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

Consultas Eficientes y Seguras a Bases de Datos Utilizando Fragmentación y Cifrado de Datos

Tesis que presenta:

Jazmín Beatrz Trejo Gatica

Para obtener el grado de

Maestro en Ciencias

en Computación

Director de la Tesis:
Dr. Debrup Chakraborty

México, D. F. Noviembre 2009

Agradecimientos

A mis padres por su apoyo incondicional y por ser mi motor a seguir. Porque juntos hemos logrado cada una de mis metas.

A mis hermanos, Hugo y Deyanira por confiar ciégamente en mí.

A Edgar, por su compañía, su sinceridad, su amistad, paciencia y preocupación. Por su apoyo en cada momento dificil y por compartir grandes cosas juntos.

A mis sobrinos Andrea y Rodrigo, por verme como ejemplo a seguir y por motivarme cada día más a ser una mejor persona.

Al Dr. Debrup Chakraborty, por su gran paciencia y preocupación a lo largo de mi trabajo. Por ser una persona admirable y sabia, que siempre me regaló el tiempo necesario sin escatimar en él.

Al Dr. Luis Gerardo de la Fraga, por su motivación y por su buena actitud ante cualquier duda, por tomarse el tiempo de auxiliarme y ayudarme con su gran conocimiento.

Al Dr. Guillermo Morales por su gran paciencia y por el tiempo que le dedico a esta tesis en la revisión y corrección.

Al CINVESTAV por permitirme formar parte de esta reconocida institución.

A CONACyT por el apoyo, brindado a lo largo de mi estancia en el cinvestav.

Al proyecto de investigación SEP-CONACyT titulado “VISION POR COMPUTADORA EVOLUTIVA” (Ref. 80965), cuyo responsable es el Dr. Luis Gerardo de la Fraga.

Resumen

Mantener la privacidad en la información ha cobrado mayor importancia en las últimas décadas. Con el surgimiento de Internet y la centralización de los datos, es cada vez más necesario dotar de seguridad a nuestra información para protegerla contra robos, plagios, extorsiones, etc. Gracias a este gran avance tecnológico en la comunicación global nació DaaS (*Database as a Service*), una arquitectura que ofrece el servicio de *hosting* de bases de datos brindando la comodidad de administrar las bases de datos y permitir consultas desde cualquier terminal que pueda conectarse a Internet. A pesar de que estas empresas que prestan el servicio de *hosting* de datos ofrecen cierto nivel de seguridad para que solamente el poseedor de la misma pueda consultarla, el prestatario del servicio tiene acceso total a nuestra información. Para cubrir este aspecto se ha hecho uso de las bondades de seguridad que la Criptografía ofrece. Han surgido diversos esquemas que cifran por completo la base de datos con el fin de que el prestatario del servicio no pueda ser capaz de interpretar la información; lo malo es que para realizar consultas a la base de datos cifrada el tiempo de espera del cliente es bastante significativo ya que para realizar la consulta se deben realizar diversas funciones de descifrado y así obtener la información en claro.

Es por esta razón que nos sentimos motivados a diseñar un modelo que permita satisfacer consultas eficientes y seguras en una base de datos bajo el contexto DaaS. Este trabajo de tesis presenta un esquema que en vez de ci-

frar por completo la base de datos, fragmenta la información y gracias a esto, gran parte de la información puede permanecer en texto claro. Fragmentar la base de datos implica separar aquellos atributos que juntos nos denotan información valiosa, como puede ser el nombre de un empleado y su sueldo, y cifrar la relación existente entre estos datos. Así, fragmentar los datos trae grandes beneficios tanto en tópicos de seguridad como en eficiencia; debido a que las consultas se realizan a los datos en texto claro, éstas son realizadas en tiempo real y generadas por el motor de búsqueda del Sistema Manejador de Bases de Datos. Además, nuestro esquema no sólo se preocupa por la seguridad de la información, sino también de la autenticidad de la misma; si un dato no ha sido autenticado, no podemos saber si éste ha sido modificado de forma intencional o accidental. Es por este motivo que los datos cifrados en nuestro esquema son autenticados mediante un cifrado con autenticación.

Por otra parte, y ya que la fragmentación de la base de datos es un problema NP-difícil debido a que corresponde al mismo problema de coloración de hipergrafos, se diseñó e implementó una heurística que permitiera resolver este problema y minimizar el número de fragmentos. Se diseñó un algoritmo genético que permitiera realizar la fragmentación y se comparó directamente contra otra heurística mostrada en la literatura; los resultados obtenidos fueron bastante satisfactorios, nuestro algoritmo genético obtuvo significativamente menos fragmentos de los obtenidos con la otra heurística.

Abstract

During the last decade privacy of information has gained a lot of attention. With the rapid expansion of the internet and centralized data storage mechanisms it has become very important to protect the stored information against non-authorized access and processing. Rapid technological development has brought many new data storage solutions, one of them is Database as a Service (Daas). DaaS is a specific architecture providing database hosting services. In the DaaS model, the owner of the data delegates the duty of maintaining his/her data to a third party (called the server) which hosts the database and process his/her queries. These kind of service providers which provides the service of data hosting may add some security mechanism on the stored data, but the service provider has full access to it. It is undesirable as the owner may not want to reveal the data to the server. Numerous cryptographic techniques can be used by the owner to protect the data. But such an encryption applied to the whole data brings in a new problem: the server needs to process queries on the encrypted data. Last said would be very inefficient, as a good encryption scheme would destroy all the structure which exists in the original data, hence indexing and other techniques generally applied for efficient query processing can no longer be applied. Thus specialized cryptographic schemes are required.

In this thesis we deal with the problem of secure storage in the DaaS model which is friendly to efficient query processing. We present a scheme

in which instead of encrypting the total database, we fragment the information and encrypt only parts of it. By using this strategy the majority of the information stays un-encrypted, and thusquery processing can also be done efficiently. Fragmenting a database implies to separate those attributes which have a sensitive association between them. For example, in an employees database the individual attributes employee name and salary are not themselves sensitive, but the association between them is not revealed.

Fragmentation of a database provides some security to the data without hampering the ease of query processing, and they can be done by the database management system without much extra overhead. Our scheme also adds an additional security service of authentication along with encryption. Authentication allows detection of accidental or intentional changes made within a database. We obtain authentication by using a primitive called authenticated encryption.

The other part which the thesis deals with is the problem of creating fragments. The problem of creating minimum fragments is the same as the minimum hyper-graph coloring problem which is known to be NP hard. We developed a heuristic based on genetic algorithm to create the minimum number of fragments given a database. Our scheme gives significantly lesser number of fragments compared to other heuristics reported in the literature.

Índice general

Índice de figuras

Introducción

Gracias al mundo globalizado en el cual nuestro quehacer diario se desarrolla y a la necesidad de integrarse e interactuar en una sociedad cada vez más compleja, cada ente social[1] requiere de mecanismos para lograr pertenecer activamente en el aspecto laboral, económico y social de su contexto; y así satisfacer sus necesidades, según Maslow [31] y su pirámide, desde las más básicas hasta las de autorealización. Estos mecanismos que son utilizados por las entidades sociales para visualizar su cometido de plenitud, estriban su eficiencia en el aprovechamiento de las oportunidades presentadas; mismas que son extraídas del entorno e identificadas mediante información pertinente que, a su vez, genera cierto conocimiento en el ente para el mejor aprovechamiento de posteriores oportunidades.

El conocimiento de cada ente para sobrevivir en su entorno, depende de la información obtenida del ambiente y de los datos que haya almacenado a lo largo de su existencia; es decir, cada entidad social depende fírmemente de su propia información para lograr satisfacer en mayor medida sus necesidades. Como ya se había mencionado anteriormente, un ente social puede ser desde

[1]Denominamos ente social a cualquier individuo u organización

un individuo hasta una gran compañía y, por que no, una nación; pero, para acotar nuestro enfoque, hablemos de las empresas.

La competencia entre las organizaciones por el dominio de los mercados es sumamente alta; es por ello que las empresas buscan estar bien posicionadas de manera comercial y económica, tomando todo tipo de estrategias y cuidando en lo mejor posible sus recursos. De una manera estricta, podemos tomar al conjunto de estrategias y recursos de toda organización como conocimiento de la misma; es decir, información propia que le permite avanzar frente al resto de las compañías.

Hoy en día, la información se ha convertido en el activo más importante de toda organización, es por ello que se busca que esta gran cantidad de datos se puedan interpretar de manera rápida y sencilla. Debido a la gran revolución informática que se ha presentado en las últimas décadas, la mejor manera de tener información clara, rápida y oportuna se presenta mediante el uso de las bases de datos y explotando el rendimiento de las computadoras actuales.

1.1 Bases de datos relacionales

Cualquier compañía aún la mas pequeña, cuenta con información propia como su nómina, sus estados financieros, sus estudios de mercados, etc; y la gran mayoría, si no es que todas ellas, hacen uso de las bases de datos relacionales debido a su necesidad de abstraer aspectos importantes de la organización; esto, porque las bases de datos relacionales permiten modelar problemas reales y a su vez administrar los datos de forma dinámica.

Así, las compañias al hacer uso de las bases de datos relacionales pueden mantener los datos de manera ordenada y automatizada, facilitando la explotación y lectura de los mismos para una mejor toma de decisiones en cualquier nivel jerárquico de la organización.

A pesar de que el ordenamiento y fácil lectura de los datos son aspectos de vital importancia para las organizaciones, es de igual o mayor relevancia la seguridad de los mismos ante cualquier tipo de riesgo.

La preciada información de las organizaciones en manos incorrectas puede causar plagios en el mejor de los casos y robos, extorsiones o fraudes en el peor de ellos. Sólo por mencionar un ejemplo, tenemos el suceso ocurrido en junio de 2005 donde los nombres y los números de tarjeta de más de 40 millones de tarjetahabientes de Master Card fueron revelados [12]. Este tipo de incidentes genera altas pérdidas monetarias para las organizaciones y un enorme daño a la sociedad. Por esta situación, cada vez es más necesario contar con medidas preventivas para la seguridad de los datos.

Muy de la mano con lo anterior, mencionemos la importancia de Internet, la red de área global, que desde su expansión mundial en 1995 ha resultado en una gran herramienta para mantener al mundo comunicado, ya que la transferencia de archivos e información puede viajar a lo largo del mundo de una manera rápida y económica. Con ello, la tendencia mundial de las organizaciones ha resultado en, primeramente hacer uso de esta red global y así expander sus mercados; y por otro lado, centralizar sus bancos de información mediante el uso de la computación nube, en inglés *cloud computing*[2], que permite a las compañías deslindarse de la administración de sus grandes bancos de información; a esta bondad también se le denonima modelo *outsourcing* de bases de datos.

El modelo *outsourcing* de bases de datos[3] ha tenido gran crecimiento en los últimos años ya que a las empresas les resulta una mejor opción contratar el servicio de administración de sus datos a alguna otra organización, en vez de realizar el trabajo ellas mismas; esto porque contratar el servicio es más barato y más sencillo que realizarlo con recursos propios. La manera en que el modelo funciona es la siguiente: la empresa contrata el servicio de *hosting* y entrega sus bancos de información al prestatario del servicio, el cual los almacena en su propio servidor; el prestatario se responsabiliza de administrar los datos y mante- nerlos accesibles de consultas solamente a la empresa a la que pertenecen.

Como ya se mencionó anteriormente, esta forma de mantener los datos

[2] La computación nube es una tecnología que permite ofrecer servicios computacionales a través de Internet, entre los cuales se presenta el servicio de *hosting* de datos.

[3] ODB, por sus siglas en inglés *Outsource Database Model*.

centralizados en un servidor remoto resulta benéfica para la compañía, ya que ésta solo se preocupa de consultar sus datos confiando en que no son accesibles a ninguna otra persona ajena; no obstante, los datos no son completamente seguros, al ser vulnerables al prestador del servicio.

Debido a lo anterior, ha surgido la necesidad de inyectarle seguridad a nuestros datos antes de contratar el servicio de *hosting*; pero surge la incógnita de *¿cómo realizarlo?*.

1.2 Lograr la seguridad

Considerando que las bases de datos están bajo el poder de un tercero, se requiere poner énfasis en la seguridad de los mismos para evitar que la información sea interferida. Para cubrir esta necesidad, se ha recurrido al uso de la criptografía como medio para mantener segura e ilegible la información; generalmente se acude a esta área debido a que las funciones criptográficas presentan bajos costos en una arquitectura computacional.

Han surgido diversos esquemas que utilizan la criptografía para asegurar los datos remotos; el principio es cifrar por completo la base de datos y posteriormente alojarla remotamente. Esta idea es súmamente segura, ya que si no se cuenta con las llaves de cifrado, los datos son ilegibles y por tal no pueden ser interpretados en el servidor. El gran aspecto negativo con el que cuentan estos esquemas, es que para poder realizar alguna consulta a estos datos es necesario generar diversas funciones de transformación de los mismos en el servidor y, posteriormente verificar en el cliente sí este dato satisface o no la condición de la consulta; todas estas funciones de descifrado y comparación resultan costosas en los tiempos de respuesta de la consulta, esto sin considerar que una consulta puede arrojar miles y miles de registros. Es por esta razón que el exceder el uso de cifrado de la información resulta ineficiente como una solución viable.

Por otra parte, y retomando la definición de información[4], otra posible solución es utilizar la fragmentación de los datos, en vez del cifrado total

[4]Definimos información a un conjunto de datos que se interelacionan y denotan cierto conocimiento.

de la base de datos. El principio del esquema de fragmentación, es que no todos los datos son igual de susceptibles en cualquier base de datos; es decir, cifrar únicamente aquellos datos que son por sí solos sensibles y dejar en texto claro o sin cifrar el resto pero de forma aislada para que juntos no denoten información alguna; así, los datos pueden permanecer en texto claro pero las relaciones entre ellas deben ser secretas. La manera en que se hacen secretas las relaciones de los datos es separándolos en diferentes tablas que llamaremos fragmentos. Así, las consultas a la base de datos fragmentada pueden realizarse de manera ordinaria, haciendo uso del motor de búsqueda del manejador de bases de datos (software mediador que permite hacer uso e interacciones con la base de datos); a diferencia de aquellas consultas generadas para bases de datos que se cifran por completo y cuyo proceso de búsqueda requiere de mecanismos adicionales que hacen que las búsquedas no sean realizadas por este software mediador haciendo que éstas no sean eficientes. Además, gracias a que los datos son en texto claro, se pueden realizar todo tipo de consultas a la base de datos fragmentada; aspecto que siempre queda limitado en las bases de datos totalmente cifradas.

Cabe mencionar que la obtención de la fragmentación de los datos es un problema complejo computacionalmente hablando. Para ello se requiere de heurísticas que resuelvan este problema buscando que el número de fragmentos obtenidos de la base de datos sea siempre el menor posible. El algoritmo que se ha utilizado anteriormente para resolver este problema es el denominado Fragmentaciones Mínimas.

A pesar de que este esquema de fragmentaciones combina los beneficios de la criptografía para la seguridad de los datos sin dejar de lado la eficiencia de las consultas; no considera el hecho de que un dato cifrado pueda ser modificado desde el servidor por el prestatario, es decir, únicamente se enfoca a la seguridad sin preocuparse de la autenticidad [5] de los datos.

Por otra parte, y un problema completamente independiente del anterior, el algoritmo de Fragmentaciones Mínimas no es del todo eficiente; lo anterior, debido a que el número de fragmentos obtenidos es muy alto con respecto al número de atributos en la base de datos.

[5] Autenticidad es verificar que los datos no hayan sido modificados

1.3 Propuesta y contribuciones

Por todo lo anterior, nos sentimos motivados para realizar un trabajo de investigación que presenta aportaciones sustanciales en dos diferentes áreas de oportunidad. La primera referente a las consultas eficientes y seguras en una base de datos remota, y la segunda en el problema la generación de fragmentos de la base de datos.

Con respecto a las consultas eficientes y seguras, se concluyó que es necesario consolidar un esquema lo suficientemente robusto que satisfaga diversos aspectos con el mismo nivel de importancia; este esquema debe mantener los datos completamente seguros sin abandonar el aspecto de eficiencia tanto en los tiempos de respuesta de la consulta, como en la diversidad de consultas a las que éste debe responder, considerando que los datos deben ser autenticados. Como bien sabemos, las consultas a una base de datos alojada en un servidor remoto se efectúan mediante la conexión a Internet, así que es necesario que el tiempo de transacción (envio y obtención de datos) sea siempre el menor posible, para que no existan pérdidas de conexión y fallas en las transacciones; es por eso, que se busca que el tiempo de respuesta en una consulta sea el menor.

Hablando acerca del problema de la fragmentación de la base de datos, observamos que éste presenta ciertas deficiencias reflejadas principalmente en el número de fragmentos obtenidos. Así, se decidió presentar a este problema como uno de optimización el cual debía ser resuelto con alguna heurística esperando obtener mejores resultados que el algoritmo Fragmentaciones Mínimas.

Con respecto a las contribuciones obtenidas, se dividen en dos vertientes que denominaremos las de cifrado y las de fragmentación.

Las contribuciones obtenidas en el cifrado tenemos algunas; se logró un esquema de fragmentación para consultas seguras y eficientes con autenticación de datos que presenta diversas bondades: i) realiza las consultas en tiempo real, ii) permite generar todo tipo de consultas, iii) mantiene los datos seguros mediante el uso de separación de fragmentos y iii) existe coherencia del cifrado con las operaciones de consulta y autenticación.

En el área de fragmentación, resolvimos el problema de optimización, más explícitamente uno de minimización, proponiendo un algoritmo genético que resolviera minimizar el número de fragmentos. Las aportaciones del genético son significativas ya que obtiene menos fragmentos que el algoritmo Fragmentaciones Mínimas reportado en [7]. Además que el proceso de fragmentación es alternativo a procedimientos de diseño atendiendo a formas normales y permite la consistencia entre operaciones del sistema manejador de bases de datos, del cifrado y de la autenticación.

1.4 Estructura de la Tesis

El capítulo 2, llamado Preeliminares, está dedicado a dar un bosquejo de los aspectos técnicos que son tocados en la presente tesis, facilitando el entendimiento técnico durante la lectura de este trabajo en diversas áreas computacionales como: Bases de Datos, Criptografía, esquemas simétricos de cifrado y finalizando con el Cómputo Evolutivo. En el capítulo 3 se presenta el Estado del Arte de nuestra investigación, presentando un gran compendio de trabajos realizados a lo largo de las décadas que se han desarrollado para atacar el mismo problema que a nosotros nos compete; estas investigaciones fueron elegidas de acuerdo a la diversidad de soluciones al problema y a su aportación a nuestra línea de investigación. El capítulo 4 corresponde a nuestra propuesta de fragmentación de bases de datos; en él es definido claramente el esquema, la funcionalidad y los beneficios. El capítulo 5 referente al algoritmo genético, presenta nuestra heurística comparándola directamente contra otro algoritmo ya propuesto, mostrando gráficamente el desempeño, eficiencia y efectividad de ambos. En el capitulo sexto se muestran algunos diagramas de la aplicación realizada. Y finalmente, el capítulo 7 presenta nuestras conclusiones del trabajo realizado y las posibles áreas de oportunidad que esta investigación pudiera generar como trabajo a fututo.

Preliminares

Para entender mejor lo expresado a lo largo de este trabajo de investigación, es necesario dar un recorrido por los diversos tópicos que en esta tesis se tocan; haciendo mención de aquellos que sólo son cruciales de citar, y enfatizando los que es necesario profundizar debido a la afinidad que tienen con el trabajo presentado.

En este capítulo se explican a detalle las técnicas, algoritmos y herramientas que fueron fundamentalmente la base de nuestra investigación; cruzando por diveras áreas computacionales como Bases de Datos, Criptografía y el Cómputo Evolutivo.

Hay interrelación entre las Bases de Datos y la Criptografía para atacar el problema del cifrado. El cómputo evolutivo es méramente utilizado para fragmentación.

De esta manera, nuestra tesis está enfocada a diversos lectores; no sólo a aquellos expertos de las áreas inmersas en esta investigación, sino también a aquellos que tengan el interés simple en este trabajo.

2.1 Modelo relacional de bases de datos y procesamiento de consultas

Una base de datos es conjunto de tablas. Una tabla es un conjunto de filas y columnas; donde las columnas, que también son conocidas como atributos o campos, denotan el nombre del dato a almacenar e.g. en la Fig. 2.1 la columna *Edad*; y las filas, también denominadas registros, son el conjunto de los valores que toman cada uno de los atributos de una tabla e.g. en el atributo *Edad* el valor *27*.

El modelo relacional de bases de datos, inventado por Edgar Codd en 1970 [8], ha sido de gran utilidad en los grandes bancos de información a procesar. Una base de datos relacional es una base de datos convencional, pero con ciertas especificaciones; primeramente, consta de un conjunto de tablas de las cuales dos de ellas no pueden tener el mismo nombre. Las tablas de igual manera que en una base de datos ordinaria están compuestas de filas y columnas; las filas o registros representan a un objeto del mundo real, cada registro contiene tantos valores como el número de atributos existentes en la tabla, i.e. un valor para cada atributo, y además cada registro es diferente del resto debido al uso de llaves primarias[1]; las columnas en una base de datos relacional deben de tener nombres diferentes en una misma tabla.

Hoy en día las bases de datos son manipuladas mediante el uso de computadoras, el rendimiento de las computadoras es explotado al máximo para lograr que el acceso a los datos sea lo más rápido posible; esto no puede realizarse si no se tiene un software que pueda hacer las funciones de un intérprete entre la máquina y el usuario. Un sistema manejador de bases de datos (DBMS por sus siglas en inglés *Data Base Management System*) permite principalmente administrar los datos que se encuentran en el manejador de archivos; el DBMS además cuenta con un motor de búsqueda que comúnmente indexa los datos mediante estructuras de árbol-B[2] para

[1]Una atributo que funge como llave primaria, es un identificador único en la tabla que permite que no haya duplicidad de los datos, i.e. no pueden existir dos registros con el mismo valor de llave primaria.

[2]Un árbol-B [2] es una estructura que se utiliza en bases de datos para mantener los datos ordenados y que las inserciones y eliminaciones de datos se realicen en tiempo logarítmico.

que cualquier consulta en los datos sea rápida y sencilla. Por otra parte, los DBMS funcionan bajo la arquitectura de cliente-servidor[3]; donde el DBMS funge como servidor y la aplicación del usuario es el cliente. Un ejemplo de sistema de gestión de bases de datos es PostgreSQL, que genera consultas bajo el estándar del lenguaje SQL[4] (por sus siglas en inglés *Structured Query Languaje*, o bien Lenguaje Estructurado de Consultas) que permite realizar gran número de operaciones en las bases de datos, tales como consultas, agregaciones, eliminaciones, inserciones, etc.

Cualquier instrucción en SQL para la manipulación de los datos consta de varias partes; primeramente de las palabras reservadas[5] que habitualmente son presentadas en letras mayúsculas, y para el nombre de la tabla y atributos que se recomienda sean presentados en letras minúsculas para diferenciarlas de las palabras reservadas. Para realizar una consulta de datos en SQL, tenemos un gran número de opciones que recaban diversos tipos de información. En la Figura 2.1 se muestra una tabla llamada `tabla`, formada por cuatro atributos (`nombre`, `edad`, `sexo` y `salario`) y cuatro registros. Un conjunto de consultas sencillas se presenta en la Fig. 2.2; que como podemos observar, en el primer caso se pide un único atributo (`edad`), en la segunda consulta se esperan dos atributos y en la última consulta el `*` indica que deseamos todos los atributos pero con una doble condición, i.e. donde la edad sea menor a diez y el sexo sea 'F'.

NOMBRE	EDAD	SEXO	SALARIO
Mónica	34	F	15 000
Fabián	23	M	10 300
Itzel	30	F	17 600
Jessica	24	F	4 500

Figura 2.1: Tabla de ejemplo

Para realizar una inserción en `tabla` la consulta sería de la siguiente manera: `INSERT INTO tabla VALUES('Nuevo',22,'M','1 000')`, donde los da-

[3]En una arquitectura cliente-servidor, el cliente realiza peticiones al servidor que presta servicios.

[4]SQL es el lenguaje por excelencia en los DBMS

[5]Una palabra reservada es aquella que para el lenguaje de programación tiene un significado gramatical que denota cierta acción.

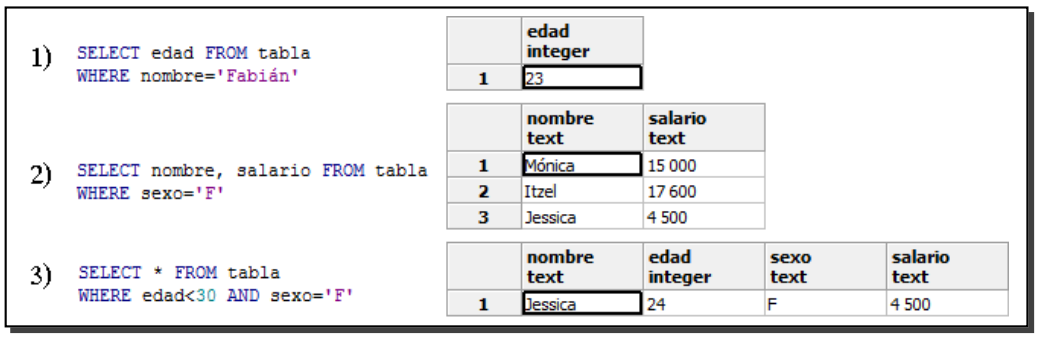

Figura 2.2: En la consulta 1 deseamos conocer la edad de Fabián. En la 2 se desea saber el nombre y el salario de las mujeres. En la tercer consulta buscamos toda la información de las mujeres menores de 30

tos dentro del paréntesis son los valores que tomará cada atributo (columna); como edad es numérico no es necesario colocarle comillas a diferencia de los otros tres que son de tipo texto. De esta forma se tiene un nuevo registro en la tabla.

2.2 Esquemas simétricos de cifrado

La criptografía cuyo nombre proviene del griego *kryptos* y *grafos*, es el arte utilizada para cifrar y descifrar datos mediante el uso de procedimientos matemáticos y lógicos para el libre intercambio de información entre dos entes, de modo que únicamente ellos puedan entenderlo. Así, la criptografía es una herramienta que se utiliza para que dos entes, llamémosles Alberto y Alicia, puedan comunicarse de manera segura mediante un canal inseguro, tal que una tercera entidad, digamos Oscar, que observa la conversación por el canal no pueda entender lo que Alberto y Alicia se transmiten. El canal inseguro puede ser una línea telefónica o la propia red global.

Para lograr que los mensajes enviados sólo puedan ser interpretados por Alberto y Alicia se utilizan *criptosistemas* que son denotados por el conjunto (P,C,K,E,D), donde P es el conjunto de todos los posibles textos en claro o textos planos[6], C es el conjunto de todos los posibles textos cifrados[7] (en

[6]Texto en claro es el formato de los datos tal como los utilizamos para comunicarnos. Pueden ser frases, números, texto, etc.

[7]Texto cifrado es el mensaje que ya no es entendible y puede viajar por el canal inseguro.

español se le llama "texto cifrado"), el conjunto K consiste de las posibles llaves para cifrar, E es la función biyectiva que transforma el texto en claro en texto cifrado y finalmente D denota la función inversa de E, D es biyectiva y es la transformación del texto cifrado a texto en claro.

Han existido muchos *criptosistemas* sencillos como el *Shift Cipher* o también llamado Cifrado de Julio César, el Cifrado de Substitución, el *Affine Cipher*, el *Vigenere Cipher*, el *Hill Cipher*, el cifrado de permutación, etc.; todos ellos son denominados criptosistemas clásicos y han sido la base para la realización de sistemas criptográficos más complejos y robustos. Todos estos criptosistemas no son utilizados hoy en día en aplicaciones reales, debido a que existen técnicas para realizar el *criptoanálisis*[8] de cada uno de ellos y estas técnicas son de dominio público. A este tipo de cifrados se les denomina cifrados simétricos o de llave privada; que son, como su nombre lo indica, aquellos en los que las llaves de cifrado sólo son conocidas por los protagonistas de la comunicación; existen otros esquemas de cifrado denominados de llave pública o asimétricos los cuales poseen dos llaves para el envío de mensajes, una pública y una privada, dentro de este tipo de esquemas encontramos al *RSA*, *ElGamal*, el llamado de *Diffie-Hellman*, las curvas elípticas, entre otros.

Hoy en día los cifrados simétricos de bloque más utilizados son los llamados cifrados de producto[9], sólo por mencionar algunos tenemos a *DES* (Data Encryption Standard), *AES* (Advanced Encryption Standard), *SERPENT*, *Feistel*, *Triple DES*, *IDEA*, *Skipjack*, *Blowfish*, *RC2*, etc.

2.2.1 El estándar de cifrado avanzado

AES [9] es un cifrado de bloques por Rijmen y Daemen que surgió en el 2000 como reemplazo de DES. *AES* requiere que la longitud de bloque del texto plano sea de 128 bits y puede soportar llaves de 128, 192 y 256 bits. El modo en que *AES* funciona se muestra en la Fig. 2.3, donde `Nr` es el número de rondas a realizar y está denotado por el tamaño de la llave; si la llave

[8]El criptoanálisis se refiere a las técnicas utilizadas para romper algún texto cirfado. Si se conoce el metodo clásico de cifrado, es relativamente fácil llegar al texto en claro.

[9]Los cifrados de producto frecuentemente incorporan operaciones de permutación y sustitución.

es de 128 bits `Nr`=10, para 192 bits `Nr`=12 y cuando la llave es de 256 bits, `Nr`=14. Cada bloque del algoritmo explica brevemente:

- TEXTO EN CLARO. Entrada del algoritmo AES, y resulta en una matriz de números hexadecimales de orden cuatro por cuatro, i.e. 16 bytes.
- `AddRoundKey`. Función que toma el bloque de texto claro y lo suma mediante un `xor` ($\oplus$) con la llave en curso. Más adelante se presenta el algoritmo de generación de llaves que indica qué llave usar en cada llamada a esta función.
- `SubBytes`. Función que hace el reemplazo de cada valor de la matriz orden cuatro por cuatro por uno de la caja `S` mostrada en la Fig. 2.4 (el nuevo valor también puede ser obtenido mediante operaciones algebraicas [42]). Se identifica el valor de la caja `S` tomando ambos valores hexadecimales como un par de coordenadas, i.e. si nuestro número hexadecimal a ingresar es `43`, éste es reemplazado por `1A` (renglón `4`, columna `3`).
- `ShiftRows`. Función en la que se realizan desplazamientos de bytes a la izquierda de la tabla orden cuatro por cuatro; en la primer fila (fila 0) no se hacen corrimientos,en la segunda (fila 1) se realiza un corrimiento a la izquierda, en la tercera (fila 2) se corren dos lugares y en la cuarta (fila 3) se cumple con el corrimiento de tres espacios.
- `MixColumns`. Función que genera la multiplicación de cada columna de la tabla orden cuatro por cuatro por cierta matriz de elementos del campo F_{2^8}, la suma es módulo dos y no es más que una función `xor`.

Para la función `AddRoundKey`, se realiza una función `xor` de la matriz orden cuatro por cuatro con la llave de ronda en curso; la llave en curso se obtiene expandiendo la llave inicial. Para la primer llamada de la función, antes del ciclo, se suma el texto en claro jutno con la llave inicial; las posteriores 9 llamadas en el ciclo y la final requieren de una llave diferente. Se el procedimiento de expansión de llaves para bloques de 128 bits en el Algoritmo 1.

El algoritmo para descifrar se presenta en la Fig. 2.6 que como puede observarse es muy parecido al cifrador; cabe mencionar que las llaves se usan

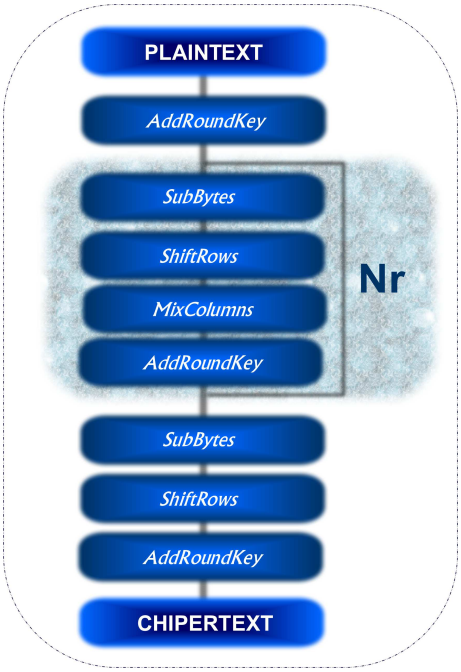

Figura 2.3: Algoritmo de cifrado *AES*

Y / X	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	63	7C	77	7B	F2	6B	6F	C5	30	01	67	2B	FE	D7	AB	76
1	CA	82	C9	7D	FA	59	47	F0	AD	D4	A2	AF	9C	A4	72	C0
2	B7	FD	93	26	36	3F	F7	CC	34	A5	E5	F1	71	D8	31	15
3	04	C7	23	C3	18	96	05	9A	07	12	80	E2	EB	27	B2	75
4	09	83	2C	1A	1B	6E	5A	A0	52	3B	D6	B3	29	E3	2F	84
5	53	D1	00	ED	20	FC	B1	5B	6A	CB	BE	39	4A	4C	58	CF
6	D0	EF	AA	FB	43	4D	33	85	45	F9	02	7F	50	3C	9F	A8
7	51	A3	40	8F	92	9D	38	F5	BC	B6	DA	21	10	FF	F3	D2
8	CD	0C	13	EC	5F	97	44	17	C4	A7	7E	3D	64	5D	19	73
9	60	81	4F	DC	22	2A	90	88	46	EE	B8	14	DE	5E	0B	DB
A	E0	32	3A	0A	49	06	24	5C	C2	D3	AC	62	91	95	E4	79
B	E7	C8	37	6D	8D	D5	4E	A9	6C	56	F4	EA	65	7A	AE	08
C	BA	78	25	2E	1C	A6	B4	C6	E8	DD	74	1F	4B	BD	8B	8A
D	70	3E	B5	66	48	03	F6	0E	61	35	57	B9	86	C1	1D	9E
E	E1	F8	93	11	69	D9	8E	94	9B	1E	87	E9	CE	55	28	DF
F	8C	A1	89	0D	BF	E6	42	68	41	99	2D	0F	B0	54	BB	16

Figura 2.4: La caja `S` para *AES*

Algoritmo 1 Algoritmo de expansión de llaves

Entrada: Llave inicial, matriz auxiliar `RCon` mostrada en la Fig. 2.5

1: inicializamos `número_de_llave_generada` en cero
2: **repeat**
3: Un vector auxiliar `aux`, guarda el vector generado al realizar un corrimiento de la última columna de la llave `número_de_llave_generada`-1 (si es la primer llave a generar se toma la llave inicial) en una posición hacia arriba
4: Cambiamos los valores de `aux` por los de la caja `S`
5: La columna 1 de la llave `número_de_llave_generada` guarda la suma de `aux`, la primer columna de la llave `número_de_llave_generada`-1 (si es la primer llave a generar se toma la llave inicial) y la columna `número_de_llave_generada` de la tabla `RCon`
6: La columna 2 de la llave `número_de_llave_generada` se obtiene mediante la operación `xor` de la columna 1 de la llave `número_de_llave_generada` (formada en el paso anterior) y de la columna 2 de `número_de_llave_generada`-1 (si es la primer llave a generar se toma la llave inicial)
7: La columna 3 de la llave `número_de_llave_generada` se obtiene sumando la columna 2 de la llave `número_de_llave_generada` y la columna 3 de `número_de_llave_generada`-1 (si es la primer llave a generar se toma la llave inicial)
8: La columna 4 de la llave `número_de_llave_generada` se obtiene sumando la columna 3 de la llave `número_de_llave_generada` y la columna 4 de `número_de_llave_generada`-1 (si es la primer llave a generar se toma la llave inicial)
9: Incrementar en uno el valor de `número_de_llave_generada`
10: **until** `número_de_llave_generada` sea 10
11: **return** 10 nuevas llaves

01	02	04	08	10	20	40	80	1B	36
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00
00	00	00	00	00	00	00	00	00	00

Figura 2.5: La caja auxiliar `RCon` para la expansión de llaves

de la 10 a la inicial. Para la función `InvSubByte` es el mismo caso que en el cifrado, se genera el reemplazo por el valor de la caja `S` invertida. En la función `InvShiftRows` los desplazamientos se realizan hacia la derecha (a diferencia que en el cifrado que se hacian a la izquierda). Para la operación de `InvMixColumns` se hace una multiplicación con la matriz de cuatro por cuatro bytes que escrita por filas es `M=(0e,0b,0d,09), (09,0e,0b,0d), (0d,09,0e,0b), (0b,0d,09,0e)`. Para la función `InvAddRoundKey` se realiza un `xor` con el producto de la llave de ronda por la matriz orden cuatro por cuatro.

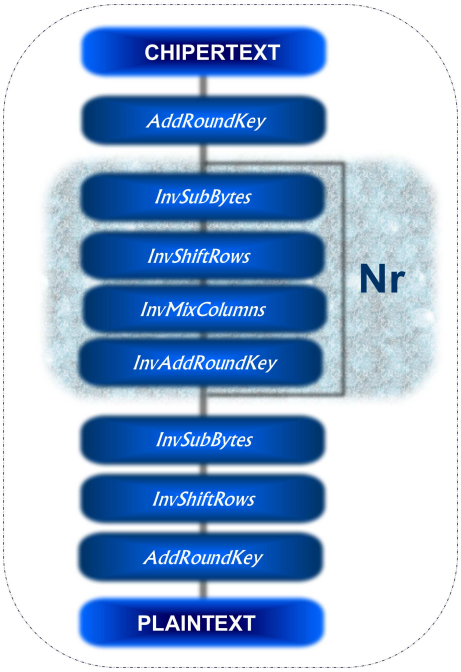

Figura 2.6: Algoritmo de descifrado *AES*

Estos esquemas son muy útiles y funcionan a la perfección si el tamaño del texto plano es igual al tamaño de bloque de cifrado; debido a que esto no ocurre a menudo, han surgido diversos modos de operación que junto con estos cifradores de bloques logran cifrar mensajes de longitud mayor que el bloque de cifrado. De entre estos modos de operación se encuentran el ECB (*Electronic CodeBook Mode*), CBC (*Cipher Block Chaining*), OFB (*Output Feed Back*), CFB (*Cipher Feed Back*), CTR (*Counter Mode*) que aseguran

la confidencialidad pero no la autenticidad[10] de los datos.

2.2.2 Cifrado con autenticación

El cifrado con autenticación AE (*Authenticated Encryption*) es un término que define a aquellos criptosistemas que protegen la autenticidad y la confidencialidad de las comunicaciones. Un MAC (*Message Authentication Code*) es una pequeña porción de información que corresponde a un breve resumen del mensaje y permite autenticar al mismo; un algoritmo MAC es conocido también como una función *hash* criptográfica de llaves y acepta como parámetros una llave secreta y un mensaje a autenticar, y retorna un MAC (también es llamado *tag*). Existen tres diferentes formas de construir sistemas de cifrado con autenticación utilizando MACs:

- **MAC después cifrar**. Primero se hace la función *hash* del mensaje M y la llave K_1 para obtener el *tag* σ; posteriormente se cifra el par (M,σ) con la llave K_2.
- **Cifrar después MAC**. Primero se cifra el mensaje M con la llave K_2 para obtener C; posteriormente se obtiene $\sigma \leftarrow \mathrm{MAC}_{K_1}(C)$ y generamos (C,σ).
- **Cifrar y MAC**. Primero se cifra el mensaje M con la llave K_2 para obtener C; posteriormente se obtiene $\sigma \leftarrow \mathrm{MAC}_{K_1}(M)$ y generamos (C,σ).

Bajo estas tres formas de autenticar los mensajes han surgido diversos esquemas como el modo CCM (*Counter with CBC-MAC*) [47], el EAX [3], el GCM (*Galois Counter Mode*) [34], entre otros; estos esquemas son llamados Esquemas de Cifrado Autenticado de dos fases (*Two Pass Authenticated Encryption Schemes*).

De igual forma existen Esquemas de Cifrado Autenticado de una fase los cuales no caen en ninguna de las tres categorias anteriormente explicadas. Algunos de los Esquemas de Cifrado Autenticado de una fase son el IAPM

[10]La autenticidad de los datos garantiza conocer si los mismos no han sido modificados durante el proceso de envio por el canal.

[22], el IACBC [25], OCB [40] y el OCB1 [39],etc. Los esquemas de una fase son más eficientes que aquellos de dos fases. Enseguida hablaremos a detalle del modo **OCB1** [39].

2.2.2.1 OCB1

OCB1 [39] es un modo de operación que usa un *Tweakable Block Cipher* [30][11] que surgió para perfeccionar el modo de operación OCB [40], ya que es más simple y más fácil que el original. Este esquema permite, entre otras cosas, agregar autenticación a los datos cifrados y por otra parte, tiene la ventaja de que puede manejar cualquier longitud de mensaje, es decir no necesita que se utilicen los bloques completos como otros esquemas, esto es una ventaja debido a que OCB1 no trabaja con información innecesaria. El modo de operación de `OCB1` se muestra en la Fig. 2.7 donde N es el *nonce* (Un *nonce* es una cadena que no puede ultilizarse en más de una ocasión); $P_1, P_2, \ldots, P_{m-1}, P_m$ es el conjunto de bloques del *plaintext* y $C_1, C_2, \ldots, C_{m-1}, C_m$ son los bloques de cifrado resultantes. Como ya se mencionó este esquema recibe cadenas de longitud variable y respeta esta misma longitud para el dato cifrado; la longitud de $P_1, P_2, \ldots, P_{m-1}$ es de 128 bits, mientras que la longitud del último bloque P_m es menor o igual a 128 bits y finalmente, la longitud de N es de 128 bits.

Cabe mencionar que el cifrado del *nonce* $E_k(N)$ es de 128 bits, así consideremos a n=128; por otra parte, podemos considerar a $E_k(N)$ como un elemento del Campo de Galois $GF(2^n)$ [29]. Cualquier elemento de $GF(2^n)$ puede ser representado de diversas formas: (i) como un punto abstracto en el campo; (ii) como una simple cadena de n bits $a_n \ldots a_1 a_0 \in \{0,1\}^n$; (iii) como un polinomio formal de grado menor que n $a(x) = a_n x^n + \cdots + a_1 x + a_0$ con coeficientes 0 y 1; (iv) como un entero entre 0 y $2^n - 1$; para nuestro caso representaremos cualquier elemento de campo como un polinomio formal. Otra parte que debemos considerar es que si deseamos multiplicar dos polinomios que pertenezcan al campo $GF(2^n)$, necesitamos multiplicarlos y reducir el resultado con un polinomio irreducible de grado n. Para la seguridad de `OCB1` es necesario que el polinomio irreducible sea primitivo. El costo computacional

[11] Un *Tweakable Block Cipher* es un cifrado de bloque que además de considerar el mensaje (plaintext) y la llave de cifrado, agrega una tercera entidad llamada "tweak".

de multiplicar elementos en el campo depende del tamaño del polinomio irreducible. Es necesario encontrar un polinomio que debe poseer el mayor número de coeficientes cero, en nuestro caso usaremos $p_{128} = x^{128} + x^7 + x^2 + x + 1$. Además, para obtener $xE_k(N)$ necesitamos multiplicar el cifrado $E_k(N)$ por el monomio x; lo que significa, $a = a_{127} \dots a_1 a_0$ multiplicado por x es igual a $a_{127}x^{128} + a_{126}x^{127} + \cdots + a_1 x^2 + a_0 x$; como podemos observar, multiplicar el polinomio por x resulta en un corrimiento a la izquierda (el símbolo es $\ll$) de a en un bit. Al multiplicar un elemento de un campo por otro del mismo, puede darse el caso que el resultado del producto no pertenezca al mismo campo; e.g. $xP_1(x) = x(x^{127} + x^5 + x^2 + x) = x^{128} + x^6 + x^3 + x^2$, es por tal motivo que en este caso el resultado debe ser reducido a un elemento que pertenezca al mismo $GF(2^n)$. Para la reducción, si el bit mas significativo es cero (es decir el bit que sale), se realiza el corrimiento de forma normal ya que no se pierde nada, es decir si el bit más significativo es cero $x(a) = a \ll 1$; y por el lado contrario, si este bit resulta ser uno, se requiere sumarlo al polinomio de la siguiente manera: $x(a) = (a \ll 1) \oplus x^{128}$; para poder realizar esto, recordemos que el polinomio irreducible que utilizamos es $p_{128} = x^{128} + x^7 + x^2 + x + 1 = 0$, si despejamos obtenemos $x^{128} = x^7 + x^2 + x + 1$ que es el polinomio que se suma; es decir, agregar x^{128} significa realizar un `xor` con $0^{120}10^41^3$ (que en binario representa el número 135; esto debido a que $2^7 = 128, 2^2 = 4, 2^1 = 2y2^0 = 1$, por lo tanto 128+4+2+1=135).

El mecanismo de cifrado de `OCB1` se muestra en el Algoritmo 2 y el de descifrado se presenta en el Algoritmo 3.

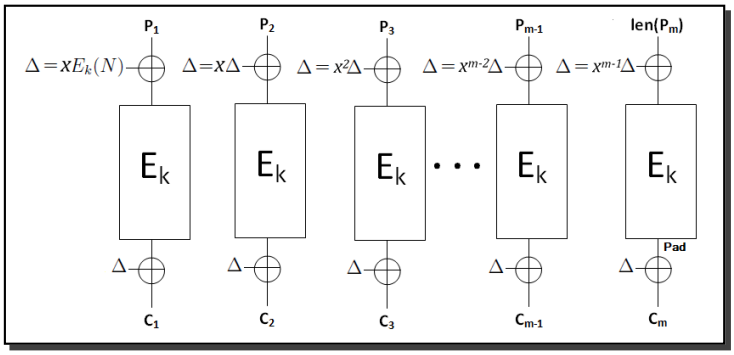

Figura 2.7: Modo de operación de `OCB1`

Ya para finalizar esta sección, y saliéndonos un poco de los tópicos crip-

Algoritmo 2 OCB1.Encrypt$_K^N(P)$

Partition P into $P_1, P_2, \cdots, P_m$
$\Delta \leftarrow xE_K(N)$
$\Sigma \leftarrow 0^n$
for all $i = 1$ to $m-1$ **do**
  $C_i \leftarrow E_K(P_i \oplus \Delta) \oplus \Delta$
  $\Delta \leftarrow x\Delta$
  $\Sigma \leftarrow \Sigma \oplus P_i$
end for
Pad $\leftarrow E_K(\text{len}(P_m) \oplus \Delta)$
$C_m \leftarrow P_m \oplus$ Pad
$\Sigma \leftarrow \Sigma \oplus C_m 0^* \oplus$Pad
$\Delta \leftarrow (1+x)\Delta$
Tag $\leftarrow E_K(\Sigma \oplus \Delta)$
$T \leftarrow$ Tag[first τ bits]
return $C \leftarrow C_1 \,||\, C_2 \,||\, \cdots \,||\, C_m \,||\, T$

Algoritmo 3 OCB1.Decrypt$_K^N(C)$

Partition C into $C_1, C_2, \cdots, C_m, T$
$\Delta \leftarrow xE_K(N)$
$\Sigma \leftarrow 0^n$
for all $i = 1$ to $m-1$ **do**
  $P_i \leftarrow E_K^{-1}(C_i \oplus \Delta) \oplus \Delta$
  $\Delta \leftarrow x\Delta$
  $\Sigma \leftarrow \Sigma \oplus P_i$
end for
Pad $\leftarrow E_K(\text{len}(C_m) \oplus \Delta)$
$P_m \leftarrow C_m \oplus$ Pad
$\Sigma \leftarrow \Sigma \oplus C_m 0^* \oplus$Pad
$\Delta \leftarrow (1+x)\Delta$
Tag $\leftarrow E_K(\Sigma \oplus \Delta)$
$T' \leftarrow$ Tag[first τ bits]
if $T = T'$ **then**
  return $P \leftarrow P_1 \,||\, P_2 \,||\, \cdots \,||\, P_m$
else
  return INVALID
end if

tográficos, hablemos del sistema de numeración posicional **Base 64**.

Base 64 se ha utilizado en la codificación de diversasa aplicaciones Web; éste se usa debido a que al cifrar cierto texto plano se obtienen muchos caracteres que no pueden ser reconocidos en la gran mayoría de aplicaciones. Base 64 utiliza un alfabeto de 64 elementos: A-Z, a-z y 0-9 en los que cualquier texto cifrado puede ser codificado.

La transformación se realiza al tomar todos los caracteres e dentificar su número correspondiente en ASCII[12] [18], posteriormente se convierten a binario dichos números, cada uno en cadenas de ocho bits. Después concatemanos todas las cadenas de ocho bits, y se van tomando de seis en seis bits (6 bits forman como máximo 64 valores diferentes en binario: 2^6), el número binario resultante de estos seis bits nos indica el caracter de Base 64 que se utilizará.

Para fines didácticos, mostremos un ejemplo.

Pensemos en la cadena *Man*; si codificamos estas letras en ASCII, son almacenadas como los bytes 77, 97 y 110, es decir, 01001101, 01100001, 01101110 en base 2.

Al concatenar estos bits, tenemos la cadena 010011010110000101101110. Posteriormente tomamos subcadenas de 6 en 6 bits y leemos el valor binario: 010011=19, 010110=22, 000101=5 y 101110=46; los valores 19, 22, 5 y 46 corresponden a T, W, F y u respectivamente.

Así *Man* codificado en Base 64 es *TWFu*.

2.3 Algoritmos genéticos

Los algoritmos genéticos son técnicas computacionales que se utilizan para obtener soluciones exactas o de aproximación en problemas de optimización[13]. Los algoritmos genéticos fueron inventados por John Holland [23]

[12] ASCII (*American Standard Code for Information Interchange*) es el Código Estadounidense Estándar para el Intercambio de Información.

[13] Optimizar implica buscar la mejor manera de realizar cualquier cosa.

en los 1960's y son uno de los paradigmas del cómputo evolutivo, de hecho el más popular. Otros paradigmas del cómputo evolutivo son la programación evolutiva [32] y las estrategias evolutivas [36]; algunas otras técnicas son la optimización por colonia de hormigas [15], la búsqueda dispersa [28], la evolución diferencial [43], la programación genética [27], la optimización mediante cúmulos de partículas [26, 33], los sistemas inmunes artificiales [14] y los algoritmos culturales [37].

El principal problema de los algoritmos genéticos estriba en lograr abstraer un problema del mundo real, que se desee optimizar, en una expresión muy simple; mientras mejor esté representado el problema, mejores resultados arroja el algoritmo. Debido a que los algoritmos genéticos están inspirados en la evolución biológica y en los mecanismos de sobreviviencia de la selección natural de Darwin [11], mediante la preservación del más apto, utilizan términos biológicos tales como cromosoma, gene, cruza, mutación, población, aptitud, etc. Un **cromosoma** es una de las cadenas de ADN[14] que se encarga de la transmisión de información genética y principalmente se compone de genes; **gene** es una sección de ADN que es escencial para llevar a cabo cierta función química, y es la unidad fundamental de la herencia; se denomina **genoma** al conjunto total de genes que posee un organismo; un **individuo** es un miembro de una población; una **población** es un grupo de individuos que interactúan juntos; el **genotipo** es la información genética de un organismo, no es visible a simple vista; el **fenotipo** es la interpretación del genotipo, son los ragsos observables de un individuo; la **mutación** se aprecia como cambios heredables del genoma; la **aptitud** de un individuo es su capacidad de sobrevivir y repoducirse; la **selección** es el mecanismo por el cuál los individuos son elegidos para reproducirse; la **reproducción** o **cruza**, implica crear nuevos individuos. estos terminos biológicos tienen un significado análogo bajo el contexto de la computación evolutiva:

- **Cromosoma**: estructura de datos que contiene una cadena de parámetros de diseño o genes. Puede almacenarse en una cadena de bits o un arreglo de enteros.

- **Gene**: subsección del cromosoma que representa un único parámetro.

[14]ADN significa Ácido desoxiribonucleico y es el material genético fundamental de todo ser vivo

- **Genotipo**: codificación de los parámetros que representan una de las posibles soluciones del problema.
- **Fenotipo**: decodificación del cromosoma.
- **Individuo**: miembro de la población de soluciones posibles de un problema. Un individuo es poseedor de un cromosoma.
- **Aptitud**: es el valor que se le asigna a cada individuo, que califica que tan bueno es el individuo con respecto al resto de la población para la solución del problema.
- **Generación**: es la creación de una nueva población a partir de la cruza y mutación de individuos.

Los algoritmos genéticos trabajan a nivel genotipo, es decir, únicamente toman el gene de los individuos. Los elementos de un algoritmo genético son:

Representación: Se utiliza para simbolizar las variables y así obtener un cromosoma manipulable para el algoritmo. Las representaciones mas usadas son la real, la binaria y la entera.

Función objetivo: Es la función a optimizar, ya sea para maximizarla o minimizarla. Cada individuo es evaluado en la función objetivo y así se mide qué tan buen invididuo es (su aptitud).

Población: Es el conjunto de individuos donde cada individuo posee cierta aptitud.

Mecanismo de selección: Se utiliza para elegir qué padres se cruzarán y así generarán nuevos individuos. Existen diversos tipos de selección como la Selección Proporcional [23], la Selección de Estado Uniforme [48] y la Selección Mediante Torneo [46].

La selección mediante torneo se basa en la comparación directa entre individuos, comunmente son dos participantes por torneo. Existen dos vertientes de este tipo de selección, la determinística y la probabilística; para ambos casos, primeramente se toma aleatoriamente a los participantes del torneo y se comparan con base en su aptitud; en el caso determinista siempre se elige

al mejor individuo, es decir, el que tenga mejor aptitud; en el caso probabilístico se elige al mejor individuo con cierta probabilidad, de lo contrario gana el peor individuo.

Operador de cruza: La finalidad es generar un nuevo cromosoma a partir de la combinación de las partes de los cromosomas padres. Existen tres tipos de cruza básicas para representaciones binarias; la de un punto [23], la de dos puntos y la uniforme [44]. Podemos observar la cruza de dos puntos en la Fig 2.8 donde se dividen los cromosoma en tres partes por medio de dos puntos; y pasan cada parte a uno de sus hijos.

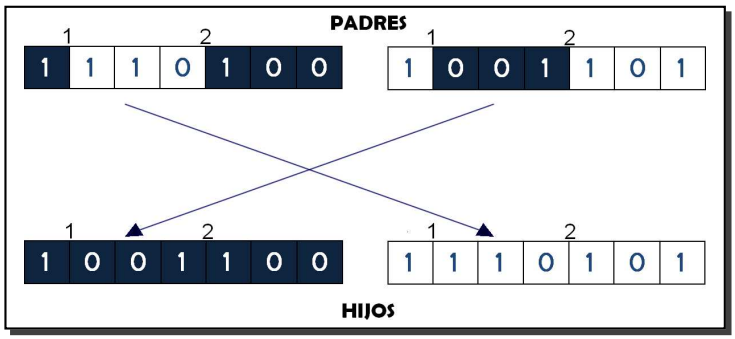

Figura 2.8: Ejemplo de una cruza de dos puntos

Operador de mutación: Este operador genera un nuevo cromosoma a partir de pequeñas modificaciones de un sólo cromosoma padre y el objetivo es generar soluciones que la cruza no puede producir.

Elitismo: Asegura que la aptitud máxima de cada generación es preservada a la siguiente generación, es decir, el mejor individuo de la generación es preservado.

El principal operador de los algoritmos genéticos es la cruza, la mutación es un operador secundario y utiliza selección probabilística; esto es lo que lo diferencía del resto de los paradigmas. La representación más comunmente utilizada en los genéticos es la binaria. El pseudocódigo de un algoritmo genético se muestra en el Algoritmo 4.

Algoritmo 4 Algoritmo Genético

```
1: Inicializar población P de forma aleatoria;
2: Evaluar la aptitud de los individuos en P;
3: repeat
4:     Seleccionar padres;
5:     Cruzar padres;
6:     Aplicar mutación a los hijos (generados de la cruza);
7:     Evaluar la aptitud de los hijos;
8:     Seleccionar nueva población de padres e hijos;
9: until Número de generaciones máximo haya concluido
```

Algunas de las aplicaciones más importantes de los algoritmos genéticos son:

- Optimización
- Aprendizaje de máquina
- Reconocimiento de patrones
- Predicción

3

Estado del Arte

El salvaguardar la información propia en contra de terceros es una preocupación que ha brotado en los últimos años a partir del surgimiento de la red global. La aparición de Internet trae consigo una serie de preocupaciones al momento de compartir información, ya que para que las comunicaciones a distancia puedan generarse, es méramente necesario que dicha información pase a través de canales inseguros, los cuales pueden permitir que cualquier persona obtenga la información y hacer mal uso de ella si así lo deseara.

Por otra parte, aprovechando las bondades que la red global ofrece surgió el cómputo en nube, que como ya se mencionó en capítulos anteriores es una tecnología que brinda servicios a través de Internet y uno de ellos es denominado **DaaS** [21] (acrónimo de *Database as a Service* que traducido al español es Base de Datos como Servicio) cuyo principio es el hospedaje de bases de datos en Internet.

DaaS [21] es una opción para todas aquellas empresas que buscan despreocuparse de la administración de sus datos, hospedando su información en un servidor remoto; la arquitectura que DaaS [21] adopta es la de cliente-

servidor[1], ya que el servidor posee la base de datos y realiza peticiones del cliente como consultas, actualizaciones, altas y bajas en la base de datos. Considerando este modelo, el hecho de que nuestra base de datos esté en el poder de un tercero hace vulnerables nuestros datos en contra de plagios, robos, fraudes, etc; ya que nada nos garantiza que el prestatario del servicio no haga mal uso de nuestra información. Es por ello que ha surgido la necesidad de inyectarle seguridad a las bases de datos bajo el contexto de DaaS [21]

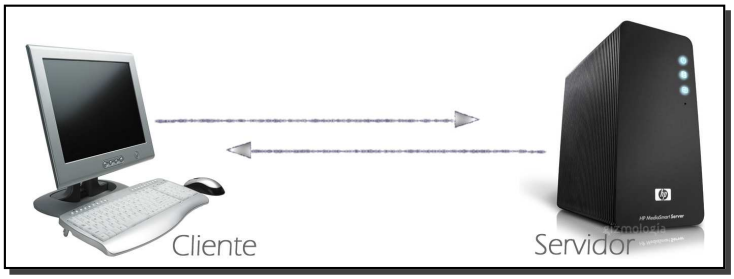

Figura 3.1: Arquitectura cliente-servidor

Para cubrir esta exigencia de la confidencialidad de la información se ha utilizado la criptografía como medio para mantener los datos incomprensibles. En el caso de DaaS [21]han surgido diversos esquemas criptográficos que permiten tener ilegibles los datos y así permanecer seguros.

En este capítulo haremos un breve recorrido a través de un compendio de variados esquemas criptográficos que han surgido por la necesidad de hermetizar los datos bajo el contexto de DaaS [21]; todos estos modelos búscan un fin común, la privacidad de los datos, estribando la seguridad del esquema en diferentes aspectos y utilizando diversas técnicas.

En 2006 propuesto por Zhiqiang Yang et. al. [49], reconociendo la importancia de la confidencialidad en bases de datos bajo el contexto DaaS [21], surgió un esquema que estriba la seguridad de la base de datos totalmente en los cifrados de bloque; i.e. genera consultas, exclusivamente de igualdad, a una base de datos que está completamente cifrada y situada en una localidad apartada. Este esquema presenta tres soluciones al problema; la primera

[1]La arquitectura cliente-servidor consiste en un cliente que realiza peticiones a otra entidad llamada servidor, que le da respuesta; esta arquitectura se muestra en la Fig. 3.1.

denominada Solución Básica se enfoca a cubrir los aspectos de seguridad más elementales, la segunda solución llamada Seguridad Mejorada inyecta mayor seguridad a la solución anterior perfilando el esquema a la seguridad total y finalmente se presenta una solución que no necesariamente contiene la seguridad de las soluciones anteriores, pero que se preocupa del desempeño de las consultas utilizando Metadatos.

Este esquema requiere de una tabla en texto claro **T** con i registros y j columnas, misma que es transformada en una tabla cifrada y almacenada en un servidor remoto.

A continuación detallaremos un modelo que responde a las consultas de tipo `Select ... from TABLA where Atributo`$_x$ = `valor`, i.e. regresa al cliente el valor que satisface a la igualdad entre lo almacenado y lo obtenido.

Primeramente hablemos de la Solución Básica. Es propio mencionar que para obtener la tabla cifrada T' se requiere cifrar con redundancia especial a T. El cifrado de cada celda T_{ij} en T' consta de dos partes $T'_{ij}[1]$ y $T'_{ij}[2]$, para ello se requiere de dos llaves s_1 y s_2 que permanecen secretas y en poder del cliente. $T'_{ij}[1]$ es un simple cifrado de T_{ij} mediante cualquier cifrador de bloque E que recibe como parámetros cadenas del conjunto $S \in 0, 1^*$, como los vistos en la sección 2.2. $T'_{ij}[2]$ es un verificador que junto con $T'_{ij}[1]$ permite conocer si la consulta satisface o no al registro. $T'_{ij}[1]$ y $T'_{ij}[2]$ satisfacen una ecuación secreta que está dada por T_{ij}. La ecuación se muestra a continuación:

$$T'_{i,j}[1] = E_s(T_{i,j}),$$
$$T'_{i,j}[2] = E_{f(T_{i,j})}(T'_{i,j}[1]),$$

donde f es una función que en los siguientes párrafos se explicará a detalle. Cuando se quiere realizar una consulta con la condición $\mathtt{A}_j$ = `valor`, únicamente envia al servidor $f(v)$, el servidor verifica en cada registro si la ecuación

$$T'_{i,j}[2] = E_{f(v)}(T'_{i,j}[1])$$

se satisface; de ser así, el registro i es enviado al cliente para posteriormente obtener la información buscada.

Es decir, T' se obtiene de la siguiente manera:

$$\begin{aligned} T'(i,j) &= (T'(i,j)[1], T'(i,j)[2]) \\ &= (E_{s_1}(T_{i,j}, r_{i,j}), E_{E_{s_2}(T_{i,j},j)}(E_{s_1}(T_{i,j}, r_{i,j}))), \end{aligned}$$

que como se puede observar la primer parte se obtiene cifrando con la llave s_1 el valor de la celda concatenado con una cadena aleatoria y la segunda parte la generamos al cifrar la primer parte con el cifrado del valor concatenado con la columna correspondiente utilizando s_2.

La seguridad de este esquema está directamente relacionada con la seguridad del cifrador de bloque, i.e. si el cifrador de bloque es seguro, el esquema lo és. Por otra parte, se recomienda utilizar cadenas pseudo aleatorias en el esquema para que así sólo se recuerde la semilla, no toda la serie de cadenas.

Para entender mejor esta SOLUCIÓN BÁSICA, mostremos un ejemplo. Tenemos la Tabla 3.1 que consta de tres atributos: *Nombre*, *Edad* y *Sexo*, y cuatro registros. La transformación de T a T' se muestra en la Tabla 3.2, donde el símbolo || implica concatenación de cadenas y las cadenas implican $r_1, r_2, \ldots, r_{12}$ cadenas aleatorias diferentes para cada caso.

Nombre	**Edad**	**Sexo**
Ron	19	M
Ana	15	F
Pam	14	F
Jon	15	M

Tabla 3.1: Tabla en texto claro T.

T' mostrada en la Tabla 3.2 es alojada en el servidor, lista para ser consultada.

Generar una consulta en este esquema es simple. Imaginemos que queremos realizar la siguiente consulta `SELECT Edad,Sexo FROM T WHERE Nombre='Pam';` lo que el cliente debe de realizar es cifrar el valor *Pam* concatenado con el nombre de la columna, en este caso es *Nombre* y cifrarlo con s_2 de la siguiente

T'		
Nombre	**Edad**	**Sexo**
$E_{s_1}(Ron\|\|r_1)$	$E_{s_1}(19\|\|r_2)$	$E_{s_1}(M\|\|r_3)$
$E_{E_{s_2}(Ron\|\|Nombre)}(E_{s_1}(Ron\|\|r_1))$	$E_{E_{s_2}(19\|\|Edad)}(E_{s_1}(19\|\|r_2))$	$E_{E_{s_2}(M\|\|Sexo)}(E_{s_1}(M\|\|r_3))$
$E_{s_1}(Ana\|\|r_4)$	$E_{s_1}(15\|\|r_5)$	$E_{s_1}(F\|\|r_6)$
$E_{E_{s_2}(Ana\|\|Nombre)}(E_{s_1}(Ana\|\|r_4))$	$E_{E_{s_2}(15\|\|Edad)}(E_{s_1}(15\|\|r_5))$	$E_{E_{s_2}(F\|\|Sexo)}(E_{s_1}(F\|\|r_6))$
$E_{s_1}(Pam\|\|r_7)$	$E_{s_1}(14\|\|r_8)$	$E_{s_1}(F\|\|r_9)$
$E_{E_{s_2}(Pam\|\|Nombre)}(E_{s_1}(Pam\|\|r_7))$	$E_{E_{s_2}(14\|\|Edad)}(E_{s_1}(14\|\|r_8))$	$E_{E_{s_2}(F\|\|Sexo)}(E_{s_1}(F\|\|r_9))$
$E_{s_1}(Jon\|\|r_{10})$	$E_{s_1}(15\|\|r_{11})$	$E_{s_1}(M\|\|r_{12})$
$E_{E_{s_2}(Jon\|\|Nombre)}(E_{s_1}(Jon\|\|r_{10}))$	$E_{E_{s_2}(15\|\|Edad)}(E_{s_1}(15\|\|r_{11}))$	$E_{E_{s_2}(M\|\|Sexo)}(E_{s_1}(M\|\|r_{12}))$

Tabla 3.2: Tabla *T'* de la Solución Básica.

forma: $consulta = E_{s_2}(Pam||Nombre)$. El cliente envía *consulta* y *Nombre* (el nombre del atributo) al servidor; el servidor recaba ambos datos, cifra en cada registro la primer parte de la columna *Nombre* usando como llave la cadena *consulta* y lo compara con la segunda parte, si resultan iguales se envia el registro entero al cliente. En nuestro caso, vemos que el tercer registro empata la igualdad, i.e. $E_{consulta}(T'_{3,Nombre}[1]) = T'_{3,Nombre}[2]$ ya que $E_{E_{s_2}(Pam||Nombre)}(T'_{3,Nombre}[1]) = T'_{3,Nombre}[2]$. Ya habiendo obtenido el o los registros que satisfacen la igualdad, el servidor envía al cliente la parte 1 de todos los atributos $T'_{3,Nombre}[1]$, $T'_{3,Edad}[1]$ y $T'_{3,Sexo}[1]$. Finalmente el cliente descifra los valores de $E_{s_1}(14||r_8)$ y $E_{s_1}(F||r_9)$ utilizando s_1; de esta forma $D_{s_1}(E_{s_1}(14||r_8))$ y $D_{s_1}(E_{s_1}(F||r_9))$ se obtienen los datos en texto claro y se quitan las cadenas aleatorias del final para sólo obtener 14 y F.

Esta Solución Básica no es cien por ciento segura, ya que revela cierta información de los atributos que aparecen en la condición `WHERE` de la consulta; decimos esto, porque si un tercer puede observar el procesamiento de una consulta en el servidor, puede darse el caso que la consulta regrese más de un registro, en este caso, debido a que sólo responde a consultas de igualdad de un atributo, el ente deducirá facilmente que ambos registros contienen el mismo valor.

Pensemos que alguien logra ingresar al servidor, en ese momento, se desea

realizar una consulta al atributo x y ésta retorna mas de un registro, retorna el registro 7 y el 32. El intruso fácilmente conoce que la primer parte de los registros 7 y 32 son equivalentes en el atributo x, $T'_{7,x} \equiv T'_{32,x}$.

Viendo esta limitación, se propone la solución SEGURIDAD MEJORADA que evita que se revele esta información. En esta solución se busca permutar los atributos de T' para evitar que se conozca qué aributo está siendo consultado.

Se utiliza una permutación pseudo aleatoria de los atributos, esto para que no se tenga que memorizar toda la permutación, únicamente la semilla que logró dicho arreglo de los atributos. Por otra parte, no es méramente necesario que todos los datos deban ser permutados. No se permuta ambas partes del atributo, solamente la segunda parte; de esta manera no se sabe que atributo está siendo consultado.

En esta solución, la ecuación secreta no está dada por $T_{i,j}$, como en el caso de la SOLUCIÓN BÁSICA, sino por $T_{i,\pi_s()}$; donde $T_{i,\pi_s()}$ es una permutación de un atributo. La nueva ecuación que se utiliza se muestra enseguida:

$$\begin{aligned} T'(i,j) &= (T'(i,j)[1], T'(i,j)[2]) \\ &= (E_{s_1}(T_{i,j}, r_{i,j}), E_{E_{s_2}(T_{i,j},j)}(E_{s_1}(T_{i,j}, r_{i,j}))) \end{aligned}$$

Esta ecuación es muy parecida a la que utilizamos en la SOLUCIÓN BÁSICA, la diferencia estriba en la forma que se obtiene la segunda parte, el cifrado que usamos como llave es diferente, ahora no ciframos el valor de $T_{i,j}$, sino el de $T_{i,\hat{j}}$ donde $\hat{j}$ es la permutación del atributo j.

Para expresar mejor lo anterior sigamos con el mismo ejemplo que teníamos, pero ahora adaptado a esta segunda solución.

Recordemos la tabla T que se presenta en texto claro, estamos utilizando la misma Tabla 3.1 usada en la SOLUCIÓN BÁSICA. Además de esta tabla, necesitamos la permutación $\pi_s()$ de cada uno de los atributos cuya tabla la mostramos en la Tabla 3.3. Como podemos observar cuenta con dos columnas, la columna j representa al orden original de los atributos, por lo tanto 1 representa al atributo *Nombre*, 2 representa a *Edad* y finalmente 3 es la columna *Sexo*; en cambio, la columna $\hat{j}$ indica la permutación de los atributos.

j	$\hat{j}$
1	3
2	1
3	2

Tabla 3.3: Permutación ejemplo $\pi_s()$.

Como ya mencionamos, únicamente sufre modificación la segunda parte del cifrado; como ya conocemos el procedimiento de la primer parte y por fines prácticos no la desarrollaremos en esta solución.

T al ser cifrada es vista como T' en la Tabla 3.4. Si buscamos realizar una consulta remota, debemos de seguir cierto protocolo. El protocolo de consultas de este esquema es casi idéntico al protocolo de la Solución Básica a diferencia que la cadena *consulta*, que es enviada al servidor, se obtiene de manera distinta haciendo una permutación previa de los atributos haciendo uso de la Tabla 3.3.

T'		
Nombre	**Edad**	**Sexo**
$T_{(1,1)}[1]$	$T_{(1,2)}[1]$	$T_{(1,3)}[1]$
$E_{E_{s_2}(M\|\|Nombre)}(T_{(1,1)}[1])$	$E_{E_{s_2}(Ron\|\|Edad)}(T_{(1,2)}[1])$	$E_{E_{s_2}(19\|\|Sexo)}(T_{(1,3)}[1])$
$T_{(2,1)}[1]$	$T_{(2,2)}[1]$	$T_{(2,3)}[1]$
$E_{E_{s_2}(F\|\|Nombre)}(T_{(2,1)}[1])$	$E_{E_{s_2}(Ana\|\|Edad)}(T_{(2,2)}[1])$	$E_{E_{s_2}(15\|\|Sexo)}(T_{(2,3)}[1])$
$T_{(3,1)}[1]$	$T_{(3,2)}[1]$	$T_{(3,3)}[1]$
$E_{E_{s_2}(F\|\|Nombre)}(T_{(3,1)}[1])$	$E_{E_{s_2}(Pam\|\|Edad)}(T_{(3,2)}[1])$	$E_{E_{s_2}(14\|\|Sexo)}(T_{(3,3)}[1])$
$T_{(4,1)}[1]$	$T_{(4,2)}[1]$	$T_{(4,3)}[1]$
$E_{E_{s_2}(M\|\|Nombre)}(T_{(4,1)}[1])$	$E_{E_{s_2}(Jon\|\|Edad)}(T_{(4,2)}[1])$	$E_{E_{s_2}(15\|\|Sexo)}(T_{(4,3)}[1])$

Tabla 3.4: Tabla T' de la solución Seguridad Mejorada.

Para la consulta `SELECT Nombre FROM T WHERE Edad=19`, en primera instancia el cliente identifica el atributo a consultar, en nuestro caso es *Edad*, cuyo número de atributo es 2, y su correspondiente permutación inversa,

i.e. $\pi_s^{-1}(2)$, si $\hat{j} = 2$ por lo tanto $j = 3 = Sexo$. Ya teniendo el valor de j procedemos en generar la cadena *consulta*; concatenamos el valor de *Edad* que buscamos junto con j y posteriormente lo ciframos con s_2, i.e. $E_{s_2}(19||Sexo) = consulta$. Se envía al servidor *consulta* y $\hat{j} = 3 = Sexo$. A partir de aquí se procede exactamente igual que en la SOLUCIÓN BÁSICA. El servidor cifra la parte uno de la columna *Sexo* utilizando como llave *consulta*. El registro que satisface nuestra consulta es el número uno; se envia $T_{(1,1)}[1]$, $T_{(1,2)}[1]$ y $T_{(1,3)}[1]$ al cliente y éste descifra el valor regresado del atributo *Nombre* con s_1 y así obtiene *Ron*.

Saliéndonos un poco de esta solución, observamos que las dos soluciones propuestas anteriormente son ineficientes. Decimos lo anterior, porque para realizar una consulta es necesario verificar cada renglón de la tabla; en nuestro ejemplo no es tan trascendental por la poca cantidad de registros, pero pensemos en una base de datos con cientos de registros. Viendo esta necesidad, Sheng Zhong et.al. [49] proponen una última solución que surge de realizar ciertas modificaciones a la solución básica para incrementar la velocidad de consulta; dicha solución fué llamada solución de METADATOS.

La solución de METADATOS reemplaza la busqueda secuencial por una búsqueda binaria[2], ya que el tiempo de búsqueda se reduce exponencialmente. Esta solución busca reducir tiempos utilizando metadatos[3] en vez de satisfacer ecuaciones.

Utilizamos metadatos ya que a cada celda le corresponden dos valores: una etiqueta y una liga, alojados en una tabla auxiliar; la etiqueta está dada por el valor de la celda y la liga es un apuntador[4] a la celda. La tabla de metadatos está de forma ordenada con respecto a la etiqueta. Cuando se genera una cosulta remota, el cliente obtiene el tag y lo envía al servidor, mismo que

[2]Al utilizar búsquedas binarias es méramente necesario que el conjunto de datos estén ordenados; este algoritmo se basa en un concepto muy utilizado en la programación: dividir para vencer. Se toma el elemento medio del vector de datos y se verifica si es el que se está buscando; en caso que no lo sea y es menor se busca en la mitad anterior de los datos, de otro modo se busca en la mitad posterior de ellos. Si se repite recursivamente el algoritmo al final o bien encontraremos el número sobre una los datos de un sólo elemento o estaremos seguros de que no se encuentra allí.

[3]Los metadatos son datos que describen a otros datos.

[4]Los apuntadores son variables que contienen direcciones de memoria.

realiza la búsqueda de forma binaria

Ilustramos el concepto anterior con un ejemplo mostrado en la Figura 3.2.

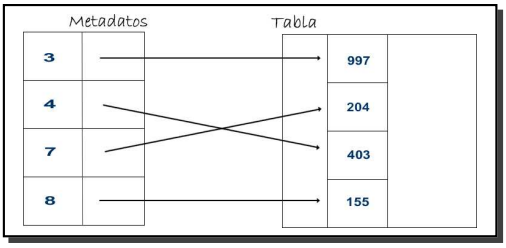

Figura 3.2: Ejemplo del uso de metadatos

No obstante, en tablas reales no podemos aplicar completamente este concepto debido a la existencia de la múltiple ocurrencia de un mismo valor; en cualquier tabla existe la posibilidad que un mismo valor o etiqueta aparezca en diferentes registros, como por ejemplo pueden existir más de una persona que tengan 23 años. Es por tal motivo que se aplica el concepto de metadatos, pero con un mapeo doble que se explica a continuación.

Se tienen dos tablas auxiliares L y B por cada atributo. L guarda un cifrado especial concatenando el valor cifrado junto con el contador actual de incidencia (iniciando en uno) y la liga que referencía a la tabla T'; existen tantos registros cifrados con el mismo valor como el número de veces que éste se repite en la tabla, diferenciando un cifrado de otro el contador de incidencia, i.e. $E_{stringA}(E_{stringB}(valor)||1)$, $E_{stringA}(E_{stringB}(valor)||2)$, $E_{stringA}(E_{stringB}(valor)||3)$, $\ldots$, $E_{stringA}(E_{stringB}(valor)||n)$, donde n es el número total de veces que aparece dicho valor en la tabla. B guarda el valor de la celda cifrado y el número total de ocurrencias del mismo valor, igualmente cifrado.

Para esta solución son necesarias tres llaves más de cifrado s_3, s_4 y s_5, que permanecen secretas en manos del cliente. Primeramente se requiere cifrar el valor con s_3 agregando ceros si es necesario para completar el tamaño de bloque de cifrado:

$$H_{i,j} = E_{s_3}(T_{ij}, 0),$$

ya teniendo el valor cifrado, comenzamos a contar las incidencias de éste en la tabla cifrando con s_4 la concatenación de H_{ij} con el número de incidencia;

si es la c_{ij}-ésima ocurrencia del valor en el atributo j se obtiene

$$I_{(i,j)} = E_{s_4}(H_{ij} + c_{ij}).$$

Cuando ya se han obtenido todas las $I\acute{}s$ de la tabla concemos el valor total de incidencias de cierto valor v. Dicho valor es cifrado con s_5 haciendo el correpondiente rellenado con ceros:

$$C_j(v) = E_{s_5}(c_j(v), 0).$$

Teniendo ya estos valores se almacenan en las tablas auxiliares de la siguiente forma

$$L = (I_{(i,j)}, \text{liga al registro } T'_i) \quad \text{y}$$

$$\begin{aligned} B = &\quad (B[1], B[2]) \\ = &\quad (E_{s_3}(v, 0), C_j(v)) \end{aligned}$$

considerando que ambas tablas deben estar ordenada por la primer columna para poder buscar los valores de forma binaria.

Ya se tienen las tres tablas necesarias para que la solución funcione. Para tener más claro, usemos la Tabla 3.2 y generemos las tablas B y L; como sabemos que por cada atributo crea B y L diferentes, usemos la columna *Sexo*. La tablas auxiliares L_{Sexo} y B_{Sexo} se muestran en las tablas 3.5 y 3.6 respectivamente.

Si se desea generar la consulta `SELECT Nombre FROM T WHERE Sexo='M'`, primeramente ciframos el valor concatenado con ceros usando s_3 y se envia la cadena al servidor, i.e. $h = E_{s_3}(M, 0)$. El servidor toma la cadena y realiza una búsqueda binaria en B_{Sexo}, al encontrarla regresa el valor en la columna incidencias; para nuestro ejemplo, la cadena h se encuentra en el primer registro y se retorna al cliente el número de incidencias cifrado: $E_{s_5}(2, 0)$. El cliente obtiene el contador de incidencias totales, lo descifra con s_5, $D_{s_5}(E_{s_5}(2, 0))$ quedando (2,0), y le quita los ceros que se le habían agregado; así obtenemos el valor 2. Teniendo el número de incidencias de ese valor, el cliente computa

I	Liga
aeml2f = $E_{s_4}((E_{s_3}(F,0)+2))$	registro 3
b6d2s6 = $E_{s_4}((E_{s_3}(F,0)+1))$	registro 2
plrlo9 = $E_{s_4}((E_{s_3}(M,0)+1))$	registro 1
zol78v = $E_{s_4}((E_{s_3}(M,0)+2))$	registro 4

Tabla 3.5: L_{Sexo}

Valor	Incidencias
$E_{s_3}(M,0)$	$E_{s_5}(2,0)$
$E_{s_3}(F,0)$	$E_{s_5}(2,0)$

Tabla 3.6: B_{Sexo}

los dos valores de I de la siguiente manera: $I_1 = E_{s_4}(h+1)$ e $I_2 = E_{s_4}(h+2)$; mismos que son enviados al servidor que los busca de forma binaria en L_{Sexo}; al encontrar los valores, el servidor verifica la liga a la tabla T' y retorna los registro completos; en nuestro ejemplo, el servidor encuentra los valores de I en el tercer y cuarto registro de L_{Sexo} mismos que apuntan a los registros de T' uno y cuatro respectivamente. Finalmente el servidor retorna los registros uno y cuatro al cliente. El cliente toma los valores de la columna *Nombre* y los descifra con s_1, obteniendo *Ron* y *Jon* como respuesta.

El esquema presentado [49] se enfoca principalmente en la seguridad de los datos bajo el contexto DaaS [21]. Su cometido es cumplido satisfactoriamente al mostrar un esquema que proporciona suficiente seguridad a la base de datos y a las consultas en ella debido a su robustés criptográfica.

A pesar que la seguridad es muy buena, tiene bastantes desventajas el esquema. En primer lugar, se procupa únicamente por la sobrecarga del procesamiento computacional, y no toma en cuenta la sobrecarga en la conexión de la red; i.e. asume que la concexión entre el cliente y servidor es un enlace dedicado sin fallas de conexión, y omite el hecho que el servidor es remoto comunicándose con el vía Internet y que la conexión puede fallar o retardarse por el tráfico. Aunado a esto, por el uso de la criptografía el servidor no rea-

liza las búsquedas con el motor del SMBD y la forma en que las búsquedas se realizan son ineficientes. Además de que para generar una consulta se requiere que el servidor haga ciertas operaciones criptográficas para lograr encontrar los registros buscados, obligando a tener una aplicación en el servidor que haga todas las operaciones; tomando esto en cuenta, bajo este esquema no sólo debemos de preocuparnos por la BD, sino tambien por nuestra aplicación ya que si ésta es modificada o removida nuestra base de datos queda inservible. A pesar que en la tercera solución, solución de METADATOS, se preocupan por la eficiencia en las consultas, quedan muy por debajo del grado de eficiencia requerido en un caso real al realizar el *hosting* de una BD; las consultas en esta tercera solución, aunque disminuye tiempos, resta eficiencia de otra forma ya que cada consulta es traducida en dos o tres consultas al servidor, sin mencionar los cifrados entre una consulta y otra que se realizan a cargo del cliente; otro aspecto negativo de esta tercera solución es, que para cada atributo de la tabla se requieren dos tablas auxiliares incrementando el tamaño de datos a almacenar, porque ahora no sólo guardamos la BD y la aplicación que ayuda a la generación de consultas, sino tambien tantas tablas extras como el doble de atributos de la tabla. Finalmente, este esquema maneja consultas limitadas debido a que responde únicamente a consultas de igualdad, dejando de lado toda la amplia gama de posibles consultas que pueden ser realizadas para la explotación de los datos.

Posiblemente la base del esquema anterior [49] fue el modelo propuesto por Dawn Xiaodong Song et.al. [41] en el 2000. Dicho artículo [41] realiza búsquedas de palabras en documentos cifrados, igualmente bajo el contexto de *hosting* de datos. A pesar que es muy similar en el fondo al esquema anterior, no lo es en la forma. La manera en que se cifran las palabras es de forma sencilla; ya que el tamaño de la palabra W es m, se genera una cadena aleatoria S de tamaño n, donde $n < m$, y se cifra de la siguiente forma $C = W \oplus S_i||f(S_i)$, f es una función que recibe como parámetros S (el tamaño de $f(S_i) = m - n$). La peculiaridad que este esquema posee es que ofrece un gran número de opciones y soluciones para buscar palabras en el texto, que van desde la búsqueda secuencial hasta búsquedas indexadas, pero cada que propone una solución enseguida la descarta por ineficiencia, falta de seguridad o no funcionalidad. Se menciona en el artículo que no sólo responde a consultas de igualdad, sino también a aquellas lógicas y de cercanía, lo malo es que no explica a profundidad cómo pueden ser realizadas. Al parecer este esquema propone un gran número de ideas para posteriores

investigaciones. La ventaja que se observa es la gran variedad de soluciones y consejos que ofrece al lector como posibles ideas de futuras investigaciones; lo malo de él, es que no ofrece ni una sola solución convencidos de que puede ser un esquema a utilizar.

En 1981 George I. Davida y David L. Wells [13] muestran un esquema que basa el cifrado de la base de datos utilizando como herramienta principal el uso de los números primos. Posiblemente es de las primeras investigaciones en el uso de la criptogafía como medio para mantener seguras las bases de datos. Este artículo da una explicación acerca de las bondades que trae el usar la criptografía en las bases de datos, para fines de seguridad; además propone un esquema basado en el Teorema del Residuo Chino [16] para cifrar la base de datos y usar subllaves para lograr descifrar independientemente cualquier bloque sin necesidad de descifrar todo; por otra parte, utiliza *MACs* (*Message Authentication Codes*) para conocer si los datos han sido vulnerados. Este artículo a pesar de su temprano surgimiento, menciona gran parte de los problemas que deben ser atacados para lograr el uso de la criptografía en la base de datos: menciona la autenticidad de los datos, el dinamismo constante de la base de datos y la seguridad requerida, a cargo de la criptografía. Las desventajas que vemos en él es que no habla acerca de las consultas que pueden ser realizadas bajo este esquema; y a pesar de que menciona la seguridad y autenticidad de los datos, no considera que la base de datos esté remota ni a cargo de un tercero, ya que no toca el tema de la eficiencia en consultas y no puede ser claramente dividido el esquema en las dos entidades cliente-servidor.

Posteriormente en el 2001, Min Wang et.al. [24] surgen con un esquema de seguridad en Bases de Datos Relacionales que estriban la seguidad de su modelo al usar un *diccionario de seguridad*; este concepto es propuesto por ellos diferenciándolo del diccionario de datos ordinal agregándole ciertas restricciones, tales como: i) éste no puede ser actualizado de forma manual por ningún usuario, ii) el acceso a él es controlado por estricta autenticación y politicas de autorización. En el diccionario de seguridad se tienen datos importantes de la base de datos como las tablas, las vistas y los usuarios junto con sus contraseñas, para realizar cualquier tipo de transacción en la base de datos el usuario requiere pasar por mecanismos de autenticación mediante contraseñas. Este esquema usa el diccionario de seguridad como base para poder inyectarle confidencialidad al esquema de la siguiente manera: de todo

el conjunto de atributos de cierta tabla, se selecciona a los que son excesivamente vulnerables y sólo ellos son cifrados; para cifrarlos se utiliza como llave la contraseña del usuario que está generando la alta del registro; así cuando un usuario requiere hacer una consulta, se genera la consulta en el resto de los atributos que quedaron intactos, sin cifrar, y se recupera el valor cifrado al descifrar utilizando como llave la contraseña. Este esquema presenta una propuesta que se enfoca en la autenticiadad mediante contraseñas, las consultas en este esquema son súmamente rápidas al tener la mayoria de campos en texto claro y por ende se puede generar la consulta utilizando el motor de búsqueda del Sistema Manejador de base de datos. Las deficiencias que saltan a la vista de este esquema son, en primer lugar, no todos los atributos de la tabla son ocultos, por otra parte si el usuario pierde su contraseña no puede recuperar los datos completamente y finalmente, recarga toda la seguridad al *diccionario de seguridad* sin considerar que no solamente puede ser modificado, sino también eliminado o cambiado.

Por otra parte, Ernesto Damiani et.al. [6] en el 2005 presenta un esquema afinando su investigación anterior [10], el cual realiza cierta indexación de los datos en la base de datos cifrada. Cada tabla de n atributos en texto en claro es reflejada en el servidor en dos tablas, la primera es un simple cifrado del valor de la tabla en texto claro, la segunda contiene un indexado directo a la primer tabla mediante un *hashing* de los índices; i.e. dos o más diferentes valores les corresponde un mismo índice, por ejemplo: si tenemos una tabla con el atributo *nombre* y para el valor *Juan* le corresponde el índice x, y por otra parte al valor *Martha* le corresponde también el índice x; al generar la consulta si deseamos regresar el registro de *Juan* cuyo índice corresponde a x, se retorna el registro de *Juan* y el de *Martha*, posteriormente en el cliente se descifra y podemos depurar el registro que en realidad se buscaba. Los beneficios que este esquema presume es, primeramente que las consultas son eficientes por el *hashing* de los datos y por otra parte son seguras al mantenerlas cifradas. Las deficiencias que encontramos en el esquema presentado son bastantes: i) no identifica el tipo de consultas a las que responde, por intuición deducimos que son solamente de igualdad, ya que propone como trabajo a futuro la búsquedad de consultas funcionales en el esquema ii)las consultas no son eficientes como se menciona, ya que el cliente requiere hacer una depuración de los datos que se le retornan, iii) la búsqueda no puede ser generada por el motor de búsqueda del servidor de la base de datos, iv) no hay una autenticación de los datos en el servidor.

Son diversos las herramientas que se utilizan para crear un esquema criptográfico robusto y seguro; algunos investigadores han girado su enfoque a utilizar homomorfismos al cifrar los datos para ganar terreno en la gama de consultas. Tal es el caso de Hakan Hacigümus et.al. [20] que en el 2004 logran perfeccionar su previo esquema de cifrado homomórfico [19] propuesto en el 2002. Este esquema resultante, se enfoca principalmente al tratamiento de los atributos a los que se les pueden aplicar operaciones aritméticas como sumas, restas y multiplicaciones para lograr generar consultas de agregación en la base de datos cifrada. Para cifrar la tabla, es necesario clasificar los atributos de acuerdo al tipo que corresponden dentro de las cuatro diferentes clasificaciones propuestas, cada tipo de atributo tiene un tratamiento diferente al resto de las demás, por ejemplo los datos como llaves primarias, nombres, etc. no pueden ser consultados; la tabla cifrada que resulta posee al menos dos columnas por cada atributo en texto en claro de la tabla original, y un atributo que concatena todo el registro y lo cifra. Como ya mencionamos cada tipo de atributo es tratado de forma distinta para que al generar una consulta, la aplicación en el cliente identifique el tipo de atributo que se está consultando y genere la consulta adecuada. Los beneficios que aporta este artículo estriban principalmente en la amplia gama de posibles consultas a realizar, debido a que se generan conultas de agregación, pueden consultarse promedios, sumas, máximos y mínimos en los datos. Sin embargo, las desventajas que presenta es: i) no se pueden generar consultas a todos los tipos de datos, ii) no se preocupa por la autenticación de los datos, iii) hablando en términos de eficiencia, para verificar que el valor de la consulta es el mismo en la base de datos cifrada se requieren de diversas operaciones matemáticas en el servidor, y posteriormente se debe de descifrar el dato retornado en el cliente; además que se debe de verificar el tipo de atributo a consultar para traducir de forma correcta la consulta, iv) la tabla cifrada resultante contiene muchos más atributos que la original, i.e. crece de forma horizontal.

Siguiendo con los homomorfismos, en el 2006 Gultekin Ozsoyoglu et.al. [35] muestran una investigación más robusta del uso de los homomorfismos para generar consultas a Bases de Datos cifradas. La idea principal de este artículo es enfocarse únicamente en los atributos numéricos robusteciendo las consultas y la seguridad en este tipo de atributos; muestran una familia de funciones de cifrado-descifrado homomórficas de forma abierta y cerrada. Este artículo enriquece el anterior al proponer más opciones del tratamiento

de datos enteros; las desventajas que vemos en él, es que deja de lado el resto de tipos de atributos en una tabla. Así, podemos ver este artículo no como un nuevo esquema, sino como un complemento del anterior.

Cambiando un poco el enfoque de la criptografía para la seguridad en Bases de Datos, han surgido otras investigaciones que buscan otras opciones para lograr seguridad y eficiencia en las consultas. Uno de ellos es el propuesto por Radek Vingralek [45] en el 2002 que propone una arquitectura que preserva la seguridad en los datos y de igual forma se preocupa por cuidar los recursos de la computadora, ya que es pensado para dispositivos móviles como teléfonos, palms, etc. que tienen resursos limitados. Esta arquitectura protege los datos contra ataques accidentales e intencionales, los primeros al generar actualizaciones constantes de los datos y los segundos mediante el uso de la criptografía; además de utilizar autenticación de los datos cifrados. Este artículo peculiarmente trata de atacar un problema más robusto que el nuestro, ya que busca que el esquema ocupe el mínimo de recursos. No obstante, debido a su limitante de recusros es bastante escueta su seguridad, y no menciona la manera en que las ocnsultas son generadas ni a qué tipo de consultas responde.

Siguiendo sobre la línea de los dispositivos y en el mismo año, surgió una investigación que conjunta el hardware y el software para atacar el problema de la seguridad en Bases de Datos bajo el contexto DaaS [21], ésta fue propuesta por Luc Bouganim et.al. [5]. Este artículo propone una arquitectura de hardware y sofware para dar seguridad a los datos; utiliza una tarjeta inteligente en la que desarrolla una aplicación para cada persona o usuario de la base de datos, esta arquitectura funge como el traductor de consultas o front end requerido en la gran mayoría de esquemas en nuestro estudio, i.e. el intermediario entre el cliente y el servidor. Los datos que son méramente sensibles son almacenados en la tarjeta y están cifrados de tal forma que nadie más pueda obtenerlos, mas que el dueño de la tarjeta, ya que las llaves de cifrado son ocultas hasta para el usuario de la misma tarjeta. Los beneficios que trae este esquema es, que la aplicación usuario se desentiende de la traducción de consultas y solamente el usuario poseedor de la tarjeta puede tener acceso a sus datos. Sin embargo, esta arquitectura es bastante costosa ya que se requiere que cada usuario cuente con su tarjeta y ésta sea programada y personalizada previamente; además, si el usuario llega a extraviar su tarjeta, cualquier persona puede hacer uso de ella y obtener sus

datos sin ningún problema; y finalmente no creemos necesario exportar la aplicación del traductor de consultas a una tarjeta.

Finalmente hablemos de otra línea de investigación del mismo problema; usando la fragmentación de datos[5]. En el 2005 Gagan Aggarwal et.al. [1] proponen un esquema innovador el cual utiliza la fragmentación de la base de datos en dos servidores remotos, no sólo en uno, con la condición que estos no tengan comunicación entre ellos. La idea principal del esquema es dividir la tabla en los dos servidores, usando fragmentación vertical de la tabla. Donde las llaves de cifrado pueden ir en ambos servidores y ciertos atributos pueden ser replicados si así se requiere en ambas entidades. Debido a que los servidores no se comunican entre ellos, algunos de los atributos pueden ir en texto en claro y así eficientar las consultas y tener una gama más amplia de ellas que los esquemas de cifrado total de la tabla. Los beneficios de este esquema son significativos, en primer lugar la libertad de consultas es más amplia al poseer datos en texto en claro y la eficiencia es bastante al generar consultas en las base de datos ya que se puede utilizar el motor de búsqueda del Sistema Manejador de Bases de Datos. Por otra parte, las desventajas que se muestran son: i) funciona bajo un ambiente utópico en el que los servidores no pueden tener ningún tipo de comunicación, ii) para recabar la información de una consulta es necesario consultar al menos dos veces, una vez a cada servidor para obtener los datos completos.

Posteriormente y con base en el artículo anterior, surgió un esquema robusto que hace uso del cifrado y fragmentación de los datos para generar consultas eficientes y seguras bajo un mismo servidor. Dicho esquema fue realizado por Valentina Ciriani et.al. [7] y es la base principal del estudio de la tesis, por tal motivo se explicará a detalle en el siguiente capítulo.

Como podemos observar a lo largo de este breve recorrido de los trabajo realizados en nuestro caso de estudio, podemos concluir que la eficiencia y la seguridad de los esquemas vistos representan a una balanza; si existe mayor seguridad, la eficiencia de consultas disminuye proporcionalmente. Lo que deseamos es encontrar la medida exacta de ambos ingredientes para

[5]La fragmentación es una técnica que parte del principio de *divide y vencerás* y es usada para eficientar búsquedas en datos, ya que también ha sido usada por Angela Bonifati et.al. [4] en el 2007 para realizar consultas en documentos XML.

así obtener un esquema completo y competitivo en ambos aspectos.

4

Procesamiento Eficiente de Consultas y Autenticación en Bases de Datos

En el capítulo anterior se mostró una gran cantidad de investigaciones, que concluimos que no se ha encontrado un esquema de cifrado con la combinación perfecta entre eficiencia y seguridad. En este capítulo mostramos nuestra propuesta, basada en la de Valentina Ciriani et.al. [7], preocupándonos no sólo por la integridad los datos, sino también por la autenticación de ellos. Además, y gracias a que se desarrolló la implementación del esquema [7], encontramos ciertas deficiencias en él que en nuestra propuesta son corregidas y detalladas.

Este esquema a diferencia de la gran mayoría no cifra la tabla por completo para mantenerla segura; sino que aisla los datos, evitando la denotación de información[1]. Bajo este principio el esquema logra la seguridad separando los

[1]Recordando que la relación existente entre dato e información podemos apreciarla perfectamente desde sus definiciones: dato es un ente aislado que por si sólo no nos dice

datos y obteniendo grandes beneficios; uno de ellos, si no el más importante, es que los datos al estar separados sin expresar información valiosa pueden permanecer en texto claro no habiendo problema alguno, así logrando eficientar las consultas generadas en ellos.

No obstante, la separación de los datos no puede realizarse sin seguir algún patrón de seguridad. La separación de los datos debe obligadamente obedecer a los requerimientos de seguridad que el poseedor de la base de datos considere necesarios para su información. Nadie mejor que el dueño de la base de datos, puede identificar qué datos deben ser forzosamente separados para salvaguardar la integridad de su información.

Es por ello que debemos aterrizar los requerimientos de seguridad para la base de datos en pequeñas expresiones que llamaremos Restricciones de Confidencialidad. Las restricciones de confidencialidad son sentencias impuestas por el poseedor de la base de datos que modelan los requerimientos de privacidad en la información. En la siguiente sección se definen propiamente estas restricciones.

Otro aspecto importante que nuestro esquema considera, es el hecho de que nuestros datos no por estar cifrados implica que están completamente a salvo. Ya que el servidor puede modificar intencional o accidentalmente los datos; cosa que repercute directamente en las consultas, ya que nos topamos con la incertidumbre de si nuestra información es verdaderamente fidedigna. Es por tal motivo que nos preocupamos no solamente de la seguridad de los datos, sino también de la autenticación de ellos. Obtenemos este beneficio al utilizar un esquema de cifrado de bloques que ofrece autenticar el mensaje, además de proporcionar seguridad a los datos.

4.1 Restricciones de confidencialidad

Las restricciones de confidencialidad son un conjunto de atributos que formalmente se definen de la siguiente manera:

---

nada y un conjunto de ellos nos manifiestan información.

Definición. *Llamemos A a un conjunto de atributos, una* ***restricción de confidencialidad*** *es un subconjunto $c \subseteq A$.*

Interpretamos a las restricciones de confidencialidad como un conjunto de atributos que no deben permanecer juntos. Existen dos tipos de restricciones de confidencialidad: las que poseen dos o más atributos y aquellas cuya cardinalidad es uno. La diferencia en semántica se muestra enseguida.

- **Restricciones de un único atributo**. Existen ciertos atributos que por sí sólos deben ser protegidos debido a que el simple dato es confidencial, a estos atributos les denominamos sensibles; como ejemplos de este tipo de atributos tenemos el número de cuentas bancarias, direcciones, cuentas de correos, etc. Las restricciones de un sólo atributo guardan justamente estos datos sensibles.
- **Restricciones de más de un atributo**. Indican que la asociación entre estos atributos es sensible, y por tal motivo, no pueden permanecer juntos; e.g. los nombres de los empleados de alguna organización y los sueldos de estos mismos.

Por otra parte, las restricciones deben estar bien definidas; es decir, no cualquier conjunto de restricciones de confidencialidad en una base de datos es válida.

Definición. *Un* ***conjunto de restricciones de confidencialidad*** $R = \{r_1, r_2, \ldots, r_n\}$ *está* ***bien definido*** *si* $\forall r_i, r_j \in R, i \neq j, r_i \not\subset r_j$ *y* $r_j \not\subset r_i$.

Una restricción está bien definida si no es subconjunto de otra, es decir si una restricción no contiene a ninguna otra. Esto se busca para evitar redundancias, ya que si una restricción a que contiene a otra b se satisface, al satisfacer a inmediatamente b está satisfecha. La forma de verificar si una restricción se satisface se muestra en la siguiente sección.

Para entender mejor cómo deben ser generadas las restricciones, tenemos un ejemplo. La Fig. 4.1 muestra una tabla del departamento de recursos humanos de cierta organización junto con sus restricciones de confidencialidad donde:

- El número de empleado, `NumEmp` por sí sólo es un dato sensible (r_0).
- No deseamos que `Nombre` esté relacionado con ningún otro atributo (r_1, r_2, r_3, r_4). El nombre de una persona nos denota identidad y si a ello le relacionamos cualquier atributo, ofrecemos muchísima información.
- La fecha de nacimiento (`FechaNac`) y el código postal (`CP`) juntos pueden denotar la identidad de alguna persona, así es que reciben el mismo tratamiento que el nombre (r_5, r_6).

RECURSOS HUMANOS

NumEmp	Nombre	FechaNac	CP	Sueldo	Horario
1234	Miriam	12/08/79	5555	11000	Matutino
1111	Felipe	01/08/89	0989	1500	Nocturno
2222	Román	25/04/86	0098	12000	Nocturno
3333	Erick	12/02/86	1265	11000	Vespertino
4444	Federico	15/04/90	6789	1500	Nocturno
5555	Ruth	23/05/85	5423	1500	Matutino

r_0= {NumEmp}
r_1= {Nombre, FechaNac}
r_2= {Nombre, CP}
r_3 = {Nombre, Sueldo}
r_4 = {Nombre, Horario}
r_5 = {FechaNac, CP, Sueldo}
r_6 = {FechaNac, CP, Horario}

Figura 4.1: Ejemplo de una tabla en texto claro y a la derecha sus restricciones de confidencialidad

Como bien se menciona, el nombre del empleado no debe ir relacionado con ningún otro atributo; y si bien se observa, es el mismo caso para el atributo `Nombre` relacionado con `NumEmp`. En este caso no es necesario generar una restricción con estos dos atributos, ya que `NumEmp` es el único elemento de r_0 y sería redundante generar una restricción para `Nombre` y `NumEmp`. Como regla, podemos decir que si un atributo es catalogado como sensible, sus relaciones con cualquiera de los atributos quedan protegidas automáticamente.

4.2 Esquema de cifrado y fragmentación

Como hemos estado diciendo nuestro esquema utiliza por una parte, fragmentación de los datos y por otro lado cifrado de los mismos para protegerlos y autenticarlos.

Utilizamos el cifrado de los datos para dos aspectos: i) mantener ilegible el registro completo y ii) autenticar estos datos. Entendemos por fragmentar

a dividir los atributos con el fin de satisfacer las restricciones de confidencialidad, y así puedan estar en texto claro sin denotarnos información sensible. A las entidades que contienen un subconjunto de atributos y los mantienen separados les denominamos fragmentos, y al conjunto total de fragmentos la llamaremos fragmentación.

Definición. *Sea RA una tabla con esquema relacional dado por un conjunto de atributos, una* ***fragmentación*** *de RA es una colección* $\mathcal{F}= \{F_1, \ldots, F_m\}$, *donde cada* F_i *es un subesquema de RA, para* $i = 1, \ldots, m$.

Hablando en términos de bases de datos, físicamente un fragmento resulta en una tabla que aloja atributos en claro y una estructura de cifrado que más adelante se explicará a detalle.

Definición. *Dado que RA es una relación con esquema relacional de atributos y* $\mathcal{F}= \{F_1, \ldots, F_m\}$ *una fragmentación de RA. Para cada* $F_i = \{a_{i_1}, \ldots, a_{i_n}\} \in \mathcal{F}$, *el* ***fragmento físico*** *es una relación* $F_i = \{nonce, enc, tag, a_{i_1} \ldots, a_{i_n}\}$.

Como observamos en la definición anterior, cada fragmento ó tabla consta de tres columnas básicas `nonce`, `enc` y `nonce`, además de los atributos en claro pertenecientes al fragmento. Cada una de estas columnas desempeña cierto rol importante en nuestro esquema: la columna `enc` es un cifrado de los atributos que no pertenecen al fragmento haciendo previamente un Xor($\oplus$) con el `nonce`, dicha columna se utiliza para lograr tener el registro completo y consultar sin problema alguno la tabla; finalmente la columna `tag` autentica, ésta nos permite verificar que la parte cifrada de la tabla, la columna `enc`, no ha sido modificada en el servidor.

En primera instancia hablemos de la obtención de la tercera y primer columna `enc` y `salt`. La columna `enc` es un cifrado de la concatenación del resto de atributos que no pertenecen al fragmento en curso, es decir el total de atributos menos los atributos del fragmento a tratar: $concatenacion = RA - F_i$, sumada con el *nonce* mediante un XOR. La concatenación de los atributos atiende a un mecanismo que más adelante se explicará.

$$t[enc] = E_k(t[concatenacion] \oplus t[nonce])$$

Por otra parte, hablemos de la columna `nonce`. El *nonce* en términos de criptografía es una cadena que no puede ser utilizada en más de una vez al cifrar; en primera instancia habíamos pensado en utilizar una cadena aleatoria para generar nuestra columna `nonce`; posteriormente nos percatamos que utilizar números aleatorios es muy complicado, ya que deben ser memorizadas todas las cadenas aleatorias previamente usadas para no repetir ninguna. Por tal motivo, decidimos que el *nonce* debía ser un contador creciente cifrado; el *nonce* se cifra debido a que si permanece en texto claro en la tabla, y si se agrega un nuevo registro, es fácil reconocer que registro fué agregado relacionando los últimos valores de la columa `nonce` en los diferentes fragmentos.

Así:

$$\begin{array}{lcl} t[nonce] & = & E_{k_1}(t[contador]) \\ t[enc] & = & E_{k_2}(t[concatenacion] \oplus t[nonce]) \end{array} \text{ ,}$$

nótese que la llave utilizada para cifrar los *nonces* es diferente a la que se utiliza para cifrar la columna `enc`.

El mecanismo de concatenación que utilizamos se realiza para lograr eficientar el parseo de los datos al momento de generarse una consulta y mostrar los valores de cada columna de forma separada. Para ello es necesario tener en el cliente una tabla auxiliar, llamémosla `Longitud Atributos`, que son los metadatos de la tabla en texto claro. Esta tabla contiene dos columnas: la primera guarda el nombre de todos los atributos de la tabla y la segunda guarda la máxima longitud del atributo en codificación binaria.

Como ejemplo consideremos la tabla en texto en claro mostrada en la Fig. 4.1, donde `NumEmp` es de tipo caracter de longitud 10, el número 10 en binario puede representarse como 1010 es decir 4 bits, `Nombre` es de 30 caracteres (5 bits), `FechaNac` ocupa 8 caracteres (4 bits), `CP` es de 4 caracteres (3 bits), `Sueldo` es de 5 caracteres (3 bits) y finalmente Horario es de 15 caracteres (4 bits). La tabla `Longitud Atributos` es mostrada en la tabla 4.1.

Ya teniendo la tabla `Longitud Atributos` se concatena de la siguiente manera. En primer lugar se identifican los atributos de a_i a concatenar. Postriormente se cuenta el número de bits necesarios que ocupa el valor de cada uno de los atributo y se representan en los n bits que le corresponden en la

Atributo	Longitud
NumEmp	4
Nombre	5
FechaNac	4
CP	3
Sueldo	3
Horario	4

Tabla 4.1: Tabla `Longitud Atributos` de la tabla mostrada en la Fig. 4.1.

tabla `Longitud Atributos`, así para cada registro. Es decir,

Definición. *$l(a_i)$ es la longitud en bits del tamaño de (a_i); $t[a_{i_1}], t[a_{i_2}], t[a_{i_3}]$ son los atributos a concatenar. Así,*

$$\begin{aligned} L &= bin_{l(a_{i_1})} len(t[a_{i_1}]) || bin_{l(a_{i_2})} len(t[a_{i_2}]) || bin_{l(a_{i_3})} len(t[a_{i_3}]) \\ &\text{y} \\ concatenacion &= L || t[a_{i_1}] || t[a_{i_2}] || t[a_{i_3}]. \end{aligned}$$

Por fines didácticos mostramos la concatenación de todos los atributos del primer registro mostrado en la Fig. 4.1. Primeramente identificamos el tamaño de cada valor y lo representamos en una cadena binaria cuya longitud está dada en la tabla `Longitud Atributos`. En la siguiente tabla mostramos el atributo, la longitud en bits, el valor, el tamaño contra la longitud máxima en caracteres y la cadena en bits generada.

atributo	longitud en bits	valor	tamaño	cadena
NumEmp	4	1234	4/10	0100
Nombre	5	Miriam	6/30	00110
FechaNac	4	12/08/79	8/8	1000
CP	3	5555	4/4	100
Sueldo	3	11000	5/5	101
Horario	4	Matutino	8/15	1000

Ya teniendo las longitudes de los valores en bits, podemos generar la cadena concatenación:

$$
\begin{aligned}
L &= 0100 \,||\, 00110 \,||\, 1000 \,||\, 100 \,||\, 101 \,||\, 1000\ , \\
concatenacion &= L \,||\, 1234 \,||\, \text{Miriam} \,||\, 12/08/79 \,||\, 5555 \,||\, 11000 \,||\, \text{Matutino} \\
&= 010000110100010010110001234\text{Miriam}12/08/795555511000\text{Matutino}
\end{aligned}
$$

Debido a que la concatenación de los atributos puede llegar a exceeder el tamaño de bloque de cifrado, decidimos cifrar *concatenación* con AES bajo el modo de operación OCB1 utilizando al *nonce* como vector de inicialización.

Gracias a que OCB1 permite autenticar los datos, la columna `tag` guarda el Tag obtenido por OCB1, cuyo procedimiento fue presentado en el capítulo 2 de la presente tesis.

La autenticación de nuestro esquema es un poco limitado debido a que solamente se autentican las columnas `nonce`, `enc` y `tag`; es decir las columnas que tienen los datos en texto claro no se autentican. Los atributos que aparecen en texto claro en un fragmento físico participan para la obtención del atributo `tag` del mismo registro. Para realizar esto utilizamos los esquemas AEAD (*Authenticated Encryption with Associated Data*) [38]; i.e. dada una fragmentación dada $F_i = \{a_1, a_2, \ldots, a_n\}$, estos atributos aparecen en texto claro, pero sólo el resto $A \setminus F_i$ son cifrados y autenticados. Así podemos considerar a los atributos $A \setminus F_i$ como los datos asociados.

Así, teniendo un fragmento F_i creamos un fragmento físico F_i^{aead} como sigue:

- $t^{aead}[nonce] \leftarrow GenSalt(K_1)$;
- $t^{aead}[a_{i_j}] \leftarrow t[a_{i_j}]$;
- $t^{aead}[enc] \leftarrow Cifra_{K_2}(t^e[nonce]; t[A \setminus F_i])$;
- $t^{aead}[tag] \leftarrow Cifra_{K_2}(t^e[nonce]; t[A \setminus F_i]) \oplus MAC_{K_3}(t^e[nonce; t[F_i]])$;

Ya teniendo cifrados los tres valores de las columnas, es necesario convertirlos a Base 64 para que puedan ser ingresados en la tabla.

Ya teniendo clara la estructura de un fragmento y lo que éste representa en nuestro esquema, es necesario indicar las restricciones que surgen para

generar un fragmento; debido a que nuestro principal objetivo es crear fragmentos que satisfagan las restricciones de confidencialidad. De igual manera que las restricciones de confidencialidad los fragmentos deben estar bien definidos.

Dada una relación de atributos RA, una fragmentación F y un conjunto de restricciones de confidencialidad R bien definido sobre RA. F hace cumplir correctamente R si:

1. $\forall F \in \mathcal{F}$, $\forall r \in R : r \not\subseteq F$ *(cada único fragmento satisface las restricciones);*
2. $\forall F_i, F_j \in \mathcal{F}$, $i \neq j : F_i \cap F_j = \emptyset$ *(los fragmentos no tienen atributos en común).*

Es decir, para determinar si un fragmento está bien definido no debe tener como subconjunto a ninguna de las restricciones de confidencialidad, esto implica que los atributos que pertenecen a restricciones de un sólo atributo no pueden aparecer en ningún fragmento; además de que ningún atributo debe aparecer en más de un fragmento; es decir, cada atributo sólo pertenece a sólo un fragmento.

El problema de generar la fragmentación correctas es bastante robusto. El procedimiento de la obtención de los fragmentos corresponde a un problema de complejidad NP-difícil ya que puede verse como el problema de coloración de hipergrafos. Por esta razón el algoritmo de generación de fragmentos que utilizamos se muestra en el siguiente capítulo.

Para concluir esta sección, en la Fig. 4.2 mostremos un ejemplo de fragmentación de la relación de atributos RA mostrada en la Fig. 4.1 haciendo uso de la tabla 4.1.

4.3 Protocolo de consultas

Para generar consultas debemos de tener claro que solamente se consultan aquellos atributos que aparecen en algún fragmento; es decir, no fueron

F_1

nonce	enc	tag	Nombre
$E_{k2}(1)$	E_{k1} ((0100100010010110001234 12/08/795555 11000Matutino)⊕ $E_{k2}(1)$)	ώ	Miriam
$E_{k2}(2)$	E_{k1} ((0100100010010001111101/08/890989 1500Nocturno)⊕ $E_{k2}(2)$)	Ξ	Felipe
$E_{k2}(3)$	E_{k1} ((0100100010010110002222225/04/860098 12000Nocturno)⊕ $E_{k2}(3)$)	έ	Román
$E_{k2}(4)$	E_{k1} ((0100100010010110103333312/02/861265 11000Vespertino)⊕ $E_{k2}(4)$)	ά	Erick
$E_{k2}(5)$	E_{k1} ((0100100010010010004441 5/04/906789 1500Nocturno)⊕ $E_{k2}(5)$)	Θ	Federico
$E_{k2}(6)$	E_{k1} ((0100100010010010005555 23/05/855423 1500Matutino)⊕ $E_{k2}(6)$)	Ψ	Ruth

F_2

nonce	enc	tag	FechaNac	CP
$E_{k2}(7)$	E_{k1} ((0100001101011000 1234Miriam 11000Matutino)⊕ $E_{k2}(7)$)	Φ	12/08/79	5555
$E_{k2}(8)$	E_{k1} ((0100001101001000 1111Felipe 1500Nocturno)⊕ $E_{k2}(8)$)	Д	01/08/89	0989
$E_{k2}(9)$	E_{k1} ((0100001011011000 2222Román 12000Nocturno)⊕ $E_{k2}(9)$)	Ж	25/04/86	0098
$E_{k2}(10)$	E_{k1} ((0100001011011010 3333Erick 11000Vespertino)⊕ $E_{k2}(10)$)	Ю	12/02/86	1265
$E_{k2}(11)$	E_{k1} ((0100010001001000 444Federiico 1500Nocturno)⊕ $E_{k2}(11)$)	П	15/04/90	6789
$E_{k2}(12)$	E_{k1} ((0100001001001000 5555Ruth 1500Matutino)⊕ $E_{k2}(12)$)	Б	23/05/85	5423

F_3

nonce	enc	tag	Sueldo	Horario
$E_{k2}(13)$	E_{k1} ((0100001101000100 1234Miriam 12/08/795555)⊕ $E_{k2}(13)$)	Ъ	11000	Matutino
$E_{k2}(14)$	E_{k1} ((0100001101000100 1111Felipe 01/08/890989)⊕ $E_{k2}(14)$)	Ю	1500	Nocturno
$E_{k2}(15)$	E_{k1} ((0100001011000100 2222Román 25/04/860098)⊕ $E_{k2}(15)$)	Ъ	12000	Nocturno
$E_{k2}(16)$	E_{k1} ((0100001011000100 3333Erick 12/02/861265)⊕ $E_{k2}(16)$)	Щ	11000	Vespertino
$E_{k2}(17)$	E_{k1} ((0100010001000100 444Federico 15/04/906789)⊕ $E_{k2}(17)$)	Х	1500	Nocturno
$E_{k2}(18)$	E_{k1} ((0100001001000100 5555Ruth 23/05/855423)⊕ $E_{k2}(18)$)	Т	1500	Matutino

Figura 4.2: Un ejemplo de fragmentos físicos de la relación de atributos Fig. 4.1

considerados como datos sensibles. Puede existir la posibilidad de generar consultas a los datos sensibles, pero estaríamos perdiendo el terreno ganado en eficiencia al generar consultas en datos en texto claro. Gracias a que las consultas son en datos en texto claro, es posible utilizar el motor de búsqueda del Sistema Manejador de Bases de Datos, en nuestro caso Postgres; así, las consultas son muy rápidas gracias a la búsqueda de árbol B[2] que este manejador utiliza.

El utilizar fragmentación de los datos, en vez de cifrar por completo la base de datos, trae gran cantidad de beneficios. Uno de ellos es que la comunicación entre el cliente y el servidor es en tiempo real, el cliente no debe esperar en línea grandes periodos de tiempo mientras que el servidor realiza funciones criptográficas para obtener la consulta; nuestro esquema considera que pueden existir fallas en la red, aspecto que la gran mayoría de esquemas criptográficos, vistos en el Capítulo 3, no consideran. Otro beneficio, muy de la mano con el anterior, es que ganamos mucho en eficiencia al realizar consultas de forma natural a la base de datos, gracias a que los datos son el texto claro. Finalmente, este esquema permite cuaquier tipo de consulta en SQL; además que permite consultas multicondicionales, es decir consultas con más de una condición.

Entonces mostremos cómo se realiza una consulta a nuestra base de datos que quedó lista desde la sección anterior.

Pensemos en dos consultas a la base de datos cifrada elaborada en la sección anterior Fig. 4.2.

q_1 = `SELECT NumEmp, Nombre`
`WHERE Sueldo='1500' AND Horario='Nocturno' AND CP='6789'`

q_2 = `SELECT Nombre WHERE NumEmp='1234' AND Sueldo='11000'`
`WHERE NumEmp='1234' AND Sueldo='11000'`

La traducción de estas consultas se muestra en la Fig. 4.3. En primera instancia veámos a la consulta q_1; en ella observamos tres condiciones, dos de ellas corresponden al tercer fragmento debido a que le pertenecen los atributos `Sueldo` y `Horario`. Así la aplicación identifica que se trata del tercer

[2]Los árboles B son estructuras de datos de árbol utilizadas en las bases de datos, que mantienen los datos ordenados e insertan y eliminan elementos en tiempo logarítmico amortizado

fragmento y genera la consulta $q_{1-servidor}$; cualquier consulta al servidor recaba las columnas `nonce`, `enc` y `tag`. Los dos registros que el servidor nos arroja posteriormente son convertidos de Base 64 a ASCII, descifrados con k_2 y autenticados mediante la columna `tag`; en el caso de que el *tag* obtenido no sea el mismo que el servidor regresó, se le avisa al usuario que los datos fueron modificados y se termina la consulta, en otro caso se procede a realizar el *parseo* de la cadena obtenida.

Para realizar el *parseo* de la tabla debemos identificar la longitud de cada atributo usando la tabla `Longitud Atributos` 4.1. Primeramente como ya conocemos que atributos fueron concatenados, buscamos las longitudes en la tabla y leemos el binario que nos indica el número de caracteres de cada atributo. De esta forma obtenemos los valores de cada atributo por separado listos para ser consultados por $q_{1-cliente}$ que requiere saber el `NumEmp` y el `Nombre` de la persona cuyo `CP` sea 06789. La consulta nos arroja que se trata de *Federico* cuyo número de empleado es el 4444; este resultado es enviado al cliente.

El procesamiento de la consulta q_1 es igual a la anterior. En esta consulta se desea conocer el nombre del empleado cuyo número de empleado es 1234 y su sueldo es de 1500. Debido a que el número de empleado no puede ser consultado, la aplicación genera la consulta al fragmento 3 que contiene el sueldo (si no hubiese existido otra condicional, la consulta no podría ser realizada y se envía un mensaje al cliente que la consulta no puede ser procesada). El servidor retorna tres registros que empatan con la condición; posteriormente son descifrados y *parseados* de la misma manera que en la consulta anterior. Y finalmente de los datos descifrados, verificamos cúal de ellos es el empleado *1234*; debido a que ninguno es, se le informa al cliente que no existe tal registro.

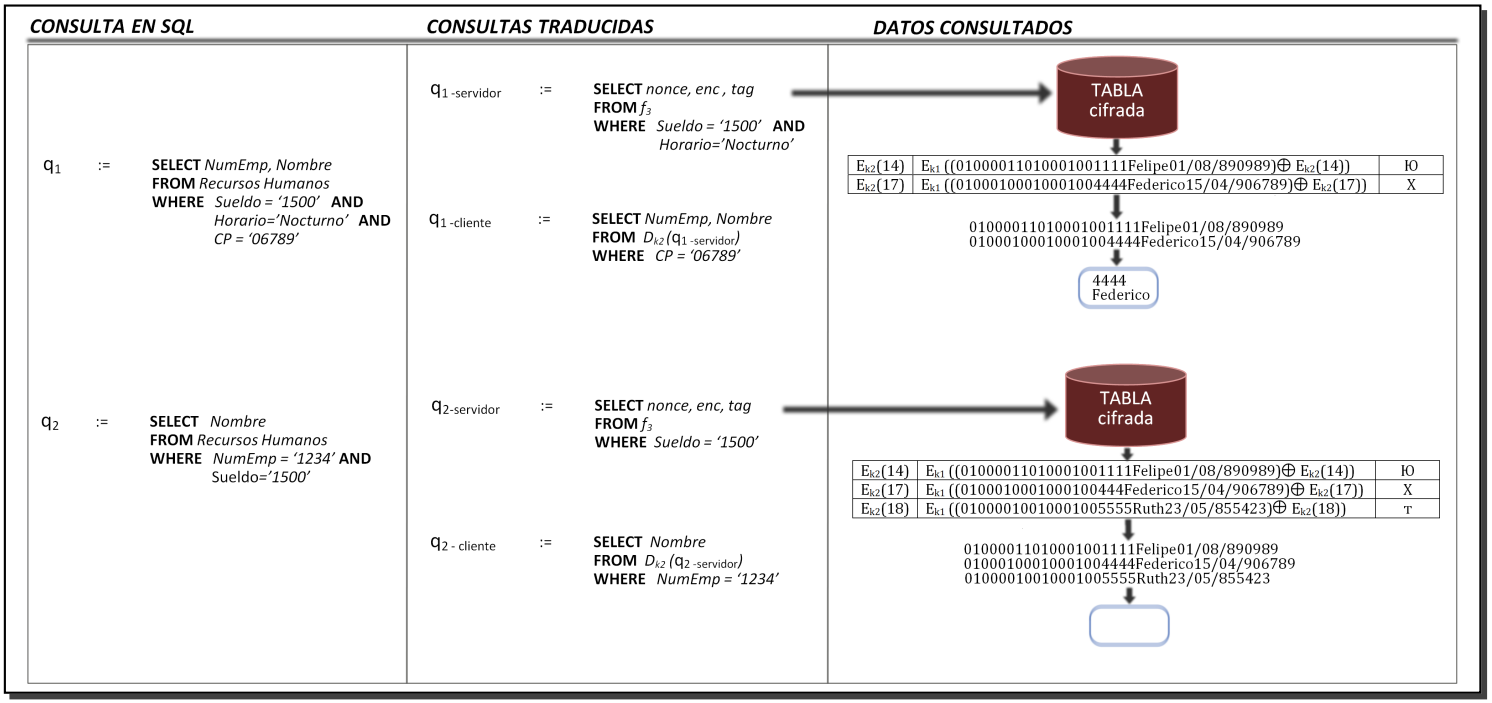

Figura 4.3: Un ejemplo de fragmentos físicos de la relación de atributos

5

Un Mejor Algoritmo para la Fragmentación

Como ya se ha mencionado a lo largo de esta tesis, el problema de fragmentación es NP-difícil, debido a que corresponde al problema de coloración de hipergrafos [17].

Definición. *Un hipergrafo HG es un conjunto de nodos V y un conjunto de hiper-aristas HA, tal que* $HA \subseteq P(V)$*, donde P(V) es el conjunto potencia de V.*

Un hipergrafo es una generalización de un grafo; y es un conjunto de nodos y aristas cuya peculiaridad radica en que sus aristas o ejes pueden conectar a más de dos nodos, a diferencia de los grafos que cada arista conecta solamente a dos nodos; a las aristas de los hipergrafos se les denomina hiper-aristas.

El problema de coloración de hipergrafo consiste en colorear los nodos de tal forma que ninguno de los hiper-ejes sea monocromático. Un ejemplo de hipergrafo coloreado se muestra en la Fig. 5.1.

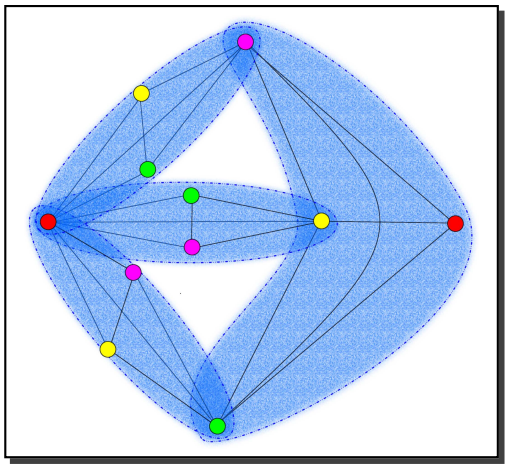

Figura 5.1: Ejemplo de un hipergrafo coloreado con cuatro hiper-aristas y número cromático 4

Decimos que nuestro problema de fragmentación empata con el de coloración de hipergrafo si: logramos ver a los nodos como atributos, las aristas como las restricciones de confidencialidad y el número cromático como el número de fragmentos. Así, de la misma forma que los hipergrafos buscamos que el número de fragmentos que obtengamos sea el menor posible; esto debido a que cada fragmento representa una instanciación completa de la Base de Datos.

Para darle solución al problema, en [7] se presenta una heurística que presume que los fragmentos obtenidos por él son mínimas. Nosotros proponemos un algoritmo genético que, a pesar de ser menos eficiente que el anterior, obtiene mejores resultados con respecto al número de fragmentos obtenidos; decimos lo anterior ya que el número de los fragmentos obtenidos con nuestro genético en comparación con aquellos del Algoritmo de Fragmentaciones Mínimas están por el orden de la mitad.

5.1 Algoritmo de Fragmentaciones Mínimas

Para resolver este problema Valentina Ciriani et.al. [7] propusieron una heurística que ataca este problema, a la cual le llamaron de FRAGMENTA-

CIONES MÍNIMAS que se muestra en el Algoritmo 5.

Este algoritmo recibe como parámetros la lista de los atributos `A_ToPlace` y la lista de restricciones a resolver `C_ToSolve`. Inicialmente el conjunto de los fragmentos $\mathcal{F}$ está vacío. Posteriormente se llenan las variables `Con` y `N_Con` para cada atributo a que pertenece a `A_ToPlace`; `Con[a]` guarda el conjunto de restricciones en las que a aparece, y `N_Con[a]` es la dimensión del conjunto `Con[a]`. Posteriormente se hace un ciclo que para hasta que ya no haya ningún atributo que se deba fragmentar; en este ciclo se elige al atributo de `A_ToPlace` que aparezca en más restricciones, es decir aquel cuyo `N_Con[a]` sea mayor; se restan las restricciones en las que aparece el atributo a de las restricciones por resolver `C_ToSolve` y se elimina dicho atributo de `A_ToPlace` para no fragmentarlo dos veces. Posteriormente se verifica que al agregar el atributo a en un fragmento de $\mathcal{F}$ no se viole ninguna restricción, si eso sucede, se prueba con el resto de los fragmentos; en el caso de que no pueda ser agregado en ningún fragmento o que no exista aún ninguno, se crea un nuevo fragmento en $\mathcal{F}$ agregando a a como elemento del mismo. Se realiza este procedimiento hasta que ya no haya más atributos en `A_ToPlace`.

Nosotros al ver, desarrollar y analizar dicho algoritmo, tuvimos la inquietud que si alguna otra heurística podría mejorar a ésta; i.e. pensamos que el algoritmo de Fragmentaciones Mínimas podía ser perfectible.

Así, pensamos en las bondades que el Cómputo Evolutivo ofrece mediante la técnica de los Algoritmos Genéticos para darle una nueva solución al problema. En la siguiente sección presentamos nuestra propuesta.

5.2 Algoritmo Genético

Un Algoritmo Genético se diferencía del resto de las técnicas utilizadas en el Cómputo Evolutivo principalmente por dos aspectos: i) el operador principal es la cruza y ii) comúnmente trabaja a nivel genotipo.

Para adentrarnos en nuestra propuesta es necesario aterrizar nuestro problema en dos principales aspectos, en primera instancia es necesario definir el problema en un gen y posteriormente identificar la función objetivo que se

Algoritmo 5 Algoritmo Fragmentaciones Mínimas

Entrada: FRAGMENT(*A_ToPlace*,*C_ToSolve*)

```
1: F := 0
2: for all a∈ A_ToPlace do
3:     Con[a] := {c ∈ C_ToSolve|a ∈ c}
4:     N_Con[a] := {c ∈ C_ToSolve|a ∈ c}
5: end for
6: repeat
7:     if C_ToSolve ≠ 0 then
8:         let attr be an attribute with the maximum value of N_Con[]
9:         for all c∈ (Con[attr]∩ C_ToSolve) do
10:            C_ToSolve := C_ToSolve − {c}
11:            for all a∈ c do
12:                N_Con[a] := N_Con[a] − 1
13:            end for
14:        end for
15:    else
16:        let attr be an attribute in A_ToPlace
17:    end if
18:    A_ToPlace:=A_ToPlace - {attr}
19:    inserted:=false
20:    for all F ∈ F do
21:        satisfies:=true
22:        for all c ∈ Con[attr] do
23:            if c ⊆ (F ∪ {attr}) then
24:                satisfies := false
25:                break
26:            end if
27:        end for
28:        if satisfies then
29:            F := F ∪ {attr}
30:            inserted := true
31:            break
32:        end if
33:    end for
34:    if NOT insert then
35:        add{attr} to F
36:    end if
37: until A_ToPlace = 0
38: return (F)
```

desea maximizar o minimizar; que en nuestro caso es una función de optimización de minimzación.

El gen de nuestra propuesta se muestra en la Fig. 5.2. Como se observa, es necesario enumerar los atributos que son candidatos a fragmentarse, i.e. aquellos que no son por sí sólos sensibles. Una fragmentación correcta del ejemplo de la Fig. 4.1 es mostrada en la Fig. 4.3. El gen consta de tres partes; la primera es una permutación de los atributos, la segunda indica la división de atributos para generar los fragmentos y la tercera es el valor de la aptitud del gen.

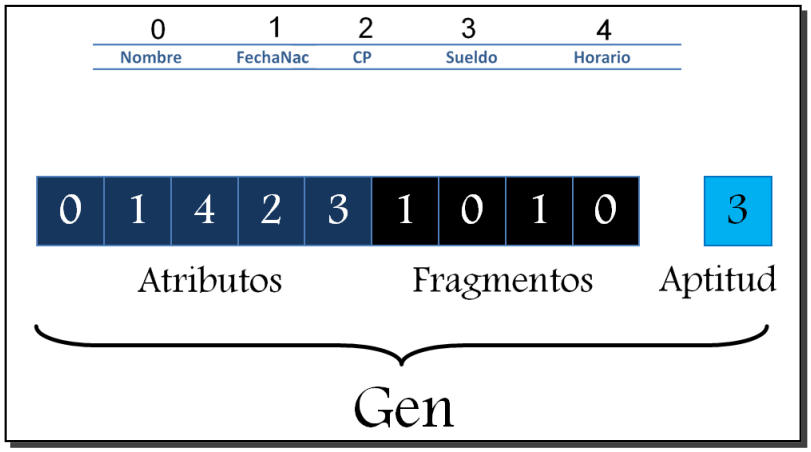

Figura 5.2: Ejemplo de un gen

El primer sector del gen como ya mencionamos antes es una permutación de los atributos, que junto con la segunda parte, nos indican el numero de fragmentos obtenidos y la distribución de los atribuos en estos. La segunda parte es una cadena binaria que, si el dígito se encuentra encendido indica que debe haber separación entre esos atributos, mientras que, si está apagado los atributos permanecen juntos en el mismo fragmento. Veámos nuesto ejemplo de la Fig. 5.2, la permutación de los atributos (la primer parte) menciona que los atributos están ordenados de la siguiente forma: `Nombre`,`FechaNac`, `Horario`, `CP` y `Sueldo`; por otro lado, en la segunda parte (la de la fragmentación) tenemos la cadena 1010. Si hacemos converger los atributos y la fragmentación, tendríamos:

`Nombre` 1 `FechaNac` 0 `Horario` 1 `CP` 0 `Sueldo`

indicándonos que `Nombre` es un único atributo del primer fragmento, `FechaNac` y `Horario` estarían en otra fragmentación y finalmente `CP` y `Sueldo` compartirían el último fragmento.

Ya una vez identificada la estructura del gen, es necesario estandarizar la longitud del mismo. Debido a que los Algorimos Genéticos trabajan a nivel genotipo es necesario tranformar la primer parte del mismo a codificación binaria. Para ello contamos con ciertas fórmulas que dependen totalmente del número de atributos a fragmentar, `atr`.

$$2^m \geq atr$$
$$tamGen = atr(m+1) - 1$$

En las fórmulas anteriores, m es el número de bits necesario para representar el número de atributos a fragmentar; y $tamGen$ guarda la longitud completa del gen en bits.

En nuestro ejemplo de la Fig. 5.2, atr es igual a 5, m es igual a 3 y `tamGen` es 19 bits. Y se mostraría de la siguiente forma:

000 001 100 010 011 1010.

La aptitud (*fitness*) de cada gen se calcula, identificando el número de fragmentos obtenidos más el número de violaciones a las restricciones de confidencialidad que dicha combinación genera. En nuestro caso obtuvimos tres fragmentos (número de unos en la sección de fragmentos + 1) y ninguna violación a las restricciones; por tal motivo el *fitnes* del gen es tan solo tres.

Nuestro genético se muestra en el Algoritmo 6.

Primeramente inicializamos la población de forma aleatoria y evaluamos la aptitud de cada individuo haciendo uso de las restricciones de condifencialidad. Posteriormente creamos una nueva generación al seleccionar los padres mediante la técnica de torneo; después se realiza la cruza de dos puntos de los padres (se eligen los dos puntos de forma aleatoria); el siguiente paso es

Algoritmo 6 Algoritmo Genético de Fragmentación

1: Inicializar población P de forma aleatoria;
2: Evaluar la aptitud de los individuos en P;
3: **repeat**
4: Seleccionar padres mediante torneo;
5: Cruzar padres usando la cruza de dos puntos;
6: Aplicar mutación a los hijos (generados de la cruza);
7: Evaluar la aptitud de los hijos;
8: Elegir al mejor individuo de la generación y compararlo directamente contra el mejor global;
9: **until** Número de generaciones máximo haya concluido

mutar a los hijos con cierta probabilidad; finalmente evaluamos la aptitud de la nueva población, elegimos al mejor individuo de la generación y lo comparamos contra el mejor individuo global[1]; si es un mejor individuo, si su aptitud es menor, se guarda como el mejor global; de otro modo el anterior sigue manteniendo el puesto del mejor global. Este proceso de crear nuevas generaciones se repite hasta que se hayan realizado totalmente el número de generaciones establecido.

Al finalizar los procesos de cruza y mutación, existe el caso de que el gen obtenido no sea válido; i.e. que repita atributos o que muestre atributos no existentes. Por ejemplo pensemos que después de uno de estos procesos nuestro gen queda de la siguiente forma:

$$\langle 2, 0, 5, 3, 2, 0, 1, 1, 1 \rangle$$

aquí, el atributo 2 se repite en dos ocaciones y el atributo 5 no existe; por otra parte los atributos 1 y 4 no aparecen en el gen. Lo que se debe hacer es reparar el gen para que pueda convertirse en uno válido.

El proceso de reparación es muy simple, y se muestra en el Algoritmo 7.

Siguiendo el Algoritmo 7; para nuestro ejemplo primeramente marcamos aquellos atributos que están incorrectos es decir el 5 y elegimos a cualquiera

[1]El mejor individuo global es aquel que ha tenido la mejor aptitud durante todo el proceso genético a lo largo de las diversas generaciones.

Algoritmo 7 Algoritmo de Reparación del Gen

Entrada: REPARACION_GEN(*Gen*,*numero_atributos*)

1: M :=numero_atributos
2: **repeat**
3: **if** El valor de la posición no es correcto **then**
4: Marcar la posición
5: **end if**
6: **if** El valor de la posición actual se ha repetido en las localidades anteriores **then**
7: Identificar las localidades de valores repetidos
8: Tomar aleatoriamente una localidad y marcarla
9: **end if**
10: **until** Se hayan recorrido las primeras M posiciones de `Gen`
11: Identificar los atributos que no aparezcan en las primeras M localidades de `Gen` y crear una lista (`restantes`)
12: Cambiar el valor de las localidades marcadas, tomando aleatoriamente un atributo de `restantes` **return** (`Gen`)

de los 2: $\langle X, 0, X, 3, 2, 0, 1, 1, 1\rangle$, posteriormente se colocan de forma aleatoria aquellos atributos que no aperecen i.e. el 4 y el 1. Una posibilidad de gen reparado sería:

$$\langle 1, 0, 4, 3, 2, 0, 1, 1, 1\rangle.$$

5.3 Comparación entre heurísticas y resultados obtenidos

Para ver el desempeño de nuestro algoritmo realizamos algunas pruebas comparándolo directamente contra el algoritmo de Fragmentaciones Mínimas. Las pruebas fueron realizadas utilizando diversos conjuntos de atributos y restricciones de confidencialidad.

La obtención de cada conjunto consiste en crear un conjunto de atributos y generar aleatoriamente las restricciones de confidencialidad. Las restricciones de confidencialidad son representadas en matrices de la siguiente forma:

como columnas tenemos a los atributos y como renglones a las restricciones de confidencialidad; las celdas xy(relglón x, columna y) de la tabla pueden tomar uno de los tres siguientes valores: un 1 indica que el atributo y aparece en la restricción x, un 3 indica que el atributo y es sensible (el valor 3 puede ir en cualquier restricción, con la condición de que no haya algun otro 3 o 1 a lo largo de esta columna) y 0 en el resto de los casos (que implica que el atributo y no aparece en la restricción x).

Para mostrar mejor el formato de la tabla de restricciones, en la Fig. 5.3 se presenta la matriz de las restricciones de la Fig. 4.1. Debido a que existe un atributo que es sensible, éste debe ser removido junto con la restricción como se muestra en la tabla inferior. Para fines prácticos en términos de programación se le agrega al final una columna que indica el número de atributos que contiene cada restricción.

	NumEmp	Nombre	FechaNac	CP	Sueldo	Horario	
r_0	3	0	0	0	0	0	
r_1	0	1	1	0	0	0	2
r_2	0	1	0	1	0	0	2
r_3	0	1	0	0	1	0	2
r_4	0	1	0	0	0	1	2
r_5	0	0	1	1	1	0	3
r_6	0	0	1	1	0	1	3

	Nombre	FechaNac	CP	Sueldo	Horario	
r_1	1	1	0	0	0	2
r_2	1	0	1	0	0	2
r_3	1	0	0	1	0	2
r_4	1	0	0	0	1	2
r_5	0	1	1	1	0	3
r_6	0	1	1	0	1	3

Figura 5.3: Matriz de restricciones del ejemplo de la Fig. 4.1

De esta manera se crearon aleatoriamente las tablas, verificando que la matriz fuera correcta. La matriz es correcta si: i) ninguna restricción es subconjunto de otra, y ii) si un atributo fué marcado como sensible, no debe aparecer en ninguna restricción con un 1.

Los parámetros utilizados para el Algoritmo Genético de Fragmentaciones fueron : i)probabilidad de cruza = 0.5, ii)probabilidad de mutación =

0.2, iii)tamaño de población = 100 indiviuos y iv)número máximo de generaciones = 100.

Realizamos 150 conjuntos de atributos con sus respectivas restricciones para generar las pruebas. Tomamos conjuntos de atributos de diez en diez hasta quinientos, i.e. 10 atributos, 20 atributos, 30 atributos, ..., 490 atributos y 500 atributos; y por cada conjunto de atributos se generaron tres conjuntos de restricciones donde: i) el número de restricciones es igual a la mitad del número de atributos, ii) el número de restricciones es igual a tres cuartos del número de atributos y iii) el número de restricciones es igual al número de atributos. Las pruebas se realizaron en un servidor **Sun Fire V2oz** cuyas especificaciones técnicas son las siguientes:

- dos procesadores AMD Opteron Serie 200 de núcleo doble
- 1 MB de caché de nivel 2 por núcleo
- 16 GB de SDRAM DDR1-333
- 3 x 3.2 GB/seg. Enlaces HyperTransport por procesador
- SUSE LINUX Enterprise Server 9

.

Decidimos utilizar tres conjuntos de restricciones por cada conjunto de atributos, ya que pensábamos que el número de fragmentos crecía directamente proporcional al número de restricciones; hipótesis que fué errónea.

Estos conjuntos de atributos-restricciones fueron ejecutados una sola vez en el algoritmo de Fragmentaciones Minimas y treinta veces en el genético; esto debido a que el genético agrega aleatoriedad en el proceso y el de Fragmentaciones Mínimas no. Los resultados del genético fueron promediados en cuanto a fragmentos obtenidos y el tiempo generado.

Priméramente mostramos en la Fig. 5.4 los resultados obtenidos cuando el número de restricciones es la mitad del número de atributos. La línea del genético además muestra los valores máximo y mínimo al ejecutar las treinta veces el algoritmo.

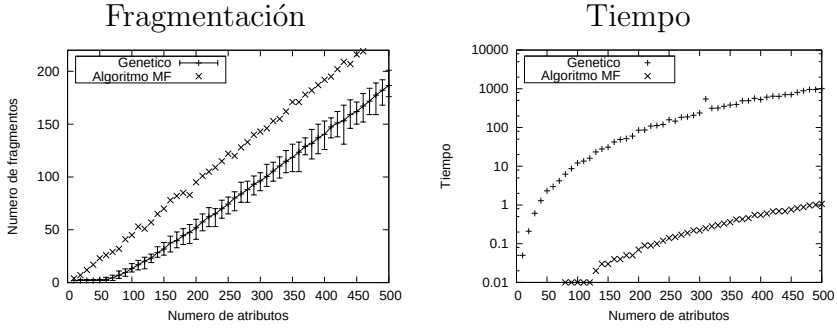

Figura 5.4: Número de restricciones=$\frac{1}{2}$(Número de fragmentos)

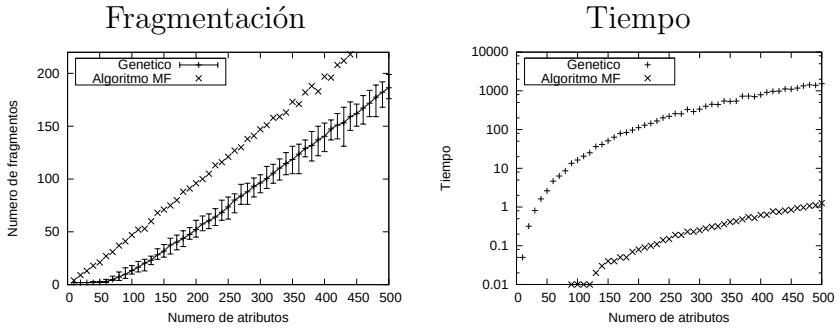

Figura 5.5: Número de restricciones=$\frac{3}{4}$(Número de fragmentos)

Los resultados obtenidos cuando el número de restricciones está en el órden de tres cuartos de los atributos se presentan en la Fig. 5.5. De igual forma el genético muestra los valores máximos y mínimos.

Finalmente se muestran en la Fig. 5.6 los resultados obtenidos cuando el número de restricciones es igual al número de atributos.

Como podemos observar, los fragmentos obtenidos en nuestro genético son mucho menores que los obtenidos en el Algoritmo de Fragmentaciones Mínimas, esto es bastante satisfactorio debido a que el propósito de obtener menos fragmentos fué cumplido. Podemos decir que nuestro genético supera por mucho al algoritmo de Fragmentaciones Mínimas propuesto en [7].

Con respecto al tiempo que se tardaron ambos algoritmos en realizar las pruebas, vemos que nuestro genético tarda mucho más que el otro. A esto no

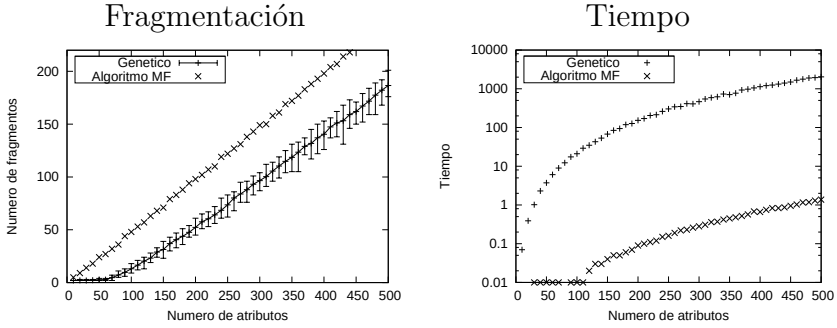

Figura 5.6: Número de restricciones=Número de fragmentos

le vemos tanto problema ya que la generación de los fragmentos se realiza solamente una vez en el cliente, es decir no es en línea. Por tal motivo ponemos más peso en encontrar menos fragmentos, sin preocuparnos mucho del tiempo; además de que creemos conveniente invertir un poco de tiempo inicialmente, pero que la dimensión de nuestra Base de Datos sea mucho menor.

6

Acerca de la implementación

En este capítulo mostramos algunos diagramas que presentan ciertas arquitecturas y paso a paso algunas transacciones de nuestra aplicación.

Los requerimientos de software considerados para nuestra aplicación, se enlistan a continuación. Primeramente era necesario seleccionar un lenguaje de programación y un Sistema Manejador de Bases de Datos que pudieran comunicarse entre sí. Por su robustés, flexibilidad y su manipulación de sentencias a bajo nivel, elegimos como lenguaje de programación a ANSI C y como Sistema Manejador de Base de Datos a pgAdmin que trabaja con PostgreSQL; nos decidimos a usar PostgreSQL debido a que fué programado en C y utiliza sentencias del estándar SQL[1]. Como plataforma utilizamos la distribución Ubuntu del sistema operativo Linux. Finalmente nos quedaba lograr que C pudiera comunicarse con Postgres y lo hicimos mediante la librería de C: `libpq`.

En la Fig. 6.1 se muestra la distribución del esquema en las dos entidades:

[1]SQL es el lenguaje de consultas estructurado actualmente más utilizado en los Sistemas Manejadores de Bases de Datos.

el cliente y el servidor. El cliente aloja la aplicación, misma que contiene el *front-end* o traductor de consultas; y por el otro lado, el servidor guarda la base de datos lista para ser consultada. La parte más robusta de este esquema es el traductor de consultas, por ello es alojado en el cliente; el *front-end* genera el proceso de transformación de una consulta normal ingresada por el usuario a una consulta que pueda ser interpretada por el Sistema Manejador de Bases de Datos.

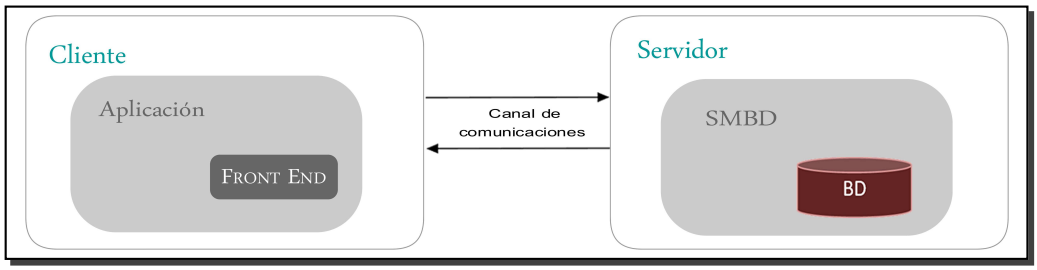

Figura 6.1: Arquitectura de distribución

Ya teniendo bien identificados los roles del cliente y del servidor, pasamos a la conversión de la tabla en texto claro a la cifrada, mostramos el proceso en la Fig. 6.2. En primera instancia es necesario encontrar los fragmentos de la Base de Datos, para ello utilizamos el Algoritmo Genético de Fragmentación; éste recibe como parámetros dos archivos de texto, en uno los atributos y en otro las restricciones de confidencialidad; y como resultado retorna una combinación de atributos y fragmentaciones que: i)no viola ninguna restricción de confidencialidad y ii)el número de fragmentos es de los menores que se pueden generar.

La aplicación toma los fragmentos con sus respectivos atributos y la tabla en texto claro; que por una parte, llena las tablas:

- Atributos Permitidos. Esta tabla guarda los atributos que pueden ser consultados, es decir que las restricciones de confidencialidad no les dictaron permanecer cifrados. Dicha tabla se utiliza al generar una consulta, es necesario verificar que el atributo está en texto claro y puede ser consultado.

- Fragmento/Atributo. Esta tabla guarda los fragmentos obtenidos y los aributos que le corresponen a cada una. Ésta es utilizada por el

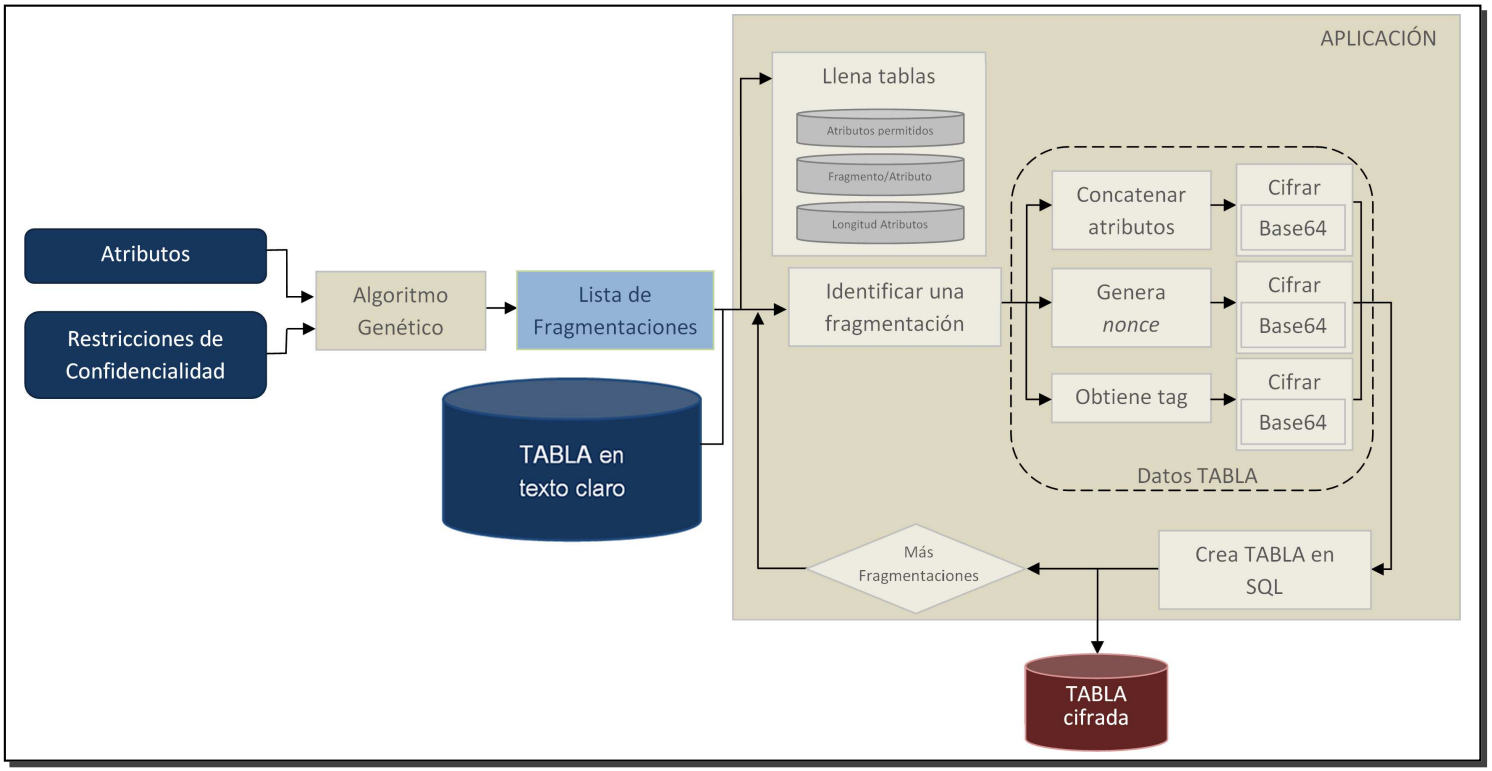

Figura 6.2: Obtención de la tabla cifrada

front-end para generar la consulta propia al servidor, identificando en qué fragmento está cierto atributo.

- Longitud Atributos. Guarda la longitud máxima en bits de todos los atributos, tanto cifrados como fragmentados. Se utiliza al momento de descifrar los valores; ya que como se concatenan los atributos que no aparecen en la fragmentación, es necesario hacer el *parseo* de los datos. El *parseo* se logra leyendo la longitud del valor de cada atributo para dicho registro en los n bits asignados en esta tabla.

Por otro lado, la aplicación crea cada una de los fragmentos obtenidos por el Algoritmo Genético de Fragmentación. Para crear cada fragmentación, además de los atributos en texto claro, son necesarias tres columnas más: el cifrado *enc*, el *nonce* y el *tag*. Para obtener *enc* se toma el primer fragmento físico y se identifican sus atributos, posteriormente se obtiene el primer registro de la tabla y se concatenan los valores de aquellos atributos que no pertenecen al fragmento actual junto con su longitud en el registro, finalmente se cifra con OCB1. El *nonce* es una cadena que no puede repetirse, así que usamos un contador como *nonce*, pero para fines de seguridad éste debe ir cifrado con otra llave diferente a la usada en la columna *enc*. El *tag* sirve para autenticar, y la obtención de éste ya fué explicada en el Capítulo 3. Cada uno de estos cifrados debe pasarse por código Base64 para que Postgres pueda ingresarlos como caracteres válidos. Dicho registro obtenido, es ingresado a la tabla cifrada. Este proceso se realiza para cada registro.

De esta manera se genera la fragmentación hasta que ya no haya ningún fragmento, y se obtiene la BD cifrada. Ésta ya puede ser alojada en el servidor.

Con lo anterior, el esquema está listo para generar cualquier consulta. El proceso de consultar la base de datos se muestra en la Fig. 6.3.

El cliente genera una consulta en SQL; la aplicación descompone la consulta en tres partes: i)el *Atributo* a consultar, ii)el *Valor* de la condición y iii)los *Atributos Respuesta*, e.g. en una base de datos del control escolar de cierta universidad, la consulta `SELECT id,nombre,carrera FROM tabla WHERE ciudad='México'` se descompone como Atributo=*ciudad*, Valor=*México* y Atributos Respuesta=*id,nombre y carrera*.

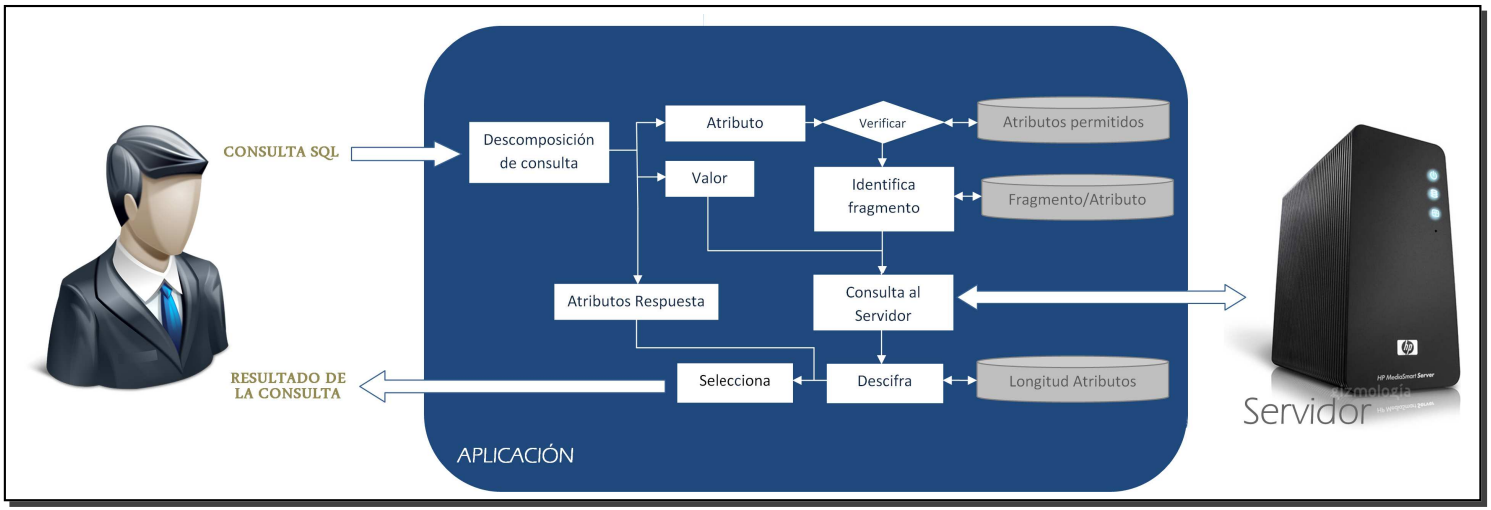

Figura 6.3: Esquema de Consulta

Posteriormente verificamos cual ATRIBUTO está permitido para ser consultado, es decir si está en texto claro; en caso contrario se le da un aviso al cliente que la consulta no puede ser procesada. Si el atributo puede ser consultado, se identifica a qué fragmentación pertenece. Posteriormente se genera la consulta apropiada preguntando por VALOR en la fragmentación correspondiente preguntando al servidor por las columnas *enc*, *nonce* y *tag*. Haciendo uso de *nonce* desciframos *enc* y lo autenticamos con *tag*; si el registro ha sido violado, se le envía un aviso al cliente que dicho registro fué modificado. Ya teniendo el registro descifrado, hacemos el *parseo* de los datos, auxiliándonos con la tabla Logitud Atributos y finalmente se envian al cliente los valores de los *Atributos Respuesta*.

7

Conclusiones y Trabajo a Futuro

En este trabajo de tesis presentamos un esquema de consultas a Bases de Datos bajo el contexto DaaS utilizando fragmentación y cifrado de los datos. Además, los datos de este esquema son autenticados ya que no solamente nos preocupamos de la seguridad de los datos, sino también de la consistencia de ellos. El esquema presenta grandes bondades en comparación con algunos otros esquemas que cifran por completo la Base de Datos; una de ellas es la eficiencia en las consultas y otra es la libertad de consultas posibles a generar.

Por otra parte, se presentó un algoritmo genético para la realización de la fragmentación, dicho problema es de complejidad NP-difícil ya que corresponde al problema de coloración de hipergrafos. Dicho algoritmo obtuvo mejores resultados, al generar menos fragmentos que otra heurística ya diseñada para este mismo problema.

Así, dicho trabajo de tesis aporta buenos resultados con respecto a otros propuestos en la literatura. No obstante, estamos conscientes de que no es el esquema perfecto para resolver este problema de consultas remotas a bases de datos, ya que la principal desventaja que apreciamos en él es que cada fragmento representa una replicación completa de la base de datos, como se

verá a continuación.

Pensemos esto en un escenario real, donde una Base de Datos ocupa un terabyte, si se obtienen a lo menos 50 fragmentos con nuestro algoritmo genético, estamos hablando de alrededor de 50 terabytes para ser alojados. Por este motivo, nuestro esquema no puede ser utilizado en las Bases de Datos con muchos atributos.

Así pues, hablemos de las posibles áreas de oportunidad que este trabajo presenta.

Una de ellas puede ser: probar con más heurísticas para tratar de reducir el número de fragmentos obtenidos. O realizar más pruebas exhautivas del genético con diferentes parámetros.

Otra área de oportunidad es pensar en una manera diferente de reducir los fragmentos mediante otras restricciones de eliminación; por ejemplo, tratar como datos sensibles a aquellos que resulten ser los únicos atributos en una fragmentación.

Finalmente, proponemos migrar el esquema a archivos XML, ya que la investigación en esa área es bastante robusta.

Bibliografía

[1] Aggarwal, G., Bawa, M., Ganesan, P., Garcia-Molina, H., Kenthapadi, K., Motwani, R., Srivastava, U., Thomas, D., and 0002, Y. X. Two can keep a secret: A distributed architecture for secure database services. In *CIDR* (2005), pp. 186–199.

[2] Bayer, R. Binary b-trees for virtual memory. In *SIGFIDET Workshop* (1971), E. F. Codd and A. L. Dean, Eds., ACM, pp. 219–235.

[3] Bellare, M., Rogaway, P., and Wagner, D. A conventional authenticated-encryption mode, 2003.

[4] Bonifati, A., and Cuzzocrea, A. Efficient fragmentation of large xml documents. In *DEXA* (2007), pp. 539–550.

[5] Bouganim, L., and Pucheral, P. Chip-secured data access: Confidential data on untrusted servers. In *VLDB* (2002), pp. 131–142.

[6] Ceselli, A., Damiani, E., di Vimercati, S. D. C., Jajodia, S., Paraboschi, S., and Samarati, P. Modeling and assessing inference exposure in encrypted databases. *ACM Trans. Inf. Syst. Secur. 8*, 1 (2005), 119–152.

[7] Ciriani, V., di Vimercati, S. D. C., Foresti, S., Jajodia, S., Paraboschi, S., and Samarati, P. Fragmentation and encryption

to enforce privacy in data storage. In *ESORICS* (2007), J. Biskup and J. Lopez, Eds., vol. 4734 of *Lecture Notes in Computer Science*, Springer, pp. 171–186.

[8] Codd, E. F. A relational model of data for large shared data banks. *Commun. ACM 13*, 6 (1970), 377–387.

[9] Daemen, J., and Rijmen, V. Rijndael for aes. In *AES Candidate Conference* (2000), pp. 343–348.

[10] Damiani, E., di Vimercati, S. D. C., Jajodia, S., Paraboschi, S., and Samarati, P. Balancing confidentiality and efficiency in untrusted relational DBMSs. In *ACM Conference on Computer and Communications Security* (2003), S. Jajodia, V. Atluri, and T. Jaeger, Eds., ACM, pp. 93–102.

[11] Darwin, C. R. *The Origin of Species by Means of Natural Selection, or The Preservation of Favoured Races in the Struggle for Life*. Penguin Books, Nueva York, EE. UU, 1959.

[12] Dash, E. Lost credit data improperly kept, company admits. *New York Times* (June 2005).

[13] Davida, G. I., Wells, D. L., and Kam, J. B. A database encryption system with subkeys. *ACM Trans. Database Syst. 6*, 2 (1981), 312–328.

[14] de Castro, L. N., and Timmis, J. *Artificial Immune Systems: A new Computational Intelligence Approach*. Springer Verlag, 2002.

[15] Dorigo, M., Maniezzo, V., and Colorni, A. The ant system: Optimization by a colony of cooperating agents. In *IEEE Transactions on Systems, Man, and Cybernetics-Part B 26* (1996), pp. 29–41.

[16] Foldes, S. Symmetries of directed graphs and the chinese remainder theorem. *J. Comb. Theory, Ser. B 28*, 1 (1980), 18–25.

[17] Garey, M. R., and Johnson, D. S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman, 1979.

[18] Gorn, S., Bemer, R. W., Green, J., and Lohse, E. Proposed american standard: bit sequencing of the american standard code for

information interchange (acsii) in serial-by-bit data transmission. *Commun. ACM 7*, 6 (1964), 333–336.

[19] Hacigümüs, H., Iyer, B. R., Li, C., and Mehrotra, S. Executing sql over encrypted data in the database-service-provider model. In *SIGMOD Conference* (2002), M. J. Franklin, B. Moon, and A. Ailamaki, Eds., ACM, pp. 216–227.

[20] Hacigümüs, H., Iyer, B. R., and Mehrotra, S. Efficient execution of aggregation queries over encrypted relational databases. In *DASFAA* (2004), Y.-J. Lee, J. Li, K.-Y. Whang, and D. Lee, Eds., vol. 2973 of *Lecture Notes in Computer Science*, Springer, pp. 125–136.

[21] Hacigümüs, H., Mehrotra, S., and Iyer, B. R. Providing database as a service. In *ICDE* (2002), IEEE Computer Society, pp. 29–38.

[22] Håstad, J. The security of the iapm and iacbc modes. *J. Cryptology 20*, 2 (2007), 153–163.

[23] Holland, J. H. *Adaptation in Natural and Artificial Systems.* Ann Arbor, University Press, 1975.

[24] Jingmin, H., and Wang, M. Cryptography and relational database management systems. In *IDEAS* (2001), M. E. Adiba, C. Collet, and B. C. Desai, Eds., IEEE Computer Society, pp. 273–284.

[25] Joux, A., Martinet, G., and Valette, F. Blockwise-adaptive attackers: Revisiting the (in)security of some provably secure encryption models: Cbc, gem, iacbc. In *CRYPTO* (2002), M. Yung, Ed., vol. 2442 of *Lecture Notes in Computer Science*, Springer, pp. 17–30.

[26] Kennedy, J., and Eberhart, R. C. *Swarm Intelligence.* Morgan Kaufmann Publishers, California, EE. UU., 2001.

[27] Koza, J. R. *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, EE. UU., 1992.

[28] Laguna, M., and Martí, R. *Scatter Search: Methodology and Implementations in C.* Kluwer Academic Publishers, 2003.

[29] Lidl, R., and Niederreiter, H. *Finite fields / Rudolf Lidl, Harald Niederreiter ; foreword by P.M. Cohn*, 2nd ed. ed. Cambridge University Press, Cambridge ; New York :, 1997.

[30] Liskov, M., Rivest, R. L., and Wagner, D. Tweakable block ciphers. In *CRYPTO* (2002), M. Yung, Ed., vol. 2442 of *Lecture Notes in Computer Science*, Springer, pp. 31–46.

[31] Maslow, A. H. A theory of human motivation. *Psychological Review 50* (1943), 430–437.

[32] Maxfield, A. C. M., and Fogel, L., Eds. *Artificial Intelligence through a Simulation of Evolution.* Biophysics and Cybernetics Systems: Proceedings of the Second Cybernetics Sciences. Spartan Books, Washington D.C., EE. UU., 1965.

[33] Maxfield, A. C. M., and Fogel, L. Artificial intelligence through a simulation of evolution. In *Biophysics and Cybernetics Systems: Proceedings of the Second Cybernetics Sciences* (Washington D.C., EE. UU., 1965), Spartan Books.

[34] McGrew, D. A., and Viega, J. The security and performance of the galois/counter mode (gcm) of operation. In *INDOCRYPT* (2004), A. Canteaut and K. Viswanathan, Eds., vol. 3348 of *Lecture Notes in Computer Science*, Springer, pp. 343–355.

[35] Özsoyoglu, G., Singer, D. A., and Chung, S. S. Anti-tamper databases: Querying encrypted databases. In *DBSec* (2003), S. D. C. di Vimercati, I. Ray, and I. Ray, Eds., Kluwer, pp. 133–146.

[36] Rechenberg, I. Evolutionsstrategie. optimierung technischer systeme nach prinzipien der biologischen evolution, 1973.

[37] Reynolds, R. G., Michalewicz, Z., and Cavaretta, M. Using cultural algorithms for constraint handling in genocop. In *In Proceedings of the Fourth Annual Conference on Evolutionary Programming* (Cambridge, Massachusetts, EE. UU., 1995), J. R. McDonnell, R. G. Reynolds, and D. B. Fogel, Eds., MIT Press, pp. 298–305.

[38] Rogaway, P. Authenticated-encryption with associated-data. In *ACM Conference on Computer and Communications Security* (2002), V. Atluri, Ed., ACM, pp. 98–107.

[39] Rogaway, P. Efficient instantiations of tweakable blockciphers and refinements to modes ocb and pmac. In *ASIACRYPT* (2004), P. J. Lee, Ed., vol. 3329 of *Lecture Notes in Computer Science*, Springer, pp. 16–31.

[40] Rogaway, P., Bellare, M., Black, J., and Krovetz, T. Ocb: a block-cipher mode of operation for efficient authenticated encryption. In *ACM Conference on Computer and Communications Security* (2001), pp. 196–205.

[41] Song, D. X., Wagner, D., and Perrig, A. Practical techniques for searches on encrypted data. In *IEEE Symposium on Security and Privacy* (2000), pp. 44–55.

[42] Stinson, D. R. *Cryptography: Theory and Practice, Second Edition.* Chapman & Hall/CRC, February 2002.

[43] Storn, R., and Price, K. Differential evolution - a simple and efficient adaptative scheme for global optimization over continuous spaces. Tech. Rep. TR-95–12, International Computer Science, Berkeley, California, March 1995.

[44] Syswerda, G. Uniform crossover in genetic algorithms. In *ICGA* (1989), J. D. Schaffer, Ed., Morgan Kaufmann, pp. 2–9.

[45] Vingralek, R. Gnatdb: A small-footprint, secure database system. In *VLDB* (2002), Morgan Kaufmann, pp. 884–893.

[46] Wetzel, A. Evaluation of the effectiveness of genetic algorithms in combinatorial optimization. Tech. rep., University of Pittsburgh, 1983.

[47] Whiting, D., Housley, R., and Ferguson, N. RFC 3610:counter with cbc-mac (ccm), 2003.

[48] Whitley, D. The genitor algorithm and selection pressure: Why rank-based of reproductive trials is best. In *Third International Conference on Genetic Algorithms*, J. D. Schaffer, Ed. Morgan Kaufmann, San Mateo, California, 1989, pp. 116–121.

[49] Yang, Z., Zhong, S., and Wright, R. Privacy-preserving queries on encrypted data. In *ESORICS* (2006), D. Gollmann, J. Meier, and A. Sabelfeld, Eds., vol. 4189 of *Lecture Notes in Computer Science*, Springer, pp. 479–495.