UNIVERSIDAD TÉCNICA NACIONAL
INGENIERÍA EN TECNOLOGÍAS DE INFORMACIÓN
SEDE DEL PACÍFICO
GUÍA DE ACTIVIDAD PRÁCTICA Y/O LABORATORIO

Curso:	ITI-912 – Administración de Bases de Datos Avanzada	Puntos	100
Profesor:	Jorge Ruiz (york)	Valor %	5%
Fecha Entrega:	24/septiembre/2024		
Tiempo:	2 Semanas		

Estudiante: Suheilyn García Umaña
Identificación: 604380960

- **Objetivos de la actividad.**
  - Evaluar los procesos para el cifrado de datos en el motor de Microsoft SQL-Server 2019 o 2022.
  - Contrastar los métodos de cifrado de datos del Microsoft SQL-Server 2019, contra otros motores de datos a elección del estudiante.
- **Instrucciones de la práctica.**

Laboratorio #3: Observaciones:

- Lea cuidadosamente cada uno de los problemas planteados y en caso de duda, puede utilizar el chat dispuesto para tal fin, que se encuentra al inicio del curso en la plataforma Moodle del Campus Virtual. U otra que indique el profesor de turno.

- Debe entregar un documento PDF tanto con los problemas planteados, así como con sus correspondientes respuestas. Debe agregar pantallazos de los procesos realizados, los scripts que se están ejecutando, planes de ejecución, estadísticas.

- El material de esta lección fue buscado, preparado y desarrollado utilizando Microsoft SQL-Server 2019.

- Su profesor hizo entrega de un script de SQL llamado 03_Encriptar_x_frase.sql y 03_Encriptar_x_llave.sql, como lo indica en el segmento general de esta práctica.

- Usted deberá de ejecutar paso a paso dicho script para poder crear el demo sugerido, recuerde que debe adaptarlo para que se pueda utilizar sobre plataforma Microsoft Windows.

Planteo del problema 1

El 18 de febrero de 2005 – Bank of America es acusado de haber perdido más de 1,2 millones de registros de clientes – aunque dijeron que no había evidencia de que la información hubiera caído en manos criminales.

El 12 de enero de 2007 – MoneyGram, un proveedor de medios de pago, informó de que un servidor de la empresa fue ilegalmente accedido desde Internet. Contenía información de unos 79.000 recibos de pago de clientes, incluyendo nombres, direcciones, números de teléfono y, en algunos casos, números de cuenta bancaria.

El 17 de enero de 2007 – TJX Companies Inc. anunció públicamente que había experimentado una intrusión no autorizada en el sistema electrónico de procesamiento de información de tarjetas de crédito/débito. En la que es considerada como la más fascinante brecha de seguridad hasta la fecha, hasta 45.700.000 números de cuenta de tarjetas de crédito/débito y más de 455.000 registros de devolución de mercancía (conteniendo nombres y números de permiso de conducción de clientes) fueron robados del sistema de TI de la empresa.

Directriz de la Industria de Tarjetas de Pago: PCI

El 'Estándar de Seguridad de Datos para la Industria de Tarjeta de Pago', PCI DSS por sus siglas en inglés, ha sido desarrollado por un comité conformado por las compañías de tarjetas (débito y crédito) más importantes (Visa, MasterCard, etc.), comité denominado PCI SSC (Payment Card Industry Security Standards Council) como una guía que ayude a las organizaciones que procesan, almacenan y/o transmiten datos de tarjetahabientes (o titulares de tarjeta), a asegurar dichos datos, con el fin de prevenir los fraudes que involucran tarjetas de pago débito y crédito.

Las compañías que procesan guardan o trasmiten datos de tarjetas deben cumplir con el estándar o arriesgan la pérdida de sus permisos para procesar las tarjetas de crédito y débito (Perdida de franquicias), enfrentar auditorías rigurosas o pagos de multas.

Protección de datos:

Uno de los requisitos para cumplir con este estándar es proteger los datos de los propietarios de tarjetas y se exige cifrar los números de tarjeta de crédito.

Que se requiere:

Este trabajo tiene como objetivo familiarizar al estudiante con la protección de datos por medio del cifrado. Para esto deberá hacer lo siguiente:

1. **Base de datos:** Crear una base de datos con una tabla para almacenar nombres de tarjeta habientes y números de tarjeta de crédito. Recuerde que cada tarjetahabiente puede tener más de una tarjeta de crédito a su nombre.

2. **Aplicación:** Desarrollar una aplicación con una sola pantalla e implementar una consulta a la base de datos. El objetivo de la consulta es saber a cuál tarjeta habiente pertenece un número de tarjeta de crédito en particular. La consulta recibe el número de tarjeta de crédito como parámetro y devuelve el nombre de la tarjeta habiente asociado a esa tarjeta.

3. **Cifrado:** Para cumplir con el estándar PCI, los números de tarjeta se deben almacenar cifrados.

4. **Eficiencia y rendimiento:** Por motivos de eficiencia y rendimiento, al procesar la búsqueda no es recomendable descifrar cada número de tarjeta de crédito y compararlo con el número de tarjeta que se busca. Por lo tanto, se recomienda explorar otras alternativas.

Consideraciones y recomendaciones:

- Usar algún algoritmo de cifrado aceptado en PCI, por ejemplo: AES, TDES, RSA.

- El proceso de cifrar los datos se puede llevar a cabo en el nivel de la aplicación o en el nivel de la base de datos, lo importante es que al final los datos se almacenen cifrados y si alguien logra tener acceso a la base de datos no pueda descifrarlos. Valorar las ventajas y desventajas de ambas alternativas.

- Los números de tarjetas de crédito tienen 16 dígitos numéricos, es aceptable que se dejen sin cifrar los últimos 4. Tampoco implica riesgos imprimir y desplegar estos dígitos en diferentes lugares como estados de cuenta, baucher, consultas en Internet, etc.

- Para hacer más eficiente la búsqueda se recomienda tomar en cuenta lo siguiente:

    - Agregar un campo adicional a la tabla, que tenga solo los últimos 4 dígitos de cada tarjeta sin cifrar y utilizarlo para limitar la búsqueda.

- Se recomienda investigar si es posible o no comparar dos valores cifrados para ahorrar recursos. Revisar qué ventajas y desventajas tiene hacer esto.

- Otra alternativa para limitar el rango de búsqueda podría estar asociado al uso de una función HASH para generar una llave a partir del número de tarjeta. Una función de hash es una función para resumir o identificar probabilísticamente un gran conjunto de información, dando como resultado un conjunto imagen finito generalmente menor. Matemáticamente, se comporta como una función cuasi inyectiva.

Entregables:

- Descripción de la propuesta y de los métodos por desarrollar.
  Desarrollo de una Api Rest que envié una solicitud a la base de datos con el método GET enviando la cedula de un tarjetabiente como parámetro para obtener el numero de la tarjeta de crédito.
- Demostración de la función de los procesos.
- Script de la base de datos generada, donde incluya las tablas y sus relaciones, procedimientos almacenados y cualquier otro objeto que ser requiera.

```
use master
go
-- Create database
create database db_bank
on primary (name='db_bank', filename='C:\mssql\data\db_bank.mdf',  size=50Mb,
maxsize=150Mb, filegrowth=25Mb)
log on (name='db_bank_log',  filename='C:\mssql\data\db_bank_log.ldf', size=30Mb,
maxsize=100Mb, filegrowth=25Mb);
go
```

```
-- Retrieve stored symmetric keys
select * from sys.symmetric_keys
go
```

```
-- Create symmetric key (master database)
create master key encryption by password =
'23987hxJKL95QYV4369#ghf0%lekjg5k3fd117r$$#1946kcj$n44ncjhdlj'
go
```

```
-- Create a certificate
create certificate secure_credit_cards with subject = 'custom credit card
number';
go
```

```
-- Create symmetric key (current database)
create symmetric key lscck_05
with algorithm = aes_256
```

```
encryption by certificate secure_credit_cards;
•
use db_bank
•
-- Create person table
create table customer(
    cedcustomer varchar(20) not null,
    nombre varchar(30),
     correo varchar(30)
);
go
•
alter table customer add constraint pk_customer primary key(cedcustomer);
go
•
-- Create example table
create table cards(
    customer varchar(20) not null,
    creditCard varchar(25) not null,
    encryptedCC varbinary(250)
);
go
alter table cards add constraint pk_cards primary key(creditCard);
go
alter table cards add constraint fk_cards_customer foreign key(customer)
references customer(cedcustomer);
go
•
-- EncryptByKey(par_1, par_2, par_3, par_4)
--   par_1: key_GUID to be used to encrypt
--   par_2: value to be stored
--   par_3: add authenticator, only if value = 1
--   par_4: authenticator value
•
-- Open the symmetric key with which to encrypt the data.
open symmetric key lscck_05 decryption by certificate secure_credit_cards;
go
•
-- Insert data
•
insert into customer
values('605960578', 'Juanito', 'juani17@gmail.com');
go
•

insert into cards
```

```
values('608960578', '6042210012564010',
EncryptByKey(Key_GUID('lscck_05'),'608960578',0));
go
```

```
-- Close the key
close symmetric key lscck_04;
```

```
-- Retrieve data
SELECT cd.creditCard FROM cards cd
INNER JOIN customer c ON cd.customer = c.cedcustomer
WHERE cd.customer = '608960578';
```

- Código fuente de la aplicación creada considerando la demanda de este trabajo

  Link de descarga https://github.com/Ach1c0d1a/lab3-flask.git

API Rest Flask con SQL Server

Required tools:

Database ...: MSSQL Server https://www.microsoft.com/es-es/sql-server/sql-server-downloads?msockid=29b39137d4f064d90f2e8410d59c65cb

Language support.....: Python https://www.python.org/

Developer tool.....: Visual Code https://code.visualstudio.com/download

- and a lot of enthusiasm to learn

Paso 01: Crear una carpeta que contendrá el proyecto

mkdir Laboratorio_03

mkdir Laboratorio_03/mssql cd Laboratorio_03/mssql

Paso 02: Crear un archivo app.py para contener la API y copiar y pegar el siguiente código

```
from flask import Flask, render_template, jsonify, request
from sqlalchemy import create_engine, text
from Crypto.Cipher import AES
from Crypto.Util.Padding import unpad
import base64

app = Flask(__name__)


# Clave de desencriptación (debe coincidir con la usada en SQL Server)
SYMMETRIC_KEY = b'lscck_04'[:32]

def decrypt_aes256(encrypted_data):
    """Desencripta un dato cifrado con AES-256"""
    try:
        cipher = AES.new(SYMMETRIC_KEY, AES.MODE_ECB)  # Modo ECB usado como ejemplo
```

```
        decrypted_data = unpad(cipher.decrypt(base64.b64decode(encrypted_data)),
AES.block_size)
        return decrypted_data.decode('utf-8')
    except Exception as e:
        print(f"Error en desencriptación: {e}")
        return None

@app.route('/get_card/<customer_id>', methods=['GET'])
def get_card(customer_id):
    """Endpoint para obtener y desencriptar tarjetas de un cliente"""
    with engine.connect() as conn:
        # Recupera los datos cifrados
        result = conn.execute(text("""
            OPEN SYMMETRIC KEY lscck_04 DECRYPTION BY CERTIFICATE secure_credit_cards;
            SELECT creditCard, CONVERT(VARCHAR, DecryptByKey(encryptedCC)) as
decryptedCC
            FROM cards WHERE customer = :customer_id;
            CLOSE SYMMETRIC KEY lscck_04;
        """), {"customer_id": customer_id})
        cards = [{"creditCard": row["creditCard"], "decryptedCC": row["decryptedCC"]}
for row in result]

    return jsonify(cards)

@app.route('/html_cards/<customer_id>')
def html_cards(customer_id):
    """Muestra las tarjetas en un archivo HTML"""
    response = get_card(customer_id)
    cards = response.json
    return render_template('cards.html', cards=cards)

if __name__ == "__main__":
    app.run(debug=True, port=5000)
```

Paso 02: Instalar las librerías

pip install flask flask-sqlalchemy pycryptodome

Paso 03: Crear una carpeta que contendrá el proyecto
mkdir Laboratorio_03/mssql/templates cd Laboratorio_03/ mssql/templates
Paso 04: Crear un archivo cards.html para contener la API y copiar y pegar el siguiente código

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tarjetas de Cliente</title>
</head>
<body>
    <h1>Tarjetas de Cliente</h1>
    <table border="1">
        <thead>
            <tr>
                <th>Número de Tarjeta</th>
                <th>Tarjeta Desencriptada</th>
            </tr>
        </thead>
        <tbody>
            {% for card in cards %}
            <tr>
                <td>{{ card['creditCard'] }}</td>
                <td>{{ card['decryptedCC'] }}</td>
            </tr>
            {% endfor %}
        </tbody>
    </table>
</body>
</html>
```

Paso 05: Ejecutar con **python app.py**

Paso 06: Copiar y pegar en Postman
http://127.0.0.1:5000/get_card?customer=605960578

Problema 2

En los scripts entregados por su profesor para examinar los procesos de protección de datos, por medios de procesos hash, cifrado por llave o cifrado por frase, investigue en al menos dos motores de datos adicionales realizar los mismos procesos de forma que pueda crear una comparativa entre los 3 motores de datos (SQL-Server y 2 más), abarcando el proceso como tal, instrucciones, formatos de salida.

Además, se requiere que entregue los scripts de dichos procesos.

Api Rest Flask con Mysql

Paso 01: Crear una carpeta que contendrá el proyecto
mkdir Laboratorio_03/mysql cd Laboratorio_03/mssql
Paso 02: Crear un archivo app.py para contener la API y copiar y pegar el siguiente código

```
from flask import Flask, request, render_template
import mysql.connector

app = Flask(__name__)

# Configuración de conexión a la base de datos
DB_CONFIG = {
    'host': 'localhost',       # Cambiar por el host de tu base de datos
    'user': 'cajero',      # Usuario de MySQL
    'password': 1234,  # Contraseña de MySQL
    'database': 'db_bank'      # Nombre de la base de datos
}

def get_db_connection():
    """Establece la conexión a la base de datos."""
    return mysql.connector.connect(**DB_CONFIG)

@app.route('/get_card', methods=['GET'])
def get_card():
    """Endpoint para obtener y desencriptar tarjetas."""
    customer_id = request.args.get('customer')  # Obtener el parámetro de la URL

    if not customer_id:
        return "El parámetro 'customer' es obligatorio.", 400

    try:
        # Conexión a la base de datos
```

```
        conn = get_db_connection()
        cursor = conn.cursor(dictionary=True)

        # Consulta para desencriptar las tarjetas
        query = """
            SELECT
                c.creditCard AS creditCard,
                AES_DECRYPT(c.encryptedCC, 'my_secret_key') AS decryptedCC
            FROM cards c
            WHERE c.customer_id = %s;
        """
        cursor.execute(query, (customer_id,))

        # Obtener resultados
        rows = cursor.fetchall()
        cards = [{"creditCard": row["creditCard"], "decryptedCC":
row["decryptedCC"].decode()} for row in rows]

        cursor.close()
        conn.close()

        # Si no hay resultados
        if not cards:
            return render_template('cards.html', customer_id=customer_id, cards=None)

        # Renderizar resultados en HTML
        return render_template('cards.html', customer_id=customer_id, cards=cards)

    except Exception as e:
        return f"Error al consultar la base de datos: {str(e)}", 500

if __name__ == '__main__':
    app.run(debug=True, port=5000)
```

Paso 03: Instalar las librerías

pip install flask mysql-connector-python

Paso 04: Crear una carpeta que contendrá el proyecto

mkdir Laboratorio_03/mssql/templates cd Laboratorio_03/mssql/templates

Paso 05: Crear un archivo cards.html para contener la API y copiar y pegar el siguiente código

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Tarjetas del Cliente</title>
```

```
</head>
<body>
    <h1>Tarjetas del Cliente: {{ customer_id }}</h1>
    {% if cards %}
        <table border="1">
            <thead>
                <tr>
                    <th>Tarjeta de Crédito</th>
                    <th>Tarjeta Desencriptada</th>
                </tr>
            </thead>
            <tbody>
                {% for card in cards %}
                    <tr>
                        <td>{{ card.creditCard }}</td>
                        <td>{{ card.decryptedCC }}</td>
                    </tr>
                {% endfor %}
            </tbody>
        </table>
    {% else %}
        <p>No se encontraron tarjetas para el cliente {{ customer_id }}.</p>
    {% endif %}
</body>
</html>
```

Paso 05: Base de datos en Mysql

```
-- Crear la base de datos
CREATE DATABASE db_bank;
USE db_bank;

-- Crear la tabla de clientes
CREATE TABLE customer (
    cedcustomer VARCHAR(20) NOT NULL,
    nombre VARCHAR(30),
    correo VARCHAR(30),
    PRIMARY KEY (cedcustomer)
);

-- Crear la tabla de tarjetas
CREATE TABLE cards (
    customer VARCHAR(20) NOT NULL,
    creditCard VARCHAR(25) NOT NULL,
    encryptedCC VARBINARY(250),
    PRIMARY KEY (creditCard),
```

```
    FOREIGN KEY (customer) REFERENCES customer(cedcustomer)
);

-- Insertar datos en la tabla de clientes
INSERT INTO customer (cedcustomer, nombre, correo)
VALUES ('605960578', 'Juanito', 'juani17@gmail.com');

-- Insertar datos en la tabla de tarjetas
-- En MySQL utilizamos AES_ENCRYPT para cifrar los datos
INSERT INTO cards (customer, creditCard, encryptedCC)
VALUES (
    '605960578',
    '6042210012564010',
    AES_ENCRYPT('6042210012564010', 'my_secret_key')
);

-- Consultar datos desencriptados
-- Utilizamos AES_DECRYPT para desencriptar
SELECT
    cd.creditCard,
    AES_DECRYPT(cd.encryptedCC, 'my_secret_key') AS decryptedCard
FROM cards cd
INNER JOIN customer c ON cd.customer = c.cedcustomer
WHERE cd.customer = '605960578';
```

Paso 05: Ejecutar con **python app.py**

Paso 06: Copiar y pegar en Postman
http://127.0.0.1:5000/get_card?customer=605960578

- **Evaluación.**

Ítem	Concepto	Puntos
1-1	Base de datos propuesta para el registro de tarjetas y propietarios	15
1-2	Aplicación de consultas.	10
1-3	Proceso de cifrado de datos de acorde al estándar PCI	20
1-4	Se realizan las consultas considerando los principios de eficiencia y rendimiento.	20
2-0	Aplicar los procesos de cifrado de datos vistos en clases en dos motores de datos diferentes al Microsoft SQL-Server	20
4	Realizar comparativa entre motores de datos.	15
	Total de Puntos	**100**