Para el descifrado, se utilizaron diferentes cantidades de atributos de usuario para cada política, de tal forma que la política se cumpliera con un número variable de atributos.

Tabla 6.3: Pruebas realizadas para la verificación del funcionamiento de los algoritmos que conforman al esquema DET-ABE.

Cifrador	Función analizada	Parámetro analizado	Valor del parámetro
CP-ABE	Encrypt KeyGen Decrypt	Nivel de seguridad	Curva *A*, Nivel de seguridad 128 bits
			curva *A*, Nivel de seguridad 192 bits
			curva *A*, Nivel de seguridad 256 bits
			curva *F*, Nivel de seguridad 128 bits
			curva *F*, Nivel de seguridad 192 bits
			curva *F*, Nivel de seguridad 256 bits
	Encrypt	Política de acceso	26 atributos
			52 atributos
			78 atributos
			156 atributos
			312 atributos
			624 atributos
	Encrypt KeyGen Decrypt	Atributos de usuario	26 atributos
			52 atributos
			78 atributos
			156 atributos
			312 atributos
			624 atributos
AES	Encrypt Decrypt	Tamaño del archivo	1 MB
			20 MB
			80 MB
			200 MB
			500 MB

6.4 Métricas y pruebas de rendimiento

En las pruebas de rendimiento se evaluó el desempeño del esquema DET-ABE de acuerdo con varias métricas. En particular, se evaluó el tiempo de procesamiento del esquema DET-ABE al variar el nivel de seguridad, los tamaños de archivo, la política de acceso y los atributos de usuario utilizados para el descifrado. Las métricas utilizadas para evaluar el desempeño del esquema DET-ABE paralelo son:

- El **tiempo de procesamiento** o **tiempo de ejecución**, el cual permite observar el crecimiento del tiempo de procesamiento en función a las configuraciones que se utilizan en los experimentos. El tiempo se usa para la medición del costo de seguridad y se compara con el tiempo de la transferencia de los datos en claro (sin seguridad).
- La **aceleración**, la cual se usa para comparar la ganancia obtenida por la implementación paralela de DET-ABE sobre la secuencial.
- El **trabajo adicional (*overhead*)**, esto es, el costo adicional que se debe al usar el esquema de seguridad DET-ABE comparado contra el no usarlo. Dicho costo, para el caso de DET-ABE, se puede evaluar desde dos puntos de vista:
  - ***Overhead* de procesamiento.** Es la diferencia entre el tiempo invertido en el cálculo de todas las funciones del esquema DET-ABE cuando se transfiere un archivo con y sin seguridad.
  - ***Overhead* de volumen.** Es la diferencia entre el espacio requerido para almacenar toda la información requerida en el sobre digital del esquema DET-ABE comparado contra el espacio requerido por la información en claro (sin el esquema de seguridad).

En los procesos de cifrado y descifrado se calculan los tiempo de procesamiento de AES (T_{AES}) y CP-ABE ($T_{CP\text{-}ABE}$), estos tiempos se calculan tanto para las versiones secuenciales (T_{AESSec} y

$T_{CP\text{-}ABESec}$) como para sus versiones paralelas (T_{AESPar} y $T_{CP\text{-}ABEPar}$). Por otra parte, los tiempos de procesamiento de DET-ABE (T_{DET}) se calcula para el cifrado (`DET-ABE.Encrypt`) y descifrado (`DET-ABE.Decrypt`) como la suma de los tiempos de procesamiento de los cifradores AES y CP-ABE, esto es, $T_{DET} = T_{AES} + T_{CP\text{-}ABE}$ corresponde al tiempo de procesamiento total del esquema DET-ABE. Este tiempo de procesamiento se calcula tanto para la versión secuencial (T_{DETSec}) como para la paralela (T_{DETPar}).

Así mismo, se mide el tiempo dedicado al proceso de transferencia T_{TranSS} y el tiempo de transferencia considerando que el volumen de la transferencia crece cuando se le asigna seguridad (T_{TranCS}). La suma del tiempo de procesamiento del esquema DET-ABE y los del proceso de transferencia generan un tiempo total (T_{Total}), el cual sirve para el cálculo del trabajo adicional absoluto tanto para el cifrado secuencial ($T_{TotalSec}$) como para el paralelo ($T_{TotalPar}$).

El *overhead* (trabajo adicional) para tiempo de procesamiento y volumen se puede definir de forma absoluta o relativa de la siguiente manera:

1. El trabajo adicional absoluto (O_{Abs}) es resultado de la diferencia de los tiempos de transferencia entre el proceso sin seguridad ($T_{TransSS}$) y el proceso con seguridad ($T_{TransCS}$) más el tiempo de procesamiento de DET-ABE (T_{DET}), de tal forma que se tiene la expresión $O_{Abs} = T_{DET} +$ ($T_{TransCS}$ - $T_{TransSS}$), usando T_{DETSec} o T_{DETPar} para calcular O_{AbsSec} o O_{AbsPar}, respectivamente. Donde ($T_{TransCS}$ - $T_{TransSS}$) usualmente es mayor que cero debido al incremento en el volumen por la información contenida en el sobre digital. Por otro lado, el trabajo adicional absoluto puede ser también calculado para la diferencia del tamaño de archivo al asignar seguridad contra el tamaño del archivo sin asegurar seguridad.

2. El trabajo adicional relativo se define como la proporción del trabajo adicional absoluto con referencia al tiempo de procesamiento total, el cual se calcula con la expresión $O_{Rel} = \frac{O_{Abs}}{T_{Total}}$, que son calculados tanto para DET-ABE secuencial (O_{RelSec}) como para su versión paralela (O_{RelPar}).

En la Figura 6.3 se muestra la interacción entre las variables descritas previamente. El inciso **a)** muestra los tiempos de transferencia con y sin seguridad, el inciso **b)** presenta la relación del *overhead* absoluto con respecto al tiempo total de DET. Por otro lado, el inciso **c)** muestra la contribución de los cifradores AES y CP-ABE en el tiempo de procesamiento de DET-ABE (sin mediciones de tiempos de transferencia), además de las variables de configuración que pueden influir en la contribución de cada parte. Así mismo, en **c)** se ilustra que el proceso de generación de las llaves de descifrado tiene dependencia únicamente del cifrador CP-ABE.

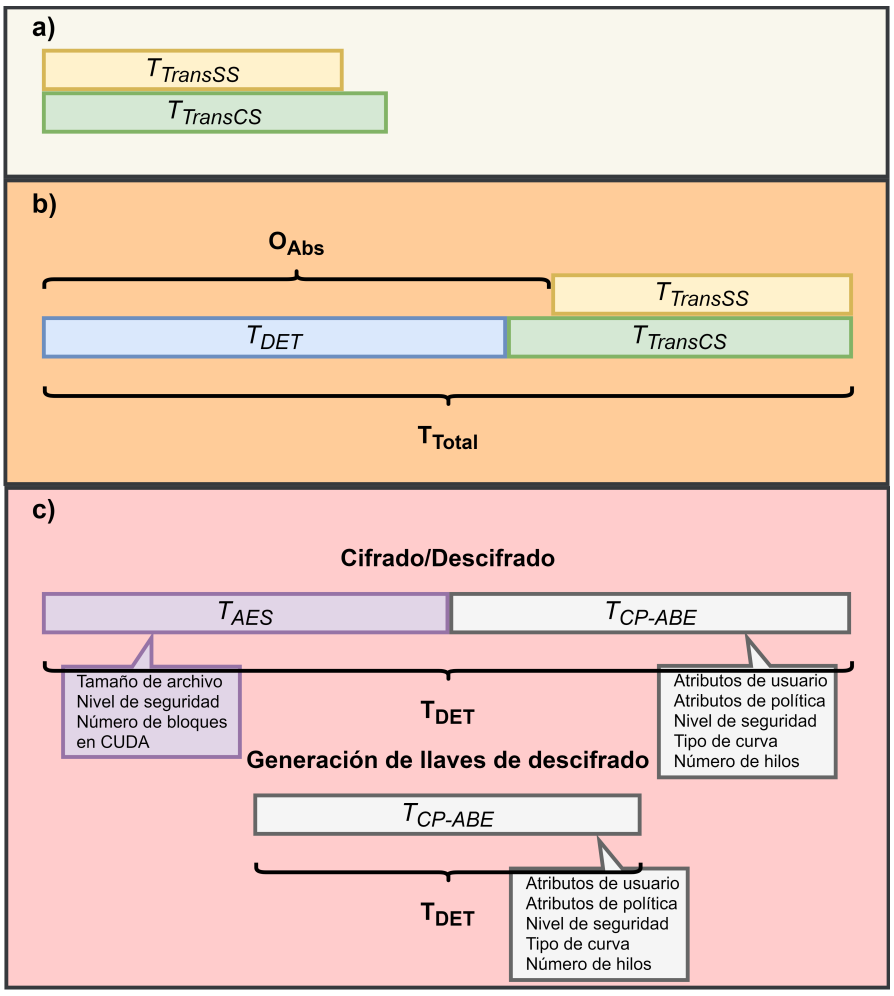


Figura 6.3: Diagrama que muestra la relación de las variables de tiempo y *overhead* en DET-ABE.

La aceleración de un proceso se calcula como la relación entre sus tiempos de ejecución secuencial y el paralelo, y define las veces que el proceso paralelo es más rápido que el secuencial. En los experimentos descritos en esta tesis, esta relación se calcula sobre tiempos de procesamiento de DET-ABE secuencial y paralelo, es decir, *Aceleración* $= \frac{T_{DETSec}}{T_{DETPar}}$.

En la Figura 6.4 se presentan cuatro regiones de interés al relacionar los dos procesos importantes de DET-ABE: AES y CP-ABE. Las unidades de los ejes horizontal y vertical están en unidades de tiempo (segundos, minutos, etc.). Las regiones I y II delimitan configuraciones que impactan a AES. En la región I se tendrían configuraciones con niveles bajos de seguridad, políticas con pocos atributos y archivos de tamaño relativamente pequeño, mientras que en la región II se representarían tamaños de archivo grandes con bajos niveles de seguridad y políticas de acceso con pocos atributos. Por otra parte, las regiones III y IV delimitan regiones para los tiempos de procesamiento con CP-ABE. La región III es en donde el tamaño de los datos es relativamente pequeño, mientras que los niveles de seguridad son altos y las políticas de acceso tienen muchos atributos. En cambio, la región IV representa tiempos para archivos grandes con niveles de seguridad altos y políticas con muchos atributos.

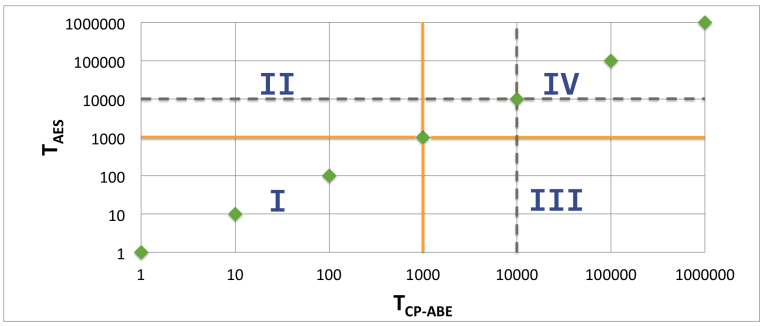


Figura 6.4: Modelo de regiones de complejidad computacional de DET-ABE desde la perspectiva del tiempo de procesamiento de AES y CP-ABE.

Para esta figura, en la diagonal principal estarían aquellos casos en donde el tiempo de procesamiento de CP-ABE es exactamente igual al tiempo de AES lo que correspondería a una

relación 50% - 50%. Aunque la versión paralela se puede aplicar a cualquier instancia, los mayores beneficios se observarían en la instancias de las regiones III y IV. Las líneas continuas mostrada en la figura delimita las 4 regiones, mientras que la línea entrecortada delimitarían las regiones factibles con la versión paralela.

La implementación paralela del esquema DET-ABE permite ampliar la región I, y por tanto poder utilizar niveles de seguridad mas grandes, políticas de acceso mas elaboradas y archivos de mayores dimensiones. Esto significa poder utilizar configuraciones de seguridad más eficientes a menores costos que en la versión secuencial de DET-ABE.

Por otra parte, también es interesante observar la relación que se puede dar entre el tiempo de transferencia y el tiempo de procesamiento de DET-ABE. En la Figura 6.5, se identifican dos regiones, A y B. La región *A* representa aquellos casos en donde los tiempos de transferencia son mayores al tiempo de procesamiento de DET-ABE, lo cual se puede dar con archivos pequeños y/o políticas simples con pocos atributos. En la región *B* aparecerían los casos en donde el tiempo de DET-ABE domina al tiempo de procesamiento debido a, por ejemplo, archivos no muy grandes, políticas con muchos atributos o niveles de seguridad altos.

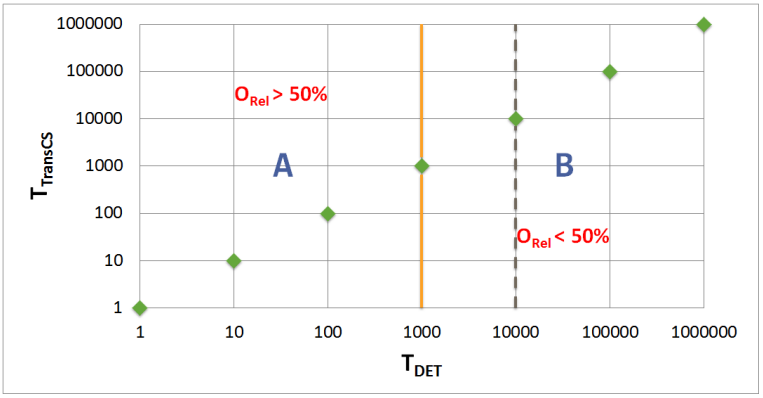


Figura 6.5: Diagrama de regiones de impacto en los tiempos de procesamiento de DET-ABE con respecto al tiempo de transferencia de los datos.

En esta Figura 6.5, la diagonal principal estarían aquellos casos en donde el tiempo de transferencia es exactamente igual al tiempo de procesamiento de DET-ABE lo que correspondería a un *overhead* relativo del 50%. La región *A* tendría un *overhead* relativo inferior al 50% y en la región *B* el *overhead* sería superior al 50%. Aunque la versión paralela se puede aplicar a cualquier instancia, los mayores beneficios se observarían en la instancias de la región *B*. La línea continua mostrada en la figura delimita la región que se puede resolver con la versión secuencial en un tiempo aceptable, mientras que la línea entrecortada delimitaría la región factible con la versión paralela.

En la Figura 6.6 se presenta la relación entre el *overhead* y la máxima aceleración posible que se puede encontrar de acuerdo con la Ley de Amdahl. Cuando el *overhead* relativo es exactamente al 50%, entonces, la máxima aceleración posible está limitado por 2X. Por tanto, para obtener aceleraciones de al menos 2X, el *overhead* relativo debe ser mayor al 50%, esto es, el tiempo de DET-ABE debe dominar al tiempo de transferencia. Esto se lograría con archivos no muy grandes, niveles de seguridad medios o altos y políticas con relativamente pocos atributos. En los experimentos se mostrarán los diferentes casos descritos previamente.

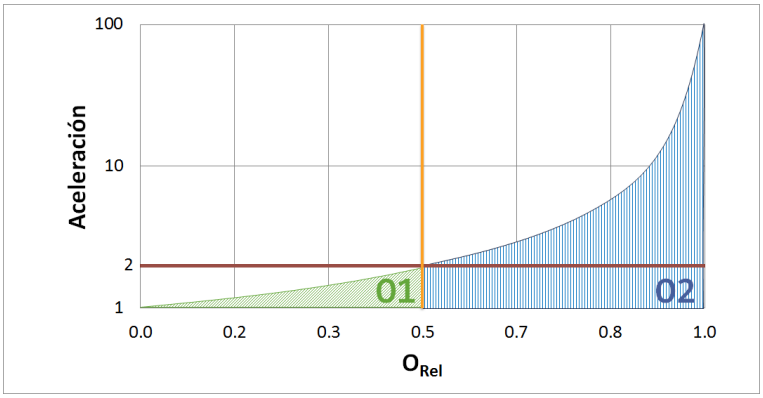


Figura 6.6: Modelo de regiones de aceleración de DET-ABE paralelo con referencia al trabajo adicional relativo.

La línea color naranja mostrada en la Figura 6.6 delimita la región en la cual se obtiene un *overhead* relativo del 50%, esto es, que el tiempo de cifrado de los datos y la aceleración son equivalentes. Por otro lado, la línea color café define una aceleración del 2X respecto a la versión

secuencial. El área de color verde delimita la máxima aceleración posible estando por debajo de un 50% del *overhead* relativo, mientras que el área de color azul define la aceleración que puede ser obtenida (teóricamente) según la ley de Amdahl.

6.5 Experimentos realizados

Los diferentes experimentos realizados para evaluar las estrategias de paralelización de DET-ABE propuestas en esta tesis se describen a continuación. Se indica el propósito del experimento y cuáles de los procesos de AES, CP-ABE y DET-ABE (cifrado, descifrado o generación de llaves) están involucrados en la evaluación.

- **Experimento 1.** Se evalúa el *overhead* que se obtiene al asignar o no seguridad a los datos al realizar una transferencia de datos a la nube. En este experimento se consideran tanto el trabajo adicional en tiempo de procesamiento y el espacio adicional requerido por el sobre digital. Se presentan los resultados para el cifrado con DET-ABE serial y paralelo en el proceso "upload" y para el descifrado en el proceso de "download".

- **Experimento 2.** Permite demostrar que los tiempos de procesamiento de DET-ABE crecen a medida que se varía el tamaño del archivo. Este experimento se realiza sobre los procesos de cifrado y descifrado de los datos de DET-ABE.

- **Experimento 3.** Permite mostrar el impacto que tiene el uso de curvas elípticas tipo *A* o F, así como el nivel de seguridad en el tiempo de procesamiento. Este experimento se realizó sobre los procesos de cifrado, generación de la llave de descifrado y descifrado de los datos de DET-ABE.

- **Experimento 4.** Se realiza con el fin de mostrar el impacto que tiene la cantidad de atributos en el proceso de cifrado de los datos con DET-ABE. En este mismo experimento se evalúa el impacto que tiene la cantidad de atributos de usuario que se utiliza para descifrar la llave AES. Este experimento se realiza sobre el proceso de generación de la llave de descifrado y el

descifrado de DET-ABE. Los atributos de la política mantienen una fuerte relación con los de usuario, de tal modo que para cada política existe una sola serie de atributos que la satisface.

- **Experimento 5.** A través de éste, se determinan las aceleraciones en el tiempo de procesamiento de DET-ABE conseguidas al variar el número de hilos que corresponden con el número de núcleos físicos que se disponen en el dispositivo emisor y receptor. Para esta serie de experimentos, se utiliza el servidor Chronos descrito en la Sección 6.1, el cual cuenta hasta con 10 hilos. Este experimento se realiza sobre los procesos de cifrado, generación de la llave de descifrado y descifrado de los datos en el cifrador CP-ABE, ya que el cifrador AES no depende de los hilos que se disponen.

- **Experimento 6.** Permite determinar la aceleración máxima obtenida en DET-ABE paralelo a nivel de esquema, donde se mide el tiempo de procesamiento tanto de cifrado y transferencia a la nube, como el de la descarga de los datos, la generación de la llave de descifrado y el proceso de descifrado sobre el prototipo experimental. Para este experimento, el dispositivo A es quien realiza todos los procesos de seguridad y transferencia de los datos.

De acuerdo con el teorema de límite central, cada experimento se repitió al menos 31 veces para obtener así una distribución normal de las variables evaluadas [24].

En la Tabla 6.4 se muestran de manera resumida los datos de entrada involucrados en cada experimento, así como los procesos del esquema DET-ABE que fueron evaluados. Los procesos no evaluados fueron considerados como irrelevantes para el experimento a desarrollar. La justificación en cada caso se presenta junto con los resultados de cada experimento, en las secciones subsecuentes.

6.5.1 Experimento 1

En este experimento se determina el trabajo adicional requerido para transferir los datos del cliente a un servidor de almacenamiento en la nube con seguridad, utilizando como referencia el tiempo de transferencia de los datos sin seguridad. En particular, se muestra como crece el tiempo

Tabla 6.4: Configuraciones para las pruebas de rendimiento de los algoritmos que conforman al esquema DET-ABE.

Experimento	Dato de entrada del esquema DET-ABE	Cifrado	Descifrado	Generación de la llave de descifrado
1	Tamaño de archivo (1, 20, 80, 200 o 500 MB)	Si	Si	No
2	Tamaño de archivo (1, 20, 80, 200 o 500 MB)	Si	Si	No
3	Tipo de curva (A o F) Nivel de seguridad (128, 192 o 256 bits)	Si	Si	Si
4	Atributos de política (26, 52, 78, 156, 312 o 624 atributos)	Si	Si	Si
5	Número de hilos (1, 2, 3, 4, 5, 6, 7 o 8 hilos)	Si	No	No
6	Tamaño de archivo (1, 20, 80, 200 o 500 MB)	Si	Si	Si

de cifrado y descifrado de DET-ABE al variar el tamaño de los datos, de tal forma que el tiempo de procesamiento en la transferencia crece a la par con el proceso de cifrado y con éstos el trabajo adicional.

En este experimento, se utiliza el esquema que se muestra en la Figura 6.7, en **a)** se establecen los parámetros fijos en la etapa de cifrado de datos, posteriormente el cifrado de la llave y por último se realiza la transferencia de los datos. El inciso **b)** muestra las mismas características para el proceso de descifrado. Sin embargo, en el descifrado se realiza primero la descarga de los datos (transferencia), posteriormente se descifra la llave y por último se descifran los datos.

El proceso de generación de llaves de descifrado no fue evaluado en el experimento 1, ya que éste depende únicamente del nivel de seguridad, la cantidad de atributos de usuario y el número de hilos con que se paralelizan los algoritmos de CP-ABE. De este modo, los datos fijos para el experimento 1 son el nivel de seguridad en 192 bits, el uso de una curva tipo *F*, una política de acceso con 624 atributos y la ejecución del esquema CP-ABE se paraleliza con 4 hilos. Los datos de entrada son los archivos de tamaño 1, 20, 80, 200 y 500 MB, referidos en lo sucesivo en las gráficas y resultados obtenidos como **TA1**, **TA20**, **TA80**, **TA200**, y **TA500**, respectivamente.

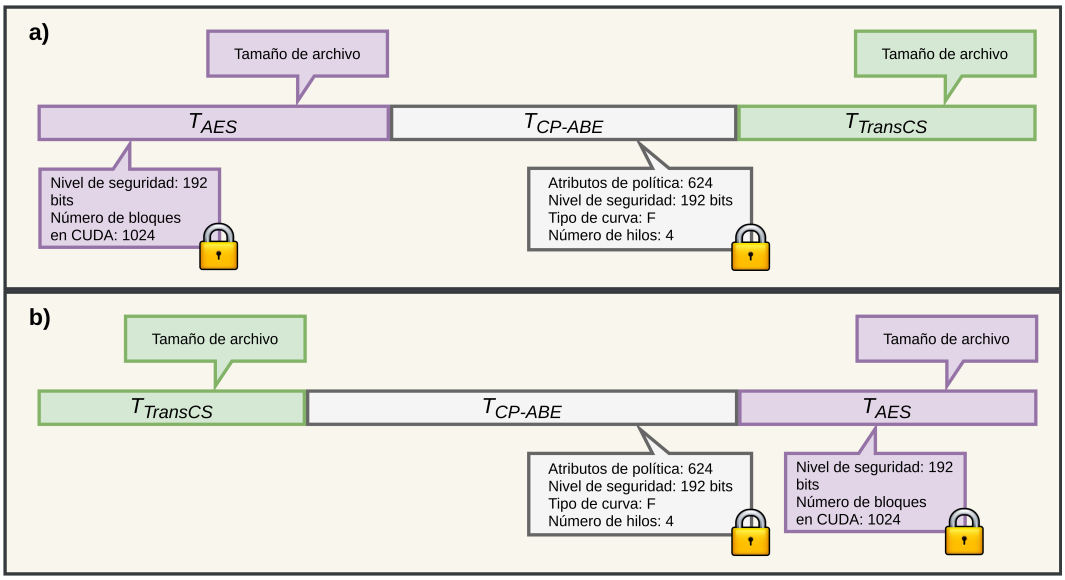


Figura 6.7: Datos de prueba para **a)** el cifrado y **b)** el descifrado de datos con DET-ABE en el experimento 1.

Debido a que al asegurar los datos existe un costo adicional en el tamaño de los datos al generar el sobre digital (concatenación de los datos cifrados a transferir, componentes de la llave cifrada y otros componentes necesarios para el descifrado), se realizó un análisis de este crecimiento de volumen (*overhead* de volumen). En la Tabla 6.5 se muestra el crecimiento en volumen cuando los datos son encapsulados en el sobre digital, donde se puede ver que la cantidad de espacio adicional que se requiere para el sobre digital es relativamente bajo (siendo de 145 kB el crecimiento máximo reportado, el cual corresponde a los datos de 500MB), de tal forma que su transferencia no implica un crecimiento significativo en el tiempo de procesamiento.

Tabla 6.5: Incremento en el volumen de los datos al asignar seguridad a éstos.

	Sin seguridad	**Con seguridad**	***Overhead***	
Tamaño (MB)	**Tamaño del paquete (Bytes)**	**Tamaño del paquete (Bytes)**	O_{Abs} **(kB)**	O_{Rel} **(%)**
1	1048576	1113381	63.286	5.820
20	20971520	21039318	66.209	0.322
80	83886080	83964352	76.438	0.093
200	209715200	209813722	96.213	0.046
500	524288000	524437437	145.935	0.028

Debido a este ligero crecimiento en la cantidad de los datos al asignar seguridad, los tiempos de transferencia del sobre digital también crecen ligeramente comparados con la transferencia sin

seguridad. Entre más pequeño es el archivo a transferir, la proporción del espacio adicional requerido por el sobre digital es mayor, como se puede apreciar en la Tabla 6.5 en la columna correspondiente al *overhead* relativo (O_{Abs}) en términos de espacio.

6.5.1.1. Cifrado

En la Tabla 6.6 se describen los tiempos de procesamiento de la etapa de cifrado de los datos utilizando DET-ABE secuencial y paralelo, en ambos casos se puede ver una tendencia creciente respecto al tamaño de los datos. También se puede ver como el crecimiento en los tiempos de transferencia son muy similares, lo que demuestra que los tiempos de transferencia no son afectados significativamente al asignar seguridad. Así mismo, en una comparación entre los tiempos de transferencia con la versión secuencial y paralela de DET-ABE no se puede apreciar una diferencia notoria, lo cual se debe a que la estrategia de paralelización no provoca ninguna modificación en el volumen de los datos y, por tanto, en su tiempo de transferencia.

Como se puede observar en la Tabla 6.6 y el inciso **b)** de la Figura 6.8, para todos los casos el tiempo de procesamiento secuencial de DET-ABE domina a la transferencia, de tal modo que corresponden a la región **B**.

Tabla 6.6: Tiempo de procesamiento total en el almacenamiento en la nube con y sin seguridad (cifrado).

	Sin seguridad	**Con seguridad**					
		Secuencial			**Paralelo**		
Tamaño (MB)	$T_{TransSS}$ **(Seg)**	$T_{TotalSec}$ **(Seg)**	T_{DETSec} **(Seg)**	$T_{TransCS}$ **(Seg)**	$T_{TotalPar}$ **(Seg)**	T_{DETPar} **(Seg)**	$T_{TransCS}$ **(Seg)**
1	0.355	11.909	11.536	0.373	5.096	4.721	0.375
20	6.431	26.281	19.828	6.453	13.743	7.292	6.451
80	26.545	71.704	45.148	26.566	39.265	12.698	26.567
200	65.295	166.544	101.231	65.313	90.292	24.982	65.310
500	163.421	405.963	242.519	163.444	218.922	55.478	163.444

Por otro lado, en la Tabla 6.7 se muestra el trabajo adicional absoluto y relativo sobre los tiempos de procesamiento totales medidos tanto para el proceso de transferencia con seguridad utilizando DET-ABE secuencial como para los medidos con su versión paralela. En estos casos, se puede ver

como el trabajo adicional relativo paralelo es menor al secuencial y que, conforme los datos son más grandes menor es el costo de la asignación de seguridad por parte del cifrador CP-ABE, el cual, para este experimento utiliza la misma configuración en todos los casos.

Tabla 6.7: Costo computacional adicional (absoluto y relativo) al asignar seguridad a los datos (cifrado).

	***Overhead* procesamiento**			
	Secuencial		**Paralelo**	
Tamaño (MB)	O_{AbsSec} **(Seg)**	O_{RelSec} **(%)**	O_{AbsPar} **(Seg)**	O_{RelPar} **(%)**
1	11.554	97.019	4.741	93.033
20	19.850	75.529	7.312	53.205
80	45.159	62.979	12.720	32.395
200	101.249	60.794	24.997	27.684
500	242.542	59.744	55.501	25.351

En la Figura 6.8, inciso **a)** se muestra una comparación entre los tiempos de procesamiento totales para los casos de transferencia de datos con y sin seguridad, donde la segunda y tercera barra para cada tamaño de archivo corresponden a los tiempos de procesamiento (incluido el tiempo de transferencia) usando el esquema DET-ABE secuencial y paralelo, respectivamente. La primera barra corresponde únicamente al tiempo de transferencia sin seguridad. Las gráficas de tiempo de procesamiento en esta figura están en escala logarítmica, y en éstas se puede ver que el proceso de transferencia sin seguridad siempre es inferior al que aporta seguridad, sin embargo, los tiempos de procesamiento son reducidos de manera significativa con la versión de DET-ABE paralela, de manera que entre más grande es el documento, menor es el costo relativo de la asignación de seguridad.

En el inciso **b)** de la misma figura se muestra una comparación entre los tiempos de transferencia y los de cifrado, de tal forma que se definen puntos que representan el tiempo de procesamiento total secuencial y paralelo, el cual equivale a la suma de los valores del eje x para el cifrado y el eje y para la transferencia. Para cada tamaño del archivo, el tiempo de DET-ABE paralelo se observa desplazado hacia la izquierda con respecto a DET-ABE secuencial. La magnitud del desplazamiento indica la ganancia obtenida con el esquema paralelo.

6.5.1.2. Descifrado

En la etapa de descifrado, los tiempos de procesamiento para recuperar la llave AES tienen un alto costo computacional, ya que se utilizaron políticas de acceso muy grandes, un alto nivel de seguridad y el uso de curvas tipo F (el impacto del nivel de seguridad y el número de atributos se mostrará más claramente en los experimentos 3 y 4 respectivamente). Debido a esto, los tiempos de procesamiento totales son mayores a los del cifrado.

En la experimentación con el descifrado de los datos se obtuvieron los resultados que se muestran en la Tabla 6.8, en donde se presentan los tiempos de procesamiento totales para la transferencia con y sin seguridad. Los tiempos de descifrado resultan muy costosos, haciendo elevar considerablemente los tiempos de procesamiento totales. Por otro lado, los tiempos de transferencia no varían considerablemente con respecto a los vistos en el cifrado, ya que los tamaños de los documentos son iguales para cada instancia y no generan costos de procesamiento adicionales.

Tabla 6.8: Tiempo de procesamiento total para recuperar documentos en la nube con y sin seguridad (descifrado).

	Sin seguridad	Con seguridad					
		Secuencial			Paralelo		
Tamaño (MB)	$T_{TransSS}$ (Seg)	$T_{TotalSec}$ (Seg)	T_{DETSec} (Seg)	$T_{TransCS}$ (Seg)	$T_{TotalPar}$ (Seg)	T_{DETPar} (Seg)	$T_{TransCS}$ (Seg)
1	0.357	158.909	158.531	0.378	78.751	78.374	0.377
20	6.421	172.579	166.116	6.463	87.800	81.341	6.459
80	26.534	218.206	191.644	26.562	115.893	89.336	26.557
200	65.195	313.029	247.800	65.229	168.353	103.132	65.221
500	163.537	551.261	387.644	163.617	297.123	133.495	163.628

En la Figura 6.8 inciso **c)** se puede ver que el crecimiento del tiempo de procesamiento en la transferencia al realizar el descifrado crece significativamente. Sin embargo, comparando los resultados mostrados en las barras dos y tres de esta gráfica se puede observar que existe una reducción del tiempo de procesamiento entre los procesos secuencial y paralelo para el descifrado de los datos.

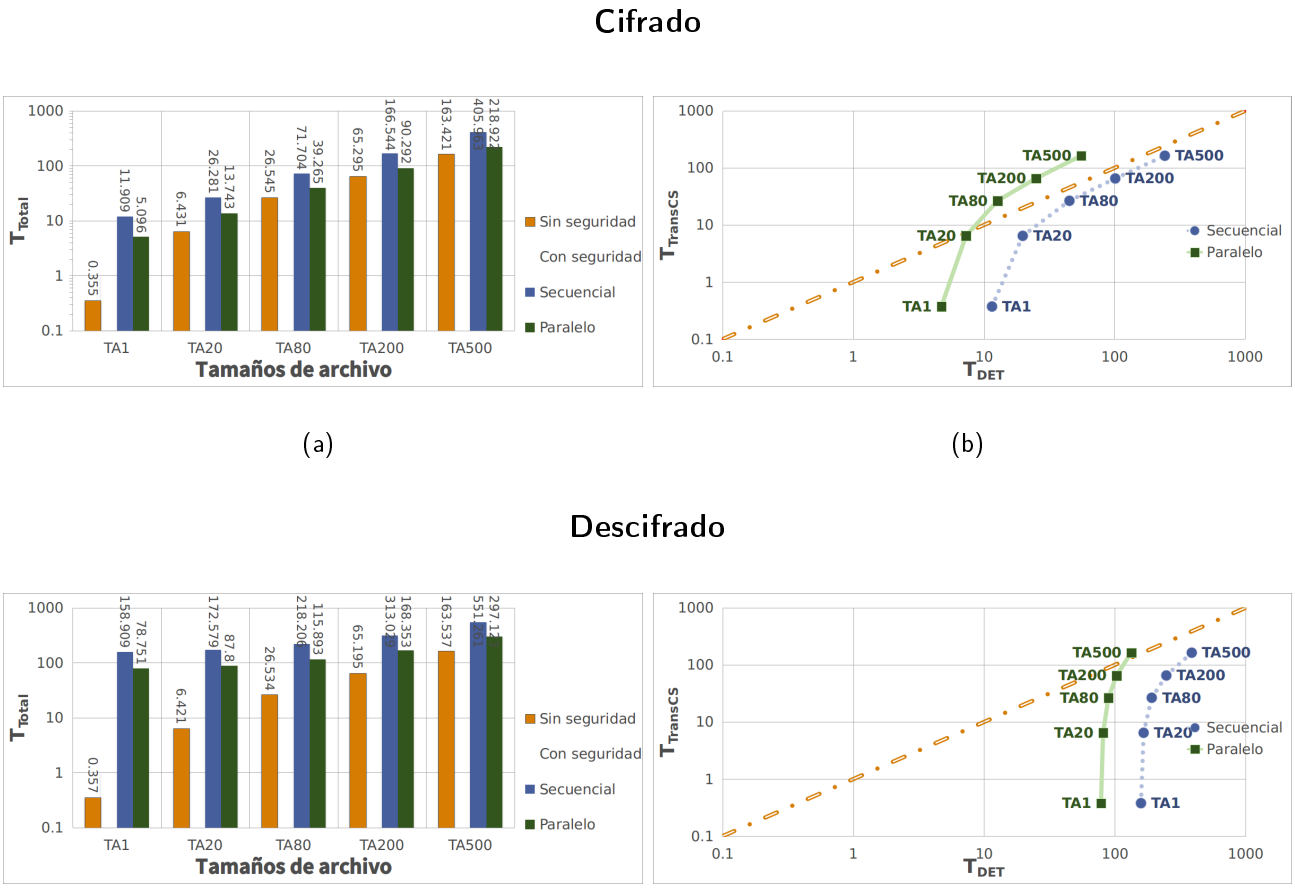


Figura 6.8: Gráficas de resultados obtenidos en el experimento 1.

En la Figura 6.8, **a)** muestra los tiempos de procesamiento en el cifrado con y sin seguridad (tanto para el esquema DET-ABE secuencial como para el paralelo), **b)** compara los tiempos de procesamiento de la transferencia y cifrado con DET-ABE, **c)** muestra los tiempos de procesamiento en el descifrado con y sin seguridad (utilizando la estrategia secuencial y paralela) y **d)** compara los tiempos de procesamiento de la transferencia y descifrado con DET-ABE. En estas gráficas se definen a **TA1**, **TA20**, **TA80**, **TA200** y **TA500** como los cinco tamaños de archivo utilizados en el experimento (1, 20, 80, 200 y 500 MB).

Por otro lado, en la Tabla 6.9 se muestra como al igual que en la etapa de cifrado, el trabajo adicional relativo decrece conforme el tamaño de los datos aumenta. Sin embargo, en el descifrado el *overhead* relativo es significativamente más grande que en el cifrado.

Así mismo, como en el caso del proceso de cifrado, y como se muestra en la Figura 6.8 inciso **d)**, se puede ver que todas las instancias en el descifrado pertenecen a la región *B*, lo cual implica que el tiempo de procesamiento de DET-ABE domina al de transferencia. Al igual que en el cifrado, los tiempo del descifrado paralelo se desplazan hacia la izquierda con respecto a DET-ABE secuencial.

Tabla 6.9: Costo computacional al asignar seguridad a los datos (descifrado).

	***Overhead* procesamiento**			
	Secuencial		**Paralelo**	
Tamaño (MB)	O_{AbsSec} **(Seg)**	O_{RelSec} **(%)**	O_{AbsPar} **(Seg)**	O_{RelPar} **(%)**
1	158.552	99.775	78.394	99.546
20	166.158	96.279	81.379	92.686
80	191.672	87.839	89.359	77.104
200	247.834	79.172	103.158	61.274
500	387.724	70.334	133.586	44.959

6.5.2 Experimento 2

El objetivo de este experimento es evaluar cómo el tiempo de procesamiento de DET-ABE varía con base en el tamaño de los datos que se cifran. En éste se determinan los tiempos de procesamiento y la aceleraciones resultantes de la paralelización del esquema DET-ABE. Así mismo, se muestran tanto los tiempos de procesamiento de los cifradores AES como CP-ABE, definiendo así la carga computacional de cada uno dentro de los procesos de DET-ABE.

Para este experimento, se utiliza un esquema de procesamiento diferente al del experimento 1, mostrado en la Figura 6.9. En a) se establecen los parámetros fijos del experimento 2 en la etapa de cifrado (de los datos y de la llave). El inciso **b)** muestra las mismas características para el proceso de descifrado, donde primero se descifra la llave AES y posteriormente los datos.

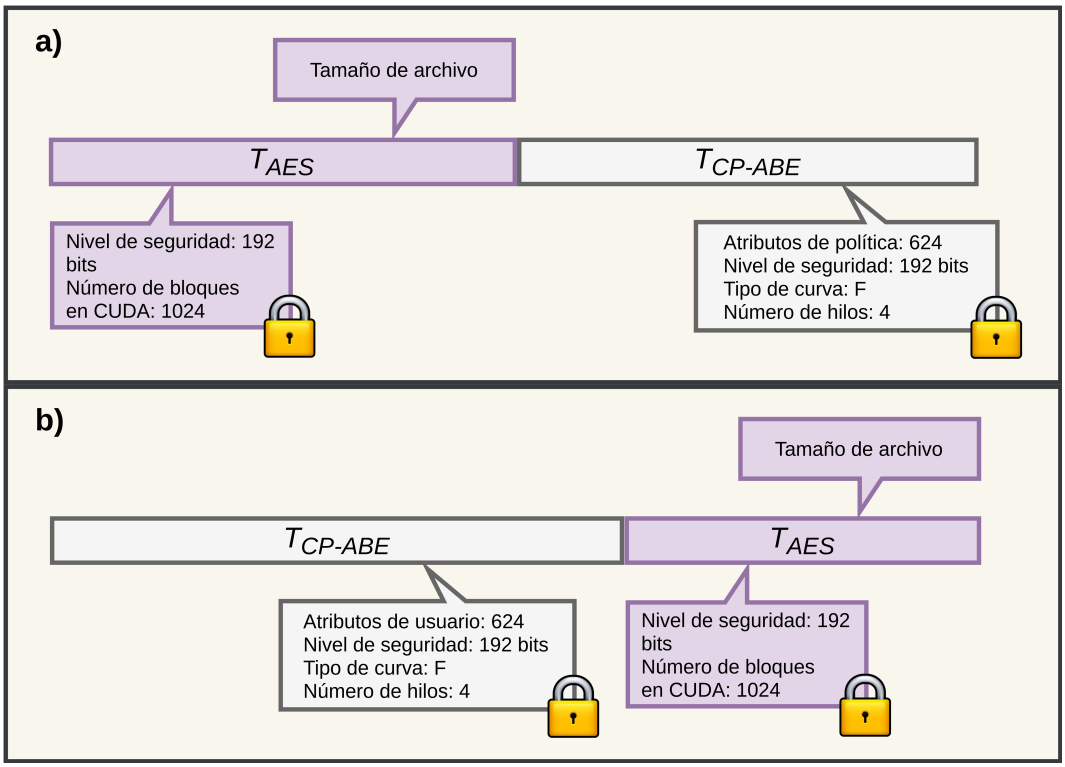


Figura 6.9: Datos de prueba y de configuración de AES y CP-ABE para **a)** el cifrado y **b)** el descifrado de datos con DET-ABE en el experimento 2.

Al igual que en el experimento 1, el proceso de generación de llaves de descifrado no fue considerado, ya que no es afectado por la variación del tamaño de los archivos cifrados. En este experimento los datos fijos son el nivel de seguridad en 192 bits, el uso de una curva tipo *F*, una política de acceso con 624 atributos y la ejecución paralela del cifrador CP-ABE se realiza con 4 hilos de procesamiento. Nuevamente se utilizan archivos de 1, 20, 80, 200 y 500 MB.

6.5.2.1. Cifrado

En la Tabla 6.10 se describen los tiempos de procesamiento de la etapa de cifrado de los datos utilizando DET-ABE secuencial y paralelo, en ambos casos se nota que el tiempo de ejecución se incrementa con el tamaño de los datos. En la misma, se puede ver como los tiempos de procesamiento de CP-ABE son muy similares, actuando así como una constante en el proceso. Debido a lo anterior,

las aceleraciones obtenidas en DET-ABE paralelo varían con base en la reducción del tiempo de ejecución de AES paralelo con respecto a su versión secuencial. Por lo tanto, como se puede ver en la columna correspondiente a las aceleraciones, a mayor tamaño de los datos, mayor es la aceleración conseguida.

En este experimento se consiguió incrementar considerablemente la aceleración obtenida con la estrategia de paralelización descrita en el Capítulo 5, como se muestran en la Tabla 6.10 y en el inciso **b)** de la Figura 6.10. Esto se debe a que a mayor tamaño del archivo, mayor es el aprovechamiento que se obtiene en el manejo de memoria y comunicaciones entre el CPU y los GPUs. En esta tabla se nota que para todos los tamaños de archivo, las aceleraciones son mayores a 2X.

Tabla 6.10: Resultados de rendimiento y acaleración de DET-ABE para distintos tamaños de los datos en la etapa de cifrado.

	Secuencial			Paralelo			
Tamaño del archivo (MB)	T_{DETSec} **(Seg)**	$T_{CP\text{-}ABESec}$ **(Seg)**	T_{AESSec} **(Seg)**	T_{DETPar} **(Seg)**	$T_{CP\text{-}ABEPar}$ **(Seg)**	T_{AESPar} **(Seg)**	**Aceleración (×)**
1	11.536	11.087	0.449	4.721	4.337	0.384	2.443
20	19.828	11.102	8.726	7.292	4.343	2.949	2.719
80	45.148	11.075	34.073	12.698	4.351	8.347	3.555
200	101.231	11.092	90.139	24.982	4.332	20.650	4.052
500	242.519	11.069	231.450	55.478	4.340	51.138	4.371

En la Figura 6.10, inciso **a)** se muestra una comparación entre los tiempos de AES y CP-ABE, de tal forma que se definen puntos que representan el tiempo de cifrado con DET-ABE secuencial y paralelo, el cual es equivale a la suma de los valores del eje x para CP-ABE y el eje y para AES. Para cada tamaño del archivo, los tiempos de AES paralelo se muestran desplazados hacia abajo (eje y), ya que se reducen los tiempos de procesamiento de la versión paralela. La magnitud del desplazamiento indica la ganancia obtenida con el esquema paralelo. Dichos desplazamientos permiten pasar de la región III a la I en instancias con tamaños de archivo menores a 80 MB. Por otro lado, las instancias mayores o iguales a 80 MB en DET-ABE paralelo producen un desplazamiento de la región III a la II.

6.5.2.2. Descifrado

El proceso de descifrado tiene un alto costo computacional debido a la configuración con que se descifra la llave AES. En la Tabla 6.11 se describen los tiempos de procesamiento de la etapa de descifrado de los datos utilizando DET-ABE secuencial y paralelo. En esta se muestra que a pesar de que el proceso de cifrado de la llave AES demanda un alto tiempo de procesamiento, el tiempo de descifrado de los datos eventualmente llega a superar este tiempo de procesamiento. Debido a lo anterior, se puede considerar que al tener un archivo suficientemente grande (como en el caso de 500 MB), teniendo un mayor tiempo de descifrado AES que de CP-ABE, la aceleración máxima obtenida incrementa.

Como se puede ver en la Tabla 6.11 y en el inciso **d)** de la Figura 6.10, las aceleraciones obtenidas en el descifrado son inferiores a las obtenidas en el cifrado. No obstante, en todas las instancias de prueba se logró una aceleración superior a 2X y el comportamiento en el incremento de éstas se mantiene. De esta forma, se considera que tanto para el cifrado como el descifrado en AES paralelo, se obtiene un mayor aprovechamiento de memoria en los GPUs entre más grande sea el archivo procesado.

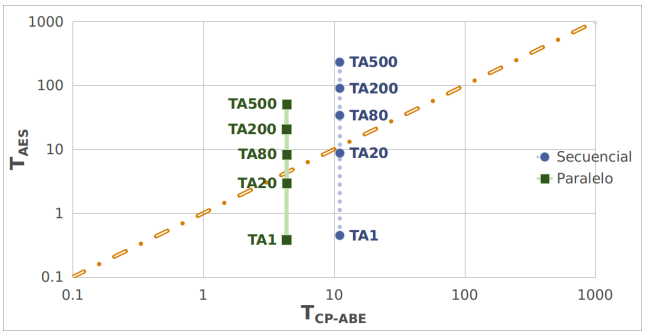
Tabla 6.11: Pruebas de rendimiento y determinación de la aceleración de DET-ABE para distintos tamaños de archivos en la etapa de descifrado.

	Secuencial			Paralelo			
Tamaño del archivo (MB)	T_{DETSec} **(Seg)**	$T_{CP\text{-}ABESec}$ **(Seg)**	T_{AESSec} **(Seg)**	T_{DETPar} **(Seg)**	$T_{CP\text{-}ABEPar}$ **(Seg)**	T_{AESPar} **(Seg)**	**Aceleración (×)**
1	158.531	158.078	0.453	78.374	78.066	0.308	2.022
20	166.116	158.125	7.991	81.341	78.058	3.283	2.042
80	191.644	158.103	33.541	89.336	78.072	11.264	2.145
200	247.8	158.091	89.709	103.132	78.051	25.081	2.402
500	387.644	158.065	229.579	133.495	78.062	55.433	2.903

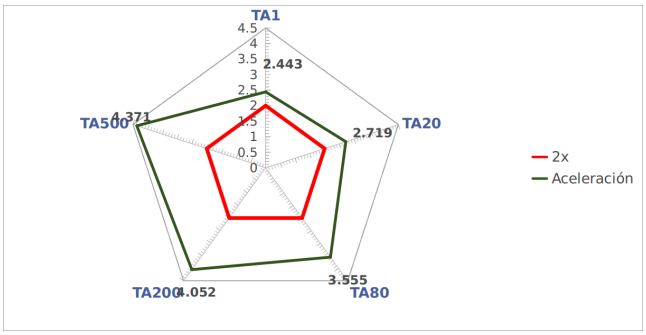
Al igual que en el cifrado, la Figura 6.10 inciso **c)** muestra una comparación entre los tiempos de AES (eje y) y CP-ABE (eje x), de tal forma que la suma de ambos equivale al tiempo de procesamiento total de DET-ABE. En esta figura se muestra que las instancias utilizadas en este experimento producen un comportamiento similar a las del cifrado, de tal forma que al incrementar el

tamaño del archivo, se pasa de la región III a la IV. Nótese que para las instancias a partir de TA80, en el cifrado CP-ABE domina a AES, mientras que para el descifrado en casi todas las instancias CP-ABE domina a AES.

Cifrado

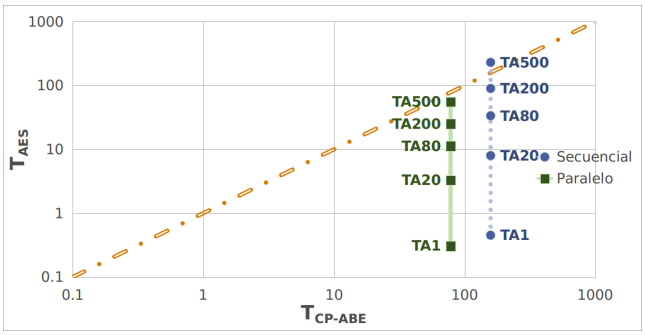


(a)

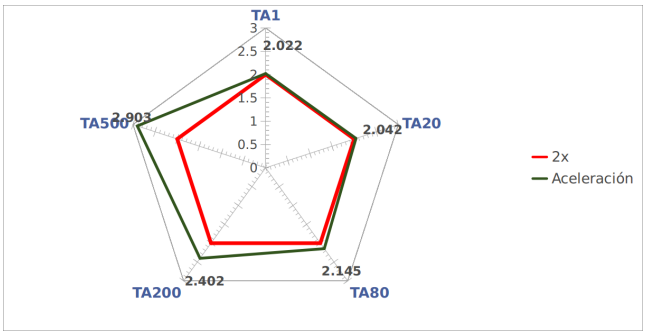


(b)

Descifrado



(c)



(d)

Figura 6.10: Gráficas de resultados obtenidos en el experimento 2.

En la Figura 6.10, **a)** y **c)** se presenta la comparación entre los tiempos de procesamiento de AES y CP-ABE en el cifrado y descifrado (respectivamente), mientras que **b)** y **d)** representan las aceleraciones medidas a partir de los tiempos de procesamientos de **a)** y **c)** (respectivamente). En estas gráficas, los puntos unidos con líneas punteadas representan al esquema DET-ABE secuencial, mientras que los cuadrados unidos con una línea continua representa a DET-ABE paralelo.

6.5.3 Experimento 3

Este experimento tiene como propósito determinar el impacto que las curvas tipo *A* o F tienen sobre los tiempos de procesamiento en el esquema DET-ABE en sus versiones secuencial y paralelo. Además de esto, con este experimento se determina el incremento en los tiempos de procesamiento debido a la variación del nivel de seguridad.

Los procesos que son evaluados en este experimento son el cifrado, generación de la llave de descifrado y descifrado con DET-ABE. De este modo, los datos fijos para este experimento son un tamaño de los datos de 20MB, una política de acceso con 624 atributos y la ejecución del esquema CP-ABE se paraleliza con 4 hilos. Los datos de entrada son niveles de seguridad de 128, 192 y 256 bits (denominados en las gráficas y resultados obtenidos en el experimento como **NS128**, **NS192** y **NS256**, respectivamente). En la Figura 6.11 se muestra el esquema bajo el cual se realiza el experimento 3 para los tres procesos que se evalúan, cifrado, descifrado y generación de llaves. Se presentan los parámteros fijos y las variables del experimento. El cifrado requiere tanto el cifrado de los datos como el de la llave AES, inciso **a)**. Para el descifrado, la llave AES se descifra primero y posteriormente los datos (inciso **b**). Por último se genera la llave con el proceso `KeyGen` con CP-ABE (inciso **c**).

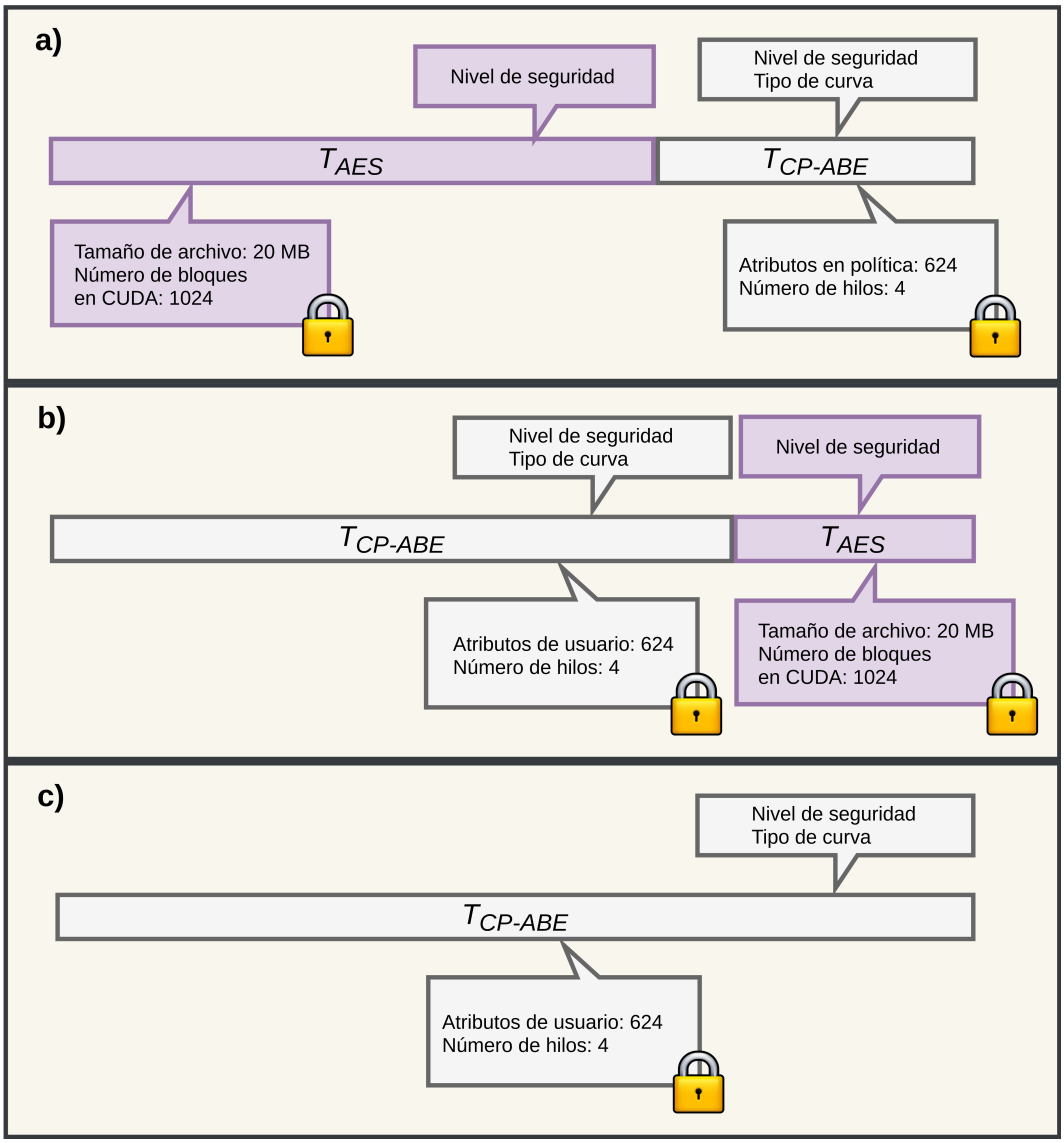


Figura 6.11: Datos de prueba y de configuración de AES y CP-ABE para **a)** el cifrado, **b)** el descifrado de datos y **c)** la generación de llaves de descifrado con DET-ABE en el experimento 3.

6.5.3.1. Cifrado

En la Tabla 6.12 se describen los tiempos de procesamiento de la etapa de cifrado de los datos utilizando DET-ABE secuencial y paralelo al variar los niveles de seguridad utilizando curvas tipo *A*. En este experimento se puede ver que los tiempos de procesamiento crecen considerablemente a medida que incrementa el nivel de seguridad. En la misma, se puede ver como los tiempos de procesamiento de AES son muy similares para todas las instancias, actuando así como una constante en el proceso. Debido a lo anterior, las aceleraciones obtenidas en DET-ABE paralelo varían con base en la reducción del tiempo de ejecución de CP-ABE paralelo con respecto a su versión secuencial. Como se puede ver en la columna referente a aceleraciones, para todos los niveles de seguridad se obtuvo una aceleración mayor a 2X.

Tabla 6.12: Resultados de rendimiento de DET-ABE variando el nivel de seguridad en la etapa de cifrado utilizando curvas tipo *A*.

		Secuencial			Paralelo			
Curva	**Nivel de seguridad (Seg)**	T_{DETSec} **(Seg)**	$T_{CP\text{-}ABESec}$ **(seg)**	T_{AESSec} **(Seg)**	T_{DETPar} **(Seg)**	$T_{CP\text{-}ABEPar}$ **(Seg)**	T_{AESPar} **(Seg)**	**Aceleración (×)**
A	128	22.624	12.475	10.139	7.171	4.824	2.417	3.154
	192	265.353	255.539	10.142	91.063	88.639	2.424	2.913
	256	1343.124	1332.966	10.158	375.310	372.851	2.459	3.602

Por otro lado, en la Tabla 6.13 se describen los tiempos de procesamiento del cifrado con DET-ABE secuencial y paralelo usando curvas tipo *F*. En ésta se puede observar que el tiempo de cifrado es considerablemente menor que cuando se usan curvas tipo *A*, de tal modo que el proceso de cifrado de la llave tiene un menor impacto sobre el tiempo de procesamiento total de DET-ABE. Sin embargo, los tiempos de CP-ABE siguen dominando a los de AES.

En este experimento la aceleración máxima obtenida con la estrategia de paralelización implementada es de 3.976X. Como se puede ver en la tabla, para todos los niveles de seguridad las aceleraciones son mayores a 2X.

Tabla 6.13: Resultados de rendimiento de DET-ABE variando el nivel de seguridad en la etapa de cifrado utilizando curvas tipo *F*.

		Secuencial			Paralelo			
Curva	**Nivel de seguridad (Seg)**	T_{DETSec} **(Seg)**	$T_{CP\text{-}ABESec}$ **(seg)**	T_{AESSec} **(Seg)**	T_{DETPar} **(Seg)**	$T_{CP\text{-}ABEPar}$ **(Seg)**	T_{AESPar} **(Seg)**	**Aceleración (×)**
F	128	10.372	0.235	10.137	2.608	0.193	2.415	3.976
	192	20.142	10.001	10.141	8.046	5.619	2.427	2.503
	256	58.441	48.280	10.161	17.436	14.974	2.462	3.351

6.5.3.2. Descifrado

En el descifrado con curvas elípticas tipo *A*, el crecimiento de los tiempos de ejecución son menores que los del cifrado, lo cual se puede observar en la Tabla 6.14. En esta se puede ver que el descifrado de la llave tiene un alto impacto en el tiempo de procesamiento del esquema DET-ABE.

Tabla 6.14: Resultados de rendimiento de DET-ABE variando el nivel de seguridad en la etapa de descifrado utilizando curvas tipo *A*.

		Secuencial			Paralelo			
Curva	**Nivel de seguridad (Seg)**	T_{DETSec} **(Seg)**	$T_{CP\text{-}ABESec}$ **(seg)**	T_{AESSec} **(Seg)**	T_{DETPar} **(Seg)**	$T_{CP\text{-}ABEPar}$ **(Seg)**	T_{AESPar} **(Seg)**	**Aceleración (×)**
A	128	12.991	3.205	9.786	3.763	1.518	2.245	3.452
	192	96.337	86.544	9.793	38.455	35.996	2.459	2.505
	256	548.892	539.082	9.810	173.044	170.581	2.463	3.171

Por otro lado, como se muestra en la Tabla 6.15 al utilizar curvas elípticas tipo *F* el crecimiento de los tiempos de ejecución son mayores que los del cifrado (contrario al uso de curvas tipo *A*). Sin embargo, la diferencia de tiempos de procesamiento no es tan notable como en el caso de las curvas tipo *A*. Los tiempos de descifrado en las Tablas 6.14 y 6.15 muestran que CP-ABE tiene un alto impacto en el tiempo de procesamiento total de DET-ABE. Tanto para curvas *A* como *F*, la aceleración obtenida del proceso de descifrado de DET-ABE es superior a 2X.

Tabla 6.15: Resultados de rendimiento de DET-ABE variando el nivel de seguridad en la etapa de descifrado utilizando curvas tipo *F*.

		Secuencial			Paralelo			
Curva	**Nivel de seguridad (Seg)**	T_{DETSec} **(Seg)**	$T_{CP\text{-}ABESec}$ **(seg)**	T_{AESSec} **(Seg)**	T_{DETPar} **(Seg)**	$T_{CP\text{-}ABEPar}$ **(Seg)**	T_{AESPar} **(Seg)**	**Aceleración (×)**
F	128	40.711	30.923	9.788	18.496	16.255	2.241	2.201
	192	166.573	156.783	9.790	76.661	74.199	2.462	2.244
	256	780.420	770.608	9.812	279.898	277.438	2.460	2.788

6.5.3.3. Generación de la llave de descifrado

Al contrario de los procesos de cifrado y descifrado de DET-ABE, la generación de las llaves de descifrado no tiene ninguna dependencia del cifrador AES, de tal modo que los tiempos de procesamiento para este proceso dependen únicamente del algoritmo `KeyGen` del cifrador CP-ABE. En la Tabla 6.16 se muestran los tiempos de procesamiento para la generación de la llave de descifrado de DET-ABE al variar los niveles de seguridad utilizando curvas tipo *A*.

El comportamiento del tiempo de procesamiento es similar al del proceso de cifrado, teniendo incrementos de tiempos de ejecución relativamente grandes al utilizar mayores niveles de seguridad. En la Tabla 6.16 se observan también las aceleraciones obtenidas con la versión paralela de `CP-ABE.KeyGen` con respecto a la secuencial.

Tabla 6.16: Resultados de rendimiento de DET-ABE variando el nivel de seguridad en la etapa de generación de la llave de descifrado utilizando curvas tipo *A*.

Curva	**Nivel de seguridad (Seg)**	T_{DETSec} **secuencial (Seg)**	T_{DETPar} **paralelo (Seg)**	**Aceleración (×)**
A	128	14.005	5.201	2.692
	192	246.388	85.370	2.886
	256	1305.610	373.540	3.495

Por otro lado, en la Tabla 6.17 se muestran los tiempos de procesamiento al variar los niveles de seguridad utilizando curvas tipo *F* en la generación de llaves de descifrado de DET-ABE. En este proceso, se observa un comportamiento similar al del proceso de cifrado, teniendo incrementos de tiempos de ejecución relativamente pequeños al utilizar mayores niveles de seguridad. En la Tabla 6.17

se observan también las aceleraciones obtenidas con la versión paralela con respecto a la secuencial. Nuevamente se puede ver que, independientemente del tipo de curva usada, la aceleración conseguida del proceso de generación de llaves de DET-ABE es superior a 2X.

Tabla 6.17: Resultados de rendimiento de DET-ABE variando el nivel de seguridad en la etapa de generación de la llave de descifrado utilizando curvas tipo *F*.

Curva	Nivel de seguridad (Seg)	T_{DETSec} secuencial (Seg)	T_{DETPar} paralelo (Seg)	Aceleración (×)
F	128	1.642	0.803	2.044
	192	11.540	4.186	2.756
	256	48.735	14.010	3.478

6.5.3.4. Comparación de curvas A y F

Cifrado

En la Figura 6.12, incisos **a)** y **b)** se muestra una comparación entre los tiempos de ejecución de AES y CP-ABE, de tal forma que en la misma gráfica se indica el tiempo de ejecución de DET-ABE secuencial y paralelo, el cual equivale a la suma de los valores del eje *x* para CP-ABE y el eje *y* para AES. Para cada nivel de seguridad, los tiempos de CP-ABE paralelo muestran desplazamientos hacia la izquierda, por la paralelización multihilo. La magnitud del desplazamiento indica la ganancia obtenida con el esquema paralelo. Como se puede ver en el inciso **a)** de esta figura, los tiempos de procesamiento de niveles de seguridad de 192 y 256 bits en curvas tipo *A* pertenecen a la región III, mientras que al utilizar un nivel de seguridad de 128 bits, las instancias se mantienen en la región I. Sin embargo, al utilizar curvas tipo *F* (inciso **b**), solamente el nivel de seguridad de 256 bits pertenece a la región III, mientras que el resto de las instancias se mantienen en la región I.

Como se puede ver en el inciso **b)**, al utilizar niveles de seguridad de 128 bits es relativamente fácil conseguir que el tiempo de AES domine al de CP-ABE, mientras que a mayor nivel de seguridad se vuelve más complicado.

Descifrado

Como en el cifrado, los incisos **c)** y **d)** de la Figura 6.12 muestran una comparación entre los tiempos de AES y CP-ABE. Como se puede ver en esta figura, a pesar de que los tiempos de procesamiento en el descifrado son menores a los del cifrado, los niveles de seguridad de 192 y 256 bits con curvas tipo *A* (inciso **c**) siguen perteneciendo a la región III, mientras que al utilizar un nivel de seguridad de 128 bits, los tiempos de procesamiento se definen en la región I. Por otro lado, al utilizar curvas tipo *F* (inciso **d**) se puede observar un crecimiento en los tiempos de procesamiento, de tal forma que para todos los niveles de seguridad se mantiene una pertenencia a la región III.

Al contrario del proceso de cifrado, el proceso de descifrado con CP-ABE tiene una tendencia a dominar al de AES.

Generación de la llave de descifrado

Los incisos **e)** y **f)** de la Figura 6.12 muestran una comparación de los tiempos de procesamiento del algoritmo de generación de llaves de DET-ABE en su versión secuencial y paralela. En estas figuras se puede ver que al utilizar curvas tipo *A*, los tiempos de procesamiento son mayores a los que se obtienen con curvas *F*. El comportamiento que tienen este proceso es muy similar al del proceso de cifrado.

En la Figura 6.12, **a)**, **c)** y **e)** son la comparación entre los tiempos de procesamiento de AES y CP-ABE en el cifrado, descifrado y generación de la llave de descifrado utilizando curvas tipo *A* (respectivamente), mientras que **b)**, **d)** y **f)** comparan los tiempos de procesamiento de AES y CP-ABE en el cifrado, descifrado y generación de la llave de descifrado utilizando curvas tipo *F* (respectivamente). En estas gráficas, los puntos unidos con líneas punteadas representan al esquema DET-ABE secuencial, mientras que los cuadrados unidos con una línea continua representa a DET-ABE paralelo.

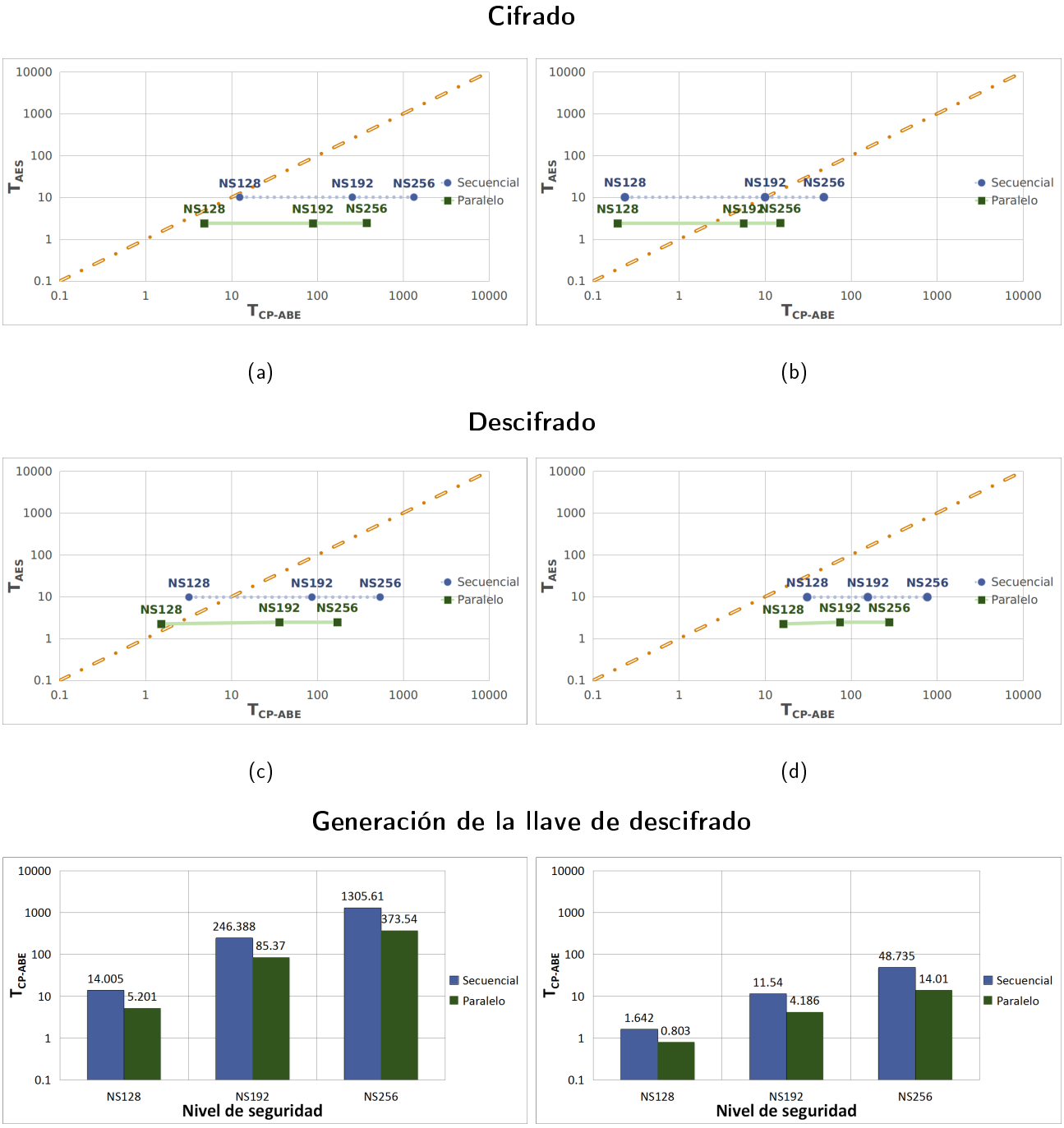


Figura 6.12: Gráficas de resultados obtenidos en el experimento 3.

6.5.4 Experimento 4

Este experimento permite observar el crecimiento en los tiempos de procesamiento a partir de la variación de los atributos utilizados para el cifrado de la llave AES (política de acceso), como los utilizados para su descifrado (atributos de usuario). De este modo, los tiempos de ejecución del esquema pueden ser considerablemente grandes, ya que el crecimiento del tiempo de éste no depende únicamente del nivel de seguridad, sino también de la cantidad de atributos con los que se realizan las operaciones. En este experimento se espera ver tanto los incrementos en los tiempos de ejecución del esquema al variar estos atributos, como la aceleración obtenida en la versión paralela de DET-ABE.

Los procesos de DET-ABE que son evaluados con este experimento son el cifrado, generación de la llave de descifrado y descifrado. Los datos fijos son el nivel de seguridad en 192 bits, el uso de una curva tipo *F*, un tamaño de datos de 20MB. La ejecución paralela del cifrador CP-ABE se realiza con 4 hilos de procesamiento. Se utilizan cantidades de atributos de usuario de 26, 52, 78, 156, 312 y hasta 624 atributos (denominados en las gráficas y resultados obtenidos como **NA1**, **NA2**, **NA3**, **NA4**, **NA5** y **NA6**, respectivamente).

La Figura 6.13 describe el experimento 4 para el cifrado (**a**), descifrado (**b**) y generación de llaves (**c**). En cada caso se presentan los parámetros fijos. Para los tres casos, la variable del experimento fue el número de atributos de la política.

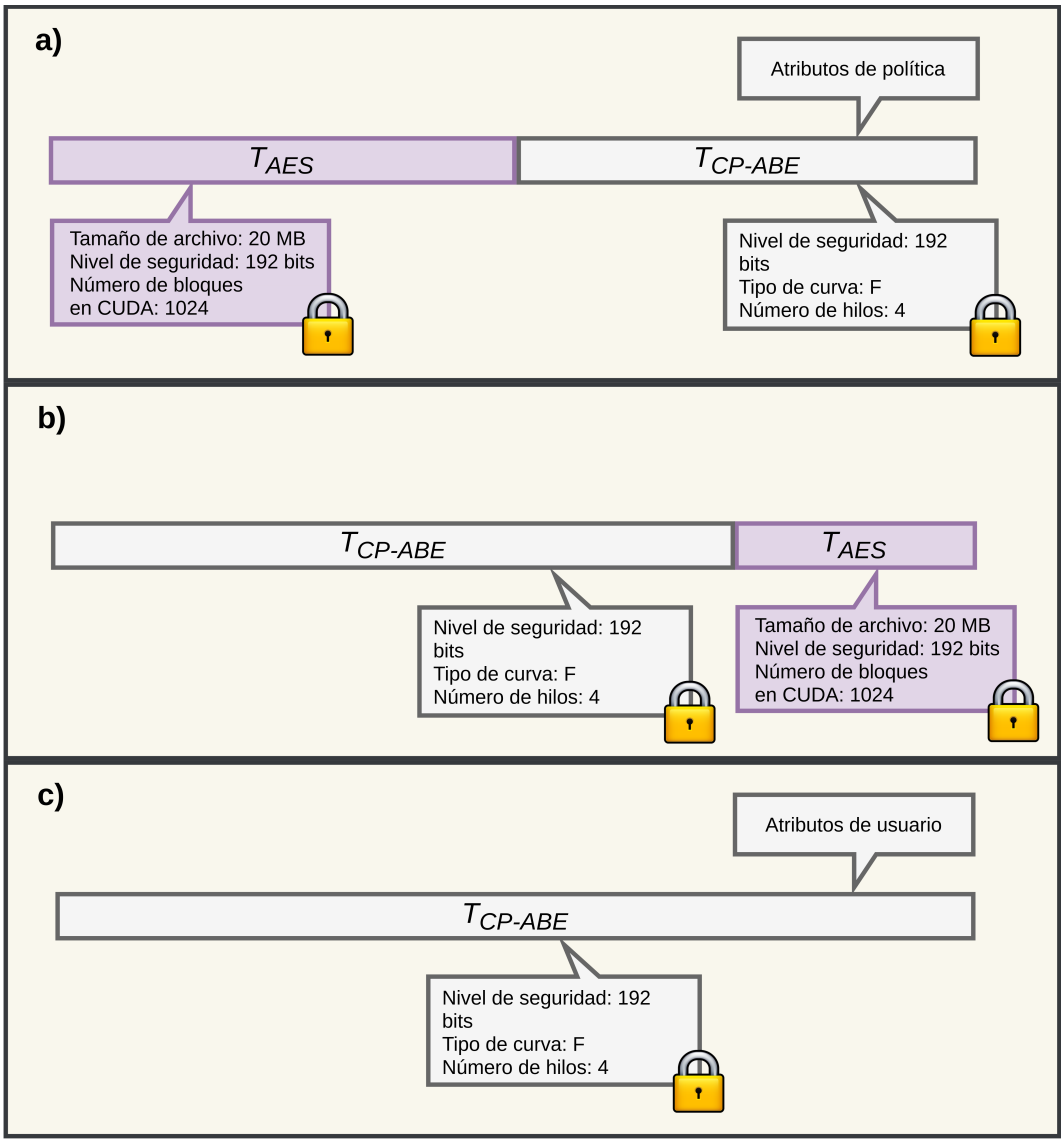


Figura 6.13: Datos de prueba y de configuración de AES y CP-ABE para **a)** el cifrado, **b)** el descifrado de datos y **c)** la generación de llaves de descifrado con DET-ABE en el experimento 4.

6.5.4.1. Cifrado

Como se mostró en el experimento 3, las curvas tipo *F* producen pequeños tiempos de procesamiento en el cifrado de tal modo que al variar la cantidad de atributos en la política de acceso, el incremento en los tiempos de procesamiento del cifrado de la llave AES es relativamente

bajo. En la Tabla 6.18 se muestran dichos tiempos, los cuales en el caso de la versión secuencial de CP-ABE crecen de manera proporcional con respecto a la cantidad de atributos de usuario en la política de acceso, teniendo así tiempos de procesamiento relativamente bajos.

En este experimento, como se puede ver en la Tabla 6.18, el tiempo de cifrado en AES domina al de CP-ABE en la mayoría de los casos, de tal forma que se considera que al variar los atributos de la política de acceso con niveles de seguridad de 192 bits en curvas F, los tiempos de cifrado tenderán a permanecer en la región I, ya que los tiempos de cifrado con CP-ABE bajo esta configuración son relativamente bajos, al igual que los tiempos de AES. Este comportamiento se puede observar en el inciso **a)** de la Figura 6.14.

Como se puede ver en la columna referente a las aceleraciones, existe una variación considerablemente grande entre las aceleraciones obtenidas en las diferentes instancias de CP-ABE. Esto se debe a que en CP-ABE, los tiempos de ejecución pueden variar mucho debido a la aleatoriedad de los componentes que se utilizan para el cifrado y a sus operaciones a bajo nivel, donde se procesan byte por byte varios cientos o miles de bits para cada componente en las operaciones. Las aceleraciones conseguidas en el cifrado con DET-ABE para este experimento se muestran en el inciso **b)** de la Figura 6.14. Entre estas, la máxima aceleración obtenida en el cifrado con DET-ABE paralelo es de 3.760X, la cual se obtiene cuando se usan un total de 26 atributos en la política de acceso.

Tabla 6.18: Resultados de rendimiento de DET-ABE variando la cantidad de atributos de la política de acceso en la etapa de cifrado.

	Secuencial			Paralelo			
Atributos en política	T_{DETSec} **(Seg)**	$T_{CP\text{-}ABESec}$ **(seg)**	T_{AESSec} **(Seg)**	T_{DETPar} **(Seg)**	$T_{CP\text{-}ABEPar}$ **(Seg)**	T_{AESPar} **(Seg)**	**Aceleración (×)**
26	10.300	0.493	9.807	2.739	0.197	2.542	3.760
52	10.760	0.941	9.819	2.918	0.379	2.539	3.529
78	11.198	1.387	9.811	3.049	0.508	2.541	3.672
156	12.588	2.779	9.809	3.571	1.034	2.537	3.525
312	15.309	5.495	9.814	5.206	2.667	2.539	2.940
624	20.904	11.087	9.817	8.156	5.619	2.537	2.563

En la Figura 6.14, inciso **a)** se muestra una comparación entre los tiempos de AES y CP-ABE, así como el tiempo de cifrado con DET-ABE secuencial y paralelo. Para las diferentes instancias del

experimento, los tiempos de CP-ABE, ya se secuencial o paralelo, crecen con el número de atributos. Comparando los tiempos de ejecución de las versiones secuencial y paralela de DET-ABE para una misma configuración de atributos, se puede ver que los que corresponden a la versión paralela se encuentran desplazados a la derecha. La magnitud del desplazamiento indica la ganancia obtenida con el esquema paralelo.

Por otro lado, el esquema paralelo puede compensar el crecimiento del tiempo de ejecución de CP-ABE. En todos los casos de este experimento, la aceleración está por arriba de 2X, como se puede ver en el inciso *b)* de la Figura 6.14. Solo para las instancias con muchos atributos se puede observan que el tiempo de CP-ABE domina ligeramente al tiempo de procesamiento de DET-ABE (para el tamaño del archivo usado). En los otros casos, la aceleración está determinada por la mejora en el tiempo de ejecución del cifrado en AES.

6.5.4.2. Descifrado

El descifrado para el caso de curvas F exhibe costos computacionales relativamente altos al utilizar un nivel de seguridad de 192 bits. Sin embargo, al procesar pocos atributos estos tiempos no son significativos, como se puede ver en la Tabla 6.19. En esta tabla se muestra la variación de los tiempos de descifrado de DET-ABE en función a la cantidad de atributos de usuario utilizados, donde se puede ver como el tiempo de descifrado de los datos con AES es superado por el tiempo de descifrado con CP-ABE al utilizar 52 atributos, mientras que el cifrado AES es superado por el cifrado CP-ABE al utilizar una política de acceso de 624 atributos. Ya que CP-ABE domina el tiempo de procesamiento de DET-ABE, las aceleraciones obtenidas con DET-ABE paralelo en el tiempo de descifrado son reducidas a medida que la cantidad de atributos de usuario crece.

En el descifrado con DET-ABE, al contrario que en el cifrado, CP-ABE domina a AES en la mayoría de los casos, de tal forma que se considera que al variar los atributos de la política de acceso con niveles de seguridad de 192 bits en curvas F, los tiempos de descifrado tenderán a permanecer en las regiones III y IV. Este comportamiento se puede observar en el inciso **c)** de la Figura 6.14,

la cual muestra el tiempo de descifrado DET-ABE, secuencial y paralelo, al incrementar la cantidad de atributos de usuario.

Tabla 6.19: Resultados de rendimiento de DET-ABE variando la cantidad de atributos de usuario en la etapa de descifrado.

	Secuencial			Paralelo			
Atributos de usuario	T_{DETSec} **(Seg)**	$T_{CP\text{-}ABESec}$ **(seg)**	T_{AESSec} **(Seg)**	T_{DETPar} **(Seg)**	$T_{CP\text{-}ABEPar}$ **(Seg)**	T_{AESPar} **(Seg)**	**Aceleración (×)**
26	16.266	6.473	9.793	5.541	2.908	2.633	2.935
52	22.719	12.934	9.785	8.422	5.795	2.627	2.697
78	29.026	19.240	9.786	10.719	8.093	2.626	2.707
156	47.858	38.078	9.780	18.562	15.926	2.636	2.578
312	85.233	75.450	9.783	39.004	36.384	2.620	2.185
624	167.853	158.078	9.775	76.832	74.199	2.633	2.184

Para la configuración de DET-ABE utilizada en este experimento, los efectos de la paralelización en el descifrado AES hacen que, aun paralelizando CP-ABE, este último sigue dominando al descifrado en todas las instancias.

Por otro lado, en el inciso *d)* de la misma figura se pueden observar las aceleraciones obtenidas con DET-ABE paralelo en el proceso de descifrado. Al realizar una comparación entre éstas y las correspondientes en la etapa de cifrado (inciso *b*), se puede observar que existe un menor impacto en la paralelización del descifrado que en el cifrado. No obstante, en todos los casos las aceleraciones son superiores a 2X.

6.5.4.3. Generación de la llave de descifrado

En la Tabla 6.20 se presentan los tiempos de ejecución totales del proceso de generación de la llave de descifrado para diferentes instancias de atributos de usuario. Se consideró aquí la misma cantidad de atributos de usuario usados en el cifrado[1].

En el inciso **e)** de la Figura 6.14 se muestra el crecimiento de los tiempos de ejecución del proceso de generación de llaves de descifrado de DET-ABE variando la cantidad de atributos de usuario. Los tiempos de procesamiento de la versión secuencial y paralela tienen un crecimiento directamente

[1] Esto no necesariamente se requiere, ya que basta con la cantidad de atributos de usuario suficiente para satisfacer la política de acceso que se utiliza en el cifrado con CP-ABE.

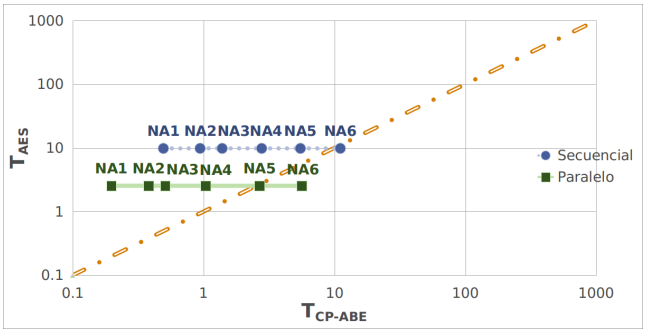
Tabla 6.20: Resultados de rendimiento de DET-ABE variando la cantidad de atributos de usuario en la etapa de generación de la llave de descifrado.

Atributos de usuario	T_{DETSec} (Seg)	T_{DETPar} (Seg)	Aceleración (×)
26	0.501	0.195	2.569
52	0.921	0.416	2.214
78	1.388	0.506	2.743
156	2.738	1.065	2.571
312	5.493	2.556	2.149
624	11.54	4.186	2.757

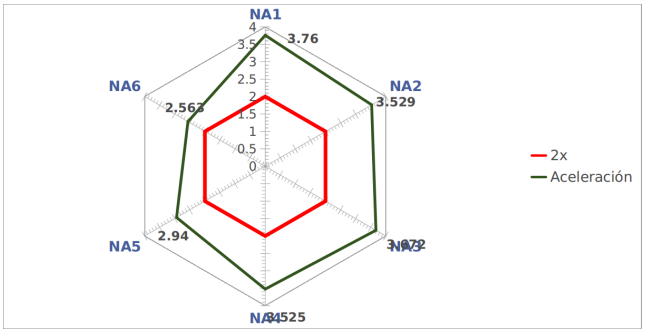
proporcional a la cantidad de atributos de usuario. Las aceleraciones mostradas en el inciso **f)** de la misma figura demuestran la eficiencia de la paralelización realizada de DET-ABE, y que superan la meta planteada en ésta tesis.

En la Figura 6.14, **a)**, **c)** y **e)** son la comparación entre los tiempos de procesamiento de AES y CP-ABE en el cifrado, descifrado y generación de la llave de descifrado (respectivamente), mientras que **b)**, **d)** y **f)** representan las aceleraciones medidas a partir de los tiempos de procesamientos de **a)**, **c)** y **e)** (respectivamente). En estas gráficas, los puntos unidos con líneas punteadas representan al esquema DET-ABE secuencial, mientras que los cuadrados unidos con una línea continua representa a DET-ABE paralelo.

Cifrado

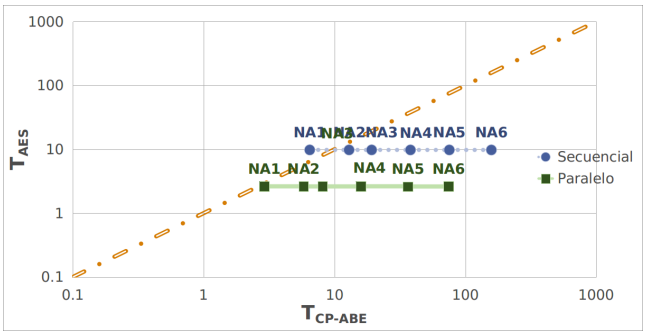


(a)

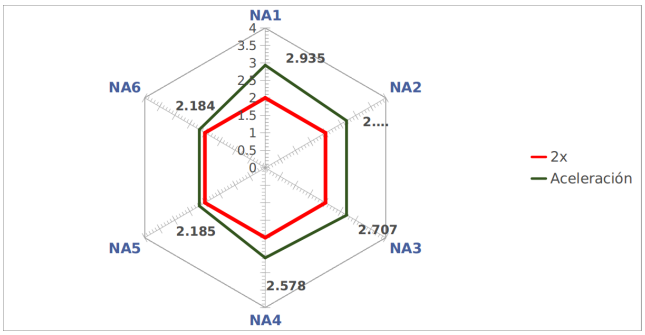


(b)

Descifrado

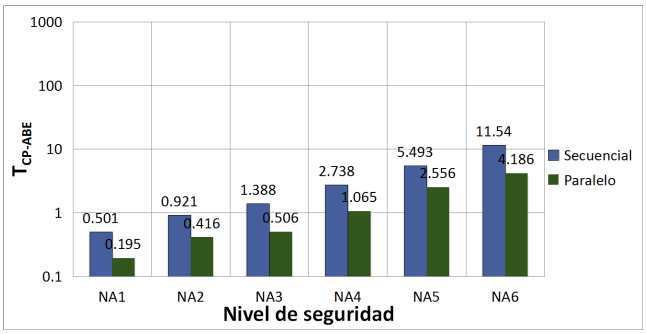


(c)

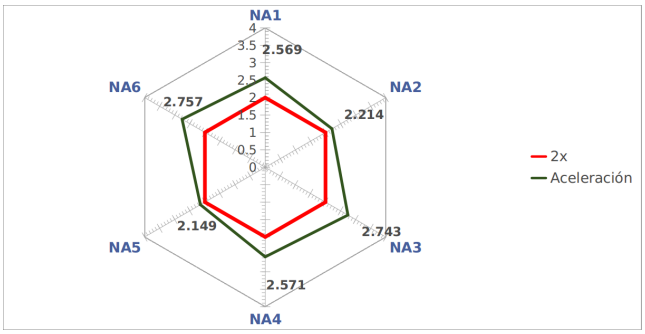


(d)

Generación de la llave de descifrado



(e)



(f)

Figura 6.14: Gráficas de resultados obtenidos en el experimento 4.

6.5.5 Experimento 5

En este experimento se tiene el propósito de identificar la variación en los tiempos de ejecución, y su impacto en las aceleraciones, cuando se utiliza un mayor número de hilos en el procesamiento de CP-ABE. En este experimento se utiliza un dispositivo de cómputo de altas prestaciones, el servidor Chronos descrito en la Sección 6.1.

Se analiza únicamente el cifrado con DET-ABE, fijando tanto el nivel de seguridad en 192 bits, el tipo de curva elíptica a F y el tamaño de los datos de 20MB. Para esto, se varían los hilos de procesamiento entre 4, 6, 8, y 10 en la versión paralela de CP-ABE. Además, se utilizan políticas de acceso con 156 y 624 atributos. Se utiliza los esquema que se presenta en la Figura 6.15.

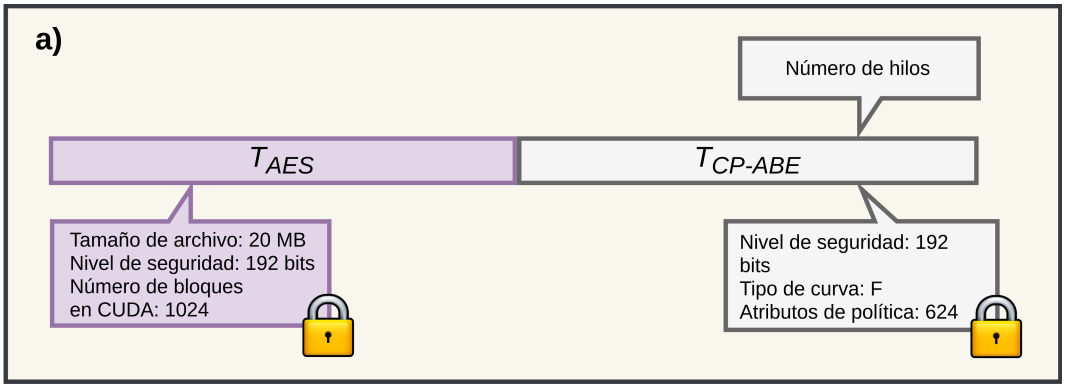


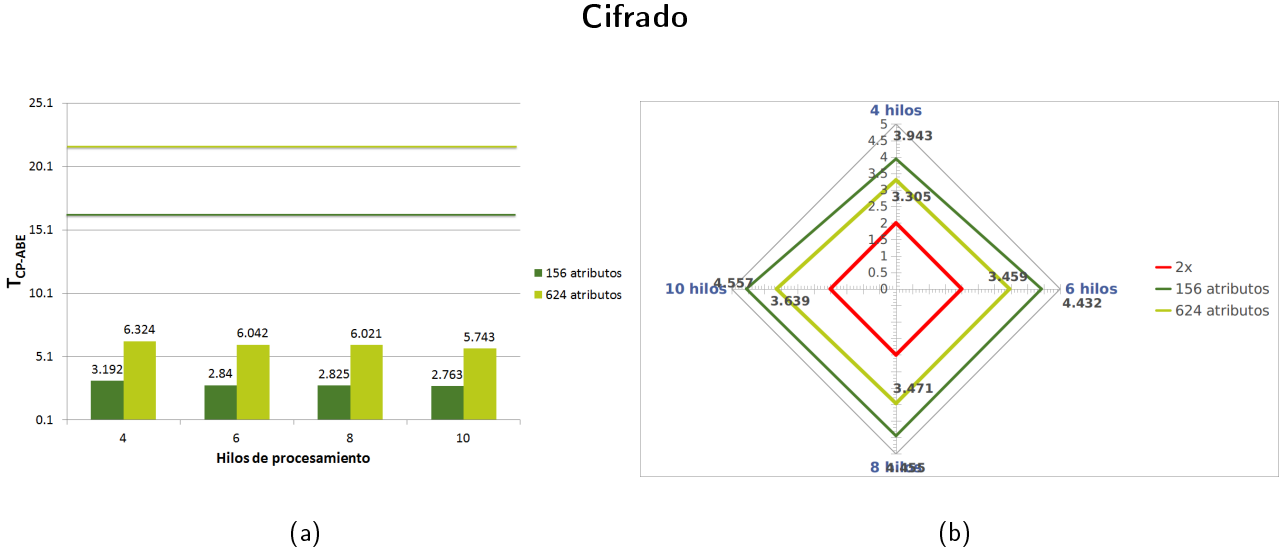
Figura 6.15: Datos de prueba y de configuración de AES y CP-ABE para el cifrado con DET-ABE en el experimento 5.

En la Tabla 6.21 se muestran los tiempos de procesamiento obtenidos para el cifrado de la llave AES tanto para la instancia con 156 atributos como para la de 624. Para este experimento se tomó como referencia el tiempo secuencial DET-ABE para realizar el cálculo de las aceleraciones obtenidas al paralelizar CP-ABE. En esta tabla, puede notarse que a mayor número de hilos, mayor es la reducción del tiempo de cifrado de CP-ABE.

Tabla 6.21: Pruebas de rendimiento variando la cantidad de hilos con que se acelera el cifrado con CP-ABE.

		Secuencial			Paralelo			
Número de atributos	Hilos	DET-ABE (Seg)	CP-ABE (seg)	AES (Seg)	DET-ABE (Seg)	CP-ABE (Seg)	AES (Seg)	Aceleración (×)
156	4	12.588	2.779	9.809	3.192	1.135	2.057	3.943
	6				2.840	0.771	2.069	4.432
	8				2.825	0.758	2.067	4.455
	10				2.762	0.700	2.062	4.557
624	4	20.904	11.087	9.817	6.324	3.297	3.027	3.305
	6				6.042	3.026	3.016	3.459
	8				6.021	2.998	3.023	3.471
	10				5.743	2.728	3.015	3.639

En el inciso **a)** de la Figura 6.16 se muestran los tiempos de procesamiento del cifrado CP-ABE. En esta figura, las líneas horizontales indican el tiempo de procesamiento para cada conjunto de atributos usando la versión secuencial de CP-ABE. Así mismo, en el inciso **b)** se muestran las aceleraciones obtenidas para ambas instancias. En todos los casos la aceleración es superior a 2X.



(a) (b)

Figura 6.16: Gráficas de resultados obtenidos en el experimento 5.

En la Figura 6.16, **a)** muestra los tiempos de procesamiento en el cifrado CP-ABE secuencial y el paralelo, mientras que **b)** muestra las aceleraciones medidas a partir de los tiempos de procesamiento descritos en **a)**.

6.5.6 Experimento 6

El propósito de este experimento es medir los tiempos de procesamiento al ejecutar el esquema DET-ABE completo, incluyendo los procesos de carga y descarga en la transferencia de la información a un servidor de almacenamiento en la nube. Para esto, se evalúan de forma integral los tres procesos: cifrado, descifrado y generación de llaves.

De esta forma, el tiempo de procesamiento total se conforma de los tiempos de ejecución de las tareas que se mencionan a continuación.

- Generación de forma dinámica de una llave de cifrado de los datos.
- Cifrado de los datos.
- Construcción del sobre digital.
- Transferencia a la nube del sobre digital.
- Descarga de la nube del sobre digital.
- Extracción de los objetos que conforman el sobre digital.
- Descifrado de los datos.

Para la evaluación de este experimento se consideró el prototipo experimental mostrado en la Figura 6.1 de la Sección 6.1. Para esto, se define al dispositivo A como propietario y receptor de los datos, de tal forma que la medición de los tiempos de procesamiento se toman de éste, mientras que el dispositivo B realiza la tarea del servidor de almacenamiento en la nube.

Por otra parte, los datos fijos son el nivel de seguridad en 192 bits, el uso de una curva tipo F, un total de 624 atributos en la política de acceso y la ejecución paralela del cifrador CP-ABE se realiza con 4 hilos de procesamiento. Para esto, se utilizan los tamaños de archivos TA1, TA20, TA80, TA200, y TA500 descritos en los experimentos previos.

El esquema utilizado en este experimento se muestra en la Figura 6.17. Nótese que el proceso requiere un proceso de cifrado, un proceso de descifrado, y dos transferencias. En un escenario real se hace una carga y una o varias descargas por separado.

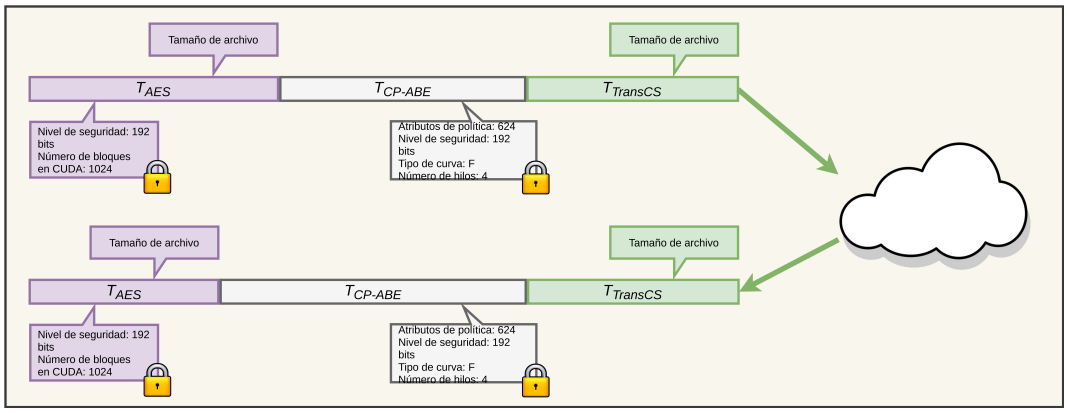


Figura 6.17: Datos de prueba y de configuración de AES y CP-ABE para el procesamiento de almacenamiento y recuperación de datos en la nube con DET-ABE (experimento 6).

En la Tabla 6.22 se muestran los tiempos de procesamiento derivados de la ejecución del experimento 6. Como se observa a partir de los datos de la tabla, los costos computacionales al ejecutar el esquema completo son relativamente grandes. Los tiempos de transferencia sin seguridad se presentan como elemento para comparar contra los tiempos de transferencia con seguridad, los cuales representan un ligero incremento en el tiempo de procesamiento debido al incremento en volumen de los datos a transferir.

Tabla 6.22: Resultados de rendimiento del esquema completo de transferencia de documentos hacia y desde la nube con y sin los servicios de seguridad de DET-ABE.

	Sin seguridad	**Con seguridad**						
	$T_{TransSS}$	**Secuencial**			**Paralelo**			
Tamaño (MB)	**(Seg)**	$T_{TotalSec}$ **(Seg)**	T_{DETSec} **(Seg)**	$T_{TransCS}$ **(Seg)**	$T_{TotalPar}$ **(Seg)**	T_{DETPar} **(Seg)**	$T_{TransCS}$ **(Seg)**	**Aceleración (×)**
1	0.702	172.170	171.460	0.710	84.918	84.188	0.730	2.027
20	12.838	212.773	199.931	12.842	114.994	102.104	12.890	1.850
80	52.900	344.428	291.360	53.068	209.174	155.726	53.448	1.647
200	130.320	612.731	482.341	130.390	390.645	259.715	130.930	1.569
500	324.638	1288.844	961.770	327.074	848.997	517.515	331.482	1.518

Tanto en DET-ABE secuencial como paralelo, los tiempos de transferencia de datos son muy similares, por lo que la variación en el tiempo solo se debe a la reducción debida a la paralelización

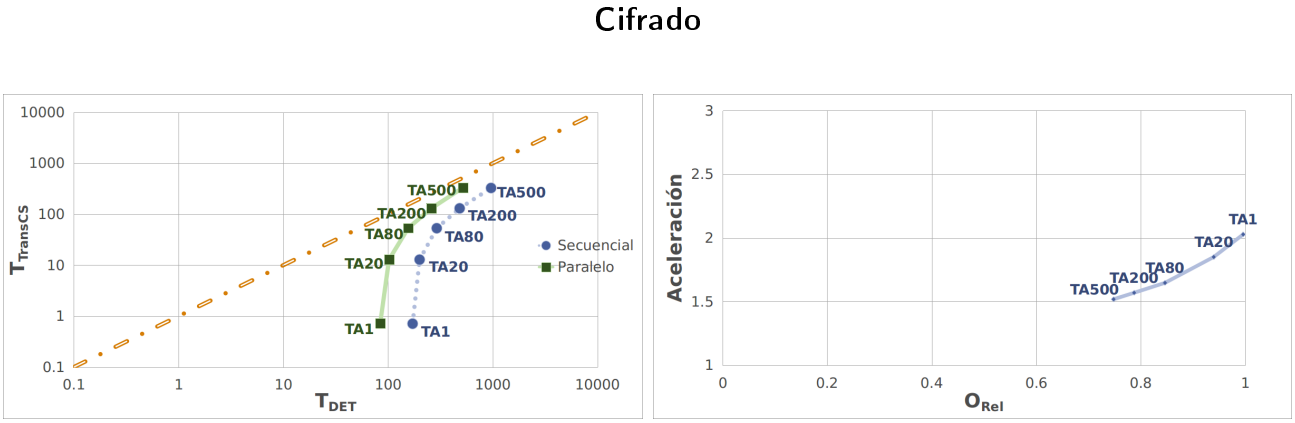
del esquema DET-ABE. Por otro lado, los tiempos de transferencia de los datos son relativamente grandes, ya que en este experimento se realizan dos procesos de transferencia (Figura 6.17), por lo tanto el tiempo de transferencia total es prácticamente el doble que en el experimento 1. Sin embargo, los tiempos de transferencia total no consiguen dominar a los tiempos de DET-ABE.

En la Figura 6.18, inciso **a)** se muestra una comparación entre los tiempos de ejecución de DET-ABE y los tiempos de transferencia. Para cada tamaño de archivo, el tiempo de DET-ABE paralelo se observa desplazado hacia la izquierda con respecto a DET-ABE secuencial. La magnitud del desplazamiento indica la ganancia obtenida con el esquema paralelo y, como se puede observar, se reducen gradualmente conforme al crecimiento del tamaño del archivo, lo cual implica un decremento en la aceleración del proceso.

Los tiempos de transferencia se comportan en la aplicación integral como la parte que no es mejorada por la paralelización del esquema DET-ABE. Desde el punto de vista de la **Ley de Amdahl** [23], la transferencia contribuye a la parte inheremente secuencial la cual limita la posible mejora obtenida con la paralelización.

Ya que los tiempos de transferencia crecen conforme se incrementa el tamaño de los datos, la fracción inherentemente secuencial crece, de tal manera que, por ejemplo, para la instancia de 500 MB, la fracción inherentemente secuencial es de 0.253 (medido con la operación $T_{TransCS}/T_{TotalSec}$ en la sección correspondiente al proceso secuencial en la Tabla 6.22), que es un cuarto del tiempo de procesamiento total.

En la Figura 6.18 inciso **b)** se relaciona el *overhead* con la aceleración obtenida. Se puede ver que el *overhead* es mayor con instancias de tamaño pequeño por lo que las aceleraciones son mayores. Conforme el *overhead* se reduce con archivos más grandes, la aceleración es menor. No obstante, a pesar de que las aceleraciones en el esquema completo son inferiores a 2X, se consiguen reducciones de procesamiento absoluto grandes. Lo anterior, contribuye a aligerar el costo de asignar seguridad con el esquema DET-ABE.



(a) (b)

Figura 6.18: Evaluación del esquema completo para transferencia de documentos hacia y desde la nube, incluyendo los servicios de seguridad de DET-ABE.

En **a)** de la Figura 6.18 se muestra una comparación entre los tiempos de procesamiento de la transferencia y cifrado con DET-ABE. En **b)** se muestran las aceleraciones medidas a partir de los tiempos de procesamiento descritos en **a)**. En el inciso **a)**, los puntos unidos con líneas punteadas representan al esquema DET-ABE secuencial, mientras que los cuadrados unidos con una línea continua representa a DET-ABE paralelo.

6.6 Resumen

En este capítulo se describieron los resultados obtenidos del trabajo de experimentación para evaluar el método propuesto para la implementación del esquema DET-ABE. Los experimentos que se definieron en esta sección se enfocaron en la demostración del impacto de acelerar el esquema DET-ABE considerando un prototipo experimental, en el cual se transfiere un conjunto de datos a un servidor de almacenamiento en la nube. En estos experimentos, se mostraron diferentes configuraciones en la seguridad asignada por el esquema.

En el experimento 1 se mostró que el *trabajo adicional* es mayor a medida que el tamaño del archivo crece. Sin embargo, las estrategias de paralelización propuestas consiguen reducir considerablemente el tiempo de procesamiento del esquema DET-ABE. Sin embargo, como se muestra en el experimento 6, las aceleraciones obtenidas a nivel de esquema se reducen considerablemente a medida que el tiempo de transferencia crece junto a los tiempos de procesamiento de DET-ABE.

En el experimento 2 se mostró cómo los *tiempos de ejecución* del esquema DET-ABE incrementan a medida que el tamaño del documento crece. Considerando esto y las reducciones de los tiempos de procesamiento en la versión paralela, las reducciones del trabajo adicional relativo mostradas en el experimento 1 se deben principalmente al cifrado de los datos. Sin embargo, como se mostró en el experimento 3, los altos costos de procesamiento en el descifrado con DET-ABE descritos en el experimento 1 se deben principalmente al descifrado de la llave AES, debido a que al utilizar curvas tipo F, el tiempo de procesamiento crece relativamente rápido con respecto al nivel de seguridad. En el caso de niveles de seguridad altos y curvas tipo F es evidente que el costo computacional de CP-ABE crece de manera importante al grado de superar, para archivos no extremadamente grandes, los costos del descifrado de los datos.

Por otro lado, en el experimento 4 se consiguió demostrar que la variación del los atributos en política tienen un alto impacto en el tiempo de procesamiento, el cual es afectado dependiendo de la configuración del nivel de seguridad y del tipo de curva elíptica. Por último, en el experimento 5 se demostró que la variación del número de hilos en CP-ABE únicamente resulta beneficioso en el cifrado al tener niveles de seguridad mayores de 192 bits para curvas de tipo F y mayores de 128 bits para curvas tipo A.

7

Conclusiones y trabajo futuro

En este capítulo se presentan las conclusiones y contribuciones del trabajo de tesis realizado. A partir de los resultados obtenidos se identifican posibles líneas de trabajo futuro.

7.1 Conclusiones

En este trabajo de investigación el problema abordado fue la paralelización de un esquema de seguridad para mejorar la eficiencia cuando se implementan servicios de seguridad para proteger los datos que se almacenan en un entorno de nube. La metodología seguida para lograr los objetivos de esta investigación consistió en el estudio e implementación de los cifradores AES y CP-ABE, la identificación de las secciones paralelizables en los algoritmos que conforman al esquema DET-ABE, el particionamiento de los mismos para su implementación paralela, el diseño de las estrategias de paralelización para cada uno de ellos y la evaluación de las estrategias propuestas. Una vez que fue diseñada y validada la estrategia de paralelización, se realizó la optimización de las implementaciones paralelas.

A continuación se presentan las conclusiones generales obtenidas en el desarrollo del trabajo.

- Al realizar el estudio y comprensión de los algoritmos de DET-ABE se identificó que los algoritmos que conforman todos los módulos de ambos cifradores son más fáciles de implementar y comprender al estudiarlos por separado. Aquellos módulos de CP-ABE que usan aritmética con números grandes en grupos resultaron ser más complejos para implementar bajo un enfoque paralelo. Por ello, los algoritmos aritméticos de CP-ABE que se encuentran en la capa baja no fueron considerados para ser paralelizados. Su implementación se realizó usando la biblioteca de uso libre PBC.

- El proceso de generación de la estructura de acceso genera muy poca carga computacional, de tal forma que éste proceso puede realizarse tanto en el dispositivo emisor como en el receptor (no se transfiere la estructura de acceso, sino que se transfiere la política de acceso).

- Se propusieron dos estrategias de paralelización de DET-ABE, una para acelerar de forma independiente a `AES.Encrypt` y `CP-ABE.Encrypt` y la otra para la ejecución paralela de ambas. La segunda obtuvo una mayor aceleración, sin embargo no es significativa. Esto se debe a que la paralelización a este nivel requiere del uso dedicado de 2 hilos de ejecución, dejando menos hilos disponibles para la aceleración de CP-ABE.

- El cifrado de la llave AES (llave de cifrado) puede no generar un gran costo computacional. Sin embargo, si se utilizan políticas de acceso muy extensas, además de niveles de seguridad altos, los tiempos de procesamiento del cifrado, descifrado y/o generación de la llave de descifrado en CP-ABE pueden ser relativamente grandes, lo cual depende del tipo de curva que se utilice. De este modo, si se requiere de muchas consultas de datos cifrados y almacenados en la nube, lo mas conveniente es usar curvas tipo *A*. De este modo, el descifrado de los datos es más rápido, ya que, como se mostró en el experimento 3, el descifrado con este tipo de curva tiene un costo considerablemente menor. Además, al contar con una versión paralela, es posible

asignar niveles de seguridad mayores con tiempos de procesamiento equivalentes a un nivel de seguridad inferior.

- Las mayores aceleraciones obtenidas en el esquema a partir de su paralelización son obtenidas en el cifrado y descifrado de AES, lo cual se debe a su alto nivel de paralelización. En el caso particular de la implementación descrita en esta tesis, el costo de la paralelización de AES se debe directamente a dos factores, el primero es la transferencia de datos del dispositivo CPU a los GPUs, mientras que el segundo al uso de múltiples vectores inicializadores, los cuales son generados de forma secuencial, previo al cifrado de los datos.

- Ya que en CP-ABE los tiempos de procesamiento en el descifrado usando curvas tipo *A* es relativamente pequeño, es posible utilizar la versión secuencial de la función `CP-ABE.Decrypt`, dejando la paralelización en los procesos `AES.Encrypt`, `CP-ABE.Encrupt`, `AES.Decrypt` y `CP-ABE.KeyGen`. Por otro lado, si se utilizan curvas tipo *F*, el tiempo de cifrado y generación de la llave de descifrado con CP-ABE son relativamente pequeños, de tal forma que pueden usarse sus versiones secuenciales para su ejecución.

- Tras la experimentación, comparando los tiempos de procesamiento de DET-ABE secuencial y paralelo, se consiguió reducir los tiempos de procesamiento de los cinco algoritmos paralelizados (`AES.Encrypt`, `AES.Decrypt`, `CP-ABE.KeyGen`, `CP-ABE.Encrypt` y `CP-ABE.Decrypt`), de tal modo que las aceleraciones máximas mostradas en DET-ABE en la experimentación fueron de 4.557× para el cifrado, 3.452× para el descifrado y 3.495× para la generación de la llave de descifrado, consiguiendo así una aceleración mayor a *2x* en los procesos DET-ABE y superando la meta del trabajo de investigación.

7.2 Trabajo futuro

Como se mostró en los experimentos 3 y 4, los módulos de CP-ABE (`CP-ABE.Encrypt`, `CP-ABE.KeyGen` y `CP-ABE.Decrypt`) pueden tener gran impacto en el tiempo de procesamiento del esquema DET-ABE, lo cual sucede al utilizar muchos atributos en la política de acceso, niveles de seguridad altos y en por lo menos uno de los tres módulos al variar el tipo de curva elíptica (descifrado en curvas tipo A y los procesos de descifrado y generación de la llave de descifrado al utilizar curvas tipo F). Ya que la complejidad de las operaciones en estas curvas se deben a la cantidad de bits procesados a bajo nivel, resulta atractiva la paralelización de estos procesos y así conseguir una mayor aceleración en la versión paralela del esquema DET-ABE.

Dado que no se utilizó la estrategia de paralelismo masivo con dispositivos GPU para acelerar los algoritmos de CP-ABE en su capa baja, a causa de las limitaciones de cómputo de CUDA (baja disponibilidad del uso de bibliotecas dentro de los kernels), se proponen los siguientes puntos como posible trabajo futuro:

- Realizar un estudio de las operaciones aritméticas a bajo nivel en grupos, campos finitos y emparejamientos bilineales, para así crear una forma eficiente de implementar éstas operaciones dentro del entorno de CUDA.
- Acelerar aún más el proceso de cifrado, descifrado y generación de llave de DET-ABE por medio de estrategias híbridas en la capa baja de CP-ABE.

Además, ya que dependiendo de la aplicación es posible que algunos procesos del esquema DET-ABE tengan mayor relevancia en la ejecución de éste, es posible generar algunas versiones de DET-ABE paralelo, donde sean paralelizados los procesos que se consideren necesarios, algunos ejemplos de estos casos se muestran a continuación.

- Si en una aplicación se considera cifrar tamaños de archivo mayores a 1 GB, considerando políticas de acceso con curvas tipo F y niveles de seguridad de 128 o 192 bits, el impacto

recae directamente en el cifrado de los datos con AES. Debido a esto, es posible mantener los procesos de CP-ABE bajo su ejecución secuencial y dedicar la paralelización del esquema únicamente en el cifrado AES.

- Si se propone una aplicación en la cual se requiera de un alto nivel de seguridad utilizando documentos menores a 100MB, es posible dedicar la paralelización a los procesos con CP-ABE, dando así prioridad a un dispositivo con más de 4 núcleos de procesamiento para obtener una aceleración mayor a 4×.

Bibliografía

[1] Adams, C. and Lloyd, S. (2002). *Understanding PKI: Concepts, Standards, and Deployment Considerations*. Addison-Wesley Publishing Company, Inc., 1 edition.

[2] Adams, R. and Pereira, R. (1998). The ESP CBC-Mode Cipher Algorithms. *IDEA*, 128(128):128.

[3] Agrawal, H. and Sharma, M. (2010). Implementation and analysis of various symmetric cryptosystems. *Indian Journal of Science and Technology*, 3(12):1173–1176.

[4] Barker, E. B., Barker, W. C., Burr, W. E., Polk, W. T., and Smid, M. E. (2007). Sp 800-57. Recommendation for Key Management, Part 1: General (Revised). Technical report, Gaithersburg, MD, United States.

[5] Bethencourt, J., Sahai, A., and Waters, B. (2007). Ciphertext-policy attribute-based encryption. In *2007 IEEE Symposium on Security and Privacy (SP '07)*, pages 321–334.

[6] Boneh, D. (2012). Pairing-Based Cryptography: Past, Present, and Future. In Wang, X. and Sako, K., editors, *Advances in Cryptology – ASIACRYPT 2012: 18th International Conference on the Theory and Application of Cryptology and Information Security, Beijing, China, December 2-6*, pages 1–1, Berlin, Heidelberg. Springer Berlin Heidelberg.

[7] Boneh, D. and Franklin, M. (2001). Identity-Based Encryption from the Weil Pairing. In Kilian, J., editor, *Advances in Cryptology — CRYPTO 2001: 21st Annual International Cryptology Conference, Santa Barbara, California, USA, August 19-23*, pages 213–229, Berlin, Heidelberg. Springer Berlin Heidelberg.

[8] Brookwood, N. (2009). NVIDIA solves the GPU computing puzzle.

Disponible en: `http://www.nvidia.com/content/PDF/fermi_white_papers/N.Brookwood_NVIDIA_Solves_the_GPU_Computing_Puzzle.pdf`, Última visita 2012-03-20.

[9] Chu, X., Zhao, K., and Wang, M. (2008). Massively parallel network coding on gpus. In *2008 IEEE International Performance, Computing and Communications Conference*, pages 144–151.

[10] Chu, X., Zhao, K., and Wang, M. (2009). *Practical Random Linear Network Coding on GPUs*, pages 573–585. Springer Berlin Heidelberg, Berlin, Heidelberg.

[11] Dagum, L. and Menon, R. (1998). Openmp: an industry standard api for shared-memory programming. *IEEE Computational Science and Engineering*, 5(1):46–55.

[12] Diffie, W. and Hellman, M. (1976). New directions in cryptography. *IEEE Transactions on Information Theory*, 22(6):644–654.

[13] Dworkin, M. (2001). Recommendation for block cipher modes of operation. Methods and techniques. Technical report, National Institute of Standars and Technology Gaithersburg MD Computer Security Division, USA.

[14] Glaskowsky, P. N. (2009). NVIDIA's fermi: The First Complete GPU Computing Architecture (2009). *URL http://www.nvidia.com/content/PDF/fermi_white_papers/P.Glaskowsky_NVIDIA's_Fermi-The_First_Complete_GPU_Architecture.pdf*, (Ultima visita: Junio 2016).

[15] Goyal, V., Jain, A., Pandey, O., and Sahai, A. (2008). Bounded Ciphertext Policy Attribute Based Encryption. In *Proceedings of the 35th International Colloquium on Automata, Languages and Programming, Part II*, ICALP '08, pages 579–591, Berlin, Heidelberg. Springer-Verlag.

[16] Goyal, V., Pandey, O., Sahai, A., and Waters, B. (2006). Attribute-based Encryption for Fine-grained Access Control of Encrypted Data. In *Proceedings of the 13th ACM Conference on Computer and Communications Security*, CCS '06, pages 89–98, New York, NY, USA. ACM.

[17] Gueron, S. (2010). Intel® Advanced Encryption Standard (AES) New Instructions Set. *Intel Corporation*,

pages 5–52. Disponible en: `https://www.intel.com.bo/content/dam/doc/white-paper/advanced-encryption-standard-new-instructions-set-paper.pdf`, Última visita 2016-10-05.

[18] Guim, F. and Rodero, I. (2013). *Arquitecturas basadas en computación gráfica (GPU)*. OpenLibra, 1 edition.

[19] Hammond, K. and Michaelson, G. (2012). *Research directions in parallel functional programming*. Springer Science & Business Media, 1 edition.

[20] Hankerson, D., Menezes, A. J., and Vanstone, S. (2006). *Guide to elliptic curve cryptography*. Springer Science & Business Media, 1 edition.

[21] Harrison, O. and Waldron, J. (2007). AES Encryption Implementation and Analysis on Commodity Graphics Processing Units. In Paillier, P. and Verbauwhede, I., editors, *9th International Workshop, Cryptographic Hardware and Embedded Systems - CHES*, pages 209–226, Berlin, Heidelberg. Springer.

[22] Hashem, I. A. T., Yaqoob, I., Anuar, N. B., Mokhtar, S., Gani, A., and Ullah Khan, S. (2015). The Rise of "Big Data" on Cloud Computing. *Inf. Syst.*, 47(C):98–115.

[23] Hennessy, J. L. and Patterson, D. A. (2006). *Computer Architecture: A Quantitative Approach, 4th Edition*. Morgan Kaufmann, 1 edition.

[24] Hoeffding, W., Robbins, H., et al. (1948). The central limit theorem for dependent random variables. *Duke Mathematical Journal*, 15(3):773–780.

[25] Holzmann, G. J. and Bosnacki, D. (2007). The design of a multicore extension of the SPIN model checker. *IEEE Transactions on Software Engineering*, 33(10):665–669.

[26] Jeong, H. and Park, J. (2012). *An Efficient Cloud Storage Model for Cloud Computing Environment*, pages 370–376. Springer Berlin Heidelberg, Berlin, Heidelberg.

[27] Joseph, A. and Sundaram, V. (2011). Cryptography and steganography–A survey. *International Journal of Science and Research (IJSR)*, 2(3):626–630.

[28] Kaaniche, N. (2014). *Cloud data storage security based on cryptographic mechanisms*. PhD thesis, Institut National des Télécommunications. France.

[29] Kamara, S. and Lauter, K. (2010). Cryptographic Cloud Storage. In Sion, R., Curtmola, R., Dietrich, S., Kiayias, A., Miret, J. M., Sako, K., and Sebé, F., editors, *Financial Cryptography and Data Security: FC 2010 Workshops, RLCPS, WECSR, and WLC 2010, Revised Selected Papers*, pages 136–149, Tenerife, Canary Islands, Spain. Springer.

[30] Kanaoka, A. (2013). TEPLA - LCIS, University of Tsukuba. *Laboratory of Cryptography and Information Security. Disponible en: http://www.cipher.risk.tsukuba.ac.jp/tepla/index_e.html. Última visita 2017-02-15.*

[31] Lee, C., Ro, W. W., and Gaudiot, J.-L. (2014). Boosting cuda applications with CPU–GPU hybrid computing. *International Journal of Parallel Programming*, 42(2):384–404.

[32] Lee, J., Samadi, M., Park, Y., and Mahlke, S. (2013). Transparent CPU-GPU Collaboration for Data-parallel Kernels on Heterogeneous Systems. In *Proceedings of the 22Nd International Conference on Parallel Architectures and Compilation Techniques*, PACT '13, pages 245–256, Piscataway, NJ, USA. IEEE Press.

[33] Levy, S. (2001). *Crypto: How the code rebels beat the government–saving privacy in the digital age*. Penguin Random House, 1 edition.

[34] Li, L., Chen, X., Jiang, H., Li, Z., and Li, K.-C. (2016). P-CP-ABE: Parallelizing Ciphertext-Policy Attribute-Based Encryption for clouds. In *2016 17th IEEE/ACIS International Conference*

on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD). Shanghai, China, pages 575–580. IEEE.

[35] Li, Y., Zhao, K., Chu, X., and Liu, J. (2010). Speeding up K-Means Algorithm by GPUs. In *2010 10th IEEE International Conference on Computer and Information Technology*, pages 115–122.

[36] Liberty, J. and Horvath, D. B. (2000). *Aprendiendo C++ para Linux en 21 días*. Pearson educación, 1 edition.

[37] Liu, G., An, H., Han, W., Xu, G., Yao, P., Xu, M., Hao, X., and Wang, Y. (2009). A Program Behavior Study of Block Cryptography Algorithms on GPGPU. In *2009 Fourth International Conference on Frontier of Computer Science and Technology. Shanghai, China*, pages 33–39. IEEE.

[38] Liu, Z., Cao, Z., and Wong, D. S. (2010). Efficient generation of linear secret sharing scheme matrices from threshold access trees. *Cryptology ePrint Archive: Listing*, 42(2):3–26.

[39] Lynn, B. (2006). PBC library: The Pairing-Based Cryptography Library. *Crypto Stanford. Disponible en: http://crypto.stanford.edu/pbc, Última visita 2017-05-23*.

[40] Mao, W. (2003). *Modern Cryptography: Theory and Practice*. Prentice Hall Professional Technical Reference, 1 edition.

[41] Mei, C., Jiang, H., and Jenness, J. (2010). CUDA-based AES parallelization with fine-tuned GPU memory utilization. In *2010 IEEE International Symposium on Parallel & Distributed Processing, Workshops and Phd Forum (IPDPSW). Atlanta, GA, USA*, pages 1–7. IEEE.

[42] Mell, P. and Grance, T. (2011). The NIST definition of cloud computing. *Recommendations of the National Institute of Standards and Technology*, (Special Publication 800-145):2–5. Disponible en: http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf, Última visita 2017-07-18.

[43] Menezes, A. (2009). An introduction to pairing-based cryptography. *Recent trends in cryptography*, 477(1):47–65.

[44] Miller, V. S. (1986). *Use of Elliptic Curves in Cryptography*, pages 417–426. Springer Berlin Heidelberg, Berlin, Heidelberg.

[45] Morales-Sandoval, M. and Diaz-Perez, A. (2015). Det-abe: A java api for data confidentiality and fine-grained access control from attribute based encryption. In *Proceedings of the 9th IFIP WG 11.2 International Conference on Information Security Theory and Practice - Volume 9311*, pages 104–119, New York, NY, USA. Springer-Verlag New York, Inc.

[46] Morales-Sandoval, M., Gonzalez-Compean, r. L., Diaz-Perez, A., and Sosa-Sosa, V. J. (2017). A pairing-based cryptographic approach for data security in the cloud. *International Journal of Information Security*, 16(80):1–21.

[47] Munoz-Hernandez, M. D., Morales-Sandoval, M., and Garcia-Hernandez, J. J. (2016). An end-to-end security approach for digital document management. *The Computer Journal*, 59(7):1076–1090.

[48] Niewiadomska-Szynkiewicz, E., Marks, M., Jantura, J., and Podbielski, M. (2012). A hybrid CPU/GPU cluster for encryption and decryption of large amounts of data. *Journal of Telecommunications and Information Technology*, 3(3):32–39.

[49] Nishikawa, N., Iwai, K., and Kurokawa, T. (2012). High-performance symmetric block ciphers on multicore CPU and GPUs. *International Journal of Networking and Computing*, 2(2):251–268.

[50] Olguin-Carbajal, J. C. I. R.-Z. J. H. L. M. (2009). GPU Programming Paradigm. *Journal of Theoretical and Applied Information Technology*, 9(2):133–138.

[51] Pu, S. and Liu, J.-C. (2014). EAGL: An Elliptic Curve Arithmetic GPU-Based Library for Bilinear Pairing. In Cao, Z. and Zhang, F., editors, *Pairing-Based Cryptography – Pairing 2013:*

6th International Conference, Beijing, China, November 22-24, 2013, Revised Selected Papers, pages 1–19, Cham. Springer International Publishing.

[52] Rijmen, V. and Daemen, J. (2001). Advanced encryption standard. *Proceedings of Federal Information Processing Standards Publications, National Institute of Standards and Technology*, 3373(197):19–22.

[53] Ristenpart, T., Tromer, E., Shacham, H., and Savage, S. (2009). Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 199–212, New York, NY, USA. ACM.

[54] Rodríguez-Henríquez, F., Saqib, N. A., Perez, A. D., and Koc, C. K. (2007). *Cryptographic algorithms on reconfigurable hardware*. Springer Science & Business Media, 1 edition.

[55] Rogaway, P., Bellare, M., and Black, J. (2003). OCB: A block-cipher mode of operation for efficient authenticated encryption. *ACM Transactions on Information and System Security (TISSEC)*, 6(3):365–403.

[56] Rosenberg, B. (2010). *Handbook of financial cryptography and security*. CRC Press, 1 edition.

[57] Sahai, A., Seyalioglu, H., and Waters, B. (2012). Dynamic credentials and ciphertext delegation for attribute-based encryption. In *Advances in Cryptology–CRYPTO 2012*, pages 199–217. Springer, Berlin, Heidelberg.

[58] Schaefer, E. (2009). An introduction to cryptography and cryptanalysis. *California's Silicon Valley: Santa Clara University*.

[59] Shamir, A. (1985). Identity-Based Cryptosystems and Signature Schemes. In Blakley, G. R. and Chaum, D., editors, *Advances in Cryptology: Proceedings of CRYPTO 84*, pages 47–53, Berlin, Heidelberg, Germany. Springer.

[60] Soong, T. W.-P. (2004). *Master Thesis. LPthread: A Work-efficient Thread Model for Loop Parallelization*. PhD thesis, University of Illinois at Urbana-Champaign.

[61] Takatoshi, N. and Daisuke, T. (2011). Implementation of Multiple-Precision Floating-Point Arithmetic Library for GPU Computing. In *Proceedings of the 23rd iasted international conference on parallel and distributed computing and systems*, pages 343–349. ACTAPRESS.

[62] Thorsteinson, P. and Ganesh, G. G. A. (2003). *NET security and cryptography*. Prentice Hall Professional, 1 edition.

[63] Tittle, E., Stewart, J. M., and Chapple, M. (2006). *CISSP: Certified information systems security professional study guide*. John Wiley & Sons, 1 edition.

[64] Unterluggauer, T., Wenger, E., Spreitzer, R., Werner, M., and Hölbling, R. (2014). "Pairings in C". *Library for Cryptographic Pairings.*

[65] Wan, Z., Liu, J., and Deng, R. H. (2012). HASBE: a hierarchical attribute-based solution for flexible and scalable access control in cloud computing. *IEEE Transactions on Information Forensics and Security*, 7(2):743–754.

[66] Wang, J. and Kissel, Z. A. (2015a). *Introduction to Network Security: Theory and Practice*. Wiley.

[67] Wang, J. and Kissel, Z. A. (2015b). *Introduction to network security: theory and practice*. John Wiley & Sons, 1 edition.

[68] Wang, X., Li, X., Zou, M., and Zhou, J. (2011). AES finalists implementation for GPU and multi-core CPU based on OpenCL. In *2011 IEEE International Conference on Anti-Counterfeiting, Security and Identification*, pages 38–42. IEEE.

[69] Yamanouchi, T. (2007). AES encryption and decryption on the GPU. *GPU Gems*, 3(1):785–804.

[70] Yanez-Sierra, J., Diaz-Perez, A., Sosa-Sosa, V., and Gonzalez, J. L. (2015). A digital envelope scheme for document sharing in a private cloud storage. In *2015 12th International Conference & Expo on Emerging Technologies for a Smarter World (CEWIT)*. IEEE.

[71] Yong, P., Wei, Z., Feng, X., Dai, Z.-h., Yang, G., and Chen, D.-q. (2012). Secure cloud storage based on cryptographic techniques. *The Journal of China Universities of Posts and Telecommunications*, 19(2):182–189.

[72] Yu, S. and Guo, S., editors (2016). *Big Data Concepts, Theories, and Applications*. Springer International Publishing, 1 edition.