CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS
DEL INSTITUTO POLITÉCNICO NACIONAL
DEPARTAMENTO DE COMPUTACIÓN

Esquema de cifrado para la ejecución de consultas en bases de datos cifradas

Tesis que presenta

Lil María Xibai Rodríguez Henríquez

Para obtener el grado de

Maestra en Ciencias

en Computación

Director de la Tesis: **Dr. Debrup Chakraborty**

México, D. F. Diciembre, 2009

A mi mamita por haberme impulsado desde que era niña a buscar nuevos horizontes,
por mostrarme con su ejemplo que la dedicación es la base del éxito
y por continuar esforzandose por ser el mejor apoyo en mi vida.

Agradecimientos

Gracias a *don pepe* por seguir trabajando arduamente para ayudar y apoyar a su hija.

Gracias a *la abuelita Rosita* por todas las enseñanzas que me brindó, por que fue y continua siendo mi *compañera mi casi hermana*.

A mis hermanos por creer en mi, por compartir conmigo sus experiencias y darme los más valiosos consejos ante las situaciones que presenta la vida.

A toda la familia por su apoyo incondicional, con especial cariño a mi primo René por compartir conmigo sus sueños y hacer suyos los mios.

A Raymundo Domínguez Colín por compartir conmigo estos dos años, por brindarme calma ante los muchos momentos de estrés vividos y ayudar a construir las alegrías.

A todos mis compañeros con los que comparti largas jornadas de trabajo, quienes estuvieron siempre dispuestos a dar su conocimiento, pero sobretodo les agradezco su amistad.

Gracias al Dr. Debrup Chakraborty, quien con mucho empeño y dedicación me ayudo hacer una realidad este trabajo de investigación, por estar dispuesto a compartir experiencias y cultura.

A mis lectores Dr. Guillermo Morales, Dr. Francisco Rodríguez; por sus valiosas aportaciones a mi trabajo.

A Sofia Reza por estar siempre disponible, dispuesta ayudar y pendiente de mi.

Al Cinvestav, por permitirme adquirir los conocimientos necesarios para desenvolverme en el arte de la Criptografía y en general en las Ciencias de la computación. Gracias por el apoyo económico que hizo posible la culminación de este trabajo.

A México por crear instituciones como el Cinvestav en las que podemos prepararnos y estudiar con profesores de alto nivel. Por contar con el Conacyt que me otorgo apoyo económico para poder realizar esta maestría.

Contenido

Lista de Figuras

Lista de Tablas

Resumen

En la actualidad la mayoría de compañías requieren manejar una gran cantidad de información para que su funcionamiento sea correcto. Con el objetivo de que estos datos sean almacenados, administrados y procesados; el dueño (*el cliente*) debe invertir recursos especializados con los que posiblemente no cuente. Por ello el cliente puede delegar esta tarea a un tercero (*el servidor*). Este modelo es conocido como *Database as a Service (DAS)* y ha cobrado popularidad e importancia para la tendencia del *cómputo nube*. Además brinda a los clientes la capacidad de *crear, almacenar, modificar y consultar* información desde un lugar remoto; todas estas operaciones son realizadas por el servidor como representante del cliente.

Al delegar la gestión de la base de datos a un tercero surgen nuevos problemas, entre los más importantes se encuentran la confidencialidad y la disponibilidad de los datos. Debido a que el servidor tiene acceso a toda la información, éste podría *divulgarla, modificarla o destruirla*. Por lo que surge una importante pregunta que debe ser resuelta en este contexto: ¿Bajo qué condiciones es posible delegar una base de datos a un servidor que no es confiable?

La solución obvia es el cifrado de la información. El cliente puede cifrar sus datos antes de almacenarlos en el servidor y con esto se soluciona el problema de confidencialidad. El servidor tiene otras responsabilidades además de almacenar la información, como es el ejecutar ciertas operaciones: procesar consultas, modificar, etc. El utilizar criptografía clásica trae consigo otros problemas, ya que un buen esquema de cifrado debe destruir la estructura natural presente en la información, lo que generalmente causa la reducción de la eficiencia en el procesamiento de consultas y otro tipo de operaciones como (*insertar, modificar, consultar,* etc.), debido a que no pueden ser efectuadas bajo los mecanismos que una base de datos utiliza normalmente.

Entonces, el reto consiste en construir un esquema de cifrado que permita que la información del cliente sea cifrada y almacenada en el servidor no confiable sin que la confidencialidad pueda ser violada, además las primitivas criptográficas utilizadas deben permitir que las operaciones que son de interés para el cliente puedan ser ejecutadas por el servidor en la información cifrada sin tener conocimiento de los datos originales.

El esquema de cifrado propuesto en esta tesis logra dos objetivos que se encuentran en conflicto, el brindar seguridad y también permitir que las consultas puedan ser procesadas. Nuestra propuesta utiliza diferentes tipos de primitivas criptográficas: un cifrador por bloques (AES), un cifrador basado en homomorfismo (Paillier 1999) y un cifrador que preserva el orden (Boldyreva et.al. 2009); los cuales se aplican a diferentes tipos de datos, permitiendo así que una gran variedad de operaciones puedan ser procesadas en información cifrada de manera eficiente.

Abstract

Now-a-days almost every company needs to handle an enormous amount of information for its proper functioning. Storage, maintenance and processing of such enormous amount of data needs lot of resources and expertise which the owner of the data (*the client*) may not have. Thus the client may delegate the duty of storage and maintenance of his/her data to a third party (*the server*). This model commonly known as as *Database as a Service (DAS)* has gained lot of popularity in the recent times, also this model is very important in the context of the emerging concept of *cloud computing*. The DAS model allows the client to perform operations like create, modify and retrieve from a remote location. These operations are performed by the server on behalf of the client.

Delegating the duty of storing and maintaining the database to a third party brings in some new problems. The privacy and availability of the data being of primary importance among them. As the server which stores and maintains the data has access to all the information, can potentially disclose, modify or damage the stored information. Thus the important question which has to be solved in this context is: under what conditions one can delegate a database to an un-trusted server.

The most obvious solution comes through data encryption. The client can encrypt the data before delegating it to the server and thus may provide a solution to the problem of privacy. However, normal encryption brings in some new problems. The server is not only responsible for storing the data, but it also performs certain operations on the data like processing queries, updating etc. on behalf of the client. A good encryption scheme is supposed destroy all natural structure present in the data which is generally detrimental to efficient query processing and performing other important operations on the stored data. Moreover, the normal database mechanisms to perform the operations (like *create, modify, retrieve* etc.) can no longer be used on the encrypted information.

Thus, the challenge is to come up with an encryption mechanism which allows the storage of the encrypted information in an un-trusted server without breach of privacy, moreover the encryption applied should be such that normal operations which are of interest to the client can be performed on the encrypted data by the server without the knowledge of the original data.

In this thesis we propose an encryption mechanism which achieves the two conflicting objectives of providing security and also being friendly to query processing. Our proposal uses different types of encryption primitives depending on the type of the data. In particular, we use a block cipher (AES), a homomorphic encryption scheme (Paillier 1999) and an order preserving encryption scheme (Boldyreva et al. 2009) in a specific manner such that it allows a variety of operations can be carried out on the encrypted data. Our encryption mechanism allows a large class of queries to be processed on the encrypted data with significant efficiency.

Capítulo 1

Introducción

> La imaginación es más importante que el conocimiento. El conocimiento es limitado. La imaginación circunda el mundo.
>
> *Albert Einstein*

En la actualidad la mayoría de empresas e instituciones realizan una cantidad considerable de transacciones al día, las cuales generan flujos de información, produciendo datos que deben ser registrados o actualizados.

Por ejemplo podemos pensar en un supermercado, en el que muchos clientes lo visitan y compran más de un producto. Esto hace necesario la actualización del inventario y las cantidades mostradas en las cuentas de ingreso.

Adicionalmente esta información que se recaba día con día es utilizada para generar estrategias con respecto al futuro de las empresas, ya que el personal puede consultar datos financieros, de ventas, de producción, etc. y en base a éstos tomar decisiones oportunas.

Esta es una de las razones por las cuales se considera que la información es considerada uno de los recursos más valiosos de toda organización. Para facilitar su administración se utilizan diferentes herramientas que desde luego incluyen sistemas de cómputo y particularmente bases de datos [1].

Sin embargo, la gestión de las bases de datos implica para las empresas e instituciones contar con computadoras dedicadas y personal calificado, generando altos costos de administración.

Por lo que **la subcontratación del servicio de administración de la base de datos** (*outsourced database*) ha surgido como una opción efectiva para la reducción de costos y ahorro de recursos, permitiendo así que las organizaciones dirijan sus esfuerzos al giro directo de la empresa, y deleguen en terceros el manejo de su base de datos. Este servicio es conocido en la literatura como *Database as a Service (DAS)*.

[1] Conjunto de datos pertenecientes a un mismo contexto y almacenados sistemáticamente para su uso posterior

1.1 Modelo de bases de datos subcontratadas

El servicio de administración de base de datos *(DAS)*, brinda a los clientes la capacidad de *insertar, almacenar, modificar, consultar y eliminar* información desde un lugar remoto; conocido como servidor [2]. Aprovechando el hardware con que el proveedor del servicio cuenta, que suele ser mucho más potente que el del cliente. Haremos referencia a estas capacidades como operaciones relacionales.

Bajo estas condiciones podemos plantear el siguiente escenario: supongamos que **el cliente** tiene una base de datos con millones de registros, y no tiene los recursos necesarios para almacenarla y gestionarla; entonces **el cliente** delega en una segunda entidad, **el proveedor**, el almacenamiento y mantenimiento de la misma. Sin embargo, **el cliente** no considera al proveedor confiable [3] (ver figura 1.1).

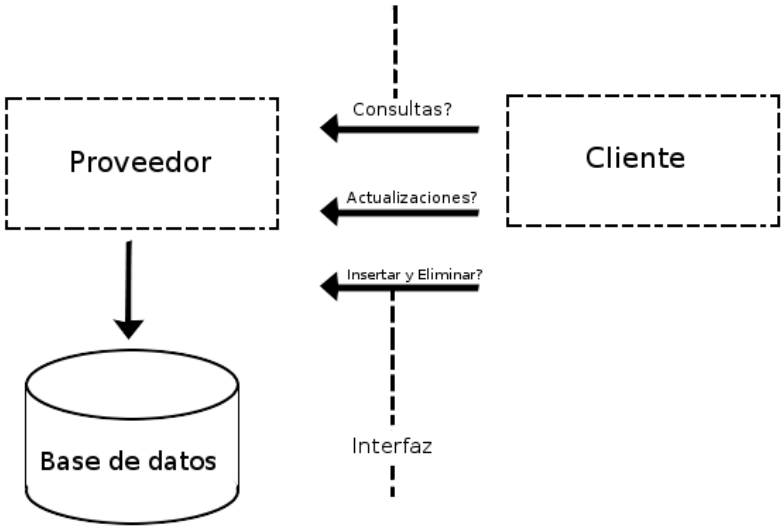

Figura 1.1: Modelo de bases de datos subcontratadas.

Al utilizar este esquema surge una serie de retos en diferentes áreas, entre las que se puede mencionar: infraestructura, tecnología y seguridad. Nuestro interés está enmarcado en la última área y puntualmente en la confidencialidad de los datos, con respecto a la cual existen al menos dos grandes retos:

- El proveedor debe proteger la información del cliente contra los posibles ataques de un tercero, entre los que se pueden mencionar: lectura, modificación o destrucción de los datos que le han sido confiados.
- Además de la seguridad contra terceros, el cliente debe proteger sus datos ante un

posible ataque del proveedor (lectura, modificación, destrucción o divulgación), si éste no es confiable.

Particularmente estudiaremos el segundo caso, donde el **proveedor** del servicio representa un posible oponente, y **el cliente** está confiando su información.

Por lo que este modelo nos conduce a una pregunta: ¿Bajo qué condiciones es posible delegar una base de datos sin que la confidencialidad de los datos sea comprometida?

Como un primer acercamiento a una solución, podemos pensar en técnicas criptográficas [2] para asegurar la confidencialidad de la información, permitiendo así que los datos cifrados sean almacenados en el servidor.

Sin embargo, aún cuando no se ha incursionado en el tipo de algoritmos que es posible utilizar, hay que resaltar que si el cliente cifra su información, el modelo enfrentará otro problema: las operaciones relacionales sobre la base de datos no pueden ser efectuadas bajo los procesos que ésta brinda de forma natural. Esto se debe a que todo algoritmo criptográfico busca inyectar aleatoriedad, la cual rompe con la estructura implícita de cualquier base de datos.

1.2 Planteamiento del problema

Tomando en cuenta lo descrito en secciones anteriores podemos definir con mayor claridad el problema al que nos enfrentamos:

En un modelo de bases de datos subcontratadas, donde el proveedor es en sí un posible oponente, cómo se puede construir un mecanismo que permita que la información del cliente sea cifrada y almacenada en el servidor, y que las funciones del servicio de administración de base de datos, sean realizadas remotamente de manera transparente y eficiente.

Así el modelo de bases de datos subcontratadas luciría como en la figura 1.2, en donde se debe destacar que la base de datos aparece cifrada y queremos mantener las funciones del servicio de administración de base de datos: *insertar, eliminar, actualizar, consultar*, etc.

Para poder mantener la funcionalidad que brinda una base de datos en claro, es necesario diseñar el esquema de cifrado de tal forma que permita mantener los mecanismos mediante los que la información es indizada cuando ésta se encuentra en claro. Ya que esto permite que el motor de la base de datos pueda realizar las operaciones relacionales de forma eficiente.

[2] Permiten que dos personas a las que nos referimos normalmente como Alicia y Beto puedan comunicarse en un canal inseguro de tal forma que el oponente Oscar no pueda comprender lo que se está diciendo. El canal inseguro puedes ser una línea telefónica, internet, etc [4] . En este caso en particular se considera que el proveedor del servicio es un posible oponente.

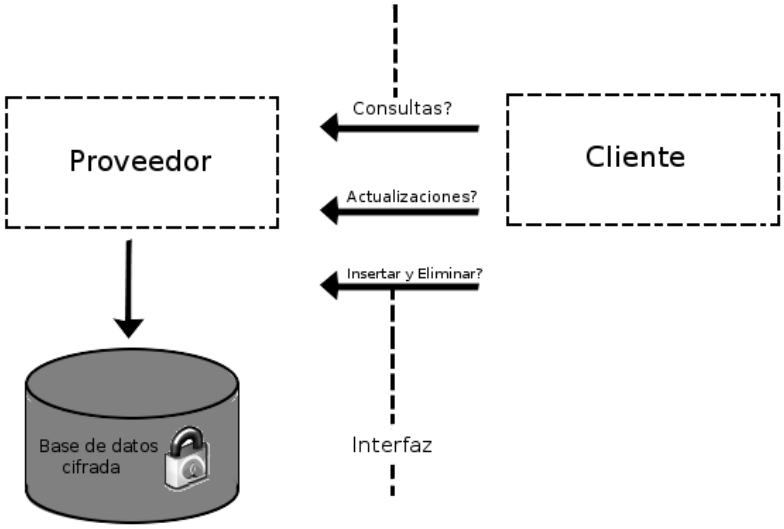

Figura 1.2: Modelo de bases de datos cifradas subcontratadas.

Dentro de estas funciones que realiza la base de datos, la operación de *consulta* tiene características que la convierten en un reto importante dentro del modelo, por lo que haremos un amplio análisis de ella. Para ello describiremos las expresiones que pueden requerir ser evaluados en una consulta, e idealmente todas ellas deben ser soportadas por nuestro sistema. Estas se listan enseguida:

- **Intervalos** Implica el uso de un límite inferior y otro superior. Por ejemplo si hablamos de los productos que ofrece un supermercado, y se requiere saber cómo se encuentran distribuidos los artículos con respecto a su precio, es decir, cuales productos se ofrecen a un costo mayor que un precio y menor que algún otro precio

- **Agregación** Implica el análisis de varios datos para obtener un sólo resultado, e.g. el máximo, mínimo, totales, etc. Una vez más en el ejemplo de los productos del supermercado hay alguno que tiene el precio más alto que todos, o bien dentro de una categoría específica (lácteos). Otro caso es cuando se desea conocer el total de los productos adquiridos por un cliente

- **Operaciones aritméticas** Algunas veces se requiere obtener sumas, restas, multiplicaciones e inclusive divisiones de la información numérica contenida en la base de datos

1.3 Bases de datos y estructura de las operaciones

El **cliente** resguarda la base de datos de su empresa o institución en el servidor administrado por el **proveedor**, la cual contiene registros que corresponden a la actividad que desempeña, lo que implica que cada base de datos es diferente. Sin embargo, es posible establecer características comunes de su estructura y patrones para cada una de las operaciones relacionales que se pueden realizar en ella.

La base de datos con la que trabaja el esquema propuesto es relacional. Este tipo de bases de datos almacena la información en tablas (*relaciones*). Cada tabla es una colección de filas (*registros*) y cada fila tiene el mismo conjunto de columnas (*atributos*). Finalmente cada columna tiene un tipo específico de dato, entre los que se puede mencionar: *alfanuméricos y numéricos*.

Por ejemplo, en la figura 1.3 podemos ver una base de datos básica que está compuesta por dos tablas, las cuales son: **Empleados** y **Departamentos**. La primera tiene por atributos: *el Identificador del Empleado, el Nombre del Empleado, el Salario del Empleado, Comisión, Dirección y el Identificador del departamento para el cual trabaja* y la segunda tiene los atributos: *Identificador de departamento, el Nombre del departamento*.

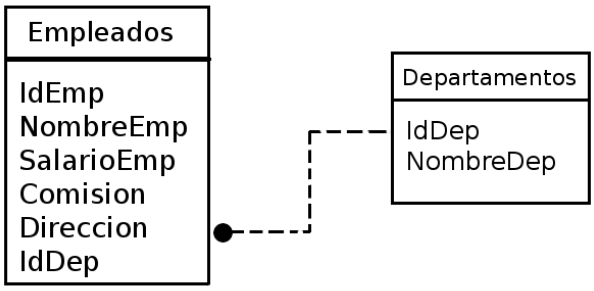

Figura 1.3: Base de datos en claro.

1.3.1 Estructura de las operaciones

El lenguaje bajo el cual se estructura una consulta, es un estándar *Structured Query Language* SQL, por lo que es posible identificar una sintaxis bien definida para cada una de las operaciones: *insertar, consultar, modificar, eliminar*; las cuales serán presentadas a continuación:

La operación *insertar* permite poblar una tabla con registros, y se escribe:

```
INSERT INTO <relación> VALUES ('texto',45,50)
```

La cláusula <*relación*> establece la tabla donde se ingresarán los datos. Cada tipo de dato tiene una forma de entrada: las constantes son valores alfanuméricos que deben ir entre comillas simples (') y en caso de ser un tipo numérico basta con colocar el valor.

Por ejemplo para ingresar un nuevo registro a la tabla **Departamentos** el enunciado quedaría como:

```
INSERT INTO Departamentos VALUES ('Fin','Finanzas')
```

El hecho de haber ingresado el registro correspondiente al departamento Finanzas, permite que puedan ser ingresados registros de empleados que trabajan en él, utilizando un enunciado como el siguiente:

```
INSERT INTO Empleados

VALUES ('23','Tom',10k,2k,'Av.Politecnico','Fin')
```

Sin embargo, si se quisiera introducir un registro de un empleado que trabaja para otro departamento que no se encuentre en la tabla de **Departamentos**, el motor de la base de datos no lo permitiría precisamente por que es relacional y se deben respetar las reglas de integridad.

La operación *consultar* permite seleccionar ciertos registros de una tabla o más y se escribe:

```
SELECT <atributos>, <función de agregación>

FROM <relaciones>

WHERE <predicados>

GROUP BY <atributos de agrupación>
```

Ahora revisaremos con mayor detalle cada una de sus cláusulas, algunas de ellas son opcionales, es decir, que no necesariamente deben formar parte de la consulta.

- **Atributos:** son las columnas que se quieren seleccionar de una o más tablas
- **Función de Agregación**: toma como parámetro un conjunto de valores y retorna sólo un resultado (opcional). E.g. **avg:** promedio, **min:** el valor minimo, **max:** el valor máximo, etc
- **Relaciones:** establecen las tablas de donde se obtendrán los atributos
- **Predicados:** definen las condiciones que debe respetar la consulta, como uniones (*joins, equijoins, outerjoins*, etc), funciones de rango, entre otras
- **Atributos de agrupación**: permiten presentar los resultados agrupados por algún atributo en particular (opcional)

Enseguida veremos algunos ejemplos de consultas aplicados a la base de datos mostrada en la figura 1.3.

- Una consulta básica sería *seleccionar el nombre y salario de todos los empleados*.

```
SELECT NombreEmp, SalarioEmp FROM Empleados
```

- Se puede complicar la consulta anterior si se requiere *seleccionar el nombre y salario de todos los empleados que trabajan en el departamento de Finanzas*.

```
SELECT NombreEmp, SalarioEmp FROM Empleados
WHERE IdDep = 'Fin'
```

- Una consulta que incluye uniones selecciona información de más de una tabla, por ejemplo, *seleccionar el nombre, salario y nombre del departamento para el que trabaja*.

```
SELECT NombreEmp, SalarioEmp, NombreDep
FROM Empleados AS emp JOIN Departamentos AS dep
ON emp.IdDep=dep.IdDep
```

- Un ejemplo de consulta de intervalos es *seleccionar el nombre y salario de los empleados cuyo monto es mayor que 5k* y luciría de la siguiente manera:

```
SELECT NombreEmp, SalarioEmp
FROM Empleados WHERE SalarioEmp>5k
```

- Un ejemplo de consulta de agregación es *calcular el total de salarios de los empleados que trabajan en el departamento de finanzas* y se escribe:

```
SELECT SUM(SalarioEmp) FROM Empleados WHERE IdDep = 'Fin'
```

Existe una variedad de consultas que pueden ser utilizadas, pero las que aquí han sido presentadas son representativas de lo que el cliente puede requerir, para mayor información de SQL remitimos al lector al apéndice C de este documento.

La operación *modificar* permite actualizar una o más columnas de un registro, y se escribe:

```
UPDATE <relación> SET <atributo_1> = <valor_1>
WHERE <atributo_2> = <valor_2>
```

Las cláusulas utilizadas por esta operación se detallan a continuación:

- **Relación:** indica la tabla que será modificada.

- **Atributo_1:** es la columna que será modificada.
- **Valor_1:** es el nuevo valor que debe tomar la columna.
- **Atributo_2:** establece que los registros que serán modificados deben contener en esta columna el valor indicado por **valor_2**.

Por ejemplo, el cliente puede requerir *modificar el salario del empleado cuyo identificador es igual a 23 con el valor de 15k*, el enunciado luciría así:

```
UPDATE Empleados SET SalarioEmp = 15k
WHERE IdEmp = '23'
```

Finalmente la operación *eliminar* permite borrar un registro o más, y se escribe:

```
DELETE FROM <relación> WHERE <atributo> = <valor_1>
```

Las cláusulas que se incluyen en este enunciado son:

- **Relación:** indica la tabla de la que se eliminarán registros.
- **Atributo:** establece que los registros que serán eliminados deben contener en esta columna el valor indicado por **valor_1**.

Si suponemos que el cliente requiere eliminar un registro de la tabla **Empleados** debido a que el empleado ya no labora para la empresa, el enunciado luciría así:

```
DELETE FROM Empleados WHERE IdEmp = '23'
```

En esta sección se han descrito las operaciones que deben poder ejecutarse en una base de datos aún cuando ésta se delegue, pero cuando la base de datos se encuentra cifrada mantener esta funcionalidad constituye un problema difícil, por lo que actualmente se están dedicando muchos esfuerzos para lograrlo. En la próxima sección se presenta un panorama general del estado del arte.

1.4 Esquemas existentes

Actualmente existe una serie de esquemas que trabajan en el modelo de bases de datos subcontratadas y establecen mecanismos que permiten implementar algunas de las operaciones relacionales del servicio de administración de la base de datos.

En general, estas técnicas brindan altos niveles de seguridad, pero tienden a sacrificar la funcionalidad de la base de datos, pues le restan velocidad y eficiencia.

Las soluciones ofrecidas hasta ahora se pueden clasificar en dos categorías [5]:

a) Enfoque basado en nuevas técnicas de cifrado

Estas técnicas permiten el uso de operadores aritméticos y/o de comparación en la información cifrada, por lo que pueden utilizarse para ejecutar consultas que incluyen agregación [6] [7] [8]. Sin embargo, no permiten incluir uniones, es decir, seleccionar datos contenidos en más de una tabla.

Existen otros trabajos que permiten preservar el orden en el cifrado, y con ello es posible ejecutar comparaciones, ordenamientos, selecciones, agrupamiento; pero no agregación [9] [10].

b) Enfoque basado en información escondida

Estas ténicas se basan en almacenar información auxiliar junto con la información cifrada para facilitar la evaluación de comparaciones y operadores aritméticos, al igual que ordenamientos, pero no permiten realizar consultas de agregación [11] [12] [13].

Existen pocos trabajos que hagan uso de ambos enfoques para la ejecución de consultas cifradas, entre los que podemos citar [6],[14]. En esta tesis proponemos un esquema de cifrado que permite ejecutar consultas de manera eficiente en bases de datos relacionales (cifradas), para lo cual hacemos uso de varias técnicas de cifrado que permiten implementar prácticamente todas las funciones del servicio de administración de base de datos.

1.5 Organización y alcance de la tesis

El objetivo principal de esta tesis es construir un esquema de cifrado que permita resolver los problemas relacionados a la ejecución de consultas en *bases de datos cifradas subcontratadas*. Para ello, fue necesario implementar varios algoritmos de cifrado: un cifrador por bloques (AES), un cifrador basado en homomorfismos (Paillier) y un cifrador que preserva el orden (Boldyreva et.al.); posteriormente se unificaron en un solo esquema, el cual permite al **usuario** realizar operaciones relacionales en la base de datos cifrada de forma remota.

En los capítulos siguientes se describe con mayor detalle esta problematica y los mecanismos que fueron encontrados e implementados para resolverla. A continuación se presenta un mapa del contenido del resto de los capítulos.

En el segundo capítulo se describen varios esquemas criptográficos que buscan resolver algún tipo de consultas en información cifrada, permitiendo así, que el lector tenga idea del estado del arte hasta el momento de realización de esta tesis.

Posteriormente, en el tercer capítulo se explica qué es el cifrado basado en homomorfismo, así como algunos esquemas bien conocidos que brindan esta propiedad, enseguida se trata en detalle el homomorfismo de Paillier [15], que es uno de los algoritmos utilizados en el esquema propuesto y finalmente se explica cómo es posible utilizarlo en consultas ejecutadas sobre bases de datos cifradas.

En el cuarto capítulo se aborda el algoritmo de cifrado con preservación de orden, una propuesta hecha por Boldyreva et.al. [10], que permite realizar un cifrado que mantiene el orden numérico del texto en claro; después se explica cómo este algoritmo puede ser utilizado para efectuar consultas que manejan intervalos en texto cifrado y finalmente se describen los cambios que se realizaron para su implementación y uso en bases de datos cifradas.

El esquema de cifrado completo se presenta en el capítulo cinco, se explica cómo conjuntando el algoritmo de cifrado de Paillier, el algoritmo de cifrado que preserva el orden de Boldyreva et.al. y el algoritmo de cifrado por bloques AES; es posible obtener un esquema lo suficientemente robusto para el manejo de prácticamente todas las consultas que se pueden generar en una base de datos en claro.

El capítulo seis aborda los detalles de la implementación del esquema, el funcionamiento de nuestra aplicación que permite probar el concepto del esquema propuesto, la tecnología utilizada y finalmente los resultados de los experimentos realizados.

Por último el capítulo siete contiene los resultados obtenidos en la elaboración de la tesis y las conclusiones a las que se llegó en base a éstos. También se tratan algunas mejoras que permitirían que la aplicación fuera más eficiente y algunas líneas de investigación como trabajo futuro.

Además, se incluyen cuatro apéndices que permiten que el lector se familiarice con algunos conceptos básicos para la mejor comprensión del manuscrito; el primer apéndice trata conceptos preliminares acerca de la aritmética utilizada en las operaciones involucradas en el esquema, el segundo contiene la descripción de los algoritmos básicos utilizados: AES y base64, el tercero presenta las operaciones permitidas en una base de datos relacional en claro y finalmente el cuarto apéndice incluye los *manuales* de la aplicación e información relacionada.

Capítulo 2

Análisis de los esquemas existentes

> Los grandes conocimientos engendran las grandes dudas.
>
> *Aristóteles*

El uso de técnicas criptográficas permite resolver una gama importante de problemas, entre los que cabe mencionar la confidencialidad de la información. Sin embargo, introduce un aumento en el procesamiento reduciendo así, el nivel de desempeño de las aplicaciones.

Por ello el manejo de información cifrada requiere la correcta elección de los algoritmos criptográficos, así como la búsqueda de un modelo adecuado de almacenamiento y recuperación de la información y el desarrollo de técnicas que permitan que los datos sean consultados.

En este sentido, ha habido muchas investigaciones dirigidas a introducir seguridad y confidencialidad en las bases de datos, desarrollando nuevos tipos de cifrado que permiten mantener la funcionalidad que se tiene en las bases de datos en claro, entre las que se puede mencionar [2] [6] [1] [16] [17] [18] [19].

En este capítulo se hace una revisión detallada de tres esquemas que trabajan bajo el modelo de *Database as a Service* y que son de interés para nuestro trabajo. Se hace énfasis en las técnicas que han sido propuestas para realizar consultas bajo el estándar SQL en bases de datos relacionales cifradas.

2.1 Búsqueda de palabras en documentos cifrados

El esquema presentado en [1] está orientado a la búsqueda de palabras en documentos cifrados, bajo el modelo de subcontratación y trabaja de acuerdo al siguiente escenario:

Sea **Alicia** la dueña de un conjunto de documentos $D = \{d_1, d_2, \cdots, d_n\}$. Un docu-

mento está compuesto por un conjunto de palabras $d = \{w_1, w_2, \cdots, w_n\}$ donde cada palabra w_i pertenece al vocabulario de todas las posibles palabras W, es decir, $w_i \in W$. Alicia decide almacenar todos sus documentos en un servidor remoto, así se convierte en **cliente** del **proveedor** que presta el servicio de almacenamiento y administración de información. Puesto que el servidor no es confiable Alicia decide enviar los documentos cifrados.

Cada documento es cifrado a nivel de palabras. Periódicamente Alicia realiza una consulta con el fin de que el servidor devuelva un subconjunto de documentos que cumplan con las condiciones establecidas, por ejemplo el seleccionar todos los documentos que contengan una palabra en específico.

2.1.1 Esquema de cifrado

El esquema de cifrado propuesto en [1] funciona de forma intuitiva calculando el XOR exclusivo entre el texto en claro y una secuencia de bits pseudoaleatorios que mantienen una estructura especial, la cual permite la búsqueda de palabras en los documentos cifrados. El algoritmo de cifrado hace uso de la siguiente notación:

- d: Es un documento en claro conformado por n palabras $d = \{w_1, w_2, \cdots, w_n\}$
- dc: Es un documento conformado por n palabras cifradas $dc = \{c_1, c_2, \cdots, c_n\}$
- $x||y$: Representa la concatenación de x, y
- $x \oplus y$: Es la suma bit a bit módulo 2 entre x y y

Además hace uso de varias primitivas criptográficas que se describen a continuación:

- Sea $G : \mathcal{K}_G \rightarrow \{0,1\}^*$ un *generador de secuencias pseudoaleatorias*, donde el espacio de llaves es $\mathcal{K}_G$.
- Sea $F : \mathcal{K} \times \{0,1\}^{n-m} \rightarrow \{0,1\}^m$ una *función pseudoaleatoria*, donde $\mathcal{K}_F$ es el espacio de llaves y $n > m$. Escribiremos $F_k(x)$ en lugar de $F(k, x)$
- Sea $f : \mathcal{K}_F \times \{0,1\}^* \rightarrow \mathcal{K}_F$ una *función pseudoaleatoria* adicional e independiente de F. Escribiremos $f_k(x)$ en lugar de $f(k, x)$
- Sea $E : \mathcal{K}_E \times \{0,1\}^n \rightarrow \{0,1\}^n$ un cifrador por bloques, donde $\mathcal{K}_E$ es el espacio de llaves. Escribiremos $E_k(x)$ en lugar de $E(k, x)$

Los autores en [1] proponen realizar el cifrado del documento con el algoritmo 2.1:

Algoritmo de cifrado(k, k', k'', d)

1. **For** todas $w_i \in d$
2. $X_i \leftarrow E_{k''}(w_i)$
3. $X_i \leftarrow L_i||R_i$ // Donde L_i tiene $n - m$ bits.
4. $k_i \leftarrow f_{k'}(L_i)$ // Se genera la llave k_i requerida para la función F
5. $S_i \leftarrow G(k)$ // La longitud de S_i es de $n - m$ bits
6. $T_i \leftarrow S_i||F_{k_i}(S_i)$// Donde F es una función pseudoaleatoria
7. $C_i \leftarrow E_{k''}(w_i) \oplus T_i$
8. **Return** C_i

Tabla 2.1: Algoritmo de cifrado en [1]

El algoritmo de cifrado de la tabla 2.1 recibe como entrada tres llaves: $k \in \mathcal{K}_G$, $k' \in \mathcal{K}_F$ y $k'' \in \mathcal{K}_E$ y el documento en claro. El primer paso consiste en realizar un pre-cifrado de la palabra w_i que está siendo analizada. Posteriormente el usuario calcula la llave k_i haciendo uso de la función pseudoaleatoria f a la que se le brinda como entrada L_i que corresponde a los primeros $n-m$ bits de la palabra w_i cifrada. El cálculo de S_i se realiza haciendo uso de un *generador pseudoaleatorio*, esta cadena tiene una longitud de $n - m$ bits. Los otros m bits restantes son calculados con la *función pseudoaleatoria* $F_{k_i}(S_i)$. De este modo es posible calcular la etiqueta $T_i := S_i||F_{k_i}(S_i)$ y realizar la operación final de cifrado $C_i = E_{k''}(w_i) \oplus T_i$.

El esquema que se ha descrito es presentado en la figura 2.1.

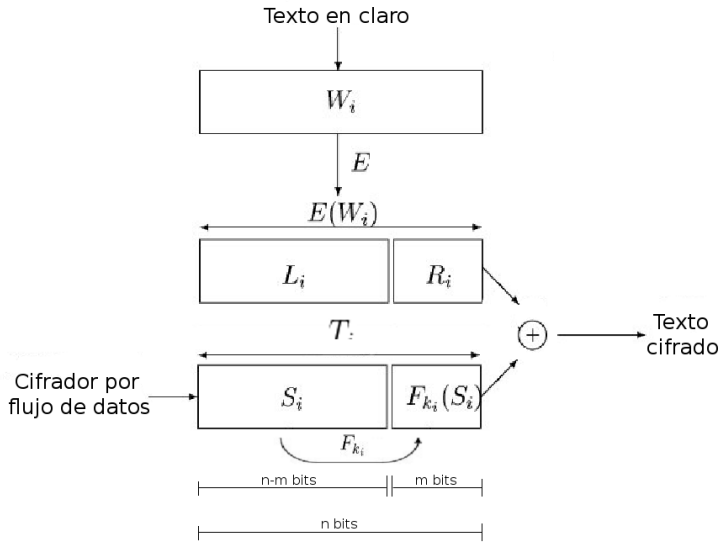

Figura 2.1: Esquema de cifrado en [1]

2.1.2 Consulta de información

Cuando Alicia quiere realizar una búsqueda de documentos que contienen la palabra w_i, calcula $X := E_{k''}(w_i)$. Este dato es enviado al servidor. Posteriormente éste aplica el algoritmo de búsqueda de la tabla 2.2.

Algoritmo de búsqueda(X)

1. **For** todas $c_i \in d_c$ // Sea d_c un documento cifrado
2. **If** $X \oplus c_i$ es de la forma $S_i || F_{k_i}(S_i)$
3. **Return** d_c

Tabla 2.2: Algoritmo de búsqueda en [1]

Una vez que Alicia obtiene todos los documentos cifrados que contienen la palabra w_i, ella procede a descifrar, para lo cual aplica el algoritmo mostrado en la tabla 2.3.

Algoritmo de descifrado(k, k', k'', dc)

1. **For** todas $c_i \in d_c$ // d_c Documento cifrado
2. $S_i \leftarrow G(k)$ // La longitud de S_i es de $n - m$ bits
3. $L_i \leftarrow S_i \oplus c_i$ // Se utilizan los $n - m$ bits más significativos
4. $k_i \leftarrow f_{k'}(L_i)$
5. $T_i \leftarrow \langle S_i, F_{k_i}(S_i) \rangle$
6. $X_i \leftarrow c_i \oplus T_i$
7. $w_i \leftarrow E^{-1}_{k''}(X_i)$
8. $d \leftarrow D || w_i$
9. **Return** D

Tabla 2.3: Algoritmo de descifrado en [1]

El algoritmo de descifrado de la tabla 2.3 recibe como entrada las mismas tres llaves que el algoritmo de cifrado de la tabla 2.1 y el documento cifrado. Para descifrar Alicia genera S_i usando el generador pseudoaleatorio (ya que Alicia conoce la llave), y una vez que se tiene S_i puede recuperarse L_i realizando un XOR entre los $n - m$ bits de C_i con S_i (ver línea 3 del algoritmo 2.3). Con ello es posible conocer la k_i utilizada y recuperar T_i. Enseguida obtiene X_i al aplicar un XOR entre c_i y T_i. Finalmente se debe aplicar el algoritmo de descifrado a X_i para obtener el texto en claro w_i.

2.2 Esquema de cifrado para bases relacionales de datos utilizando homomorfismo

El esquema presentado en esta sección permite realizar varios tipos de consultas [6], acercándose a la funcionalidad esperada en una base relacional de datos en claro.

- Consultas que implican uniones (*joins*)
- *Consultas de agregación (total, promedio, conteo)*. Este tipo de consultas permite realizar operaciones aritméticas
- *Consultas de intervalos*. Para poder efectuar consultas de este tipo los autores proponen índices basados en particiones del dominio del texto en claro

2.2.1 Homomorfismo de privacidad

Los homomorfismos son funciones $E_k : T \rightarrow T'$ del espacio de textos en claro T al espacio de textos cifrados T'; y haciendo uso de la llave k permiten realizar un conjunto F' de operaciones sobre los datos cifrados (en T') sin tener ningun conocimiento de la función de descifrado D_k.

Al aplicar este conjunto de operaciones F' se obtiene como resultado una salida que corresponde a la cifra del texto en claro que sería obtenido si el conjunto de operaciones F hubiera sido utilizado en T (texto en claro). Con el conocimiento de D_k es posible descifrar esta salida [20].

La aplicación de este tipo de funciones para procesar información cifrada es conocida como homomorfismos de privacidad (*Privacy homomorphisms (PH)*) y fueron introducidos de manera formal en [8].

El uso de esquemas de cifrado basados en homomorfismos de privacidad se trata de forma detallada en el capítulo 3.

2.2.2 Esquema de cifrado

Los autores proponen hacer una clasificación de los atributos en base al tipo de consultas en los que se verán involucrados. A continuación se presenta las categorías que son planteadas en este artículo:

- De Agregación: Son aquellos que se verán involucrados en totales u operaciones aritméticas. Estos son cifrados haciendo uso del homomorfismo de privacidad planteado en el artículo

- Cifrados determinísticamente: Son aquellos que participan en consultas de igualdad, *joins*, etc. Se cifran con un algoritmo determinístico
- De Partición: Permiten hacer consultas de intervalos o selecciones de *joins* para atributos numéricos

Cifrado basado en homomorfismo

El esquema de cifrado se describe a continuación:

- La llave está constituida de la siguiente manera $k = (p, q)$, donde p y q son números primos elegidos por el cliente, estos dos números son privados
- Se calcula $n = p \cdot q$, este valor calculado es conocido por el servidor; la seguridad según describen los autores se basa en la dificultad del problema de factorización
- El cifrado es realizado bajo la siguiente primitiva $E_k(a) = (a \mod p, a \mod q)$, donde $a \in \mathbb{Z}_n$
- La primitiva de descifrado es $D_k(d_1, d_2) = d_1 q q^{-1} + d_2 p p^{-1} \pmod n$, donde el inverso multiplicativo de q es tal que $qq^{-1} = 1 \pmod p$ y el inverso multiplicativo de p es tal que $pp^{-1} = 1 \pmod q$

Este esquema tiene fundamento en el teorema chino del residuo y referimos al lector a consultarlo en [4].

Ejemplo:

Sea $p = 5$ y $q = 7$ y $n = p \cdot q = 5 \cdot 7 = 35$ la llave es $k = (5, 7)$. Si el cliente quiere sumar $a_1 = 5$ y $a_2 = 6$, entonces el cifrado de ambos números es $E(a_1) = (0, 5)$ y $E(a_2) = (1, 6)$ los cuales son almacenados en la base de datos. El servidor es capaz de realizar la suma sin tener que descifrar los datos, lo cual es equivalente a $E(a_1) + E(a_2) = (0+1, 5+6) = (1, 11)$. El resultado $(1, 11)$ es enviado al cliente, y éste puede descifrar utilizando la primitiva:

$(d_1 q q^{-1} + d_2 p p^{-1}) \mod n = (1 \cdot 7 \cdot 3 + 11 \cdot 5 \cdot 3) \mod 35 = 11$; donde 11 es la suma de 5 y 6.

Consultas de agregación

Supongamos que se quiere calcular el salario neto de un trabajador, el cual está dado por la suma del salario y de la comisión $SUM(salario + comision)$, datos que están contenidos en la tabla de empleados.

Habiendo utilizado el esquema de homomorfismo explicado en la sección anterior, la tabla empleados (cifrada) contiene dos atributos que conforman la columna *salario* de la tabla en claro: ($salario\_p$, $salario\_q$) que corresponden a $salario \mod p$ y $salario \mod q$ respectivamente. De forma similar, la tabla empleados contiene dos atributos para la comisión: ($comision\_p$, $comision\_q$).

La consulta original puede ser evaluada por el servidor al calcular la operación de agregación componente a componente:

```
SELECT SUM(salario_p + comision_p) as s1,
SUM(salario_q + comision_q) as s2
FROM empleados
```

El cliente puede descifrar el resultado evaluando la expresión:

$$s1 \mod p \times q \times q^{-1} + s2 \mod q \times p \times p^{-1} \pmod n$$

Comparaciones lógicas

Los autores clasifican las comparaciones lógicas en dos: relación de igualdad y relación de desigualdad. En el primer caso la consulta puede ser evaluada directamente haciendo uso del cifrado determinístico (AES). Para la relación de desigualdad, utilizan la estrategia propuesta en [2], la cual se describe a continuación.

La tabla de empleados contiene los siguientes atributos: *id del empleado, nombre, salario, ciudad y id del departamento*; y **el cliente** requiere realizar consultas que implican intervalos.

Para poder evaluar una condición como $salario > 60k$, un índice es almacenado como atributo en la tabla cifrada, que se llama $salario\_id$. Este índice se obtiene de hacer particiones previas del dominio de los salarios en claro. Si por ejemplo los salarios de los empleados se encuentran en el rango de $[0, 100k]$, un esquema de posibles particiones es el siguiente:

$salario = \{[0k, 25k], (25k, 50k], (50k, 75k], (75k, 100k]\}$;

cada partición tiene asociado un índice, que puede ser obtenido con una función *picadillo* [1].

La tabla 2.4 muestra la asociación de índices. Un empleado en particular cuyo salario es de $70k$, tendrá un índice en la tabla cifrada igual a 81.

[1] Este tipo de funciones permite obtener una *huella digital* del mensaje que recibe como parámetro de entrada. En caso de que el mensaje sea alterado la huella digital ya no será válida. Aún cuando la información es almacenada en un lugar inseguro es posible revisar que no haya sido alterada al calcular de nuevo la función picadillo y compararla con la obtenida la primera vez [4].

Particiones	ID
$[0, 25k]$	59
$(25k, 50k]$	49
$(50k, 75k]$	81
$(75k, 100k]$	7

Tabla 2.4: Tabla de particiones de salario de empleados

Consultas de intervalos

Bajo el esquema anterior, cuando un cliente quiere seleccionar los empleados cuyo salario es mayor que $60k$, la consulta debe ser re-escrita para evaluar la siguiente condición: $salario\_id = 81\ OR\ salario\_id = 7$ en el servidor. Si el registro cumple con está nueva condición es retornado al cliente para que efectúe el descifrado y compruebe si es parte del conjunto de datos que deseaba obtener.

Este último paso es requerido porque la consulta re-escrita hace uso de una expresión más amplia que la original, lo que implica que algunos de los registros devueltos no son parte de la respuesta correcta, estos registros que no deben ser tomados en cuenta son conocidos como *falsos positivos*.

2.3 Esquema que permite consultas básicas en bases de datos relacionales cifradas

Una propuesta para realizar consultas básicas de la forma:

`SELECT` $\cdots$ `FROM` T `WHERE` $A_j = v$

es presentada en [3], donde T es una tabla de la base de datos y sus columnas se encuentran en el conjunto A, por lo que la expresión $A_j = v$ implica que se están buscando los registros en T donde el valor de la columna A_j sea igual a un valor particular v.

2.3.1 Esquema de cifrado

Consideraremos una función de cifrado por bloques $E : \{llaves\} \times \{mensajes\} \rightarrow \{cifrados\}$. Para cada lave $s \in \{llaves\}$ escribiremos E_s para referirnos a la función de cifrado $\{mensajes\} \rightarrow \{cifrados\}$ propia de la llave s utilizando la función E.

Para cifrar cada celda se utiliza un esquema redundante. Específicamente para cada

celda $T_{i,j}$ se tiene una celda cifrada que está constituida por dos elementos $T'_{i,j} = (T'_{i,j}\langle 1\rangle, T'_{i,j}\langle 2\rangle)$.

El primer elemento resulta de aplicar un cifrador por bloques $E_{s_1}()$ sobre la celda $T_{i,j}$, donde s_1 es la llave secreta. El segundo elemento actúa junto con el primero como medio de verificación para saber si la tupla satisface las condiciones de la consulta. Para calcularlo se hace uso del mismo cifrador por bloques, utilizando una segunda llave s_2.

Para crear la tabla T en la base de datos, el usuario realiza los siguientes pasos:

- Primero elige dos llaves secretas s_1, s_2 de forma independiente y uniforme
- Por cada celda $T_{i,j}$ elige $r_{i,j}$ como una secuencia de k_2 bits elegidos aleatoriamente en $\{0,1\}^{k_2}$
- Por cada celda $T_{i,j}$ almacena en el servidor:

$$\begin{aligned} T'_{(i,j)} &\triangleq (T'_{i,j}\langle 1\rangle, T'_{i,j}\langle 2\rangle) \\ &= (E_{s_1}(T_{i,j}, ri, j), E_{E_{s_2}(T_{i,j},j)}(E_{s_1}(T_{i,j}, r_{i,j}))) \end{aligned}$$

Protocolo de consultas

Sea A_j el j-ésimo atributo de T. Supongamos que se quiere efectuar una consulta que tiene el siguiente formato:

`SELECT` $A_{j1}, \cdots, A_{jl}$ `FROM` T `WHERE` $A_{j_0} = v$.

Para resolver esta consulta, el cliente calcula $q = E_{s_2}(v, j_0)$ y envía al servidor j_0 y q además de las columnas que desea seleccionar $A_{j1}, \cdots, A_{jl}$.

El servidor revisa en cuáles de las celdas que van desde $i = 1, \cdots, n$, la condición $T'_{i,j_0}\langle 2\rangle = E_q(T'_{i,j_0}\langle 1\rangle)$ se cumple.

Por cada celda que haya cumplido la condición anterior, el servidor devuelve la tupla correspondiente al cliente, y finalmente éste descifra usando la llave s_1 y descarta los k_2 bits que le agregó al texto en claro.

2.3.2 Mejoras al esquema

Los autores presentan dos posibles mejoras, en cuanto a:

Seguridad

El esquema anterior revela qué atributos están siendo probados, es decir, los contenidos en la clausula **where**. Para evitar que esta información se filtre los autores proponen permutar los atributos en la tabla cifrada de forma uniformemente aleatoria.

Eficiencia en las consultas

En el esquema descrito, se obliga al servidor a probar fila por fila para determinar si la tupla cumple con las condiciones de la consulta. Lo cual es ineficiente si se maneja un número grande de registros. Por lo que los autores sugieren agregar un mapeo de metadatos numéricos que permiten sustituir la búsqueda lineal por una búsqueda binaria.

2.4 Comparación de los esquemas existentes

Al trabajar con bases de datos cifradas, existe un compromiso entre seguridad y funcionalidad que debe ser resuelto. Como se ha mostrado en secciones anteriores, existen varios enfoques propuestos, pero aunque éstos ofrecen altos niveles de seguridad desde el punto de vista de la criptografía, no resultan eficientes en términos de bases de datos. Esto se debe a que en su mayoría ofrecen una complejidad lineal con respecto al tamaño de la base de datos, lo cual es ineficiente si se maneja un número grande de registros. Por lo que resulta de interés buscar mecanismos que tengan una complejidad logarítmica o al menos sublineal con respecto al número de registros en la base de datos.

En esta sección se analizan las ventajas y desventajas ofrecidas por los esquemas mostrados en el capítulo y los retos que deben ser resueltos para lograr un equilibrio entre seguridad y eficiencia.

Xiaodong et. al. en [1], muestran un esquema que permite hacer búsquedas en documentos cifrados. Este esquema resulta costoso ya que requiere $O(n)$ operaciones criptográficas por documento donde n es el número de palabras.

El esquema podría ser adaptado para trabajar en base de datos cifradas, sin embargo, la complejidad sería lineal con respecto al número de registros.

Hacigümüs et. al. muestran que es posible ejecutar un conjunto amplio de consultas al utilizar el esquema de cifrado que proponen en [6]. Sin embargo, éste presenta algunas desventajas:

- En cuanto al homomorfismo: la primitiva de cifrado es $E_k(a) = (a \mod p, a \mod q)$, donde $a \in \mathbb{Z}_n$, lo que implica que todo elemento menor que p y q producirá como texto cifrado su mismo valor.

  Por ejemplo, sea $p = 5$ y $q = 7$ y $n = p \cdot q = 5 \cdot 7 = 35$. Si el cliente quiere cifrar $a_1 = 4$, el cifrado es $E(a_1) = (4, 4)$. Lo cual constituye un problema significativo, debido a que por aspectos de seguridad p y q deben ser primos grandes. Y definitivamente la mayoría de números que se cifren serán menores que ellos

- En cuanto a la seguridad: el homomorfismo presentado en este esquema fue propuesto por Rivest et.al. en [8], sin embargo, Brickell y Yacobi en [21] demostraron que este esquema puede ser comprometido bajo el ataque de texto en claro conocido

- En cuanto a las consultas de intervalo: Su solución implica dividir los dominios de los atributos en particiones y asociarlas con identificadores. Esto implica un compromiso entre seguridad y eficiencia, ya que si las particiones son grandes una cantidad menor de información es revelada, pero el servidor enviará un mayor número de falsos positivos. Si las particiones son pequeñas, disminuye el número de falsos positivos pero se incrementa la información que se filtra

Yang et. al. en [3], presentan un esquema que tiene la ventaja de obtener para cada celda un texto cifrado diferente, aunque el texto en claro que la generó sea el mismo. Esto se debe a que cada atributo que va ser cifrado se le agrega una secuencia de k_2 bits pseudoaleatorios.

Lo anterior es una característica muy deseable, ya que si el oponente observa los datos cifrados no puede distinguir si dos celdas corresponden al mismo texto en claro. Sin embargo, desde el punto de vista de las bases de datos implica que no es posible mantener un esquema relacional.

El esquema básico tiene una complejidad lineal, y aun cuando ofrece una mejora para que la búsqueda sea binaria, para lograrlo ésta incrementa la comunicación, pues para resolver una consulta es necesario que los mensajes cliente servidor se efectúen en dos pasos, lo cual lo vuelve ineficiente. Además está enfocado a consultas muy sencillas del tipo `SELECT` $\cdots$ `FROM` T `WHERE` $A_j = v$ y no estudia la demás gama de consultas que sí es tratada por Hacigümüs et al. en [6].

Existe otro tipo de propuesta para lograr de forma eficiente consultas que implican intervalos, haciendo uso de esquemas de cifrado que preservan el orden, este concepto fue introducido por Agrawal et. al.[9].

Este enfoque también permite asociar índices al contenido de las tablas y el procesamiento de la consulta es igual al realizado en texto en claro, ya que para resolver la búsqueda se utiliza directamente el motor de la base de datos.

Posteriormente, fue publicado en Boldryreva et. al. [10] otro esquema de cifrado de preservación del orden, en donde se hace un estudio de la seguridad ofrecida desde el punto de vista de la criptografía.

Hasta ahora todos los esquemas tratados se basan en criptografía de llave simétrica, sin embargo, existen otras propuestas que utilizan criptografía de llave pública que buscan resolver alguno de los problemas relacionados con la ejecución de consultas en bases de datos cifradas, entre los que cabe mencionar: [22] [23] [24] [25].

Es en este contexto que nuestro trabajo busca ofrecer un esquema de cifrado lo suficientemente robusto para ofrecer todas la características de una base de datos relacional en claro, atacando problemas como: inserción, actualización, consultas con *joins*, consultas de agregación, consultas de intervalos y eliminación.

En el próximo capítulo abordaremos el cifrado basado en homomorfismos de privacidad, estudiaremos detenidamente el algoritmo de Paillier, ya que como ha sido mencionado constituye uno de los bloques básicos de nuestra propuesta.

Capítulo 3

Homomorfismos de privacidad

La desconfianza es la madre de la seguridad.

Aristofanes

El cifrado basado en homomorfismos es una técnica criptográfica donde es posible realizar una operación algebraica específica en el texto en claro al efectuar la misma o posiblemente otra operación en el texto cifrado.

Esta propiedad puede ser usada para crear sistemas de votación segura [26] [27], funciones *picadillo* resistentes a colisiones [28] y esquemas de recuperación de información privada [6] [29]. Es en esta última área es que se enmarcan los sistemas que trabajan bajo el modelo de *DAS*.

En este capítulo se estudia formalmente los conceptos de homomorfismo, isomorfismo y homomorfismos de privacidad; posteriormente se revisan algunos criptosistemas que hacen uso de estas propiedades, además se revisa con detalle el algoritmo de cifrado de Paillier y se explica cómo puede ser utilizado para el manejo de información cifrada.

3.1 Homomorfismo

Un homomorfismo es una función que va de un objeto matemático a otro de la misma categoría (grupos (ver apéndice A), anillos, espacios vectoriales) y es compatible con toda la estructura. La palabra *homomorfismo* viene del griego *homos* que significa **mismo** y *morfos* que significa **forma**.

Definición 1. *Un homomorfismo desde un grupo* $\langle \mathbb{G}, o_{\mathbb{G}} \rangle$ *hacia un grupo* $\langle \mathbb{H}, o_{\mathbb{H}} \rangle$*, donde* $o_{\mathbb{G}}, o_{\mathbb{H}}$ *son respectivamente las operaciones de grupo de* $\mathbb{G}$ *y* $\mathbb{H}$*, es una función* $f : \mathbb{G} \rightarrow \mathbb{H}$ *que cumple con:*

$$f(g_1 \; o_{\mathbb{G}} \; g_2) = f(g_1) \; o_{\mathbb{H}} \; f(g_2) \qquad \forall \; g_1, \; g_2 \; \in \; \mathbb{G}$$

Ejemplo

Sea $\mathbb{G}$ el conjunto de todas las matrices 2×2 de números reales e invertibles. Este grupo está definido bajo la multiplicación de matrices $(\cdot)$. Sea $\mathbb{H} = \mathbb{R}$ el conjunto de todos los números reales con excepción del cero, bajo el producto de números reales $(\times)$. Sea la función $f(A) = \det(A)$, donde $A \in \mathbb{G}$ y $\det()$ es el determinante de la matriz. Entonces la función determinante es un homomorfismo, porque:

$f(A \cdot B) = \det(A \cdot B) = \det(A) \times \det(B)$ donde $A, B \in \mathbb{G}$

3.2 Isomorfismo

Un isomorfismo de un grupo $\mathbb{G}$ proporciona una forma alternativa de pensar en él, siendo ésta equivalente.

Definición 2. *Sean $\mathbb{G}, \mathbb{H}$ grupos con respecto a las operaciones $o_{\mathbb{G}}$ y $o_{\mathbb{H}}$ respectivamente. Una función $f : \mathbb{G} \to \mathbb{H}$ es un isomorfismo si:*

1. *f es una función biyectiva*
2. *Para todos $g_1 \cdot g_2 \in \mathbb{G}$ se tiene que $f(g_1 \; o_{\mathbb{G}} \; g_2) = f(g_1) \; o_{\mathbb{H}} \; f(g_2)$*

Si existe un isomorfismo de $\mathbb{G}$ hacia $\mathbb{H}$, entonces se dice que estos grupos son isomorfos y esta relación se escribe como $\mathbb{G} \backsimeq \mathbb{H}$.

En esencia, un isomorfismo es simplemente renombrar los elementos de $\mathbb{G}$ como elementos de $\mathbb{H}$, por lo tanto si $\mathbb{G}$ es finito $\mathbb{H}$ también lo es y debe tener el mismo tamaño que $\mathbb{G}$.

Si existe el isomorfismo f de $\mathbb{G}$ hacia $\mathbb{H}$, entonces f^{-1} también es un isomorfismo que va de $\mathbb{H}$ hacia $\mathbb{G}$.

Ejemplo

Sea $\mathbb{G} = \mathbb{R}$ el conjunto de números reales positivos bajo la operación del producto, y $\mathbb{H}$ el conjunto de números reales con la suma, el logaritmo natural es un isomorfismo, porque $\ln(g_1 \cdot g_2) = \ln(g_1) + \ln(g_2)$. Esto significa, que cada enunciado sobre el producto de números reales positivos tiene un enunciado equivalente en términos de la suma de números reales.

3.3 Homomorfismo de privacidad

La demanda de privacidad en los sistemas computacionales ha crecido en los últimos años. Para el almacenamiento y recuperación de la información, existen diferentes propuestas que incluyen técnicas criptográficas para asegurar la integridad de los datos y la confidencialidad [30].

Sin embargo, el problema se complica cuando se requiere que dicha información (cifrada) pueda ser procesada de forma pública (por un tercero) y que se continue brindando estos servicios de seguridad.

Es en este contexto donde los sistemas criptográficos basados en homomorfismo pueden ser utilizados, debido a que éstos permiten realizar operaciones en información cifrada.

En 1978, Rivest et.al. [8] utilizó las propiedades que brinda el homomorfismo en un sistema criptográfico. Desafortunadamente algunas debilidades a nivel de seguridad fueron descritas por Brickell y Yacobi [21].

Posteriormente han surgido diferentes propuestas buscando eficiencia y seguridad, entre las que se puede mencionar [7],[6]; además se continua trabajando en la definición de un homomorfismo bajo más de una operación, específicamente la suma y la multiplicación.

En la siguiente sección estudiaremos dos ejemplos de criptosistemas homomórficos: RSA y el gamal, para dar paso a Paillier del cual se hace un estudio más detallado.

3.4 Criptosistemas homomórficos

Los algoritmos de cifrado basados en homomorfismo pueden ser determinísticos o probabilísticos. Cuando el algoritmo es probabilístico, se obtienen diferentes textos cifrados para un mismo texto en claro, dependiendo de una variable aleatoria. A continuación se describen algunos criptosistemas que son homomórficos.

3.4.1 RSA

El esquema de RSA es un ejemplo de un algoritmo homomórfico determinista bajo la multiplicación. Éste realiza los cálculos en $\mathbb{Z}_n$ donde $n = p \cdot q$ y p, q son primos distintos. Para ese entero n se sabe que $\phi(n) = (p-1)(q-1)$, donde ϕ es la función tociente de Euler. El espacio de los textos en claro T es igual que el de los textos cifrados T' y queda expresado como: $T = T' = \mathbb{Z}_n$.

El esquema de cifrado queda formalmente definido de la siguiente manera:

$$\begin{aligned} n &= p \cdot q \quad \text{donde } p \text{ y } q \text{ son primos} \\ T &= T' = \mathbb{Z}_n \\ K &= \{(n, p, q, e, d) : ed \equiv\ 1 \pmod{\phi(n)}\} \\ E_k(a) &= a^e\ \bmod\ n \\ D_k(a') &= (a')^d\ \bmod\ n \end{aligned}$$

donde $\mathbb{Z}_n$ es el conjunto de enteros modulo n, (n, e) constituye la llave pública y d es la llave secreta. Además a y $a' \in\ \mathbb{Z}_n$ son respectivamente el texto en claro y texto cifrado.

Si se define el conjunto $\mathbb{G} = \mathbb{H} = \mathbb{Z}_n\{\cdot\}$, donde $\cdot$ denota la multiplicación modular sobre $\mathbb{Z}_n$ entonces la propiedad de homomorfismo se cumple.

$$E_k(a) \cdot E_k(b) = (a^e\ \bmod\ n)(b^e\ \bmod\ n)\ \bmod\ n = (a \cdot b)^e\ \bmod\ n = E_k(a \cdot b)$$

3.4.2 El Gamal

El Gamal es un criptosistema que basa su seguridad en el problema del logaritmo discreto. Este problema puede ser descrito en un grupo multiplicativo $(\mathbb{G}, \cdot)$, para un elemento $\alpha \in \mathbb{G}$ de orden n se define el subgrupo generado por α como:

$$\langle\alpha\rangle = \{\alpha^i : 0 \leq i \leq n - 1\}$$

$\langle\alpha\rangle$ es un subgrupo ciclico de $\mathbb{G}$, que tiene orden n. En este contexto el problema del logaritmo discreto en un subgrupo $\langle\alpha\rangle$ de un grupo $(\mathbb{G}, \cdot)$ se enuncia como: encontrar un entero a donde $0 \leq a \leq n - 1$ tal que $\alpha^a = \beta$. Este entero a es equivalente a la expresión $\log_\alpha\ \beta$.

La utilidad de este problema en criptografía se debe a que encontrar logaritmos discretos en ciertos grupos es difícil, pero la operación inversa que es la exponenciación se calcula de forma eficiente. El Gamal es un ejemplo de criptosistema homomórfico probabilístico ya que el cifrado depende no sólo del texto en claro sino también de un valor elegido aleatoriamente por la persona que realiza el cifrado.

El esquema se presenta a continuación:

Sea p un primo tal que el problema del Logaritmo Discreto en $(\mathbb{Z}_p^*, \cdot)$ es difícil y sea $\alpha \in\ \mathbb{Z}_p^*$ un elemento primitivo. Entonces el espacio de texto en claro es $P = \mathbb{Z}_p^*$, el espacio de texto cifrado es $C = \mathbb{Z}_p^*$ x $\mathbb{Z}_p^*$ y se define:

$$K = \{(p, \alpha, a, \beta) : \beta \equiv \alpha^a (\bmod\ p)\}$$

Los valores p, α, β constituyen la llave pública y a es la llave privada.

Para $K = (p, \alpha, a, \beta)$ y un número secreto aleatoriamente elegido $k \in \mathbb{Z}_{p-1}$ se tiene:

$$e_K(x, k) = (y_1, y_2) \text{ donde}$$
$$y_1 = \alpha^k \mod p$$
$$y_2 = x\beta^k \mod p$$

Para $y_1, y_2 \in \mathbb{Z}_p^*$:

$$d_K(y_1, y_2) = y_2(y_1^a)^{-1} \mod p$$

Este esquema tiene la propiedad de homomorfismo y los grupos están definidos bajo la multiplicación como se muestra a continuación:

$$e_K(x_1) \cdot e_K(x_2) = (\alpha^{k_1}, x_1\beta^{k_1})(\alpha^{k_2}, x_2\beta^{k_2}) = (\alpha^{k_1+k_2}, (x_1 \cdot x_2)\beta^{k_1+k_2}) = e_K(x_1 \cdot x_2)$$

3.5 Esquema de cifrado Paillier

El esquema de cifrado de Paillier al igual que RSA, basa su seguridad en el problema de factorización de un número N que ha sido construido a partir del producto de dos primos. Para apreciar el funcionamiento del esquema de cifrado de Paillier es necesario estudiar la estructura del grupo $\mathbb{Z}_{N^2}^*$, algunas propiedades interesantes de éste grupo son definidas en la proposición 3.1. Otro grupo al que haremos referencia es $\mathbb{Z}_N \times \mathbb{Z}_N^*$, con las operaciones respectivas $(+, \cdot)$, es decir, $(\mathbb{Z}_N, +)$ y $(\mathbb{Z}_N^*, \cdot)$. La información aquí reportada se obtuvo principalmente de las siguientes fuentes: [31],[15].

Proposición 3.1. *Sea $N = p \cdot q$ donde p y q son primos diferentes de la misma longitud. Entonces:*

1. $m.c.d.(N, \phi(N)) = 1$
2. *Para cualquier entero $a \geq 0$, se tiene $(1+N)^a = (1+aN) \mod N^2$. Como consecuencia el orden de $(1 + N)$ en $\mathbb{Z}_{N^2}^*$ es N. Lo que significa que $(1 + N)^N = 1 \mod N^2$ y $(1 + N)^a \neq 1 \mod N^2$ para cualquier a con $1 \leq a < N$.*
3. *$\mathbb{Z}_N \times \mathbb{Z}_N^*$ es isomorfo con $\mathbb{Z}_{N^2}^*$. La función de isomorfismo es $f : \mathbb{Z}_N \times \mathbb{Z}_N^* \to \mathbb{Z}_{N^2}^*$ dado por:*
$$f(a, b) = [(1 + N)^a \cdot b^N \mod N^2]$$

Prueba. 1. Se sabe que $\phi(N) = (p-1)(q-1)$ sin pérdida de generalidad hacemos la siguiente suposición: $p > p - 1 > q - 1$. Entonces $m.c.d.(p, \phi(N)) = 1$ de forma similar $m.c.d.(q, q-1) = 1$.

Ahora bien, si $m.c.d.(q, p-1) \neq 1$ entonces $m.c.d.(q, p-1) = q$, debido a que q es un primo. Pero entonces $(p-1)/q \geq 2$ y esto contradice la suposición que p y q tienen la misma longitud.

2. Con el uso del teorema de expansión binomial tenemos:

$$(1+N)^a = \sum_{i=0}^{a} \binom{a}{i} N^i$$

Reduciendo el lado derecho modulo N^2, todos los sumandos para $i \geq 2$ se convierten en 0. Entonces $(1+N)^a = 1 + aN \mod N^2$. El exponente positivo más pequeño de a que permite que se cumpla la expresión $(1+N)^a = 1 \mod N^2$ es $a = N$.

3. El término $(1+N)^a \cdot b^N$ no tiene factor común con N^2 ya que $m.c.d.((1+N), N^2) = 1$ y $m.c.d.(b, N^2) = 1$ (ya que $b \in \mathbb{Z}_N^*$). Entonces $(1+N)^a \cdot b^N \mod N^2$ se encuentra en $\mathbb{Z}_{N^2}^*$.

f es un isomorfismo. Para probarlo, primero se demuestra que es una función biyectiva.

$$\begin{aligned} \mid \mathbb{Z}_{N^2}^* \mid = \phi(N^2) = p \cdot (p-1) \cdot q \cdot (q-1) &= pq \cdot (p-1)(q-1) \\ &= |\mathbb{Z}_N| \cdot |\mathbb{Z}_N^*| = |\mathbb{Z}_N \times \mathbb{Z}_N^*| \end{aligned}$$

Por lo anterior es suficiente demostrar que f es una función uno a uno. Sea $a_1, a_2 \in \mathbb{Z}_N$ y $b_1, b_2 \in \mathbb{Z}_N^*$ tal que $f(a_1, b_1) = f(a_2, b_2)$ entonces:

$$(1+N)^{a_1 - a_2} \cdot (b_1/b_2)^N = 1 \mod N^2 \tag{3.1}$$

Elevando ambos lados a $\phi(N)$ y utilizando el hecho de que el orden de $\mathbb{Z}_{N^2}^*$ es $\phi(N^2) = N \cdot \phi(N)$ obtenemos:

$$\begin{aligned} (1+N)^{(a_1-a_2)\cdot\phi(N)} \cdot (b_1/b_2)^{N\cdot\phi(N)} &= 1 \mod N^2 \\ \Rightarrow (1+N)^{(a_1-a_2)\cdot\phi(N)} &= 1 \mod N^2 \end{aligned}$$

Como ya ha sido mencionado $(1+N)$ tiene orden N módulo N^2. Entonces $(a_1 - a_2) \cdot \phi(N) = 0 \mod N$ y por lo tanto N divide a $(a_1 - a_2) \cdot \phi(N)$. La expresión $m.c.d.(N, \phi(N)) = 1$ entonces $N|(a_1 - a_2)$; y esto sólo puede ocurrir si $a_1 = a_2$ ya que $a_1, a_2 \in \mathbb{Z}_N$.

Regresando a la ecuación 3.1 y sabiendo que $a_1 = a_2$ entonces se tiene que $b_1^N = b_2^N \mod N^2$, esto implica que $b_1^N = b_2^N \mod N$, debido a que N es primo relativo a $\phi(N)$ que es el orden de $\mathbb{Z}_N^*$, la exponenciación a la N es una función biyectiva en $\mathbb{Z}_N^*$.

Esto significa que $b_1 = b_2 \mod N$, debido a que $b_1, b_2 \in \mathbb{Z}_N^*$ entonces $b_1 = b_2$; por lo que es posible concluir que f es uno a uno y sobreyectiva.

Para probar el isomorfismo debemos mostrar que:

$$f(a_1, b_1) \cdot f(a_2, b_2) = f(a_1 + a_2, b_1 \cdot b_2)$$

Entonces:

$$\begin{aligned} f(a_1, b_1) \cdot f(a_2, b_2) &= ((1+N)^{a_1} \cdot b_1^N) \cdot ((1+N)^{a_2} \cdot b_2^N) \mod N^2 \\ &= (1+N)^{a_1+a_2} \cdot (b_1 b_2)^N \mod N^2 \end{aligned}$$

Debido a que $(1+N)$ tiene orden N modulo N^2 (ver proposición punto número 2) podemos obtener:

$$\begin{aligned} f(a_1, b_1) \cdot f(a_2, b_2) &= (1+N)^{a_1+a_2} \cdot (b_1 b_2)^N \mod N^2 \\ &= (1+N)^{a_1+a_2 \mod N} \cdot (b_1 b_2)^N \mod N^2 \end{aligned} \tag{3.2}$$

Todavía falta probar que $b_1 b_2$ puede operar en módulo N puesto que en la ecuación 3.2 se encuentra en modulo N^2. Sea $b_1 b_2 = r + \gamma N$ donde γ, r son enteros entre $1 \leq r < N$. Note que $r = b_1 b_2 \mod N$ y además se tiene:

$$\begin{aligned} (b_1 b_2)^N &= (r + \gamma N)^N \mod N^2 \\ &= \sum_{k=0}^{N} \binom{N}{k} r^{N-k} (\gamma N)^k \mod N^2 \\ &= r^N + N \cdot r^{N-1} \cdot (\gamma N) = r^N = (b_1 b_2 \mod N)^N \mod N^2 \end{aligned}$$

Utilizando el teorema de expansión binomial y sustituyendo en la ecuación 3.2 se obtiene el resultado deseado:

$$\begin{aligned} f(a_1, b_1) \cdot f(a_2, b_2) &= (1+N)^{a_1+a_2 \mod N} \cdot (b_1 b_2 \mod N)^N \mod N^2 \\ &= f(a_1 + a_2, b_1 b_2) \end{aligned}$$

probando que f es un isomorfismo desde $\mathbb{Z}_N \times \mathbb{Z}_N^*$ hacia $\mathbb{Z}_{N^2}^*$

□

3.5.1 El problema de decisión del residuo compuesto

Se ha explicado que $\mathbb{Z}_N \times \mathbb{Z}_N^*$ es isomorfo al grupo $\mathbb{Z}_{N^2}^*$, como consecuencia un elemento cualquiera $c \in \mathbb{Z}_{N^2}^*$ corresponde a una pareja $(a, b) \in \mathbb{Z}_N \times \mathbb{Z}_N^*$. Se escribirá esto como $c \leftrightarrow (a, b)$.

Definición 3. *$c \in \mathbb{Z}_{N^2}^*$ es un residuo N-ésimo módulo N^2 si c es una N-ésima potencia, es decir, si existe una $x \in \mathbb{Z}_{N^2}^*$ que $c = x^N \mod N^2$. El conjunto de residuos N-ésimos es denotado por $Res(N^2)$.*

Veamos cómo luce un residuo N-ésimo, si tomamos cualquier elemento $x \in \mathbb{Z}^*_{N^2}$ representado como $x \leftrightarrow (a, b)$ elevandolo a la N-ésima potencia se tiene:

$$x^N \mod N^2 \leftrightarrow (a, b)^N = (N \cdot a \mod N, b^N \mod N) = (0, b^N \mod N)$$

Entonces cualquier elemento c de la forma $c = (0, b)$ es un residuo N-ésimo. Esto puede comprobarse ya que el $m.c.d.(N, \phi(N) = 1$ lo que implica que su inverso existe $d = [N^{-1} \mod \phi(N)]$ y entonces se tiene:

$$(a, [b^d \mod N])^N = (Na \mod N, [b^{dN} \mod N]) = (0, b) \leftrightarrow c$$

para toda $a \in \mathbb{Z}_N$, con ello se ha demostrado que el conjunto de residuos N-ésimos ($Res(N^2)$) corresponde al conjunto $\{(0, b)|b \in \mathbb{Z}^*_N\}$.

Además el número de raíces que tiene c es exactamente N, es decir, que calcular N-ésimas elevaciones es una función N a 1. Por lo que si se elige de manera aleatoria y uniforme un elemento $r \leftarrow \mathbb{Z}^*_{N^2}$ entonces $[r^N \mod N^2]$ es un elemento uniformemente distribuido en $Res(N^2)$.

Podemos decir de forma general que *el problema de decisión del residuo compuesto* consiste en la dificultad de distinguir un elemento aleatorio en $\mathbb{Z}^*_{N^2}$ de un elemento en $Res(N^2)$.

Como ya se ha mencionado antes, los elementos en $\mathbb{Z}^*_{N^2}$ tienen la forma (r', r) con r' y r aleatorias en cada uno de los grupos correspondientes, mientras que los residuos N-ésimos tienen la forma $(0, r)$ con r aleatoria. La *decisión del residuo compuesto* (DCR) dice que es difícil distinguir elementos del primer tipo con respecto al segundo tipo.

Esto nos sugiere una forma de cómo cifrar un mensaje $m \in \mathbb{Z}_N$ con respecto a la llave pública N, se elije de forma aleatoria un residuo N-ésimo $(0, r)$ y el cifrado está dado por:

$$c \leftrightarrow (m, 1) \cdot (0, r) = (m, r)$$

donde $(\cdot)$ representa la operación en el grupo $\mathbb{Z}_N \times \mathbb{Z}^*_N$.

Podemos ver que este cifrado es seguro debido a que un residuo enésimo $(0, r)$ no es distinguible de un elemento (r', r). Por lo tanto el cifrado construido es indistinguible de un cifrado construido como:

$$c' \leftrightarrow (m, 1) \cdot (r', r) = ([m + r' \mod N], r)$$

para un elemento elegido aleatoriamente $r' \in \mathbb{Z}_N$.

A continuación presentamos cómo es posible realizar el cifrado y descifrado. En la próxima sección se formaliza el esquema de Paillier.

Cifrado: El cifrado se realiza en el grupo $\mathbb{Z}^*_{N^2}$. El interesado en cifrar genera un texto cifrado $c \in \mathbb{Z}_{N^2}$, eligiendo un número aleatorio $r \in \mathbb{Z}^*_N$ y calculando:

$$c = [(1 + N)^m \cdot r^N \mod N^2]$$

Y se puede observar que:

$$c = ((1+N)^m \cdot 1^N) \cdot ((1+N)^0 \cdot r^N) \mod N^2] \leftrightarrow (m,1) \cdot (0,r)$$

entonces $c \leftrightarrow (m,r)$ como se buscaba.

Descifrado: El descifrado puede realizarse de forma eficiente, si se cuenta con la factorización de N, mediante los siguientes pasos:

- Calcule: $\tilde{c} = [c^{\phi(N)} \mod N^2]$
- Posteriormente realice la siguiente operación sobre los enteros: $\tilde{m} = \frac{(\tilde{c}-1)}{N}$
- Finalmente: $m = [\tilde{m} \cdot \phi(N)^{-1} \mod N]$

Prueba de que el método es correcto

Sea $c \leftrightarrow (m,r)$ para un valor aleatorio $r \in \mathbb{Z}_N^*$ entonces:

$$\begin{aligned}\tilde{c} &= [c^{\phi(N)} \mod N^2] \\ &= \big([m \cdot \phi(N) \mod N], [r^{\phi(N)} \mod N]\big) \\ &= ([m \cdot \phi(N) \mod N], 1)\end{aligned}$$

Debido a que $f(a,b) = [(1+N)^a \cdot b^N \mod N^2]$(proposición punto 3) entonces $\tilde{c} = (1+N)^{[m\cdot\phi(N) \mod N] \mod N^2}$. Ahora bien como $(1+N)^a = (1+aN) \mod N^2$ (proposición punto 2), sabemos que:

$$\tilde{c} = (1+N)^{[m\cdot\phi(N) \mod N]} = (1 + [m \cdot \phi(N) \mod N] \cdot N) \mod N^2$$

Debido a que $1+[m\cdot\phi(N) \mod N]\cdot N$ es siempre más pequeño numéricamente que N^2 podemos olvidarnos del $\mod N^2$. La operación $\frac{(\tilde{c}-1)}{N} = [m \cdot \phi(N) \mod N]$, finalmente:

$$m = [\tilde{m} \cdot \phi(N)^{-1} \mod N]$$

3.5.2 Esquema de cifrado Paillier

El esquema de cifrado queda formalmente definido por:

Sea GenModulus un algoritmo en tiempo polinomial que con una entrada 1^n genera a la salida (N,p,q) donde $N = p \cdot q$, p y q son primos de n-bits. Las primitivas criptográficas son las siguientes:

- Generación de llaves: con una entrada 1^n se obtiene (N, p, q). La llave pública es N y la privada es $(N, \phi(N))$.
- Cifrado: Como entrada requiere la llave pública N y el mensaje $m \in \mathbb{Z}_{\mathbb{N}}$. Se elige aleatoriamente un valor $r \in \mathbb{Z}^*_{\mathbb{N}}$ y se obtiene como salida el texto cifrado.

$$c = [(1+N)^m \ \cdot \ r^N \mod N^2]$$

- Descifrado: La entrada es la llave privada $(N, \phi(N))$ y el texto cifrado. A la salida se obtiene el texto en claro m:

$$m = \left(\frac{[c^{\phi(N)} \mod N^2] - 1}{N} \cdot \phi(N)^{-1} \mod N \right)$$

Homomorfismo

Este esquema de cifrado es un ejemplo de criptosistema homomórfico probabilístico bajo la suma. En efecto, se cumple la propiedad si se cifra los mensajes $m_1, m_2 \in \mathbb{Z}_N$ con la llave pública N:

$$enc_N(m_1) \cdot enc_N(m_2) = enc_N([m_1 + m_2 \mod N]) \text{ para todos } m_i \in \mathbb{Z}_N$$

donde enc_N es la función de cifrado.

Esto puede ser verificado al observar:

$$\begin{aligned} &((1+N)^{m_1} \cdot r_1^N \mod N^2) \cdot ((1+N)^{m_2} \cdot r_2^N \mod N^2) \\ &= (1+N)^{m_1+m_2 \mod N} \cdot (r_1 \cdot r_2)^N \mod N^2 \end{aligned}$$

La expresión anterior es equivalente al cifrado para el mensaje $[m_1 + m_2 \mod N]$

Ejemplo Numérico

Sea $N = 11 \cdot 17 = 187$ entonces $N^2 = 34969$. El mensaje que se quiere cifrar es $m = 175$. El valor aleatorio elegido es $r = 83 \in \mathbb{Z}_{187}$. Para calcular el texto cifrado realizamos la siguiente operación:

$$c = [(1+187)^{175} \cdot 83^{187} \mod 34969] = 23911$$

El resultado obtenido para $(175, 83) = 23911$. Para descifrar este valor, se utiliza el $\phi(N) = 160$ y los calculos que se deben realizar son:

$$\tilde{c} = [23911^{160} \mod 34696] = 25620$$

Restando 1 y dividiendo entre 187.

$$\tilde{m} = \frac{(25620 - 1)}{187} = 137$$

Se calcula el inverso: $90 = 160^{-1} \mod 187$ el mensaje es recobrado asi:

$$m = [137 \cdot 90 \mod 187] = 175$$

3.6 Homomorfismo de Paillier aplicado a bases de datos cifradas

Hay que recordar que el reto consiste en permitir ejecutar consultas en un modelo de bases subcontratadas. Y como se ha explicado anteriormente, en este modelo el dueño de la información es el cliente. El proveedor del servicio no es confiable y como consecuencia es necesario contar con mecanismos para administrar la base de datos del cliente sin que el servidor pueda interpretarla.

Ahora veamos un ejemplo básico de la información con la que puede contar un cliente. La base de datos está constituída por una sola tabla llamada **Empleados** (ver la tabla 3.1) y los atributos que contiene son: *el Identificador del Empleado, el Nombre del Empleado, el Salario del Empleado, Comisión, Dirección y el Identificador del Departamento para el cual trabaja*.

IdEmp	NombreEmp	SalarioEmp	Comisión	Dirección	IdDep
23	Tom	10K	2k	Av.Politecnico	Fin
860	Mary	20K	1k	La Rioja	Comp
320	John	5k	1k	Bamba	Comp

Tabla 3.1: Tabla Empleados en claro

Las consultas de texto en claro se originan desde el cliente, y la información contenida en la tabla 3.1 debe ser resguardada en el servidor de forma cifrada. En esta sección nos concentraremos en los atributos numéricos (*SalarioEmp, Comisión*), que potencialmente pueden intervenir en consultas que requieren cálculos. Para realizar el cifrado de estos atributos utilizaremos el esquema de cifrado de Paillier.

La relación **Empleados** se representa en el servidor con una tabla que contiene información cifrada, la cual luce como en la tabla 3.2. Los atributos de dicha tabla han sido cifrados, especificamente los campos: *SalarioEmp y Comisión* fueron cifrados con el algoritmo de Paillier.

IdEmpC	NameEmpC	SalarioC	ComisiónC	DirecciónC	IdDepC
?xEw	bcd	7	27	a34?	dre?
xr?t	ts=	18	17	dklj	kie
abtx	nmr	2	12	aiop	kie

Tabla 3.2: Tabla Empleados cifrada

Este tipo de cifrado se utiliza para poder ejecutar consultas de agregación (e.g. totales (*sum*), promedios (*avg*), etc.).

Tomemos como ejemplo que se desea calcular el monto total en salarios que esta empresa invierte en sus empleados, donde el salario de un empleado está compuesto por su salario nominal y la comisión.

Es decir, la consulta se escribe como:

```
SELECT SUM (salario + comision) FROM empleados
```

Para realizar la consulta es necesario re-escribirla como se muestra a continuación:

```
SELECT E.salario * E.comision AS subtotal FROM Empleados as E
```

Posteriormente se realiza la multiplicación de todos los subtotales, obteniendo un valor acumulado.

$$total = total * subtotal$$

El cliente puede descifrar el resultado calculando:

$$m = \left(\frac{[total^{\phi(N)} \mod N^2] - 1}{N} \cdot \phi(N)^{-1} \mod N \right)$$

Hasta aquí se ha revisado como los homomorfismos de privacidad son útiles para permitir que la información cifrada pueda ser procesada por un tercero (servidor). Este tipo de cifrado permite realizar consultas de agregación que requieren la evaluación de una o más sumas. Sin embargo, este esquema no es suficiente para realizar otros tipos de consulta (intervalos, comparaciones) que también son necesarios para mantener la funcionalidad de una base de datos en claro. Por ello, en el siguiente capítulo se presenta el esquema de cifrado que preserva el orden, él cual permite resolver consultas intervalos.

Capítulo 4

Esquema de cifrado que preserva el orden

> Si revelas tus secretos al viento no le eches la culpa al viento por revelárselo a los árboles.
>
> *Khalil Gibran*

El concepto de *cifrado simétrico que preserva el orden* (OPE), fue presentado en la comunidad que trabaja en bases de datos por Agrawal et.al. [9]. Este tipo de esquema utiliza una función de cifrado que preserva el orden numérico del texto en claro.

La importancia de este tipo de cifrado radica en que permite efectuar consultas que implican la evaluación de un intervalo en información cifrada, es decir, que es posible solicitar la devolución de textos cifrados en la base de datos donde los textos en claro (descifrados) caen dentro de dos límites dados $[a, b]$.

El cifrado que preserva el orden permite indizar y procesar las consultas exactamente con la misma eficiencia (tiempo logarítmico) que como se realiza en información en claro; utilizando las *estructuras de árbol* que normalmente maneja una base de datos.

Posterior al trabajo de Agrawal et.al., Boldyreva et.al. [10] presentan un estudio de la seguridad brindada por este tipo de esquemas y además proponen un nuevo algoritmo que permite realizar este tipo de cifrado; el cual se ha retomado como base para este trabajo y será descrito en las secciones 4.1, 4.2 y 4.3. Sin embargo, para que el algoritmo propuesto en [10] sea aplicable para la resolución de consultas de intervalos en bases de datos cifradas subcontratadas, es necesario realizar algunos ajustes referentes a la eliminación de estructuras compartidas entre el mecanismo de cifrado y el de descifrado, éstos ajustes son descritos con detalle en la sección 4.3.2 y nuestra propuesta para resolverlos es presentada en la sección 4.4.

4.1 Nociones básicas

En esta sección abordaremos como primer punto la notación y convenciones que serán ocupadas a lo largo del capítulo, posteriormente se definen conceptos preliminares que permiten presentar el esquema de cifrado propuesto por Boldyreva et.al. de forma clara, entre los que cabe mencionar: *función y cifrado que preserva el orden*.

4.1.1 Notación y Convenciones

En esta sección se presentan la notación y convenciones que serán ocupadas a lo largo del capítulo.

- Sea $[X]$ el conjunto de números enteros consecutivos de 1 a X.
- $|X|$ denota la cardinalidad del conjunto $[X]$
- Para todo $\ell \in \mathbb{N}$ se denota por 1^ℓ a la cadena de ℓ bits
- Sea $\mathcal{S}$ un conjunto. Escribiremos $x \xrightarrow{\$} \mathcal{S}$ para denotar que x es seleccionada de forma uniformemente aleatoria en $\mathcal{S}$
- $min(R)$ Límite inferior de R
- $max(R)$ Límite superior de R
- $bin_n(x)$ Si se tiene un entero x, nos referimos como $bin_n(x)$ a la representación binaria de x, bajo la suposición que el número de bits de x es menor o igual que n. Si el número de bits en x es menor que n, entonces se realiza un *pad* con ceros de izquierda a derecha
- $x \| y$ Concatenación, por ejemplo si $x = 0010$ y $y = 1000$ entonces $x \| y = 00101000$
- $AES_K(M)$ Es el texto cifrado de M calculado mediante el esquema AES utilizando una llave de 128 bits

4.1.2 Función y cifrado que preservan el orden

Para $A, B \subseteq \mathbb{N}$ donde $|A| \leq |B|$ una *función que preserva el orden (no decreciente)* es aquella $f : A \to B$ tal que para todo $i, j \in A,\ f(i) > f(j)$ si y solamente si $i > j$.

Sea un esquema de cifrado determinístico $\mathcal{SE} = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec)$ donde $\mathcal{K}$ es el espacio de llaves, $\mathcal{E}nc, Dec$ las funciones de cifrado y descifrado respectivamente, D el

espacio de textos en claro y R el espacio de textos cifrados. Entonces, se dice que $\mathcal{SE} = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec)$ es un esquema de *cifrado que preserva el orden* si $\mathcal{E}nc(k, \cdot)$ es una función que lo hace, que va D hacia R para toda $k \in \mathcal{K}$.

De aquí en adelante nos referimos a $OPF_{D,R}$ como el conjunto de todas las funciones que preservan el orden de D hacia R.

Estos conceptos serán el punto de partida para explicar como la distribución hipergeométrica es una función que preserva el orden, lo cual será tratado de manera formal en la siguiente sección.

4.2 Conexión con la distribución hipergeométrica

Hasta aquí todavía no es claro la relación que existe entre una *función que preserva el orden* y la distribución hipergeométrica, para mostrar esta conexión es necesario revisar la proposición 4.1 presentada por Boldyreva et.al. en [10].

Proposición 4.1. *Existe una biyección entre el conjunto $OPF_{D,R}$ que contiene todas las funciones que preservan el orden del dominio D de tamaño M al rango R de tamaño $N \geq M$ y el conjunto de todas las posibles combinaciones de M en N donde los elementos están ordenados.*

Prueba. Sin perder generalidad, es suficiente probar el resultado para el dominio $[M]$ y el intervalo $[N]$. Si se tiene un plano cuyas abcisas están marcadas con los enteros que van desde 1 hasta M y sus ordenadas $y = f(x)$ estan marcadas con enteros que van desde 1 hasta N, entonces S es el conjunto de los M números enteros diferentes de $[N]$, es posible construir una función que *preserva el orden* desde $[M]$ hasta $[N]$ proyectando cada $i \in [M]$ hacia el elemento i-ésimo en S. Entonces, la elección de M en N corresponde a una única *función que preserva el orden*. Si ahora tenemos la función que preserva el orden f desde $[M]$ hasta $[N]$. La imagen de f define el conjunto de M objetos distintos en $[N]$, entonces una *función que preserva el orden* corresponde a una única combinación de M elegido en N. □

A partir de lo expresado anteriormente podemos definir el lema 1.

Lema 1. *Para $M, N \in \mathbb{N}$ y para cualquier $x, x+1 \in [M], y \in [N]$ se tiene:*

$$\Pr[f(x) \leq y < f(x+1) : f \xleftarrow{\$} OPF_{[M],[N]}] = \frac{\binom{y}{x} \cdot \binom{N-y}{M-x}}{\binom{N}{M}} \tag{4.1}$$

donde $f \xleftarrow{\$} OPF_{[M],[N]}$ denota que f es seleccionada de forma uniformemente aleatoria del conjunto $OPF_{[M],[N]}$

Prueba. Partiendo de la proposición 4.1 es claro que el seleccionar la función aleatoria de $OPF_{[M],[N]}$ es equivalente a elegir un conjunto S de tamaño M en $[N]$. Entonces el número total de formas en que S puede ser elegido es $\binom{N}{M}$. Además el número de eventos favorables es igual al número de formas en que S puede ser seleccionado tal que el siguiente enunciado se cumpla:

el x-ésimo elemento más pequeño en S es menor o igual que y y $(x+1)$-ésimo elemento en S es mayor que y

Entonces deben existir x elementos en S que son menores que y y éstos pueden ser elegidos en $\binom{y}{x}$ formas. También podemos decir que existen $(M-x)$ elementos que son mayores que y, éstos pueden ser elegidos de $\binom{N-y}{M-x}$ formas. El número de eventos favorables son: $\binom{N-y}{M-x}\binom{y}{x}$, entonces la probabilidad está dada por:

$$\Pr[f(x) \leq y < f(x+1) : f \xleftarrow{\$} OPF_{[M],[N]}] = \frac{\binom{y}{x} \cdot \binom{N-y}{M-x}}{\binom{N}{M}}$$

□

4.2.1 Distribución hipergeométrica

Esta distribución se refiere a un espacio muestral donde hay elementos de dos tipos, por ejemplo: se tienen N pelotas en una caja, de las cuales M son negras y $N-M$ son blancas (defectuosas). En cada paso se elige una pelota de manera aleatoria, sin reemplazo. La variable aleatoria X describe la probabilidad de obtener x pelotas negras (éxitos) después de que una muestra de tamaño y ha sido elegida previamente. Esta variable aleatoria sigue una distribución hipergeométrica, y la probabilidad de que $X = x$ para los parámetros N, M, y es:

$$HGD(x; N, M, y) = \frac{\binom{y}{x} \cdot \binom{N-y}{M-x}}{\binom{N}{M}}$$

Se debe notar que el lado derecho de esta ecuación es igual al de la 4.1, por lo que podemos decir que:

$$Pr[f(x) \leq y < f(x+1) : f \xleftarrow{\$} OPF_{[M],[N]}] = HGD(x; N, M, y)$$

Ejemplo ilustrativo de funciones que preservan el orden

Sea el conjunto D el dominio cuyos elementos son: $D = \{1, 2, 3, 4\}$ y su tamaño es $M = 4$; el conjunto R es el rango y sus elementos son: $R = \{1, 2, 3, 4, 5, 6, 7, 8\}$ y su

tamaño es $N = 8$. Lo que se busca es proyectar cada elemento que pertenece a D hacia un elemento en R de tal manera que el orden de los elementos en D se preserve. Es posible definir distintas funciones que cumplen con ello y algunas de ellas se muestran en la tabla 4.1. El total de funciones que preservan el orden está dado por $\binom{8}{4}$.

Elementos en D	OPF 1	OPF 2	OPF 3
1	1	3	4
2	2	4	5
3	3	6	7
4	4	7	8

Tabla 4.1: Ejemplos de funciones $OPF_{D,R}$ donde $D = \{1, 2, 3, 4\}$ y $R = \{1, 2, 3, 4, 5, 6, 7, 8\}$, dominio y rango respectivamente.

La función que preserva el orden mostrada en la columna 2 es el caso trivial, pues cada elemento ha sido proyectado hacia el mismo en R; la columna 3 y 4 muestran otras posibilidades.

Para motivar cómo se puede construir un esquema de cifrado que preserva el orden haciendo uso de la función de distribución hipergeométrica presentamos un ejemplo ilustrativo.

Ejemplo ilustrativo de cifrado que preserva el orden

Retomando el ejemplo anterior donde el dominio está dado por el conjunto $D = \{1, 2, 3, 4\}$ y el rango por $R = \{1, 2, 3, 4, 5, 6, 7, 8\}$, si se quiere realizar el cifrado de todos los elementos en D se debe realizar el siguiente experimento:

Se tienen 8 pelotas de las cuales 4 son negras y las restantes son blancas, entonces se debe elegir una y dependiendo de su color, se asignará o no el elemento del rango al elemento del dominio que está siendo evaluado (negras = éxito = asignación), ver la tabla 4.2.

Muestreos	Pelota Resultante	Proyección de Puntos	Total de Pelotas
0	-	-	4 Blancas y 4 Negras
1	Negra	1 - 1	4 Blancas y 3 Negras
2	Blanca	2 - x	3 Blancas y 3 Negras
3	Blanca	2 - x	2 Blancas y 3 Negras
4	Blanca	2 - x	1 Blanca y 3 Negras
5	Negra	2 - 5	1 Blanca y 2 Negras
6	Blanca	3 - x	0 Blancas y 2 Negras
7	Negra	3 - 7	0 Blancas y 1 Negra
8	Negra	4 - 8	0 Blancas y 1 Negra

Tabla 4.2: Cifrado que preserva el orden

En la primera fila queda establecido que se inicia con 4 pelotas blancas y 4 negras. En este ejemplo en particular, se selecciona la primera pelota y ésta es de de color negro, por ello el elemento $\{1\}$ del dominio es proyectado al primer elemento en el rango $\{1\}$.

Debido a que en las siguientes tres selecciones se obtiene una pelota blanca el segundo elemento del dominio no es asiganado a los 3 elementos que se pierden en el rango $\{2, 3, 4\}$ y es hasta la extracción número cinco en la que se obtiene de nuevo una pelota negra y queda establecido que el segundo elemento del dominio será proyectado al quinto elemento en el rango.

Para terminar de realizar la proyección se continúa bajo el mismo procedimiento. Los números obtenidos en la proyección pueden ser utilizados como el texto cifrado de los números originales (texto en claro).

En la próxima sección estudiaremos de manera formal cómo construir este tipo de esquema.

4.3 Esquema de cifrado que preserva el orden

En esta sección se presenta el esquema de cifrado que preserva el orden propuesto por Boldyreva et.al. en [10], se explica la estrategia que siguen los algoritmos de cifrado y descifrado que lo componen y los ajustes que requieren para poder ser utilizados.

El esquema de cifrado que preserva el orden es simétrico $\mathcal{SE} = (\mathcal{K}, \mathcal{E}nc, \mathcal{D}ec)$ y está asociado al espacio de texto en claro D y al espacio de texto cifrado R y consiste de tres algoritmos:

- Algoritmo de generación de llaves $\mathcal{K}$ que devuelve una llave secreta k
- Algoritmo de cifrado $\mathcal{E}nc$ que recibe como entradas la llave secreta k, (D, R) respectivamente los espacios de texto en claro y texto cifrado, y el texto en claro que se desea cifrar m. Como salida se obtiene el texto cifrado c
- Algoritmo de descifrado $\mathcal{D}ec$ toma la llave secreta k y los espacios de texto en claro y cifrado (D, R) y el texto cifrado c. A la salida se obtiene el correspondiente texto en claro m o en su defecto $\perp$ indicando que el texto cifrado es inválido

Para construir este esquema, Boldyreva et.al. proponen dos algoritmos: **Cifrado_PO** (D, R, m) (ver la tabla 4.3 a la izquierda) y **Descifrado_PO** (D, R, c) (ver la tabla 4.3 a la derecha).

Los algoritmos **Cifrado_PO** (D, R, m) (ver la tabla 4.3 a la izquierda) el cual va de D a R siendo $|D| \leq |R|$ y **Descifrado_PO** (D, R, c) (ver la tabla 4.3 a la derecha) el

cual va en sentido contrario, es decir, de R a D; no dependen de una llave por lo que comparten y mantienen dos arreglos que permiten definir el estado actual I y F, los cuales incialmente están vacíos.

Ambos algoritmos utilizan dos funciones básicas, la primera es: **HGD** que toma como entrada D, R, y $y \in R$ y retorna $x \in D$ tal que para cada $x^* \in D$ se tiene $x = x^*$ con probabilidad $P_{HGD}(x - d; |R|, |D|, y - r)$ obtenida utilizando cierto número de bits aleatorios, donde $d = min(D) - 1$ y $r = min(R) - 1$.

La segunda es **GetCoins** la cual tiene como entrada $1^\ell, D, R, y$, y devuelve cierto número de bits que son utilizados por la función **HGD** o bien para establecer el valor del rango que será asignado como texto cifrado (ver líneas $6, 7$ y $10, 11$ del algoritmo **Cifrado_PO** 4.3).

La eliminación de los arreglos I y F al igual que la construcción de las funciones **HGD** y **GetCoins** será tratado en detalle en la sección 4.4.

Algoritmo Cifrado_PO(D, R, m)

1. $M \leftarrow |D|$; $N \leftarrow |R|$
2. $d \leftarrow min(D) - 1; r \leftarrow min(R) - 1$
3. $y \leftarrow r + \left\lfloor \frac{N}{2} \right\rfloor$
4. **If** $|D| = 1$
5. **If** $F[D, R, m]$ no está definido entonces
6. $cc \xleftarrow{\$} GetCoins(1^{\ell_R}, D, R, 1||m)$
7. $F[D, R, m] \xleftarrow{cc} R$
8. **Return** $F[D, R, m]$
9. **If** $I[D, R, y]$ no está definido entonces
10. $cc \xleftarrow{\$} GetCoins(1^{\ell_1}, D, R, 0||y)$
11. $I[D, R, y] \xleftarrow{\$} HGD(D, R, y; cc)$
12. $x \leftarrow d + I[D, R, y]$
13. **If** $m \leq x$
14. $D \leftarrow \{d + 1, \cdots, x\}$
15. $R \leftarrow \{r + 1, \cdots, y\}$
16. **Else**
17. $D \leftarrow \{x + 1, \cdots, d + M\}$
18. $R \leftarrow \{y + 1, \cdots, r + N\}$
19. **Return** Cifrado_PO(D, R, m)

Algoritmo Descifrado_PO(D, R, c)

1. $M \leftarrow |D|$; $N \leftarrow |R|$
2. $d \leftarrow min(D) - 1; r \leftarrow min(R) - 1$
3. $y \leftarrow r + \left\lfloor \frac{N}{2} \right\rfloor$
4. **If** $|D| = 1$ entonces $m \leftarrow min(D)$
5. **If** $F[D, R, m]$ no está definido entonces
6. $cc \xleftarrow{\$} GetCoins(1^{\ell_R}, D, R, 1||m)$
7. $F[D, R, m] \xleftarrow{cc} R$
8. **If** $F[D, R, m] = c$ **then return** m
9. **Else return** $\perp$
10. **If** $I[D, R, y]$ no está definido entonces
11. $cc \xleftarrow{\$} GetCoins(1^{\ell_1}, D, R, 0||y)$
12. $I[D, R, y] \xleftarrow{\$} HGD(D, R, y; cc)$
13. $x \leftarrow d + I[D, R, y]$
14. **If** $c \leq y$
15. $D \leftarrow \{d + 1, \cdots, x\}$
16. $R \leftarrow \{r + 1, \cdots, y\}$
17. **Else**
18. $D \leftarrow \{x + 1, \cdots, d + M\}$
19. $R \leftarrow \{y + 1, \cdots, r + N\}$
20. **Return** Descifrado_PO(D, R, c)

Tabla 4.3: A la izquierda: algoritmo de cifrado que preserva el orden, a la derecha: algoritmo de descifrado que preserva el orden

A continuación describiremos la estrategia que utiliza este algoritmo para obtener tex-

tos cifrados que preservan el orden.

4.3.1 Estrategia del algoritmo de preservación de orden

Para determinar la imagen que se le asigna a un mensaje m, **Cifrado_PO** utiliza una estrategia de relacionar intervalos que se localizan en el rango con los que se encuentran en el dominio.

Un intervalo está definido por una barrera imaginaria entre dos puntos, para definir los intervalos que quedarán relacionados el algoritmo utiliza búsqueda binaria.

El algoritmo de Boldyreva es de tipo recursivo, las líneas $1-3$ son de inicialización, las líneas $4-8$ constituyen el caso base. De la línea 4 a la 19 es el caso recursivo, finalmente en la 19 el algoritmo se llama a el mismo.

Cuando el algoritmo se ejecuta, primero relaciona la mitad del intervalo del rango "y", es decir, la mitad de las dos fronteras del rango; con un intervalo en el dominio.

Para establecer esta relación se hace uso de la distribución hipergeométrica (línea 11 del algoritmo de la tabla 4.3) quien define cuántos puntos estarán incluidos en el intervalo del dominio, y estos valores son almacenados en el arreglo I.

Así podemos ver cómo se ha relacionado el intervalo entre y y $y+1$ con x y $x+1$. En posteriores llamadas se trabaja con las partes bajas del rango y dominio, y se siguen proyectando intervalos.

Finalmente en la línea 7, el algoritmo elige aleatoriamente un valor dentro de un intervalo del rango el cual será la imagen asignada a m. Esta información es almacena en el arreglo F.

4.3.2 Ajustes requeridos

Estos algoritmos (**Cifrado_PO** tabla 4.3 lado izquierdo, **Descifrado_PO** tabla 4.3 lado derecho) no pueden ser directamente aplicados, debido a que comparten los arreglos I y F que determinan el estado actual.

Para que el esquema pueda ser aplicado dichos arreglos deben ser eliminados, Boldyreva et.al. presentan en su artículo cómo implementar la subrutina **GetCoins** y las referencias necesarias para implementar **HGD** de tal forma que la información contenida en los arreglos pueda ser calculada en línea según sea requerida.

En la próxima sección se plantea la forma en que este problema es resuelto en el esquema propuesto; a fin de que el algoritmo sea implementable en el modelo *DAS*.

4.4 Cambios realizados sobre el algoritmo de preservación de orden

En la subsección 4.3.2 se explica que el algoritmo de cifrado que preserva el orden, que se muestra en la tabla 4.5 lado izquierdo, no es directamente aplicable como esquema de cifrado, debido a que hace uso de dos arreglos (I,F) que determinan el estado actual; además son compartidos con el algoritmo de la tabla 4.3 (descifrado lado derecho).

Por lo anterior, el objetivo es eliminar estos arreglos, buscando que los valores contenidos en ellos puedan ser generados en línea; para esto se requiere sustituir de forma adecuada la funciones **GetCoins** y **HGD**.

El algoritmo original se muestra en la tabla 4.5 lado izquierdo y el algoritmo con modificaciones en la tabla 4.5 lado derecho, en este último se puede observar los cambios que se ralizaron con el fin de calcular con facilidad los valores generados por las funciones **GetCoins** y **HGD**. Estos cambios serán explicados a continuación.

La función **GetCoins** brinda a la salida bits aleatorios, siendo ésta la fuente de aleatoriedad del algoritmo de cifrado que preserva el orden (tabla 4.5 lado izquierdo). Al sustituir esta función se debe cumplir que su salida dependa de una llave para que el texto cifrado pueda ser descifrado por quien conozca esta llave.

La aleatoriedad generada por esta función es utilizada para dos propósitos:

- Encontrar el índice que determina el texto cifrado. (línea 6 de la tabla 4.5 lado izquierdo)
- Para generar un número aleatorio uniformemente distribuido entre 0 y 1 que será ocupado para generar un elemento que siga la distribución hipergeométrica. (línea 10 y 11 de la tabla 4.5 lado izquierdo)

En el algoritmo modificado OPEM_Cifrado (lado derecho tabla 4.5), GetCoins es sustituída por dos llamadas al cifrador por bloque AES. Una sola llamada puede generar 128 bits aleatorios. Enseguida, describimos en detalle cada una de las llamadas al cifrador por bloque.

GetCoins para encontrar el índice del texto cifrado

Como primer paso se construye un entero de 128 bits conformado de la siguiente manera (línea 5 del algoritmo de la tabla 4.5 lado derecho):

D	R	m
32 bits	64 bits	32 bits

Tabla 4.4: Distribución de bits en GetIndex

Posteriormente se hace una llamada a AES y se obtiene un texto cifrado de 128 bits (línea 6 de la tabla 4.5 lado derecho). De los cuales los 64 bits menos significativos son utilizados para calcular el índice que determinará el texto cifrado. Este índice es un número entre los límites del intervalo $min(R)\ y\ max(R)$ (línea 8 de la tabla 4.5 lado derecho).

Algoritmo Cifrado_PO(D, R, m)

1. $M \leftarrow |D|$; $N \leftarrow |R|$
2. $d \leftarrow min(D) - 1; r \leftarrow min(R) - 1$
3. $y \leftarrow r + \left\lfloor \frac{N}{2} \right\rfloor$
4. **If** $|D| = 1$
5. **If** $F[D, R, m]$ no está definido entonces
6. $cc \xleftarrow{\$} GetCoins(1^{lR}, D, R, 1||m)$
7. $F[D, R, m] \xleftarrow{cc} R$
8. **Return** $F[D, R, m]$

9. **If** $I[D, R, y]$ no está definido entonces
10. $cc \xleftarrow{\$} GetCoins(1^{l1}, D, R, 0||y)$
11. $I[D, R, y] \xleftarrow{\$} HGD(D, R, y; cc)$
12. $x \leftarrow d + I[D, R, y]$
13. **If** $m \leq x$
14. $D \leftarrow \{d + 1, \cdots, x\}$
15. $R \leftarrow \{r + 1, \cdots, y\}$
16. **Else**
17. $D \leftarrow \{x + 1, \cdots, d + M\}$
18. $R \leftarrow \{y + 1, \cdots, r + N\}$
19. **Return** Cifrado_PO(D, R, m)

Algoritmo OPEM_Cifrado(D, R, m)

1. $M \leftarrow |D|\ \ N \leftarrow |R|$
2. $d \leftarrow min(D) - 1; r \leftarrow min(R) - 1$
3. $y \leftarrow r + \left\lfloor \frac{N}{2} \right\rfloor$
4. **If** $|D| = 1$
5. $M \leftarrow bin_{32}(D) \parallel bin_{64}(R) \parallel bin_{32}(m)$
6. $C \leftarrow AES_{128}(M)$
7. $c_0 \leftarrow C_L$ *// C_L está compuesta por los 64 bits menos significativos de C*
8. $idx = min(R) + c_0 \mod ((max(R) - min(R)) - 1)$
9. **Return** idx

10. $M \leftarrow bin_{32}(D) \parallel bin_{64}(R) \parallel bin_{32}(y)$
11. $C \leftarrow AES_{128}(M)$
12. $c_0 \parallel c_1 \parallel c_2 \leftarrow C_H$ *//C_H equivale a la parte alta de C.*
//Cada segmento c_i es de 16 bits.
13. $cc = 2^{-16} * (2^{-16} * ((2^{-16} * c_0) + c_1) + c_2)$

14. $x \leftarrow d + HGD\_Generate(D, R, y; cc)$
15. **If** $m \leq x$
16. $D \leftarrow \{d + 1, \cdots, x\}$
17. $R \leftarrow \{r + 1, \cdots, y\}$
18. **Else**
19. $D \leftarrow \{x + 1, \cdots, d + M\}$
20. $R \leftarrow \{y + 1, \cdots, r + N\}$
21. **Return** OPEM_Cifrado(D, R, m)

Tabla 4.5: A la izquierda: algoritmo de cifrado que preserva el orden Boldyreva et.al., A la derecha: algoritmo de cifrado que preserva el orden con modificaciones.

GetCoins para generar un elemento que sigue la distribución hipergeométrica

Para construir la entrada de la segunda llamada a AES el algoritmo modificado toma como entrada el dominio D, el rango R y la variable y; y son acomodados en un entero de 128 bits, quedando distribuidos tal y como se muestra en la tabla 4.6 (ver línea 10 del algoritmo mostrado en la tabla 4.5 lado derecho).

A la salida de la llamada a AES se obtiene 128 bits de texto cifrado (ver línea 11 del algoritmo mostrado en la tabla 4.5 lado derecho). De estos 128 bits se toman los 48 más

significativos para generar un número entre 0 y 1, líneas 12-13 del algoritmo mostrado en la tabla 4.5 lado derecho. Este número se utiliza como parámetro de entrada en la función **HGD_Generate** la cual convierte la distribución uniforme a una distribución hipergeométrica.

D	R	y
32 bits	64 bits	32 bits

Tabla 4.6: Distribución de bits en GetCoins

HGD_Generate

Esta función fue implementada utilizando el algoritmo presentado en MATLAB y se muestra en la tabla 4.7. MATLAB es un proyecto desarrollado por MathWorks. Existen otras bibliotecas que incluyen implementaciones de la función hipergeométrica entre las que se puede mencionar la de *Wolfram Mathematica*.

Para generar un número que siga la *distribución hipergeométrica* es necesario calcular la inversa de la distribución hipergeométrica acumulada, conocida como función de distribución acumulada (**cdf**) por sus siglas en inglés. Debido a que la función hipergeométrica es discreta, la función **cdf** consiste en buscar el entero más pequeño x tal que la función hipergeométrica **cdf** evaluada en x es igual o excede a p.

Puede pensarse a p como la probabilidad de encontrar x elementos defectuosos con n muestreos sin reemplazo, de una muestra de m elementos donde k de ellos están defectuosos.

El algoritmo de **HGD_Generate** descrito en la tabla 4.7 genera un elemento que sigue la distribución hipergeométrica con parámetros M, N, n, dado un número uniformemente distribuido p. Además este algoritmo llama a la función $hygepdf$ para calcular la distribución acumulada, la cual usa la función $gammaln(\cdot)$ que está definida por:

$gammaln(x) = \ln[\Gamma(x)]$ donde $\Gamma(x)$ es la función gamma.

4.4.1 Generalización de la estrategia

Para introducir la idea de porque es posible generalizar la estrategia del algoritmo presentado en la tabla 4.5 lado derecho, tenemos la siguiente proposición:

Proposición 4.2. *Sea f una función que preserva el orden de $[2^w]$ a $[2^u]$, donde $u > w$. Sea $g : [2^{mw}] \to [2^{mu}]$ tal que $\forall x \in [2^{mw}]$, si se escribe $x = x^{(m-1)}||x^{(m-2)}|| \cdots ||x^0$ donde $x^i \in [2^w]$; entonces se hace $g(x) = f(x^{m-1})||f(x^{m-2})|| \cdots ||f(x^0)$. Se tendrá que la función g así definida preserva el orden.*

En la implementación realizada en este trabajo de tesis el tamaño de D es 2^{16} y el tamaño de R es 2^{24}, con lo que este esquema permite cifrar números de 32 bits, y en términos generales cualquier cantidad de un mayor número de bits. Para ello el número m que se desea cifrar, puede se particionado en n segmentos de 16 bits cada uno, y posteriormente se llama como una sub-rutina el algoritmo de cifrado que preserva el orden con modificaciones (tabla 4.5 lado derecho) pasándole como parámetro de entrada cada uno de los segmentos (ver el algoritmo de la tabla 4.8).

Algoritmo HGD_Generate(N, M, n, p)

1. $m \leftarrow N$; $k \leftarrow M$;
2. $count \leftarrow 0$; $x \leftarrow 0$; $cumdist \leftarrow 0.0$; $y \leftarrow 0.0$
3. $cumdist \leftarrow hygepdf(x, m, k, n, y)$
4. **while** $((p > cumdist)\ and\ (count < n)\ and\ (count < k))$
5. $\quad count \leftarrow count + 1$
6. $\quad x \leftarrow x + 1$
7. $\quad y = hygepdf(count, m, k, n, y)$
8. $\quad cumdist = cumdist + y$
9. **Return** x

Algoritmo hygepdf(x, m, k, n, y)

10. $kx \leftarrow 0.0$; $mn \leftarrow 0.0$; $mknx \leftarrow 0.0$;
11. **If** $(x > k\ or\ (m - k - n + x + 1) \leq 0\ or\ x < 0)$ then
12. $\quad y = 0$
13. **Else**
14. $\quad kx \leftarrow gammaln(k + 1) - gammaln(x + 1) - gammaln(k - x + 1)$
15. $\quad mn \leftarrow gammaln(m + 1) - gammaln(n + 1) - gammaln(m - n + 1)$
16. $\quad mknx \leftarrow gammaln(m - k + 1) - gammaln(n - x + 1) - gammaln(m - k - ((n - x)) + 1)$
17. $\quad y \leftarrow exp(kx + mknx - mn)$
18. **Return** y

Tabla 4.7: Algoritmo de HGD_Generate

Algoritmo Cifrado_Bloque(D, R, m)

1. $m = m_n, m_{n-1}, \cdots m_1, m_0$
2. **For** $i = 0$ **to** n
3. $\quad$ idx[i]=OPEM_Cifrado(D, R, m_i)
4. $cipherText = idx[n] || idx[n-1] \cdots idx[1] || idx[0]$
5. **Return** cipherText

Tabla 4.8: Algoritmo de cifrado que preserva el orden por bloques

4.5 Cifrado que preserva el orden aplicado a bases de datos cifradas

Retomando el ejemplo de la tabla **Empleados** (ver la tabla 4.9) cuyos atributos son: *el Identificador del Empleado, el Nombre del Empleado, el Salario del Empleado, Comisión, Dirección y el Identificador del departamento para el cual trabaja*.

IdEmp	NombreEmp	SalarioEmp	Comisión	Dirección	IdDep
23	Tom	10K	2k	Av.Politecnico	Fin
860	Mary	20K	1k	La Rioja	Comp
320	John	5k	1k	Bamba	Comp

Tabla 4.9: Tabla Empleados en claro

El cliente puede requerir ejecutar una consulta que implica evaluar un intervalo. Como ejemplo podemos pensar en seleccionar todos lo empleados cuyo salario nominal es mayor o igual que $10k$ y menor o igual que $20k$. Esta consulta se escribe:

```
SELECT NombreEmp, Salario FROM Empleados
  WHERE Salario ≥ 10K AND Salario ≤ 20K
```

Para que este tipo de consulta pueda ser evaluada es necesario agregar otro campo cifrado generado al aplicar el *algoritmo de preservación del orden* al campo en claro *salario* (ver tabla 4.10).

Note que el texto cifrado de la columna *salarioPO* mantiene la estructura del orden numérico del texto en claro *SalarioEmp*, e.g. a Tom cuyo salario en texto en claro es 10K se le asignó 30 y a Mary que gana 20K se le asignó 50; es claro que los textos cifrados mantienen la relación de que Mary tiene un mejor salario que Tom.

IdEmpC	NameEmpC	SalarioC	SalarioPO	ComisiónC	ComisiónPO	DirecciónC	IdDepC
?xEw	bcd	7	30	27	7	a34?	dre?
xr?t	ts=	18	50	17	5	dklj	kie
abtx	nmr	2	15	12	5	aiop	kie

Tabla 4.10: Tabla Empleados cifrada

Ahora retomemos cómo el servidor evalúa la consulta; para ello, del lado del cliente se aplicará el algoritmo **OPEM_Cifrado** que se muestra en la tabla 4.7 lado derecho a los valores 10K y 20K y se sustituirán en la consulta original. Es decir, que la consulta re-escrita quedará como sigue:

```
SELECT NombreEmp, SalarioPO FROM Empleados
  WHERE SalarioPO ≥ 30 AND SalarioPO ≤ 50
```

Posteriormente se envía la nueva consulta al servidor y éste podrá regresar los registros coincidentes aplicando la consulta recibida. En este caso en particular, el servidor devolverá al cliente los registros de Tom y Mary, y quedará excluido el de John.

Para efectuar el descifrado de los registros devueltos por el servidor, se utiliza el algoritmo de descifrado de Boldyreva en el atributo *SalarioPO*. La manera en la que se descifra el nombre del empleado será tratado en el siguiente capítulo.

Es importante notar que haciendo uso de este otro esquema de cifrado es posible resolver ciertas consultas de agregación como **max** y **min**, que devuelven respectivamente el máximo y minímo valor sobre una columna. Por ejemplo, si el cliente quiere seleccionar el salario más alto entre todos sus empleados, la consulta luciría así:

```
SELECT max(SalarioEmp) FROM Empleados
```

En caso de que lo que qusiera seleccionar fuera el salario más bajo de todos sus empleados, la consulta se escribe así:

```
SELECT min(SalarioEmp) FROM Empleados
```

Lo interesante en estos casos es que no es necesario re-escribir la consulta, simplemente se envía al servidor y el cliente descifra con el algortimo de Boldyreva el valor devuelto por el servidor. Esto es posible gracias a que el texto cifrado mantiene el orden numérico del texto en claro.

Entonces es posible concluir que con el uso del cifrado basado en homomorfismo y con el cifrado que preserva el orden el servidor tiene la capacidad de resolver consultas de agregación e intervalos, pero queda pendiente descubrir cómo pueden realizarse consultas con atributos alfanuméricos y cómo se pueden acoplar los algoritmos que ya han sido presentados para forma un solo esquema. Son precisamente estos topícos los que se tratan en el siguiente capítulo.

Capítulo 5

Esquema de cifrado para la ejecución de consultas en bases de datos cifradas

> Nunca emprenderíamos nada si quisiéramos asegurar por anticipado el éxito de nuestra empresa
>
> *Napoleón*

En este capítulo abordaremos el problema de ejecución de consultas en bases de datos cifradas; para ello retomaremos *el modelo de bases de datos subcontratadas*.

Se presenta el esquema propuesto como una opción viable para poder ejecutar consultas de forma eficiente, cubriendo la gran mayoría de tipos de consultas provistas por SQL en bases de datos en claro.

5.1 Recordatorio del modelo de bases de datos subcontratadas

Uno de los mecanismos más ampliamente difundido para el manejo de la información son las bases de datos relacionales, y debido a la cantidad de transacciones que una compañía realiza diariamente (comercio electrónico, transacciones bancarias, etc.), la cantidad de datos que es necesario administrar crece desmesuradamente, aumentando así su tamaño y los costos de mantenimiento.

Este problema es la motivación que hizo surgir el modelo *Database as a Service* donde los clientes almacenan su información en un servidor que es administrado por un tercero (**Proveedor**); lo que permite que los altos costos de administración e infraestructura sean absorbidos por el proveedor del servicio.

Sin embargo, el proveedor podría realizar un ataque en contra de la confidencialidad

e integridad de los datos; por ello se le considera un posible adversario, es decir, **no es confiable**.

Para evitar estos posibles ataques, los clientes cifran su información antes de enviarla al servidor, y ésta permanecerá cifrada aún cuando esté siendo procesada. Por lo que el sevidor debe proporcionar mecanismos para ejecutar correctamente las operaciones relacionales generadas por el cliente en texto en claro, sin ser capaz de interpretar la información contenida en la base de datos.

5.1.1 Operaciones que deben ser soportadas

Tal y como fue explicado en la sección 1.3, el modelo de bases de datos utilizado es relacional, lo que implica que la información es almacenada en un conjunto de tablas (*relaciones*). La base de datos modelo a la que haremos referencia durante todo el capítulo luce como en la figura 5.1 y está compuesta por dos tablas **Empleados** y **Departamentos**. La primera tiene por atributos: *el Identificador del Empleado, el Nombre del Empleado, el Salario del Empleado, Comisión, Dirección y el Identificador del departamento para el cual trabaja* y la segunda tiene los atributos: *Identificador de departamento, el Nombre del departamento*.

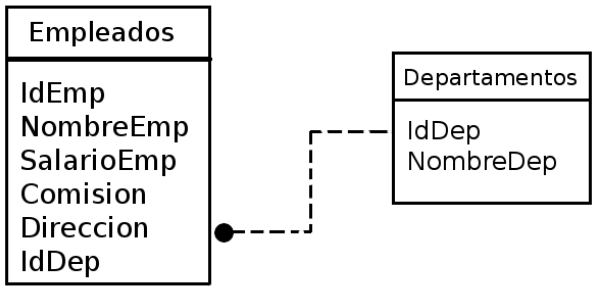

Figura 5.1: Base de datos en claro.

Al cifrar la base de datos el cliente requiere que ciertas características sigan siendo válidas al igual que las operaciones que normalmente realiza, las cuales son descritas enseguida.

- **Modelo relacional** Está constituído por varias tablas las cuales se relacionan por medio de llaves, éstas pueden ser de dos tipos: primarias y extranjeras (*primary and foreign keys*) [32].

  Llave primaria Permite identificar de manera única a un registro, en el caso de

la tabla Empleados la llave primaria es el atributo *IdEmp* y para la tabla Departamentos es el atributo *IdDep* (ver figura 5.1).

Llave extranjera Aparece en una tabla haciendo referencia a la información contenida en otra tabla. Por ejemplo, el atributo *CodigoDep* de la tabla Empleados es llave extranjera y hace referencia a la información contenida en la tabla Departamentos, de modo que es posible conocer cuales empleados trabajan en cada departamento.

Entonces un requisto importante es que la base de datos cifrada siga manteniendo esta propiedad.

- **Insertar** Implica poblar una tabla con registros y se escribe:

```
INSERT INTO Departamentos VALUES ('Fin','Finanzas')
```

- **Consultar** Permite seleccionar información de la base de datos. Existen varios tipos de consultas entre los que cabe mencionar:

  – **Consultas con atributos alfanuméricos** Estos atributos pueden intervenir de dos formas: como parte de las columnas seleccionadas o bien como parte de las condiciones de la consulta. En el primer caso la consulta consiste en *seleccionar todos los nombres de los empleados* y luciría así:

  ```
  SELECT NombreEmp FROM Empleados
  ```

  En el segundo caso la consulta consiste en *seleccionar los nombres de los empleados que pertenecen al departamento de finanzas*, se escribiría como:

  ```
  SELECT NombreEmp FROM Empleados WHERE IdDep = 'Fin'
  ```

  donde la condición de la cláusula `WHERE` implica a un atributo de tipo alfanumérico.

  – **Consultas de intervalos** Este tipo de consulta selecciona información de una tabla estableciendo ciertos límites en la cláusula `WHERE`. Por ejemplo, *seleccionar el nombre y salario de los empleados cuyo monto es mayor que 5k*

  ```
  SELECT NombreEmp, SalarioEmp
  FROM Empleados WHERE SalarioEmp>5k
  ```

  – **Consultas de agregación** Este tipo de consultas permite obtener un solo resultado calculado por la evaluación de varios registros, por ejemplo al *calcular el total de salarios de los empleados que trabajan en la empresa* y se escribe:

  ```
  SELECT SUM(SalarioEmp) FROM Empleados
  ```

- **Modificar** permite actualizar una o más columnas de un registro, por ejemplo, *actualizar el salario del empleado cuyo identificador es igual a 23* y se escribe:

```
UPDATE Empleados SET SalarioEmp = 15k
WHERE IdEmp = '23'
```

- **Eliminar** Permite borrar un registro o más, por ejemplo, *eliminar el registro del empleado cuyo identificado es igual a 23*

```
DELETE FROM Empleados WHERE IdEmp = '23'
```

En la siguiente sección describiremos cuales bloques componen al esquema propuesto y cómo éstos nos ayudan a resolver todas las operaciones antes detalladas, en el contexto de bases de datos subcontratadas.

5.2 Esquema propuesto

Nuestro esquema de cifrado está compuesto por tres algoritmos: un cifrador por bloques (AES), un cifrador basado en homomorfismos (Paillier) y un cifrador que preserva el orden (Boldyreva et.al.); cada uno de éstos desempeña un papel esencial para hacer que el esquema sea lo suficientemente robusto, pues cada uno propicia una o algunas de las operaciones relacionales que se realizan en una base de datos en claro. A continuación revisaremos la problemática que resuelve cada uno.

Cifrador por bloques

La primera característica que debe ser provista es que la base de datos cifrada sea relacional, para permitir que la información se almacene en distintas tablas y que ésta pueda ser consultada.

Debido a que un cifrador por bloques es determinístico puede ser utilizado para mantener las relaciones entre las tablas, ya que el obtener siempre el mismo texto cifrado para un texto en claro garantiza que la cifra de un atributo que es llave primaria en una tabla será exactamente igual al del atributo que es llave extranjera.

Además nos permite dar respuesta aquellas consultas que involucran atributos alfanuméricos, ya sea que éstos participen en la cláusula `SELECT` o en la cláusula `WHERE`. Por lo tanto el texto cifrado de este tipo de atributos es obtenido con un cifrador por bloques y es almacenado en la base de datos.

Cifrador basado en homomorfismos

Tal y como fue explicado en el capítulo 3 esta técnica criptográfica permite operar información cifrada sin conocer el contenido en claro. Ya que en las bases de datos en claro es usual obtener totales y subtotales, el esquema debe permitir que el servidor pueda resolver este tipo de consulta de agregación.

Las consultas de agregación operan sobre atributos numéricos por lo que éstos en nuestro esquema son cifrados con un algoritmo basado en homomorfismo bajo la operación suma, dicha cifra es almacenada en la base de datos.

Cifrador que preserva el orden

Las consultas de evaluación de intervalos también deben ser soportadas, sin embargo, el resolver este tipo de consultas constituye un problema díficil debido a que los algoritmos de cifrado buscan por naturaleza inyectar en la cifra la mayor aleatoriedad posible, con lo cual rompen toda estructura existente en el texto en claro. Entonces, los límites involucrados en este tipo de consulta no pueden ser evaluados.

Con la ídea de poder resolver estas consultas ha surgido un nuevo paradigma de cifrado cuya característica principal es el mantener en el texto cifrado el orden numérico existente en el texto en claro (ver capítulo 4); permitiendo que las consultas de intervalos puedar ser resueltas.

Entonces los atributos numéricos deben ser cifrados con un algoritmo de este tipo y ser almacenados en la base de datos.

Hasta este momento ya hemos discutido dos puntos importantes: que la base de datos cifrada sea relacional y las aportaciones que los bloques de cifrado brindan al esquema de manera general. En el siguiente apartado abordaremos por qué la elección de AES, de Paillier y Boldyreva et.al.

5.2.1 Justificación de algoritmos elegidos

Ya se ha mencionado que nuestro esquema utiliza tres algoritmos: un cifrador por bloques, un cifrador basado en homomorfismos y un cifrador que preserva el orden, sin embargo, en la literatura actual existe más de una opción por cada una de estas categorías. Entonces surge la pregunta por qué se eligió a AES, Paillier y Boldyreva et.al. Enseguida daremos respuesta a este cuestionamiento.

Comenzaremos por hablar del cifrador por bloques, en realidad no es restrictivo utilizar AES puesto que se tienen otras opciones como DES, IDEA, entre otros; que podrían substituirlo logrando la misma funcionalidad. La caracterítica básica que se debe mantener al elegir este algoritmo es que sea determinístico y utilice criptografía de llave simétrica por aspectos de eficiencia. En lo particular el esquema se implemento con AES por ser un estándar bien conocido.

En el capítulo 3 se revisaron varios esquemas de homomorfismos, sin embargo, en su mayoría trabajan bajo la operación multiplicación y como ya ha sido revisado, el objetivo es dotar al servidor la capacidad de resolver consultas que involucran sumas. Por lo tanto se buscó un algoritmo cuyas propiedades homomórficas fueran bajo dicha operación, por lo que Paillier resulto una opción viable. Sin embargo, es posible experimentar con otros algoritmos que satisfagan esta característica.

Finalmente haremos referencia al cifrador que preserva el orden, este tipo de cifrado fue revisado en detalle en el capítulo 4 y se explicó que antes del esquema propuesto

por Boldyreva et.al. únicamente existe otro algoritmo bajo este paradigma, desarrollado por Agrawal et.al. Sin embargo, este esquema presenta ciertas desventajas ya que el algoritmo de cifrado requiere tomar como entrada todos los textos en claro de la base de datos. En muchos escenarios no es posible que los cliente conozcan todos estos textos en claro con anticipación. Por lo tanto el esquema de Boldyreva et. al. brinda una caraterística muy deseable al incluir un algoritmo de cifrado que puede procesar los textos en claro en línea.

Con la discusión aquí presentada esperamos que el lector tenga una idea clara de por qué los algoritmos elegidos son una buena opción dentro de la literatura. En la próxima sección se explica cómo estos tres algoritmos han sido acoplados en un solo esquema, al igual que el funcionamiento del mismo.

5.3 Funcionamiento del esquema propuesto

En esta sección abordaremos cómo interactúan los bloques básicos de nuestra propuesta, por lo que se revisa el funcionamiento dividido en tres grandes áreas: proceso de cifrado, proceso de consulta en información cifrada y finalmente proceso de descifrado. En la primera sección se trata además las adecuaciones realizadas en el uso de los algoritmos de cifrado, ya que al ser independientes requirieron cambios para su correcto acoplamiento dentro del modelo de bases de datos subcontratadas.

5.3.1 Proceso de cifrado

Recordemos que las dos entidades básicas del modelo de bases de datos subcontratadas son: **el cliente** y **el proveedor**, el primero genera todas las operaciones relacionales, éstas deben ser ejecutadas por el servidor afectando la base de datos cifrada que reside en él.

A continuación revisaremos como se realiza el proceso de cifrado de la información que el cliente requiere introducir en la base de datos, para ello haremos referencia a la figura 5.2.

El proceso de cifrado recibe como entrada un registro en texto en claro, el cual está constituído por varios atributos que son analizados uno a uno y son clasificados en numéricos y alfanuméricos.

Cuando el atributo es del primer tipo se cifra bajo dos de los algoritmos de nuestro esquema: el cifrador basado en homomorfismos y el que preserva el orden. En caso que resultara ser alfanumérico se utiliza el cifrador por bloques y posteriormente es códificado en base64, este último paso es requerido para que la cifra este conformada

únicamente de caracteres válidos que si pueden ser almacenados en la base de datos.

Después de haber aplicado estos algoritmos según el caso que corresponda entonces se obtiene un registro con atributos cifrados que puede ser almacenado.

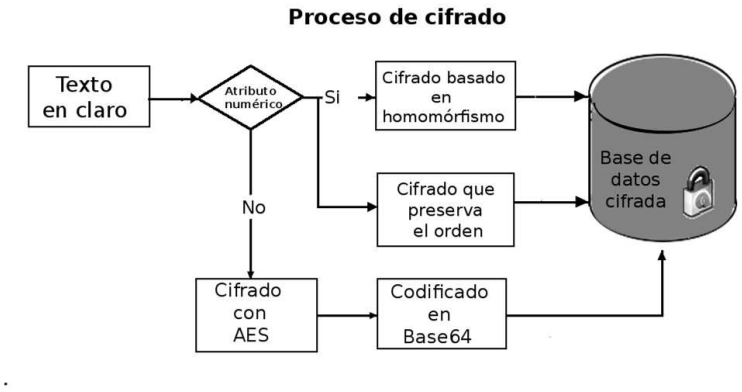

Figura 5.2: Proceso de cifrado

Ahora que ya hemos explicado el proceso de cifrado, se debe destacar dos puntos importantes: la redundancia existente para los atributos numéricos y cómo esta siendo utilizado el algoritmo de Paillier.

Debido a que en la base de datos se almacenan dos cifras para cada atributo numérico: la producida por el algoritmo de Paillier y la producida por Boldyreva et.al. se tiene dos opciones para obtener el texto en claro de dicho atributo. Sin embargo, en nuestro esquema optamos por descifrar con Paillier por ser mucho más eficiente que Boldyreva. Por lo tanto el algoritmo de descifrado de Boldyreva et.al. jamás es utilizado.

El otro punto importante es que pese a que el algoritmo de Paillier es un esquema de llave pública, dentro de nuestra propuesta no es utilizado como tal, pues el interés en este esquema radica en sus propiedades homomórficas y no en sus características de cifrado asimétrico. Debido a que la comunicación que existe entre el cliente y el servidor es únicamente en un sentido, pues el servidor jamás envía un texto cifrado al cliente. Por lo tanto, no es necesario revelar la llave pública N.

Modelo de almacenaje

Cómo ya se ha mencionado los algoritmos de cifrado se aplican a diferentes tipos de atributos. Para explicar en base a qué se decide cuál algoritmo aplicar a un determinado atributo, se hace referencia a las tablas 5.1 y 5.2 que muestran los registros de la tabla **Empleados** en *texto en claro* y cómo luciría su equivalente en *texto cifrado*.

Sea R una relación con el siguiente conjunto de atributos $R = \{r_1, \ldots, r_n\}$, e.g. **Empleados** es una relación que tiene por atributos: *el Identificador del Empleado, el Nombre del Empleado, el Salario del Empleado, Comisión, Dirección y el Identificador del departamento para el cual trabaja*.

IdEmp	NombreEmp	SalarioEmp	Comisión	Dirección	IdDep
23	Tom	10K	2k	Av.Politecnico	Fin
860	Mary	20K	1k	La Rioja	Comp
320	John	5k	1k	Bamba	Comp
44	Lucy	15k	2k	Matanzas	Comp

Tabla 5.1: Tabla Empleados en claro

IdEmpC	NameEmpC	SalarioC	SalarioPO	ComisiónC	ComisiónPO	DirecciónC	IdDepC
?xEw	bcd	7	30	27	7	a34?	dre?
xr?t	ts=	18	50	17	5	dklj	kie
abtx	nmr	2	15	12	5	aiop	kie
t5wt	hlmk	31	40	41	7	tryu	kie

Tabla 5.2: Tabla Empleados cifrada

R se representa en el servidor como R^S (tabla cifrada) la cual contiene los atributos que ya fueron cifrados basándose en la siguiente clasificación:

- Atributos cifrados determinísiticamente
- Atributos de agregación
- Atributos de intervalos

Ahora veamos cada una de estas clases.

Atributos cifrados determinísticamente

Este tipo de atributos se encuentran en R y son aquellos en los que selecciones de igualdad, *equijoins*, y agrupación pueden ser ejecutadas. Para cada atributo en R, R^S contiene un atributo $F_i^S = E_k(F_i)$, donde $E_k()$ es un cifrador por bloques (AES, descrito en el apéndice B) el cual es aplicado al valor del campo F_i. (ver la figura 5.2 Proceso de cifrado (atributo no numérico)).

E.g., en la tabla (ver la tabla 5.1), *el id del empleado, nombre, dirección y id del departamento* son considerados bajo esta categoría, y la tabla cifrada contiene únicamente un atributo por cada uno, tal y como se muestra en la tabla 5.2; estos atributos son: *IdEmpC, NameEmpC, DirecciónC, IdDepC*.

Atributos de agregación

Son atributos de R en donde se espera que se apliquen consultas de agregación, e.g., **el cliente** puede estar interesado en saber cuanto es el total que invierte la compañía en salarios, o el total de comisiones en un departamento. (ver la figura 5.2 Proceso de cifrado (atributo numérico)).

Para cada atributo en R de este tipo, R^S contiene un atributo $F_i^S = E_P(F_i)$, donde $E_P()$ es el cifrador basado en homomorfismo de Paillier (ver Capítulo 3) el cual es aplicado al valor del campo F_i E.g., en la tabla **Empleados** (ver la tabla 5.1), *SalarioEmp, Comisión* son considerados atributos de agregación debido a que son numéricos por lo que pueden participar en totales, promedios, etc. Y los atributos cifrados que los representan son: *SalarioEmpC, ComisiónC* (ver la tabla 5.2).

Atributos de intervalo

Son atributos de R en los que esperamos aplicar consultas que evalúan un intervalo. Para cada F_i, R^S contiene un atributo $E(F_i) = OPE_{D,R,F_i}$, donde OPE_{D,R,F_i} es una función que preserva el orden. (ver la figura 5.2 Proceso de cifrado (atributo numérico)).

E.g, en la tabla empleados (ver 5.1), el *SalarioEmp y la Comisión* son considerados bajo esta categoría pues son numéricos y el cliente puede estar interesado en conocer cuántos empleados ganan un salario entre dos límites. La tabla cifrada 5.2 tiene un atributo por cada uno de los originales: *SalarioPO y ComisionPO*.

Por lo tanto si el cliente desea introducir un nuevo registro en la tabla empleados, escribe la instrucción SQL en claro:

```
INSERT INTO Empleados

VALUES ('23','Tom',10k,2k,'Av.Politecnico','Fin')
```

La consulta es traducida en base a la clasificación antes presentada y luce como sigue:

```
INSERT INTO Empleados

VALUES ('?xEw','bcd',7,30,27,7,'a34','dre')
```

Finalmente es enviada al servidor para que la ejecute.

En la siguiente sección revisaremos cómo el esquema soporta los tipos de consultas discutidos previamente: consultas con atributos alfanuméricos, consultas de agregación y finalmente consultas de intervalos.

5.3.2 Proceso de consulta sobre información cifrada

Se trata con detenimiento la operación de **consulta** por tener características que la convierten en un reto importante dentro de este modelo; y porque permite comprender cómo es posible realizar las otras operaciones.

En la figura 5.3 se muestra el proceso de traducción de consultas, inicialmente el cliente escribe una consulta en claro posteriormente el traductor analiza qué tipo de consulta es y dependiendo de ello aplicará uno o más algoritmos, en caso de que la cláusula `WHERE` contenga una restricción alfanumérica este valor es cifrado con AES y posteriormente se aplica el codificado base64, si la restricción consiste en evaluar límites entonces Boldyreva et.al. es utilizado para cifrarlos, finalmente si en la cláusula `SELECT` aparece la instrucción `SUM` entonces se trata de una consulta de agregación y el cifrado de Paillier es ocupado.

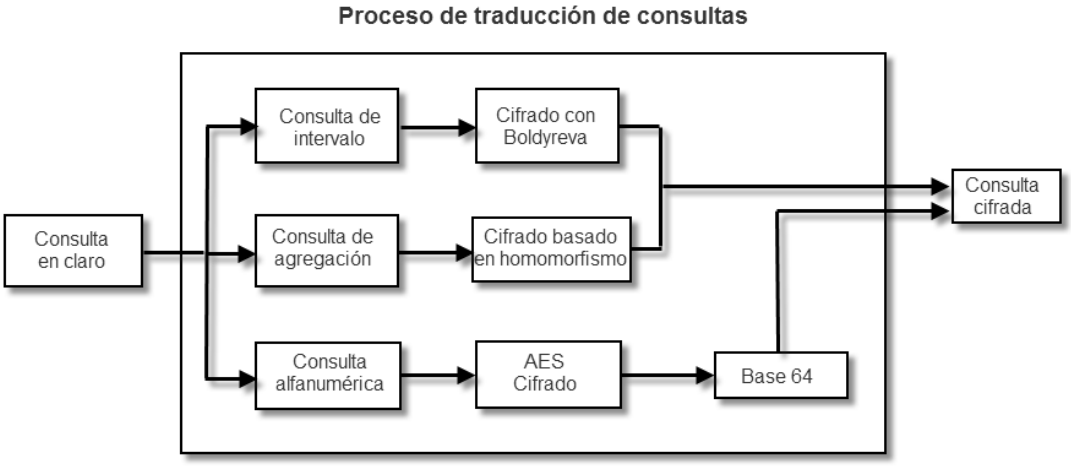

Figura 5.3: Proceso de traducción de consultas

Enseguida se analiza este proceso con consultas más complejas, para ello hacemos referencia a la tablas departamentos y su relación con la tabla Empleados. El contenido de la tabla **Departamentos** en claro luce como la mostrada en la tabla 5.3.

IdDep	NombreDep
Comp	Computación
Fin	Finanzas

Tabla 5.3: Tabla Departamentos en claro

Como ambos atributos son cifrados determinísticamente el contenido de la tabla en el

servidor luce como en la tabla 5.4.

IdDepC	NombreDepC
kie	atr@'?
dre?	k53l

Tabla 5.4: Tabla Departamentos cifrada

Esto implica que el servidor contiene ambas tablas cifradas es decir, la tabla **Empleados cifrada** (ver 5.2) y la tabla **Departamentos cifrada** (ver 5.4).

Consulta de datos numéricos y alfanuméricos

Si retomamos la consulta del capítulo 4 en la que **el cliente** requiere seleccionar *todos lo empleados cuyo salario nominal es mayor o igual que* $10k$ *y menor o igual que* $20k$. Esta consulta se escribe:

```
SELECT NombreEmp, SalarioEmp FROM Empleados
    WHERE Salario ≥ 10K AND Salario ≤ 20K
```

Primero se re-escribe la consulta haciendo uso del algoritmo de preservación de orden (tabla 4.3) a los valores 10K y 20K y se sustituyen en la consulta original. La consulta re-escrita luce como:

```
SELECT NombreEmp, SalarioC FROM Empleados
  WHERE SalarioPO ≥ 30 AND SalarioPO ≤ 50
```

Es importante notar que en la cláusula *SELECT* el campo *SalarioEmp* fue sustituido por *SalarioC*, esto se debe a que bajo el nombre de este atributo están almacenados los salarios cifrados con el algoritmo de Paillier; el cual se ocupara siempre que se haga referencia a atributos numércos.

Posteriormente se envía la nueva consulta al servidor y éste podrá regresar los registros coincidentes aplicando la consulta recibida. En este caso en particular el servidor devolverá al cliente los registros de Tom, Mary y Lucy, queda excluido el de John.

Estos tres registros son descifrados del lado del cliente: los nombres de los empledos (NombreEmp) se descifran con AES y los salarios con Paillier.

Consulta de agregación e intervalos

Analicemos el siguiente caso, **el cliente** puede requerir ejecutar una consulta que obtenga *la suma de todos los salarios de los empleados que trabajen en el departamento de Computación y cuyo salario sobre pase los 10k*. Esta consulta se escribe en SQL como se muestra a continuación:

```
SELECT SUM(E.salario)
FROM Empleados as E, Departamentos as D
WHERE E.did = D.did
AND D.IdDep ="Comp"
AND E.salary > 10k
```

Esta consulta tiene mayor complejidad ya que implica **Consultas de agregación**, **Consultas de rango** y **Comparaciones simples**.

Para procesarla se realiza primero la re-escritura, haciendo los siguientes cambios:

1. Se cifra la cadena "Comp" de forma determinística, es decir, utilizando AES.
2. Se cifra la cantidad de 10k con el cifrador que preserva el orden.
3. Finalmente al identificar que se requiere hacer una suma se multiplican todos los atributos SalarioC de los registros que sean devueltos por el servidor.

La consulta re-escrita luce asi:

```
SELECT E.SalarioC as subtotal
FROM Empleados as E, Departamentos as D
WHERE E.IdDep = D.IdDep
AND D.IdDep ="kie"
AND E.SalarioPO > 30
```

En este caso el primer filtro dice que los empleados deben ser del Departamento de Computación, por lo que unicamente los registros de Mary, John y Lucy son posibles candidatos.

Posteriormente se aplica la restricción del salario, por ello sólo los que tengan un valor mayor de 30 en el campo que preserva el orden (**SalarioPO**) deben ser registros devueltos; con lo que se elimina John. Y únicamente el salario de Mary y Lucy son devueltos.

Se realiza la multiplicación de todos los subtotales, obteniendo un valor acumulado.

$$total = total * subtotal$$

El cliente descifra por medio de Paillier y obtiene la cantidad deseada.

Ya hemos visto como se pueden resolver los distintos tipos de consultas y se ha mencionado que una vez que el servidor devuelve los registros coincidentes, el cliente puede descifrar, en la próxima sección se describe este proceso con mayor detalle.

5.3.3 Proceso de Descifrado

En la figura 5.4 se muestra el proceso de descifrado el cual recibe a la entrada una consulta cifrada, es decir, la consulta en claro ya re-escrita. El servidor analiza qué tipo de atributos contienen los registros cifrados devueltos por el servidor y los descifra basado en la siguiente clasificación si es numérico el descifrado se realiza con Paillier y si es alfanumérico utiliza base64 en modo de decodificado y finalmente realiza el descifrado con AES.

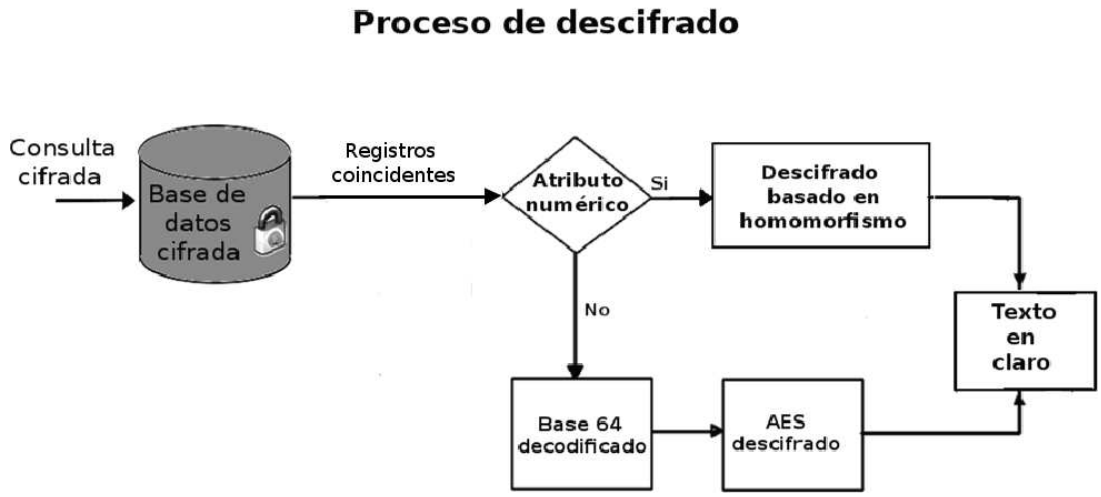

Figura 5.4: Proceso de descifrado

Hemos visto cómo el esquema soporta las operaciones de *inserción y consulta*, nos queda pendiente deducir del funcionamiento mostrado cómo operan las operaciones de *modificar y eliminar*.

El proceso es el mismo, el cliente escribe la instrucción SQL en claro y ésta es re-escrita. En el caso de la operación *modificar* se debe revisar si el atributo que se quiere actualizar es númerico o alfanumérico, en el primer caso el nuevo valor es cifrado con Paillier y Boldyreva, si existen condiciones en el `WHERE` se resuelven exactamente igual que en la consulta. Si el atributo es alfanumérico se aplica el cifrador AES y el codificado base64 para cifrar el nuevo valor y se substituye en la consulta.

Finalmente la operación *eliminar* solamente requiere que la cláusula `WHERE` sea anali-

zada, se re-escribe la instrucción y es enviada al servidor. En estas operaciones no aplica ocupar el proceso del descifrado.

Se ha presentado como el esquema cumple con todas las características y operaciones establecidas en la sección 5.1.1, entonces podemos concluir que es lo suficientemente robusto para brindar los servicios ofrecidos por una base de datos en claro.

Para poder implementar este esquema es necesario determinar algunos detalles entre los que cabe mencionar: la arquitectura de la solución, la elección de parámetros, la tecnología que es posible utilizar, entre otros. Estos puntos son retomados en el siguiente capítulo.

Capítulo 6

Implementación del esquema

La teoría es el capitán y la práctica, el soldado.

Leonardo da Vinci

En este capítulo abordaremos los detalles de implementación requeridos para el esquema de cifrado planteado en el capítulo 5. En la primera sección se trata la arquitectura con que trabaja el esquema y la estructura general de la solución, en la segunda sección se trata la organización de la base de datos, con lo que se da paso al funcionamiento de la solución, posteriormente se explica la tecnología utilizada y la organización del código. En la última sección se presentan los resultados del experimento realizado con una base de datos con un mayor número de registros, en la que se midieron tiempos de cifrado y descifrado.

6.1 Arquitectura y estructura general de la solución

El sistema trabaja en una arquitectura cliente - servidor, donde el dueño de este último provee el servicio de bases de datos. Por lo que el esquema encaja bien en la tendencia actual de cómputo nube, en la cual se ofrecen servicios de computación a través de Internet (ver la figura 6.1).

El modelo con el que se trabaja es conocido en la literatura como modelo de bases de datos subcontratadas *Database as a Service (DAS)*. El **Cliente** confía sus datos al **Proveedor** y requiere mantener la funcionalidad que tiene una base de datos local, es decir, debe poder *insertar, modificar, consultar y eliminar* información. La complejidad aumenta debido a que la base de datos se encuentra cifrada.

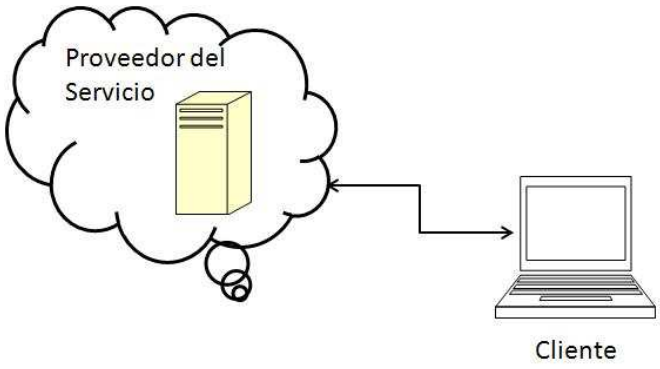

Figura 6.1: Modelo en cómputo nube: servicio de bases de datos (DAS).

A continuación se expone los módulos con los que cuenta la solución, para permitir que el cliente siga generando sus consultas en texto en claro y obtenga de forma transparente el resultado que habría obtenido si lo hubiera realizado en la base de datos en claro.

Estructura general de la solución

Debido a que la información es almacenada en forma cifrada y que el cliente genera consultas a los datos en texto en claro, los módulos básicos con los que debe contar la solución son:

- Interfaz de usuario: el cliente genera sus consultas en texto en claro.
- Traductor de consultas: transforma la consulta en claro al lenguaje comprendido por el servidor (texto cifrado).
- Módulo de descifrado: descifra los registros que constituyen la respuesta de la consulta.

Las dos entidades básicas del modelo son: **el cliente** y **el proveedor** del lado de éste último se encuentra la base de datos cifrada y es **el cliente** quien genera las consultas de texto en claro. Posteriormente se re-escriben haciendo uso de un módulo especializado, esta tarea es transparente para el cliente, sin embargo, debe ser realizada de su lado, puesto que el servidor reside en un lugar remoto y la consulta viajará por un canal inseguro. De este modo la consulta ya cifrada puede ser enviada al servidor y este la ejecuta devolviendo los registros correspondientes, finalmente del lado del cliente se realiza el descifrado para obtener los resultados. Este proceso se muestra en la figura 6.2.

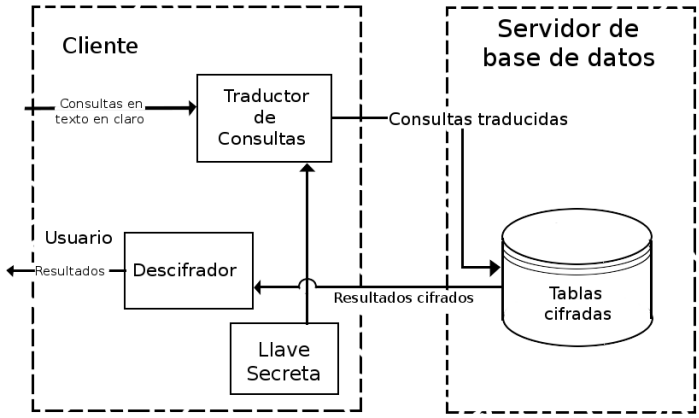

Figura 6.2: Arquitectura general de la solución.

El módulo de traductor de consultas contiene la lógica para re-escribir todas las operaciones relacionales generadas por el cliente, de tal forma que el servidor pueda procesarlas. Es decir, que el esquema mostrado en el capítulo 5 estará contenido en este módulo. Por otra parte el módulo de descifrado es el que contiene a el proceso explicado en la sección 5.3.3

6.2 Funcionamiento de la solución

En esta sección se presenta con mayor detalle cómo se comunican e interactuan los módulos de la solución, los parámetros que son generados previamente para el funcionamiento de la aplicación, y los servicios que brinda.

6.2.1 Generación de parámetros iniciales

Debido a que la aplicación utiliza el homomorfismo de Paillier, es necesario utilizar ciertos parámetros que pueden ser pre-calculados y únicamente leerlos para el uso de la aplicación. Estos parámetros son: los primos p y q, $N = p \cdot q$, N^2, $\phi(N)$, $\phi(N)^{-1}$; los cuales son almacenados en el archivo `parametros.dat`.

Cuando este archivo no se encuentra, la aplicación supone que es la primera vez que es utilizada y genera los parámetros antes mencionados, resguardandolos en el archivo; es importante tomar en cuenta que si se cambian los valores de éstos, el descifrado no será correcto.

La seguridad en el algoritmo de Pailier está ligada a la longitud en bits de p y q (ver

capítulo 3). Por lo que idealmente estos primos deben ser grandes, en está aplicación en particular se utilizo una longitud de 32 bits. Aunque esto puede parecer inseguro, hay que remarcar que en nuestro esquema ninguno de los parámetros se hace público, por lo que la seguridad brindada es la de una llave de 64 bits, es decir, la longitud de N; sabiendo que este parámetro es secreto.

Sin embargo, la única restricción para no ocupar primos más grandes es que la base de datos pierde desempeño, por lo que es recomendable fijar la longitud en bits de estos primos según sea el escenario de la aplicación.

6.2.2 Organización de la base de datos

La aplicación asume que la organización de cada tabla cifrada está estructurada de la siguiente manera:

Por cada atributo alfanumérico existente en la tabla en claro, e.g. *IdEmp*, existe únicamente un atributo en la tabla cifrada con el mismo nombre que en la tabla en claro (*IdEmp*); donde se almacena el valor del atributo cifrado con AES-128 y posteriormente el codificado base64 (ambos descritos en el apéndice B).

Por cada atributo numérico en la tabla en claro (*SalarioEmp*), existen dos atributos en la tabla cifrada: uno para el cifrado con homomórfico de Paillier y otro para el cifrado que preserva el orden, donde el primero mantendrá el mismo nombre de la tabla en claro (*SalarioEmp*) y será de tipo *numeric* y el segundo el nombre original en la tabla en claro más una 'o' (*SalarioEmpo*) y su tipo es *bigint*.

El uso de este tipo de datos se justifica ya que en el caso del homomorfismo de Paillier se busca que los primos que se utilizan sean grandes, y el texto cifrado será del orden de N^2, por lo que los 64 bits que nos ofrece el entero más grande no es suficiente.

En el caso del algortimo que preserva el orden, el entero más grande que se requiere manejar es el que una base de datos en claro usualmente utiliza, es decir, hablamos de enteros de hasta 32 bits.

Debido a que la implementación del algoritmo que preserva el orden utiliza como dominio $D = 2^{16}$ y como rango $R = 2^{24}$, para cifrar un número de 32 bits es necesario llamar al algoritmo dos veces, lo anterior implica que el cifrado tendrá a lo más 48 bits, número que puede ser almacenado en un *bigint* de 64 bits (ver sección 4.4).

6.2.3 Servicios de la aplicación

Aquí describiremos que operaciones es capaz de resolver la aplicación. Para ello revisaremos brevemente cada uno de los módulos que contiene.

La interfaz utilizada por el usuario es sencilla, la aplicación trabaja en consola, y le presenta al usuario la oportunidad de elegir si desea introducir un registro, realizar una consulta o salir del programa. Para mayor información vea el manual del usuario apéndice D de este documento.

Una vez que el cliente introduce la instrucción que desea ejecutar, el traductor de consultas puede realizar una de dos actividades: la primera es preparar un registro que el cliente desee ingresar a la base de datos y la segunda es re-escribir una consulta solicitada por el cliente.

La consulta en claro (inserción o consulta) es analizada y posteriormente re-escrita basándose en los siguientes criterios:

Inserción

Cuando se desea realizar una inserción, el usuario debe indicar el nombre de la tabla y posteriormente los valores de los atributos de dicha tabla. E.g. para ingresar un registro a la tabla **Empleados** (figura 5.3) la instrucción SQL para el texto en claro sería:

```
INSERT INTO Empleados
VALUES (IdEmp,NombreEmp,SalarioEmp,Comision,Direccion,IdDep)
```

Una vez que haya sido re-escrita quedaría como:

```
INSERT INTO Empleados
VALUES (IdEmp,NombreEmp,SalarioEmp,SalarioEmpo,

Comision,Comisiono,Direccion,IdDep)
```

Esta nueva instrucción respeta la estructura definida para cada tabla dentro de la base de datos cifrada (ver sección 6.2.2).

Consulta

En esta operación el usuario ingresa el nombre del archivo que contenga la consulta en claro que debe ser procesada. Existen distintas variantes que deben ser cubiertas:

Consulta con condiciones alfanuméricas: en este caso la aplicación realiza el análisis sintáctico sobre la cláusula `WHERE` identificando los atributos que son alfanuméricos y sustituyendo el valor por el cifrado con AES y base64. E.g. En una consulta que incluye la condición `WHERE IdDep=Comp` es sustituido por `WHERE IdEmp=kie`.

Consulta de Agregación: en este caso la aplicación realiza el análisis sintáctico de los atributos incluidos en la cláusula `SELECT` y determina si hay alguna función de

agregación, sabiendo que debe aplicar las propiedades del homorfismo de Paillier para obtener el resultado adecuado (ver capítulo 3).

Consulta de Intervalos: en este caso la aplicación realiza un análisis sintáctico sobre la cláusula `WHERE` en donde es necesario hacer dos cambios: la sustitución del nombre del atributo por el nombre del atributo más una 'o' y sustituyendo el valor de los límites por su cifrado con preservación de orden. E.g. En una consulta que incluye la condición `WHERE SalarioEmp < 10k` es sustituido por `WHERE SalarioEmpo < 30`.

Finalmente, cuando los registros han sido seleccionados, el módulo de descifrado aplica el algoritmo correspondiente: si se trata de un atributo alfanumérico se descifra aplicando AES y el decodificado de base64. Si se trata de un atributo numérico se descifra con Paillier.

6.3 Tecnología utilizada y organización del código

En esta sección abordaremos la tecnología utilizada para la implementación del esquema propuesto, podemos mencionar dos grandes áreas que fueron tratadas:

- Primitivas criptográficas: Estas fueron implementadas en el lenguaje **C** y para el manejo de variables con multiprecisión se utilizo la biblioteca **gmp**.

- Base de datos relacional: El motor es **Postgresql**, sin embargo, el esquema de cifrado es transparente para trabajar con cualquier otro motor de bases de datos relacional como SQL Server, Oracle entre otros. Para facilitar la administración usamos **pgAdmin**.

Para poder combinar C y Postgresql se utilizó la biblioteca **libpq**, que actúa como una interfaz que permite realizar consutas y recibir los resultados de las mismas desde aplicaciones que trabajan en el cliente. El Sistema Operativo bajo el que se desarrolló es Linux.

Estas herramientas interactuan entre si tal y como se muestra en la figura 6.3. La aplicación descansa en C y Postgresql, y las herramientas de administración y comunicación son respectivamente pgAdmin y libpq, y el sistema operativo es Linux.

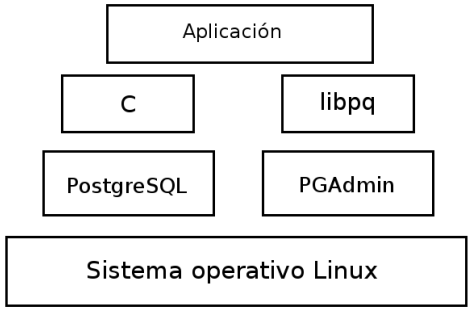

Figura 6.3: Tecnología utilizada.

Organización del código

El código de la aplicación se encuentra organizado en las siguientes bibliotecas:

- Scheme: Es el módulo principal de la aplicación. Aquí es donde se realiza la conección a la base de datos y el menú principal.
- Aritmetica: En este módulo se encuentra la aritmética utilizada, como la generación de números primos, el algoritmo extendido de Euclides para encontrar inversos, entre otros.
- AES: En este módulo se encuentra la implementación del algoritmo AES-128.
- DAES: Este módulo incluye la implementación del algoritmo de descifrado de AES-128.
- base64: En este módulo se encuentra el códificado y decódificado de base64.
- lazysample: Aqui se encuentra la implementación del algoritmo de Preservación de orden.
- homomorphic: Aqui se encuentra la implementación de Paillier.

En el apéndice D se describe cómo puede ser instalada la aplicación y las herramientas utilizadas. En la próxima sección se describen los resultados obtenidos para un experimento básico con un conjunto de datos de 200 registros.

6.4 Resultados experimentales

En esta sección se describe el experimento realizado con el fin de probar el esquema propuesto, el cual consiste de dos escenarios: una base de datos en claro y una base de datos cifrada. En ambos se ejecutaron el mismo tipo de operaciones, en la primera prueba se midieron tiempos para poblar una tabla con un conjunto de datos de 50, 100, 150 y 200 registros, posteriormente se ejecutaron las mismas consultas en ambas bases y se compararon los tiempos de respuesta. Los resultados obtenidos son presentados en los próximos apartados.

6.4.1 Estructura de la base de datos

La base de datos utilizada está constituída por tres tablas:

- **Empleados** Contiene los siguientes atributos: *CodigoEmp, NombreEmp, CodigoCiudad, CodigoDep, Salario y Comision*
- **Departamentos** Cuyos atributos son: *CodigoDep, NombreDep*
- **Ciudades** Sus atributos son: *CodigoCiudad, NombreCiudad, Pais*

Esta base de datos se muestra en la figura 6.4.

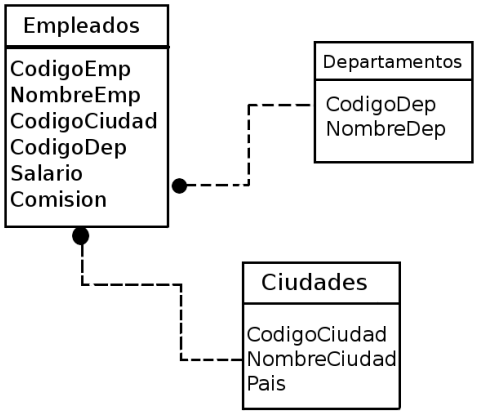

Figura 6.4: Base de datos en claro utilizada.

El contenido de las tablas Departamentos y Ciudades fue previamente inicializado con los registros de las tablas 6.1 y 6.2 respectivamente:

CodigoDep	NombreDep
Comp	Computación
Fin	Finanzas
Pro	Producción

Tabla 6.1: Contenido de la tabla Departamentos

CodigoCiudad	NombreCiudad	Pais
Gan	Guanajuato	Mexico
Gua	Guadalajara	Mexico
Pue	Puebla	Mexico
Tla	Tlaxcala	Mexico
San	SantaAna	El Salvador

Tabla 6.2: Contenido de la tabla Ciudades

6.4.2 Primera prueba

Se midieron los tiempos para poblar la tabla Empleados en texto en claro y texto cifrado. El conjunto de datos es cargado desde un archivo de texto, pero en el caso de la base de datos cifrada, se debe realizar la traducción de cada registro, en este caso en particular implica: cuatro llamadas al cifrador por bloques AES y el codificado base64 que corresponden a los atributos: *CodigoEmp, NombreEmp, CodigoDep, CodigoCiudad*; dos llamadas al cifrador basado en homomorfismos Paillier debido a los atributos: *Salario y Comisión*; finalmente se requiere dos llamadas a Boldyreva por los mismos dos atributos.

Por lo tanto, de antemano sabiamos que el proceso en la base de datos cifrada sería más lento que en la base de datos en claro, notamos que el comportamiento del tiempo es líneal con respecto al tamaño de la tabla, tarda aproximadamente un segundo para cifrar un registro e introducirlo a la base de datos.

Los resultados para un conjunto de 50, 100, 150 y 200 registros son mostrados en la tabla 6.3, la gráfica correspondiente se muestra en la figura 6.5.

Números de Registros	Texto cifrado	Texto en claro
50	49887.8480	133.80
100	102097.8427	180.69
150	153869.5556	262.16
200	204828.8212	287.51

Tabla 6.3: Tiempos para poblar la tabla Empleados en milisegundos

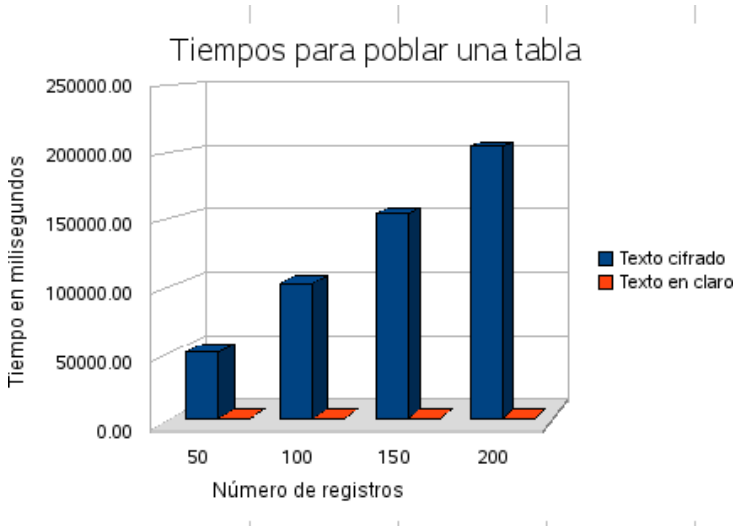

Figura 6.5: Gráfica de tiempos para poblar la tabla Empleados.

Es importante recalcar que en la base de datos en claro el tiempo para poblar una tabla también es líneal con respecto al número de registros, en la figura 6.5 da la impresión de ser constante únicamente por el factor de la escala.

El poblar tablas en texto cifrado es costoso, sin embargo, este proceso se realiza una sola vez y es fuera de línea; además es posible mejorar estos tiempos ya que nuestra implementación no es eficiente.

6.4.3 Segunda prueba

Posteriormente se ejecutó un conjunto de consultas representativas de lo provisto por el modelo, los tiempos de consulta son más rápidos que los de ingreso de información. Siendo lo costoso el proceso descifrado y no en sí la selección de los registros. Enseguida se presentan las consultas experimentadas y en la tabla 6.4 se muestra las llamadas que son realizadas por consulta a los distintos algoritmos de cifrado.

- C1: `SELECT * FROM Empleados`
- C2:
```
SELECT e.CodigoEmp,e.NombreEmp,c.NombreCiudad,c.Pais
FROM empleados AS e
JOIN ciudades as c on e.CodigoCiudad = c.CodigoCiudad
```
- C3: `SELECT SUM(e.Salario) FROM empleados as e`

- C4:
```
SELECT e.CodigoEmp,e.NombreEmp,
c.NombreCiudad,c.Pais,d.NombreDep
FROM empleados as e JOIN departamentos as d
ON e.CodigoDep = d.CodigoDep
JOIN ciudades as c on e.CodigoCiudad= c.CodigoCiudad
WHERE e.CodigoCiudad = 'Pue'
```

- C5:
```
SELECT e.CodigoEmp,e.NombreEmp,c.NombreCiudad,c.Pais
FROM empleados as e JOIN departamentos as d
ON e.CodigoDep= d.CodigoDep
JOIN ciudades as c on e.CodigoCiudad = c.CodigoCiudad
WHERE e."Salario" > 50000
```

En la siguiente tabla se presentan las operaciones que deben ser realizadas por registro y por condición en la cláusula `WHERE` por cada una de las consultas experimentadas.

Identificador	Operaciones por consulta
C1	4 llamadas AES modo descifrado,2 llamadas a Paillier descifrado
C2	4 llamadas AES modo descifrado
C3	procesar 200 salarios con homomorfismo de Paillier
C4	4 llamadas AES modo descifrado por registro y 1 más por condición alfanumérica
C5	4 llamadas AES modo descifrado y una llamada Boldyreva por condición de límites

Tabla 6.4: Costo computacional de ejecución de consultas en bases de datos cifrada

Identificador	Número de Registros	Texto cifrado (ms)	Texto en claro (ms)
C1	200	156.8063	19.6553
C2	200	121.7623	19.7046
C3	1	12.5736	3.7399
C4	34	51.1903	8.5883
C5	110	551.2373	14.5799

Tabla 6.5: Tiempos en milisegundos de consultas procesadas en ambas bases de datos (texto en claro,texto cifrado)

A continuación presentamos la gráfica de éstos tiempos figura 6.6

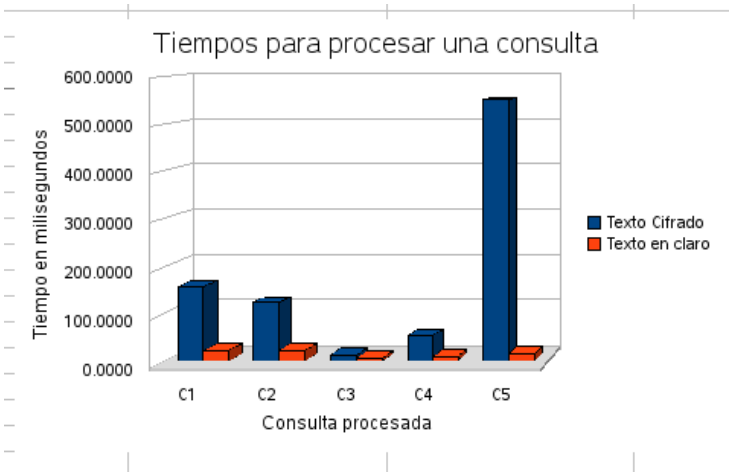

Figura 6.6: Gráfica de tiempos para ejecutar consultas.

Como conclusión tenemos que las consultas más caras son aquellas en las que el esquema de Boldyreva se ve involucrado, definitivamente es necesario realizar un mayor número de experimentos, pero hasta aquí ya tenemos una idea del comportamiento del esquema. Lo que se requiere saber para calcular el tiempo de una consulta es de cuantos atributos están siendo manejados y de que clase son, además del número devuelto de registros.

Este esquema sugiere varias mejoras que podrían realizarse entre las que cabe mencionar: la optimización de las primitivas criptográficas, la mejora del análizador sintáctico y el diseño de un optimizador de consultas específico para información cifrada.

Capítulo 7

Conclusiones

> El tiempo es el mejor autor; siempre encuentra un final perfecto
>
> *Charles Chaplin*

En este capítulo se abordan los resultados obtenidos y las conclusiones a las que se llegó en base a ellos. Posteriormente se presenta una serie de posibles líneas de investigación para trabajo futuro.

7.1 Conclusiones

Este trabajo propone un esquema de cifrado que permite realizar consulta en bases de datos cifradas de forma eficiente; estas consultas pueden involucrar funciones de agregación, evaluación de rangos y condiciones alfanuméricas.

La principal aportación de esta investigación es proporcionar un esquema de cifrado lo suficientemente robusto como para mantener la funcionalidad de una base de datos relacional en claro; cubriendo el servicio de confidencialidad y haciéndolo ideal para ser utilizado en el contexto de *bases de datos subcontratadas*.

Esta aseveración tiene fundamento en el levantamiento de las operaciones que es posible realizar en una base de datos, para lo cual tomamos de referencia la documentación de Postgresql [33]. Con esta información se construyó la tabla 7.1 donde se muestran las diferentes operaciones y cuáles de ellas son soportadas por el esquema. Para mayor comprensión de las operaciones mostradas en la tabla, consulte el apéndice C.

Ésta es una aportación importante debido a que aún cuando existen múltiples propuestas para la ejecución de consultas en bases de datos cifradas, la mayoría únicamente se concentra en un tipo específico de consulta.

Para evaluar el esquema se realizó una implementación de los algoritmos que lo cons-

tituyen: el cifrado basado en homomorfismos de Paillier con lo que se resuelve la operación de agregación SUM, el cifrado de preservación de orden que hace posible la evaluación de rangos y otras operaciones como MIN, MAX (también de agregación); el cifrado de AES-128 para restricciones alfanuméricas y el poder mantener la cualidad de que la base sea relacional.

Todas estas primitivas criptograficas se implementaron en **C**, permitiendo que la multiprecisión requerida por algunas variables y ciertas operaciones de la aritmética fueran resueltas utilizando las funciones que incluye la biblioteca **gmp**.

TABLA RESUMEN	
Operación	**Estatus**
Inserción Implicita	Soportado
Inserción Explicita	Soportado
Consultas a una tabla	Soportado
Consultas con restricciones de menor o mayor que	Soportado
Consultas con restricciones de menor igual o mayor igual que	Soportado
Consultas con el operador LIKE	**No Soportado**
Consultas con modificadores	Soportado
Consultas con JOINS	Soportado
Sub consultas	Soportado
Consultas con SUM	Soportado
Consultas con MIN, MAX, COUNT	Soportado
Consultas con AVG	Soportado
Actualizaciones	Soportado
Eliminaciones	Soportado
Consultas que implican sumas y restas	Soportado
Consultas que implican multiplicacion y divisiones	**No Soportado**
Consulta en tipos de datos enteros y cadenas	Soportado
Consulta en tipos de datos flotantes y fechas	**No Soportado**

Tabla 7.1: Operaciones permitidas y no permitidas por el Esquema de Cifrado

Haciendo uso de estos algoritmos de cifrado se realizó una pequeña aplicación que permite interactuar con una base de datos, permitiendo realizar algunas de las operaciones que soporta el esquema (*inserción y consulta*). Para ello se utilizo *PostgreSQL* como motor de base de datos.

La importancia de la aplicación radica en que muy pocos trabajos realizan una implementación ya que se concentran en el esquema de cifrado. Sin embargo, es interesante ver la funcionalidad práctica del esquema propuesto.

En nuestro caso en particular se utilizó una base de datos muy pequeña pues se encuentra constituída únicamente por tres tablas; pero fue suficiente para demostrar la

utilidad del esquema.

7.2 Trabajo futuro

Las *bases de datos cifradas subcontratadas* implican un gran reto, si se busca que sean una tecnología utilizada a nivel comercial. Para que esto sea posible se deben cubrir dos áreas: la velocidad en las transacciones y mantener la mayor (toda) funcionalidad de la bases de datos.

Con lo que respecta al primer punto, la implementación de la aritmética podría realizarse programando todas las funciones para buscar mayor eficiencia en lugar de utilizar las funciones de gmp que finalmente son genéricas; además que es posible mejorar la implementación de los algoritmos de cifrado, lo que daría como resultado mejores tiempos en las operaciones sobre los registros (*inserción, modificación, consulta y elminación*).

En cuanto a la funcionalidad de la **base de datos**, se debería experimentar la implementación de procesos más complicados como: procedimientos almacenados, desencadenantes (*stored procedures y triggers respectivamente*); para lo que con seguridad surgirían nuevos retos desde la perspectiva del esquema de cifrado.

Desde el punto de vista de la aplicación exiten varias mejoras posibles, que aunque no son de interés científico, mejorarían los aspectos prácticos: experimentar en una arquitectura real cliente-servidor, ya que en este trabajo una misma computadora funge como servidor y cliente. Crear una interfaz para que sea más amigable al usuario, realizando una versión Web; agregar más operaciones que el *traductor de consultas* pueda reconocer.

Finalmente se debe mencionar que actualmente la línea de investigación en técnicas de homomorfismo es muy fuerte, por lo que es posible buscar la adaptación de dichos esquemas al contexto de *bases de datos subcontratadas*, entre estos trabajos se puede mencionar el artículo de Damgard et.al. [34] en donde se presenta un esquema de homomorfismo más eficiente que el de Paillier.

Apéndice A

Conceptos preliminares

En este apéndice se tratan algunos conceptos básicos sobre grupos, que son requeridos para la comprensión de algunas de las secciones del manuscrito. Para mayor profundidad del tema remitimos al lector a [31].

A.1 Grupos

Sea $\mathbb{G}$ un conjunto y o una operación binaria, la cual es una función que toma dos elementos de $\mathbb{G}$. Si $g, h \in \mathbb{G}$ entonces se utiliza la notación $g\ o\ h$. Ahora describiremos que propiedades debe cumplir un grupo.

Definición: Un *grupo* es un conjunto $\mathbb{G}$ con una operación binaria o donde las siguientes condiciones se cumplen:

- *Cerradura:* Para todo $g, h \in \mathbb{G}$, $g\ o\ h \in \mathbb{G}$
- *Existencia de elemento identidad:* Existe un elemento identidad $e \in \mathbb{G}$ tal que para toda $g \in G$, $e\ o\ g = g = g\ o\ e$
- *Existencia de los inversos:* Para todo $g \in \mathbb{G}$ existe un elemento $h \in \mathbb{G}$ tal que $g\ o\ h = e = h\ o\ g$. El elemento h es llamado el inverso de g.
- *Asociatividad:* Para todo $g_1, g_2, g_3 \in \mathbb{G}$, $(g_1\ o\ g_2)\ o\ g_3 = g_1\ o\ (g_2\ o\ g_3)$

Cuando $\mathbb{G}$ tiene un número finito de elementos, se dice que es un **grupo finito**. $|\mathbb{G}|$ denota el orden del grupo, es decir, el número de elementos en $\mathbb{G}$.

Un grupo es **abeliano**, si con la operación o la siguiente propiedad se cumple:

- *Conmutatividad:* Para todo $g, h \in \mathbb{G}$, $g\ o\ h = h\ o\ g$

Es importante mencionar que el elemento identidad es único para el grupo $\mathbb{G}$ y el inverso de un elemento también es único. En general no se utiliza la notación o para denotar la operación de un grupo, ya que estos pueden ser clasificados como: *grupo aditivo* o *grupo multiplicativo*, dependiendo si se trabaja con la operación "$+$" o "$\cdot$" respectivamente.

A.1.1 Grupo $\mathbb{Z}_N^*$

El conjunto $\mathbb{Z}_N = 0 \cdots, N-1$ es un grupo bajo la suma módulo N. ¿Es posible establecer un grupo bajo la múltiplicación módulo N?, para lograr esto algunos elementos deben ser eliminados, precisamente aquellos que no tienen inverso, e.g. el 0 (no tiene inverso).

¿Qué elementos son invertibles módulo N?, son aquellos cuyo $gcd(a, N) = 1$, es decir, el grupo $\mathbb{Z}_N^*$ está compuesto por aquellos elementos que son primos relativos de N.

El orden del grupo está dado por la *función de Euler* denotada por $\phi(N) = \prod_i p_i^{e_i - 1}(p_i - 1)$.

Apéndice B

Algoritmos

En este apartado presentaremos dos algoritmos que fueron utilizados en este trabajo de investigación, el cifrador por bloques AES-128 y el codificador base64.

B.1 AES-128

El *Estándar de Cifrado Avanzado* AES (por sus siglas en inglés), establece el algoritmo de Rijndael, que es un cifrador simétrico que puede procesar bloques de 128 bits, utilizando llaves de 128, 192, y 256 bits.

El nombre Rijndael está basado en los nombres de sus inventores: Joan Daemen y Vicent Rijmen. Este algoritmo es el ganador del concurso realizado en 1997 por el gobierno de los Estados Unidos, después que el algoritmo *Data Encryption Standar (DES)* fuera considerado inseguro debido a que la longitud de la llave era pequeña con respecto a los avances de la tecnología y el poder de computo de la época.

AES es una red de substituciones y permutaciones (*substitution-permutation network*), la cual está constituída por una serie de operaciones matematicas que utiliza substituciones, también llamadas S-Box y permutaciones (P-Boxes).

Debido a la definición cuidadosa de estas operaciones cada bit a la salida depende de cada uno de los bits de entrada. En esta sección únicamente se describe AES-128, que es un algoritmo iterativo con una longitud de bloque fija y longitud de llave variable, que para nuestro caso es de 128 bits.

B.1.1 The State

Internamente, el algoritmo AES realiza las operaciones sobre una matriz de bytes llamada *state*, la cual conciste de 4 renglones y 4 columnas de bytes (véase la tabla B.1), el número de columnas es la longitud del bloque (128) dividido entre 32 y es denotado como **Nb**. De forma similar la llave es una matriz de 4 renglones y el número de columnas se denota como **Nk**, la cual es equivalente a dividir la longitud de la llave entre 32 (véase la tabla B.2).

$S_{0,0}$	$S_{0,1}$	$S_{0,2}$	$S_{0,3}$
$S_{1,0}$	$S_{1,1}$	$S_{1,2}$	$S_{1,3}$
$S_{2,0}$	$S_{2,1}$	$S_{2,2}$	$S_{2,3}$
$S_{3,0}$	$S_{3,1}$	$S_{3,2}$	$S_{3,3}$

Tabla B.1: State

$K_{0,0}$	$K_{0,1}$	$K_{0,2}$	$K_{0,3}$
$K_{1,0}$	$K_{1,1}$	$K_{1,2}$	$K_{1,3}$
$K_{2,0}$	$K_{2,1}$	$K_{2,2}$	$K_{2,3}$
$K_{3,0}$	$K_{3,1}$	$K_{3,2}$	$K_{3,3}$

Tabla B.2: Key

B.1.2 Cifrado

En la ronda inicial se aplica una sola transformación llamada **AddRoundKey**. AES utiliza un número variable de rondas, este número esta basado en la longitud de la llave, para una llave de 128 bits se tienen 10 rondas, para 192 bits se tienen 12 rondas y para 256 bits son 14 rondas.

Durante cada ronda se realizan las siguientes operaciones:

1. SubBytes: cada byte es reemplazado por otro, utilizando la S-Box de Rijndael
2. ShiftRow: cada renglón en el arreglo de $4 * 4$ es intercambiado cierto número de desplazamientos hacia la izquierda
3. MixColumns: es una transformación de columnas
4. AddRoundKey: cada byte es combinado con la llave de ronda, la cual es diferente para cada ronda y es derivada de la llave original

En la última ronda la operación **MixColumns** no se realiza. El algoritmo de cifrado se muestra en la tabla B.3 tomado directamente de [35].

Algorithm Cipher(in[$4 * Nb$], out[$4 * Nb$], w[Nb*(Nr+1)])

1. state[$4, Nb$]
2. state=in
3. **AddRoundkey**(state,w[$0, Nb - 1$])
4. **for** round=1 step 1 to Nr-1
5. **SubBytes**(state)
6. **ShiftRows**(state)
7. **MixColumns**(state)
8. **AddRoundkey**(state,w[$round * Nb, (round + 1) * Nb - 1$])

9. **SubBytes**(state)
10. **ShiftRows**(state)
11. **AddRoundkey**(state,w[$round * Nb, (round + 1) * Nb - 1$])
12. **Return** out=state

Tabla B.3: Algoritmo de cifrado AES

AddRoundkey

En esta operación una de las llaves de ronda es aplicada al **state** utilizando XOR entre los dos componente. Esto se muestra en la figura B.1.

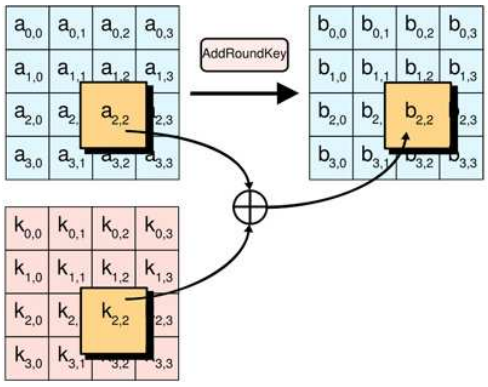

Figura B.1: Operación XOR entre el state y la llave.

SubBytes

Esta operación es una substitución de bytes no lineal, operando cada uno de los bytes del **state** de forma independiente. Esta S-Box es invertible y se muestra en la figura

B.2. Para utilizarla lo que se realiza es lo siguiente: se toma el valor del **state** digamos 53, el primer elemento es el índice de la fila $x = 5$ y la columna es $y = 3$ el elemento es sustituído por ed.

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figura B.2: Tabla de sustitución (S-Box).

ShiftRows

En esta transformación, los bytes de las últimas tres filas del **state** son intercambiadas de forma ciclica por un número de desplazamiento determinado, mientras que la primera fila no es desplazada. Es por ello que: la primera fila no es desplazada, la segunda es desplazada una posición hacia la izquierda, la tercera dos posiciones y finalmente la cuarta en tres posiciones. Tal y como se muestra en la figura B.3.

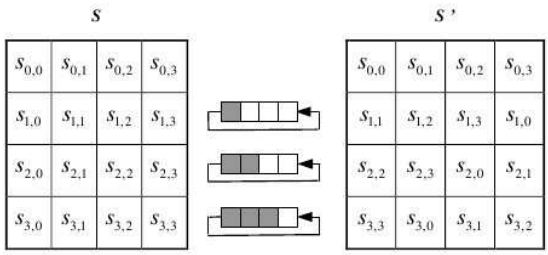

Figura B.3: Transformación de ShiftRows.

MixColumns

Esta transformación opera en la matriz **state** columna a columna y cada una es vista como un polinomio de 4 términos dentro del campo finito $GF(2^8)$. La operación puede ser vista como una multiplicación de matrices o como una multiplicación de polinomios módulo $x^4 + 1$ dentro del campo. La matriz por la que se debe multiplicar

es:

$$\begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \end{bmatrix} \begin{bmatrix} 02 & 03 & 01 & 01 \\ 01 & 02 & 03 & 01 \\ 01 & 01 & 02 & 03 \\ 03 & 01 & 01 & 02 \end{bmatrix}$$

Esta transformación se muestra en la figura B.4.

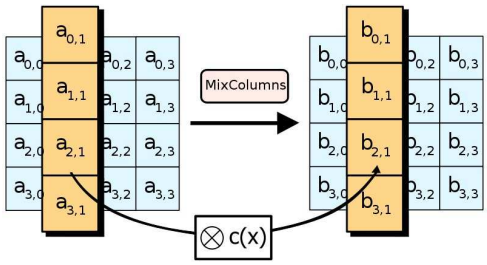

Figura B.4: Transformación de MixColumns.

B.1.3 Descifrado

Las transformaciones utilizadas en el cifrador AES, pueden ser invertidas e implementadas en sentido contrario, para obtener el algoritmo de descifrado. Estas funciones son: **InvShiftRows, InvSubBytes, InvMixColumns y AddRoundKey**.

El algoritmo de descifrado se presenta en la tabla B.4, este recibe como entrada el texto cifrado y se obtiene a la salida el texto en claro.

Algorithm InvCipher(in[$4 * Nb$], out[$4 * Nb$], w[Nb*(Nr+1)])

1. state[$4, Nb$]
2. state=in
3. **AddRoundkey**(state,w[$Nr * Nb, (Nr + 1) * Nb - 1$])
4. **for** round=Nr-1 step -1 to 1
5. **InvSubBytes**(state)
6. **InvShiftRows**(state)
7. **InvMixColumns**(state)
8. **AddRoundkey**(state,w[$round * Nb, (round + 1) * Nb - 1$])

9. **InvSubBytes**(state)
10. **InvShiftRows**(state)
11. **AddRoundkey**(state,w[$0, Nb - 1$])
12. **Return** out=state

Tabla B.4: Algoritmo de descifrado AES

InvShiftRows

InvShiftRows es la función inversa de la transformación **ShiftRows**. Los bytes de las útimas tres filas del **state** son ciclicamente desplazadas por un número específico de desplazamiento. La primera fila queda igual, y las otras tres son desplazadas hacia la derecha tal y como se muestra en la figura B.5.

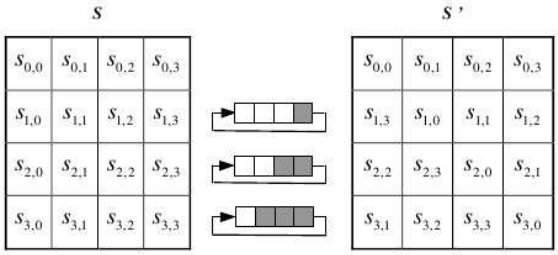

Figura B.5: InvShiftRows.

InvSubBytes

InvSubBytes es la función inversa de la transformación **SubBytes** para ello se utiliza la tabla inversa de la S-Box, la cual se muestra en la figura B.6. Para encontrar el valor que se debe sustituir, se utiliza el mismo procedimiento que en **SubBytes**.

x \ y	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f
0	52	09	6a	d5	30	36	a5	38	bf	40	a3	9e	81	f3	d7	fb
1	7c	e3	39	82	9b	2f	ff	87	34	8e	43	44	c4	de	e9	cb
2	54	7b	94	32	a6	c2	23	3d	ee	4c	95	0b	42	fa	c3	4e
3	08	2e	a1	66	28	d9	24	b2	76	5b	a2	49	6d	8b	d1	25
4	72	f8	f6	64	86	68	98	16	d4	a4	5c	cc	5d	65	b6	92
5	6c	70	48	50	fd	ed	b9	da	5e	15	46	57	a7	8d	9d	84
6	90	d8	ab	00	8c	bc	d3	0a	f7	e4	58	05	b8	b3	45	06
7	d0	2c	1e	8f	ca	3f	0f	02	c1	af	bd	03	01	13	8a	6b
8	3a	91	11	41	4f	67	dc	ea	97	f2	cf	ce	f0	b4	e6	73
9	96	ac	74	22	e7	ad	35	85	e2	f9	37	e8	1c	75	df	6e
a	47	f1	1a	71	1d	29	c5	89	6f	b7	62	0e	aa	18	be	1b
b	fc	56	3e	4b	c6	d2	79	20	9a	db	c0	fe	78	cd	5a	f4
c	1f	dd	a8	33	88	07	c7	31	b1	12	10	59	27	80	ec	5f
d	60	51	7f	a9	19	b5	4a	0d	2d	e5	7a	9f	93	c9	9c	ef
e	a0	e0	3b	4d	ae	2a	f5	b0	c8	eb	bb	3c	83	53	99	61
f	17	2b	04	7e	ba	77	d6	26	e1	69	14	63	55	21	0c	7d

Figura B.6: Tabla de substitución inversa (Inv S-Box).

InvMixColumns

Al igual que con **MixColumns**, aquí también se opera sobre el **state** columna a columna y cada una es vista como un polinomio de 4 términos dentro del campo finito $GF(2^8)$. También la operación puede ser vista como una multiplicación de matrices o como una

multiplicación de polinomios módulo $x^4 + 1$ dentro del campo. La diferencia consiste en la matriz por la que se debe multiplicar:

$$\begin{bmatrix} S_{0,c} \\ S_{1,c} \\ S_{2,c} \end{bmatrix} \begin{bmatrix} 0e & 0b & 0d & 09 \\ 09 & 0e & 0b & 0d \\ 0d & 09 & 0e & 0b \\ 0b & 0d & 09 & 0e \end{bmatrix}$$

AddRoundKey

Esta transformación es su propia inversa ya que está constituída por la operación XOR.

B.2 base64

Base 64 es un sistema de numeración que usa el 64 como base. Es la mayor potencia de dos que puede ser representada usando únicamente los caracteres imprimibles de ASCII. El alfabeto utilizado es mostrado en la tabla B.5.

Este tipo de codificado está diseñado para representar una secuencia arbitraria de octetos de tal forma que permita utilizar letras minúsculas y mayúsculas, sin que sea legible para los seres humanos.

Valor	Código	Valor	Código	Valor	Código	Valor	Código
0	A	16	Q	32	g	48	w
1	B	17	R	33	h	49	x
2	C	18	S	34	i	50	y
3	D	19	T	35	j	51	z
4	E	20	U	36	k	52	0
5	F	21	V	37	l	53	1
6	G	22	W	38	m	54	2
7	H	23	X	39	n	55	3
8	I	24	Y	40	o	56	4
9	J	25	Z	41	p	57	5
10	K	26	a	42	q	58	6
11	L	27	b	43	r	59	7
12	M	28	c	44	s	60	8
13	N	29	d	45	t	61	9
14	O	30	e	46	u	62	+
15	P	31	f	47	v	63	/

Tabla B.5: Alfabeto de codificación para **base64**.

B.2.1 Codificado

El proceso que utiliza **base64** para codificar consiste en tomar grupos de 24 bits (3 grupos de 8 bits) del mensaje original y reacomodarlos como 4 grupos de 6 bits (4 enteros entre 0 y 63), substituyendolos por el símbolo correspondiente del alfabeto. Como ejemplo, podemos presentar la siguiente secuencia:

- Representación Binaria: '01010101', '00100100' y '00010001'
- La secuencia anterior puede ser dividida en cuatro conjuntos de 6 bits: '010101', '010010', '010000' y '010001'

- Estas nuevas secuencias binarias equivalen a: 21, 34, 32 y 33. Por lo tanto, corresponden a los símbolos **VSQR** del alfabeto base64.

Padding

Cuando el texto original no es multiplo de 24 entonces es necesario utilizar un símbolo más "=". Existen dos casos que deben ser considerados, cuando hacen falta dos bytes o bién uno.

En el primer caso cuando hacen falta dos bytes, se debe realizar un *padding* de dos bytes, representados con ceros. Veamos el siguiente ejemplo:

- Representación Binaria: '00000001'
- Se realiza el *padding* con dos bytes llenos de ceros: '00000001', '00000000' y '00000000'
- Ahora se rompe la secuencia en grupos de 6 bits: '000000', '010000', '000000' y '000000'
- Se sustituyen las dos primeras secuencias y se agrega dos '=': 0 y 32. Por lo tanto, corresponden a los símbolos **AQ==** del alfabeto base64.

Finalmente cuando hace falta un solo byte, la cadena binaria se completa con un byte de ceros, y se hace el pad de sólo un '=' al final.

B.2.2 Decodificado

Para la decodificación de **base64** se realiza el proceso inverso. Se lee un conjunto de 4 caracteres del mensaje codificado y se obtienen los correspondientes índices de la tabla. Posteriormente se concatenan para formar la cadena de 24 bits que conformoran el mensaje decodificado. Es importante tomar en cuenta los dos casos de final de cadena, ya que es posible encontrar uno o dos '=' y los bytes correspondientes a estos símbolos deben ser ignorados.

Apéndice C

Lenguaje SQL

PostgreSQL en un sistema de administración de base de datos relacional (RDBMS). Esto significa que la información es almacenada en relaciones (tablas). Cada tabla es una colección de filas. Cada fila tiene el mismo conjunto de columnas, y cada columna tiene un tipo específico de dato. Existen distintos tipos de operaciones y consultas que pueden aplicarse a una o más tablas las cuales trataremos en este apartado.

C.1 Inserción

El enunciado `INSERT` es utilizado para popular una tabla con filas:

```
INSERT INTO <nombre_tabla> VALUES('texto',45,50,'fecha')
```

Cada tipo de dato tiene una forma de entrada. Las constantes son valores alfanuméricos que deben ir entre comillas simples (').En caso de ser un tipo numerico basta con colocar el valor, y los tipos de datos fechas deben ser colocados entre comillas también.

La sintaxis anterior exige el recordar el orden de las columnas, por lo que es posible listas las columnas de forma explicita.

```
INSERT INTO <nombre_tabla> (atributo 1, atributo 3, atributo 5)
VALUES ('valor 1','valor 3',valor 5)
```

C.2 Consultas

Para obtener información de una tabla, se realiza la operación de consulta. Para ello se utiliza el enunciado `SELECT`. Este enunciado contiene varias secciones: las colum-

nas que deben ser retornadas, la lista de tablas de dónde la información es obtenida, restricciones las cuales pueden ser opcionales.

```
SELECT * FROM <nombre_tabla>
```

El operando * es equivalente a solicitar todas las columnas.

También es posible utilizar restricciones numéricas:

```
SELECT * FROM <nombre_tabla> WHERE <atributo_1> = valor
```

```
SELECT * FROM <nombre_tabla> WHERE <atributo_1> < valor
AND <atributo_1> > valor
```

Para este tipo de consultas los operadores que pueden ser utilizado son los que aparecen en la tabla C.1:

Operador	Uso
< >	Menor que, Mayor que
<= >=	Menor igual que, Mayor igual que
LIKE	Se utiliza para buscar en un atributo alfanumérico un patrón.

Tabla C.1: Operadores para consultas de rango

Además es posible utilizar modificadores como se muestra en la siguiente consulta:

```
SELECT * FROM <nombre_tabla> ORDER BY <atributo_1>
```

El modificador `ORDER BY` puede ser sustituido por cualquiera de los que aparecen en la tabla

Modificador	Uso
ORDER BY	Ordena los registros en base al atributo que le sigue.
GROUP BY	Agrupa los registros en base al atributo que le sigue.
HAVING	Selecciona el grupo de registros que cumplen con la condición que le sigue.

Tabla C.2: Modificadores para consultas

Uniones entre tablas

Hasta este momento sólo se ha presentado como consultar una tabla, sin embargo, las consultas pueden acceder a más de una tabla. Para escribir una consulta de este tipo es necesario establecer las columnas que permite relacionar las tablas de las que se esta obteniendo la información, generalmente estos atributos son llaves primarias o extranjeras. La sintaxis se muestra a continuación:

SELECT <atributo_1>, <atributo_2>, <atributo_3>
FROM <tabla_1> JOIN <tabla_2> ON
<atributo_1> = <atributo_2>

Existen otro tipo de enunciados que permiten establecer el comportamiento de la unión y estos se presentan en la tabla C.3.

Tipos de Joins	Uso
INNER JOIN	Para cada fila R1 en T1, se busca las filas que satisfacen la condición contenidas en la tabla T2.
LEFT OUTER JOIN	Primero se ejecuta un INNER JOIN y posteriormente se agregan los registros de T1 que no satisfacen la condición colocando nulos en los atributos de T2.
RIGHT OUTER JOIN	Primero se ejecuta un INNER JOIN y posteriormente se agregan los registros de T2 que no satisfacen la condición colocando nulos en los atributos de T1.
FULL OUTER JOIN	Primero se ejecuta un INNER JOIN y posteriormente ambas filas son agregadas las que no cumplen la condición en T1 y las que no cumplen la condición en T2.

Tabla C.3: Tipos de JOINS

Sub consultas

Las sub consultas implican que una tabla puede ser incluida entre parentesis y debe utilizar un alias. Por ejemplo:

```
FROM (SELECT * FROM <table_1>) AS alias_name
```

Este ejemplo es esquivalente a: `FROM <table_1> AS alias_name`. Casos más complejos resultan cuando existen agregaciones o agrupaciones.

Funciones de agregación

Una función de agregación calcula sólo un valor que resulta a partir de múltiples registros. Por ejemplo, existen funciones de agregación para calcular el número de registros (count), el total, el promedio, el máximo, el minimo, de un conjunto de filas.

```
SELECT max(atributo_1) FROM <tabla_1>
```

Las funciones de agregación más comunes se muestran en la tabla C.4.

Función de Agregación	Uso
COUNT(*)	Cuenta el número de registros.
AVG(expresión)	Obtiene el promedio de los registros, evaluado en la expresión.
MAX(expresión)	Obtiene el máximo de los registros, evaluado en la expresión.
MIN(expresión)	Obtiene el mínimo de los registros, evaluado en la expresión.
SUM(expresión)	Obtiene la suma de todos los registros según la expresión.

Tabla C.4: Funciones de agregación

Además la funciones de agregación pueden ser utilizadas con la clausula `GROUP BY`, es decir, presentar el resultado de la agregación por cada grupo establecido por el `GROUP BY`. Sintaxis:

```
SELECT <atributo_1>, max(<atributo_2>)
FROM tabla_1 JOIN tabla_2 ON atributo_2 = atributo_3
GROUP BY atributo_1
```

C.3 Actualizaciones

La modificación de la información que ya esta previamente registrada en la base de datos se realiza con el comando `UPDATE`. Es posible actualizar un registro, todos los registros de la tabla o un subconjunto de ellos; esto depende de las condiciones establecidas en el `WHERE`. Para ello se usa la siguiente sintaxis:

```
UPDATE <tabla_1>
SET atributo_1 = <valor_nuevo>
WHERE atributo_2 = <valor>
```

C.4 Eliminaciones

Al igual que las actualizaciones la elminación de registros puede ser realizada por grupos de registros, un registro único o toda la tabla; y esto se establece en la cláusula `WHERE`. El comando utilizado es `DELETE` y la sintaxis es:

```
DELETE FROM tabla_1 WHERE atributo_1= <value>
```

C.5 Operadores matemáticos

Dentro del uso de una base de datos es necesario realizar operaciones como suma, resta, multiplicación, etc. En la tabla C.5 se muestran los operadores más usados.

Operador	Descripción	Ejemplo	Resultado
+	Adición	2 + 3	5
-	Substración	2 - 3	-1
*	Multiplicación	2*3	6.
/	División	4/2	2

Tabla C.5: Operadores matemáticos

C.6 Tipos de datos

Dentro de una Base de Datos se utilizan distintos tipos de datos los más usados se describen a continuación.

Numéricos

Entre este tipo de datos se encuentra los enteros y los flotantes, clasificados tal y como se muestra en la tabla C.6

Tipo de dato	Descripción	Tamaño
smallint	Rango pequeño de enteros	2 bytes
integer	El tamanõ màs usado	4 bytes
bigint	Rango largo de enteros	8 bytes.
real	Precisión variable	4 bytes
double precision	Precisión variable	8 bytes

Tabla C.6: Tipos de datos numéricos

Caracter

Este tipo de datos define a los alfanuméricos y están clasificados en la tabla C.7.

Tipo de dato	Descripción
character varyinng(n), varchar(n)	Longitud variable
character(n), char(n)	Longitud fija
text	Longitud variable ilimitada

Tabla C.7: Tipos de dato caracter

Fechas

Este tipo de datos define como se almacenan las fechas que pueden incluir la hora y están clasificados en la tabla C.8.

Tipo de dato	Descripción	Tamaño
timestamp[(p)] [without time zone]	Fecha y tiempo	8 bytes
timestamp[(p)] with time zone	Fecha y tiempo más la zona horaria	8 bytes
date	Solamente fecha	4 bytes

Tabla C.8: Tipos de datos fecha

Apéndice D

Manuales

En este apéndice incluye el manual de instalación, manual del usuario; los cuales facilitaran el uso de la aplicación. En la última sección se incluye información relacionada a las bibliotecas externas que requiere este proyecto para su compilación.

D.1 Manual de instalación

- Como primer paso debe obtener la aplicación que se encuentra en `EsquemaFinal.tar.gz` y desempaquetarla. Para ello dirijase a una terminal (*shell*) y ejecute el comando:

  ```
  tar zxf EsquemaFinal.tar.gz
  ```

- Para la correcta compilación del programa el usuario debe tener instalada la biblioteca: `gmp`, puede referirse a la sección D.3.

- Posteriormente dirijase al directorio `EsquemaFinal` y ejecute el comando: `make`

- Para poder utilizar la aplicación es necesario instalar `PostgreSQL 8.3`, en el mismo directorio se ha incluido el `rpm` para la distribución de `Linux Suse 10.3`, que fue con la que se desarrollo el proyecto. En caso que se desee trabajar con otra distribución de linux se debe hacer la instalación apropiada, para un mayor detalle consultar:

  ```
  http://www.postgresql.org/download/linux
  ```

  El `rpm` incluido en este directorio, incluye la instalación de `PGAdmin III`, se recomienda utilizarlo para facilitar la Administración de la base de datos.

- Con la instalación de PostgreSQL se instala la biblioteca `libpq`, esta biblioteca es requerida para la comunicación entre la aplicación y la base de datos (ver D.3).

- Para hacer uso de los ejemplos de consultas que incluye esta aplicación es necesario hacer un `restore` de la base datos, llamada `mydb`, el archivo que permite restaurarla se encuentra en: `EsquemaFinal/backup`.

- Para ejecutar la aplicación ingrese el comando desde consola:

  `./scheme`

D.2 Manual del usuario

- Para ejecutar la aplicación vaya a la consola y ejecute: `./scheme`

- Al usuario se le presenta un menu con tres opciones: *Introduce a register, Make a query, Quit*.

- En caso que el usuario ingrese un nuevo registro, se le preguntará por el nombre de la tabla a la que desea introducir el nuevo registro. Posteriormente debe ingresar los valores de los atributos separados por el operador *punto*.

  E.g. Para introducir un nuevo registro en la tabla **Empleados** cuyos atributos son: *el Identificador del Empleado, el Nombre del Empleado, el Salario del Empleado, Comisión, Dirección y el Identificador del departamento para el cual trabaja*, el usuario debe escribir en la consola una línea similar a la que sigue:

  `860.Mary.20.2.LaRioja.Comp`

- Para realizar una consulta el usuario deberá proporcionar el archivo escrito en texto plano (`.dat`), en el que se encuentra su consulta, respetando la siguiente sintaxis:

  - Cada sección de la consulta `SELECT, FROM y WHERE` deben ser separadas por un pipe.
  - El último pipe delimita las condiciones del `WHERE`, y cada una de ellas es separada por un *.
  - Los operadores <> son separados de los operandos por un el símbolo (_).

- Para salir del programa se utiliza la última opción: *Quit*.

Ejemplos incluidos

- Los archivos: `q1.dat`,`q2.dat` muestran la información completa de las tablas: **Ciudades, Departamentos**.

- El archivo: `qnowhere.dat` muestra la información de la tabla: **Empleados**, seleccionando explicitamente algunos de sus campos.
- `qigual.dat, qrange.dat, qrange2.dat` estos archivos contienen consultas que evalúan rangos.
- Los archivos: `qagg.dat,qagg2.dat` contiene consulta de agregación `SUM` con distintas restricciones en el `WHERE`.

D.3 Bibliotecas

En esta sección damos una breve descripción de dos bibliotecas que fueron utilizadas para la implementación del esquema propuesto, **gmp** que permite manejar las variables de multiprecisión además de facilitar la aritmética y **libpq** que permite la interfaz entre C y PostgreSQL; más información puede ser encontrada en [36] y [37], respectivamente.

D.3.1 gmp

GNU MP es una bibliteca portable escrita en C, diseñada para manejar precisión arbitraria en aritmética entera, números racionales y números de punto flotante; particularmente en este proyecto se utilizo número enteros. Esta biblioteca permite tener operaciones aritméticas a gran velocidad para aplicaciones que requieren mayor precisión que la soportada por los tipos básicos de C. Además, tiene un buen desempeño tanto para aplicaciones que requieren pocos bits de precisión como para aquellas que requieren gran cantidad.

D.3.2 libpq

libpq es una aplicación en C que provee a los programadores de una interfaz con PostgreSQL, está biblioteca contiene un conjunto de funciones que permite enviar consultas al servidor de PostgreSQL y recibir los resultados de estas consultas. Además esta biblioteca permite trabajar con C++,Perl,Python, entre otros.

Bibliografía

[1] Dawn Xiaodong Song, David Wagner, and Adrian Perrig. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy*, pages 44–55, 2000.

[2] Hakan Hacigümüs, Balakrishna R. Iyer, Chen Li, and Sharad Mehrotra. Executing sql over encrypted data in the database-service-provider model. In Michael J. Franklin, Bongki Moon, and Anastassia Ailamaki, editors, *SIGMOD Conference*, pages 216–227. ACM, 2002.

[3] Zhiqiang Yang, Sheng Zhong, and Rebecca N. Wright. Privacy-preserving queries on encrypted data. In Dieter Gollmann, Jan Meier, and Andrei Sabelfeld, editors, *ESORICS*, volume 4189 of *Lecture Notes in Computer Science*, pages 479–495. Springer, 2006.

[4] Douglas R. Stinson. *Cryptography Theory and Practice*. CRC Press,Inc, second edition.

[5] Gertz Michael and Jajodia Sushil. *Handbook of Database Security*. Springer, first edition. US, 2008.

[6] Hakan Hacigümüs, Balakrishna R. Iyer, and Sharad Mehrotra. Efficient execution of aggregation queries over encrypted relational databases. In *DASFAA*, pages 125–136, 2004.

[7] Josep Domingo-Ferrer. A new privacy homomorphism and applications. *Inf. Process. Lett.*, 60(5):277–282, 1996.

[8] Ronald L. Rivest, Len Adleman, and Michael L. Dertouzos. On data banks and privacy homomorphisms. In *Foundations of secure computation (Workshop, Georgia Inst. Tech., Atlanta, Ga., 1977)*, pages 169–179. Academic, New York, 1978.

[9] Rakesh Agrawal, Jerry Kiernan, Ramakrishnan Srikant, and Yirong Xu. Order-preserving encryption for numeric data. In Gerhard Weikum, Arnd Christian König, and Stefan Deßloch, editors, *SIGMOD Conference*, pages 563–574. ACM, 2004.

[10] Y.Lee A.Boldyreva, N. Chenette and A. O'Neill. Order-preserving symmetric encryption. In *Advances in Cryptology- Eurocrypt 2009 Proceedings*, 2009.

[11] Nabil R. Adam and John C. Wortmann. Security-control methods for statistical databases: A comparative study. *ACM Comput. Surv.*, 21(4):515–556, 1989.

[12] L. Willenborg and T. De Waal. *Elements of Statistical Disclosure Control*. Springer, 2001.

[13] L. Willenborg and T. De Waal. *Statistical Disclosure Control in Practice*. SpringerVerlag, 1996.

[14] Hakan Hacigümüs, Balakrishna R. Iyer, and Sharad Mehrotra. Query optimization in encrypted database systems. In *DASFAA*, pages 43–55, 2005.

[15] Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *EUROCRYPT*, pages 223–238, 1999.

[16] Ravi Chandra Jammalamadaka, Sharad Mehrotra, and Nalini Venkatasubramanian. Pvault: a client server system providing mobile access to personal data. In Vijay Atluri, Pierangela Samarati, William Yurcik, Larry Brumbaugh, and Yuanyuan Zhou, editors, *StorageSS*, pages 123–129. ACM, 2005.

[17] Steve Barker and Gail-Joon Ahn, editors. *Data and Applications Security XXI, 21st Annual IFIP WG 11.3 Working Conference on Data and Applications Security, Redondo Beach, CA, USA, July 8-11, 2007, Proceedings*, volume 4602 of *Lecture Notes in Computer Science*. Springer, 2007.

[18] Ravi Chandra Jammalamadaka and Sharad Mehrotra. Querying encrypted xml documents. In *IDEAS*, pages 129–136. IEEE Computer Society, 2006.

[19] Hui Wang and Laks V. S. Lakshmanan. Efficient secure query evaluation over encrypted xml databases. In Umeshwar Dayal, Kyu-Young Whang, David B. Lomet, Gustavo Alonso, Guy M. Lohman, Martin L. Kersten, Sang Kyun Cha, and Young-Kuk Kim, editors, *VLDB*, pages 127–138. ACM, 2006.

[20] Josep Domingo Ferrer. Privacy homomorphisms for statistical confidentiality. *Quaderns d'Estadística i Investigació Operativa. Segona Època (Qüestiió)*, 20(3):505–521, 1996.

[21] Ernest F. Brickell and Yacov Yacobi. On privacy homomorphisms (extended abstract). In *EUROCRYPT*, pages 117–125, 1987.

[22] Dan Boneh, Giovanni Di Crescenzo, Rafail Ostrovsky, and Giuseppe Persiano. Public key encryption with keyword search. In Christian Cachin and Jan Camenisch, editors, *EUROCRYPT*, volume 3027 of *Lecture Notes in Computer Science*, pages 506–522. Springer, 2004.

[23] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. In *NDSS*. The Internet Society, 2004.

[24] Philippe Golle, Jessica Staddon, and Brent R. Waters. Secure conjunctive keyword search over encrypted data. In Markus Jakobsson, Moti Yung, and Jianying Zhou, editors, *ACNS*, volume 3089 of *Lecture Notes in Computer Science*, pages 31–45. Springer, 2004.

[25] Joonsang Baek, Reihaneh Safavi-Naini, and Willy Susilo. Public key encryption with keyword search revisited. In Osvaldo Gervasi, Beniamino Murgante, Antonio Laganà, David Taniar, Youngsong Mun, and Marina L. Gavrilova, editors, *ICCSA (1)*, volume 5072 of *Lecture Notes in Computer Science*, pages 1249–1259. Springer, 2008.

[26] Kun Peng, Riza Aditya, Colin Boyd, Ed Dawson, and Byoungcheon Lee. Multiplicative homomorphic e-voting. In Anne Canteaut and Kapalee Viswanathan, editors, *INDOCRYPT*, volume 3348 of *Lecture Notes in Computer Science*, pages 61–72. Springer, 2004.

[27] Ron Rivest. Voting, homomorphic encryption. Technical report, Computer and Network Security, 2002.

[28] Yuval Ishai, Eyal Kushilevitz, and Rafail Ostrovsky. Sufficient conditions for collision-resistant hashing. In Joe Kilian, editor, *TCC*, volume 3378 of *Lecture Notes in Computer Science*, pages 445–456. Springer, 2005.

[29] Rafail Ostrovsky and William E. Skeith III. A survey of single-database private information retrieval: Techniques and applications. In Tatsuaki Okamoto and Xiaoyun Wang, editors, *Public Key Cryptography*, volume 4450 of *Lecture Notes in Computer Science*, pages 393–411. Springer, 2007.

[30] M. Akinwande. Advances in homomorphic cryptosystems. *Journal of Universal Computer Science*, 15(3):506–522, 2009.

[31] Jonathan Katz y Yehuda Lindell. *Introduction to Modern Cryptography*. Chapman & Hall/CRC, first edition, Agosto 2007.

[32] S.Sudarshan Avi Silberschatz, Henry F. Korth. *Database System Concepts*. McGraw-Hill, fifth edition.

[33] PostgreSQL. Documentation, 2009. `http://www.postgresql.org/docs/`.

[34] Ivan Damgård, Martin Geisler, and Mikkel Krøigaard. Homomorphic encryption and secure comparison. *IJACT*, 1(1):22–31, 2008.

[35] National Institute of Standards and Technology. Specification for the advanced encryption standard (aes). Publication 197, Federal Information Processing standards (FIPS), 2001.

[36] GNU Proyect. Gmp. http://gmplib.org/.

[37] PostgreSQL. libpq c library. http://www.postgresql.org/docs/8.1/static/libpq.html#LIBPQ-CONNECT.