



ÍNDICE

| | |
|--|-----------|
| ÍNDICE | 2 |
| 1.ENCRYPTACIÓN | 3 |
| 1.1 PROCEDIMIENTOS ALMACENADOS | 3 |
| 1.2 TRIGGERS TDE | 4 |
| 1.3 VIEWS | 7 |
| 1.4 FUNCIONES | 8 |
| 2.ENCRYPTBYPASSPHRASE | 10 |
| 3.ENCRYPTACIÓN DE COLUMNAS DE BASE DE DATOS | 12 |
| 4.BACKUP ENCRYPTADO | 15 |
| 4.1 GENERANDO BACKUP DATABASE | 15 |
| 4.2 RESTORE BACKUP DATABASE | 19 |
| 5.DATA MASKING | 20 |
| TIPOS DE ENMASCARAMIENTO | 22 |
| 5.1 FUNCIÓN DEFAULT | 22 |
| 5.2 PARTIAL DATAMASKING | 23 |
| 5.3 RANDOM MASK | 24 |
| 5.4 CUSTOM STRING DYNAMIC DATA | 25 |
| 5.5 DATE TIME (SQL SERVER 2022 (16.x)) | 26 |
| 5.6 UNMASK | 27 |
| 5.7 PERMISOS GRANULARES SQL SERVER 2022 (16.x) | 28 |
| 6.0 ROW ENCRYPTION (ROW-LEVEL SECURITY) | 30 |
| 7.ALWAYS ENCRYPTED | 36 |
| 8.AUDITORIA | 39 |
| 8.1 A NIVEL DE INSTANCIA | 39 |
| 8.1.1 APPLICATION LOG | 39 |
| 7.1.2 SECURITY LOG | 40 |
| 7.1.3 FILE LOG | 41 |
| 8.2 A NIVEL DE SERVIDOR | 43 |

1. ENCRYPTACIÓN

1.1 PROCEDIMIENTOS ALMACENADOS

[Volver al índice →](#)

Los procedimientos almacenados son fragmentos de código, los cuales pueden ser invocados desde una aplicación o desde otra parte de la base de datos, son capaces de mejorar la eficiencia en la búsqueda de información, facilitan el mantenimiento de las bases de datos, pueden también ayudar a controlar el acceso a las bases de datos, entre otras funciones.

Haciendo referencia a nuestro proyecto, teniendo en cuenta que almacenamos información referente a clientes y las tarjetas con las que pagan los mismos, creamos un procedimiento almacenado que busque o enlace rápidamente al cliente con sus tarjetas almacenadas en nuestra base de datos.

Creamos el procedimiento almacenado, y hacemos una ejecución del mismo, sin "WITH ENCRYPTION" para visualizar como sería el resultado sin estar encriptado y luego estando encriptado.

```
--PROCEDIMIENTO ALMACENADO ENCRYPTADO--  
  
--CREAMOS EL PROCEDIMIENTO ALMACENADO--  
  
CREATE PROCEDURE BUSCAR_TARJETA_DE_CLIENTE  
    @ID_Cliente INT  
AS  
BEGIN  
    SELECT ID_TARJETA, NUMERO, CVV, Fecha_Caducidad, ID_Cliente  
    FROM Tarjeta  
    WHERE ID_Cliente = @ID_Cliente  
END  
  
DROP PROCEDURE BUSCAR_TARJETA_DE_CLIENTE  
GO  
  
EXEC BUSCAR_TARJETA_DE_CLIENTE @ID_Cliente = 1  
GO
```

Ahora ejecutamos el mismo script pero está indicando WITH ENCRYPTION

Comprobamos con el comando `SP_HELPTEXT` y nos indica que el mismo se encuentra encriptado.

```

--PROCEDIMIENTO ALMACENADO ENCRYPTADO-
--CREAMOS EL PROCEDIMIENTO ALMACENADO--

CREATE PROCEDURE BUSCAR_TARJETA_DE_CLIENTE
    @ID_Cliente INT
WITH ENCRYPTION
AS
BEGIN
    SELECT ID_TARJETA, NUMERO, CVV, Fecha_Caducidad, ID_Cliente
    FROM Tarjeta
    WHERE ID_Cliente = @ID_Cliente
END

DROP PROCEDURE BUSCAR_TARJETA_DE_CLIENTE

```

10 %

Messages

The text for object 'BUSCAR_TARJETA_DE_CLIENTE' is encrypted.

Completion time: 2023-03-10T01:43:54.6099119+01:00

1.2 TRIGGERS TDE

1.2.1 ENCRYPTADO

[Volver al índice →](#)

Los triggers son objetos de programación utilizados para automatizar tareas y aplicar reglas a eventos específicos, la encriptación de los mismos proporciona una capa adicional de seguridad puesto que los usuarios no autorizados no podrán visualizar o modificar su lógica de funcionamiento.

Los mismos pueden ser encriptados con el uso de funciones hash, cifrados simétricos o asimétricos.

```
--TRIGGER ENCRYPTADO--
```

```
-- CREAMOS LA MASTERKEY A NIVEL SERVIDOR--
```

```
--PARA ELLO NOS ASEGURAMOS DE ENCONTRARNOS EN MASTER--
```

```
USE MASTER;
```

```
GO
```

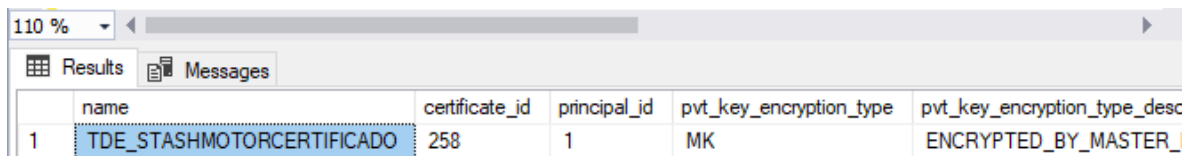
```
CREATE MASTER KEY ENCRYPTION
BY PASSWORD='STASHTRIGGERDB';
GO
```

```
--GRAMOS EL CERTIFICADO UTILIZANDO MSSQL--
```

```
CREATE CERTIFICATE TDE_STASHMOTORCERTIFICADO
WITH
SUBJECT='TRIGGERDB DATABASE ENCRYPTION';
GO
```

Comprobamos la correcta creación del certificado.

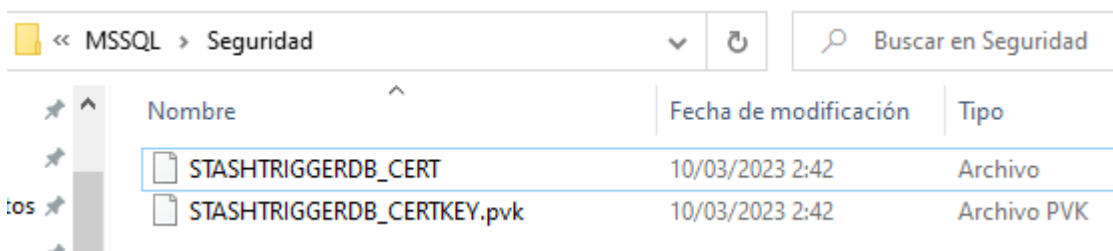
```
SELECT TOP 1 *
FROM sys.certificates
ORDER BY name DESC
GO
```



| | name | certificate_id | principal_id | pvt_key_encryption_type | pvt_key_encryption_type_desc |
|---|---------------------------|----------------|--------------|-------------------------|------------------------------|
| 1 | TDE_STASHMOTORCERTIFICADO | 258 | 1 | MK | ENCRYPTED_BY_MASTER_ |

Creamos el Backup del certificado. Previamente a ello debemos asegurarnos de poseer los permisos son la carpeta destino, ya que esto puede generar problemas.

```
BACKUP CERTIFICATE TDE_STASHMOTORCERTIFICADO
TO FILE = 'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\Seguridad\STASHTRIGGERDB_CERT'
WITH PRIVATE KEY (FILE='C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\Seguridad\STASHTRIGGERDB_CERTKEY.pvk',
ENCRYPTION BY PASSWORD='STASHTRIGGERDB')
GO
```



| Nombre | Fecha de modificación | Tipo |
|----------------------------|-----------------------|-------------|
| STASHTRIGGERDB_CERT | 10/03/2023 2:42 | Archivo |
| STASHTRIGGERDB_CERTKEY.pvk | 10/03/2023 2:42 | Archivo PVK |

[Volver al índice →](#)

--CREAMOS LA BASE DE DATOS--

--VERIFICAMOS LA EXISTENCIA PREVIAMENTE DE LA BASE DE DATOS--

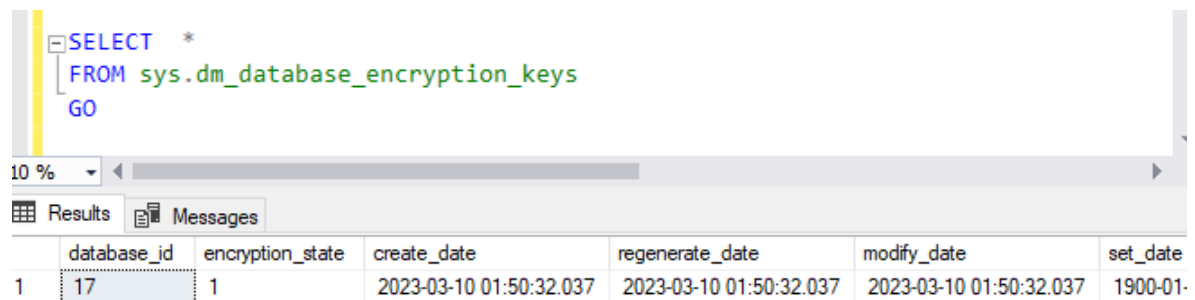
```
DROP DATABASE IF EXISTS STASHMOTOR_ENCRYPT_TDE
GO
```

```
CREATE DATABASE STASHMOTOR_ENCRYPT_TDE
GO
```

```
USE STASHMOTOR_ENCRYPT_TDE
GO
```

--ENCENDIDO DE ENCRIPCIÓN--

```
CREATE DATABASE ENCRYPTION KEY
    WITH ALGORITHM = AES_256
    ENCRYPTION BY SERVER CERTIFICATE TDE_STASHMOTORCERTIFICADO;
GO
```



The screenshot shows a SQL query window with the following text:

```
SELECT *
FROM sys.dm_database_encryption_keys
GO
```

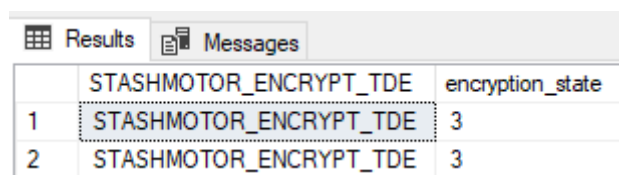
Below the query window, the 'Results' tab is active, displaying a table with the following data:

| | database_id | encryption_state | create_date | regenerate_date | modify_date | set_date |
|---|-------------|------------------|-------------------------|-------------------------|-------------------------|----------|
| 1 | 17 | 1 | 2023-03-10 01:50:32.037 | 2023-03-10 01:50:32.037 | 2023-03-10 01:50:32.037 | 1900-01- |

```
ALTER DATABASE STASHMOTOR_ENCRYPT_TDE SET ENCRYPTION ON;
GO
```

--VERIFICAMOS QUE SE ENCUENTRE REALMENTE ENCRIPADA--

```
SELECT DB_Name(17) AS 'STASHMOTOR_ENCRYPT_TDE', encryption_state
FROM sys.dm_database_encryption_keys;
GO
```



The screenshot shows a SQL query window with the following text:

```
SELECT DB_Name(17) AS 'STASHMOTOR_ENCRYPT_TDE', encryption_state
FROM sys.dm_database_encryption_keys;
GO
```

Below the query window, the 'Results' tab is active, displaying a table with the following data:

| | STASHMOTOR_ENCRYPT_TDE | encryption_state |
|---|------------------------|------------------|
| 1 | STASHMOTOR_ENCRYPT_TDE | 3 |
| 2 | STASHMOTOR_ENCRYPT_TDE | 3 |

Creamos un backup de nuestra base de datos

```
--CREAMOS EL BACKUP DE LA BASE DE DATOS--

BACKUP DATABASE STASHMOTOR_ENCRYPT_TDE
TO DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\Seg
GO
```

10 %

Messages

Processed 528 pages for database 'STASHMOTOR_ENCRYPT_TDE', file 'STASHMOTOR_ENCRYPT_TDE'
 Processed 1 pages for database 'STASHMOTOR_ENCRYPT_TDE', file 'STASHMOTOR_ENCRYPT_TDE'
 BACKUP DATABASE successfully processed 529 pages in 0.340 seconds (12.136 MB/sec).

Completion time: 2023-03-10T03:05:35.2104341+01:00

| << MSSQL > Seguridad > BACKUPTDE | | | |
|----------------------------------|-----------------------|----|--|
| Buscar en | | | |
| Nombre | Fecha de modificación | Ti | |
| STASHMOTOR_TDE_Full.bak | 10/03/2023 3:05 | A | |

1.3 VIEWS

[Volver al índice →](#)

Las vistas son consultas predefinidas que permiten a los usuarios acceder a una selección o búsqueda específica de información, encriptar las mismas puede ayudar a que ciertos datos vulnerables no sean visualizados.

Continuamos trabajando con las tablas Cliente y Tarjeta, en la cual, encriptamos el campo CVV, de manera que los usuarios que la ejecuten no sean capaces de ver el contenido del campo.

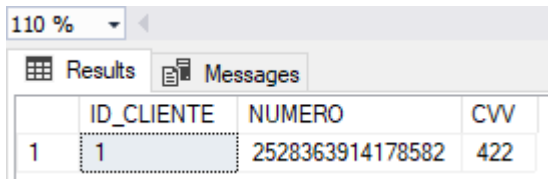
```
--VISTA ENCRYPTADA--
--HACIENDO USO DE LA TABLA CLIENTE Y TARJETA--
--CREAMOS LA VISTA QUE SOLO NOS MUESTRE EL ID DEL CLIENTE NUMERO, Y CVV
(ENCRYPTADO)--
```

```
--CREAMOS LA VISTA SIN ENCRYPTAR--
```

```
CREATE VIEW CLIENTE_TARJETA AS
SELECT ID_CLIENTE, NUMERO, CVV
FROM TARJETA;
GO
```

--HACEMOS UN SELECT PARA VER LOS CAMPOS--

```
SELECT * FROM CLIENTE_TARJETA
GO
```



| | ID_CLIENTE | NUMERO | CVV |
|---|------------|------------------|-----|
| 1 | 1 | 2528363914178582 | 422 |

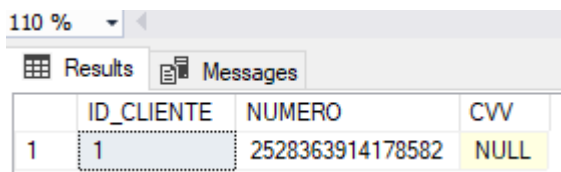
Procedemos a eliminar la vista anterior

--BORRAMOS LA VISTA ANTERIOR--

```
DROP VIEW CLIENTE_TARJETA
GO
```

--CREAMOS LA VISTA ENCRIPTADA--

```
CREATE VIEW CLIENTE_TARJETA AS
SELECT ID_CLIENTE, NUMERO, CONVERT(VARCHAR(50),
ENCRYPTBYKEY(KEY_GUID('STASHMOTOR'), CVV)) AS CVV
FROM TARJETA;
GO
```



| | ID_CLIENTE | NUMERO | CVV |
|---|------------|------------------|------|
| 1 | 1 | 2528363914178582 | NULL |

1.4 FUNCIONES

[Volver al índice →](#)

Las funciones son un objeto de base de datos que puede aceptar uno o más valores/parámetros de entrada, realizar un cálculo o una operación en base a esos valores, y luego devolver un resultado.

Las funciones se utilizan para encapsular la lógica del cálculo o la operación en un objeto reutilizable que se puede llamar desde otras partes del código.

Inicialmente crearemos una funcion sin encriptar para mostrar el comportamiento y resultado de la misma.


```
--FUNCION ENCRIPTADA--
```

```
--UTILIZAMOS LA TABLA CLIENTE Y TARJETA COMO REFERENCIA--
```

```
CREATE FUNCTION FN_CLIENTE_TARJETA_INFO(@ID_CLIENTE VARCHAR(10))
RETURNS TABLE
AS
RETURN
(
    SELECT C.ID_CLIENTE, C.NOMBRE, C.APELLIDO, T.NUMERO, T.CVV
    FROM CLIENTE C
    INNER JOIN TARJETA T ON C.ID_CLIENTE = T.ID_CLIENTE
    WHERE C.ID_CLIENTE = @ID_CLIENTE
);
GO
```

```
SELECT * FROM FN_CLIENTE_TARJETA_INFO ('1');
```

| Results | | Messages | | | |
|---------|------------|----------|----------|------------------|-----|
| | ID_CLIENTE | NOMBRE | APELLIDO | NUMERO | CVV |
| 1 | 1 | Andres | Prieto | 2528363914178582 | 422 |

Repetiremos nuevamente el procedimiento anterior, pero esta vez encriptando el campo CVV

```
CREATE FUNCTION FN_CLIENTE_TARJETA_INFO(@ID_CLIENTE VARCHAR(10))
RETURNS TABLE
AS
RETURN
(
    SELECT C.ID_CLIENTE, C.NOMBRE, C.APELLIDO, T.NUMERO,
    CONVERT(VARCHAR(50), ENCRYPTBYKEY(KEY_GUID('SymmetricKey'), T.CVV)) AS
    CVV
    FROM CLIENTE C
    INNER JOIN TARJETA T ON C.ID_CLIENTE = T.ID_CLIENTE
    WHERE C.ID_CLIENTE = @ID_CLIENTE
);
GO
```

```
SELECT * FROM FN_CLIENTE_TARJETA_INFO ('1');
```

| Results | | Messages | | | |
|---------|------------|----------|----------|------------------|------|
| | ID_CLIENTE | NOMBRE | APELLIDO | NUMERO | CVV |
| 1 | 1 | Andres | Prieto | 2528363914178582 | NULL |

Dando como parámetro de entrada el valor "1" correspondiente al ID_CLIENTE, nos da como resultado todos los datos a excepción del CVV, comprobando que ha sido correctamente encriptado.

2.ENCRYPTBYPASSPHRASE

[Volver al índice →](#)

Es una función de cifrado en Microsoft SQL Server que se utiliza para cifrar datos utilizando una clave o frase de contraseña proporcionada por el usuario. La función acepta dos parámetros: la cadena de texto que se desea cifrar y la clave de cifrado que se utilizará para cifrar la cadena.

Continuamos trabajando con la mismas tablas CLIENTE, TARJETA de nuestro proyecto, pero esta vez hemos modificado la tabla TARJETA, añadiendo el elemento **VARBINARY** para poder realizar las demostraciones.

```
--ENCRYPTBYPASSPHRASE--
```

```
--CREAMOS LA TABLA TARJETA CON EL CAMPO CORRESPONDIENTE (VARBINARY) EL CUAL SERÁ ENCRIPADO--
```

```
--ELIMINAMOS LA TABLA TARJETA EXISTENTE--
```

```
DROP TABLE TARJETA
GO
```

```
CREATE TABLE TARJETA (
    ID_TARJETA INT IDENTITY (1,1),
    NUMERO VARBINARY(128),
    CVV VARBINARY (128),
    FECHA_CADUCIDAD DATE)
GO
```

```
--INSERTAMOS UN VALOR--
```

```
INSERT INTO TARJETA (NUMERO, CVV, FECHA_CADUCIDAD)
VALUES (
    CONVERT(VARBINARY(128), '2528363914178582'),
    ENCRYPTBYPASSPHRASE('STASHMOTOR_FRASE_SECRETA', '422'),
    '2025-06-25'
);
```

```
--HACEMOS UN SELECT SOBRE LA TABLA--
```

```
SELECT * FROM TARJETA
GO
```

| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
|---|------------|------------------------------------|--|-----------------|
| 1 | 1 | 0x32353238333633393134313738353832 | 0x0200000094DDBC600B3D862E9CF6CFB4720DBC0913874C7... | 2025-06-25 |

Para poder visualizar el contenido de la misma, ejecutaremos la siguiente instrucción, previamente a conseguir el resultado deseado, que en este caso sería desenscriptar los datos y poder visualizar el contenido del mismo, demostraremos que ingresando una “frase” incorrecta, no seremos capaces.

```
--DESENCRIPTAR EL CONTENIDO--
```

```
--PRIMERAMENTE INTENTAMOS HACERLO CON UNA FRASE INCORRECTA, PARA DEMOSTRAR QUE FUNCIONA CORRECTAMENTE--
```

```
SELECT ID_TARJETA, NUMERO, CONVERT(VARCHAR(50),
DECRYPTBYPASSPHRASE('NOSELACLAVE', CVV)) AS CVV, FECHA_CADUCIDAD
FROM TARJETA;
```

```
--NOS MUESTRA EL CONTENIDO NULL, PUESTO QUE LA FRASE ES INCORRECTA--
```

| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
|---|------------|------------------------------------|------|-----------------|
| 1 | 1 | 0x32353238333633393134313738353832 | NULL | 2025-06-25 |

```
--NOS MUESTRA EL CONTENIDO NULL, PUESTO QUE LA FRASE ES INCORRECTA--
```

```
--ESTA VEZ INSERTAMOS LA FRASE CORRECTA--
```

```
SELECT ID_TARJETA, NUMERO, CONVERT(VARCHAR(50),
DECRYPTBYPASSPHRASE('STASHMOTOR_FRASE_SECRETA', CVV)) AS CVV,
FECHA_CADUCIDAD
FROM TARJETA;
```

| Results | | Messages | | |
|---------|------------|------------------------------------|-----|-----------------|
| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
| 1 | 1 | 0x32353238333633393134313738353832 | 422 | 2025-06-25 |

3. ENCRIPCIÓN DE COLUMNAS DE BASE DE DATOS

[Volver al índice →](#)

El cifrado de columna en SQL es una técnica de seguridad utilizada para proteger la información sensible almacenada en una tabla de base de datos. Dicho cifrado implica convertir el valor de una columna en un formato ilegible y seguro antes de almacenarlo en la tabla, y luego descifrar cuando se lee de la tabla.

Trabajando con la tabla Cliente del proyecto, nos interesa que la columna de teléfono se encuentra encriptada, para ello debemos hacer lo siguiente.

```
--NOS ASEGURAMOS DE ESTAR EN MASTER--
```

```
USE MASTER
GO
```

```
--CREAMOS UN LOGIN ADMINISTRATIVO DE LA BASE DE DATOS--
```

```
CREATE LOGIN STASH_ADMIN WITH PASSWORD = 'STASH1234.'
GO
```

```
USE STASHMOTOR_ENCRYPT
GO
```

```
--CREAMOS EL USUARIO DENTRO DE LA BASE DE DATOS--
```

```
CREATE USER STASH_ADMIN FOR LOGIN STASH_ADMIN
GO
```

```
CREATE TABLE CLIENTE (
    ID_CLIENTE VARCHAR(10) PRIMARY KEY NOT NULL,
    NOMBRE VARCHAR(20),
    APELLIDO VARCHAR(20),
    EMAIL VARCHAR(50),
    TELEFONO VARBINARY(150))
GO
```

```
--OTORGAMOS PERMISOS AL USUARIO SELECT, INSERT, UPDATE, DELET EN LA TABLA  
CLIENTE A STASH_ADMIN--
```

```
GRANT SELECT, INSERT, UPDATE, DELETE ON CLIENTE TO STASH_ADMIN  
GO
```

```
--GENERAMOS LA LLAVE SIMÉTRICA PARA PROCEDER A ENCRIPTAR--
```

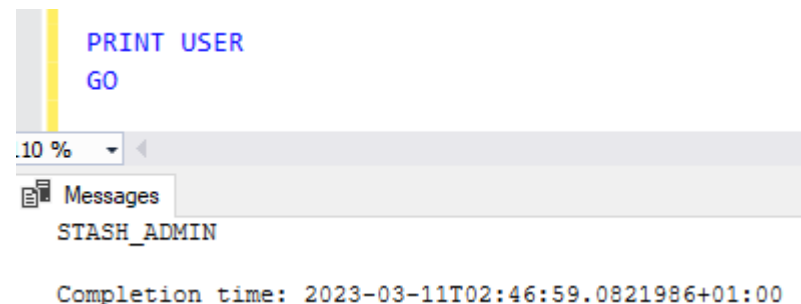
```
CREATE SYMMETRIC KEY STASH_ADMIN_KEY  
AUTHORIZATION STASH_ADMIN  
WITH ALGORITHM=AES_256  
ENCRYPTION BY PASSWORD='STASH1234.'  
GO
```

```
--IMPERSONAMOS COMO STASHADMIN
```

```
EXEC AS USER= 'STASH_ADMIN'  
GO
```

```
--UNA VEZ ESTANDO CON EL USUARIO STASH_ADMIN--  
--INSERTAMOS LOS DATOS EN LA TABLA CLIENTE--
```

```
PRINT USER  
GO
```



```
PRINT USER  
GO  
.  
10 %  
Messages  
STASH_ADMIN  
Completion time: 2023-03-11T02:46:59.0821986+01:00
```

```
OPEN SYMMETRIC KEY [STASH_ADMIN_KEY] DECRYPTION BY PASSWORD='STASH1234.'  
GO
```

```
INSERT INTO CLIENTE VALUES  
(4, 'Enrique', 'Gomez', 'kikegomez@gmail.com', EncryptByKey(Key_GUID('STASH_AD  
MIN_KEY'), '82'))  
INSERT INTO CLIENTE VALUES  
(5, 'Mercedes', 'Lopez', 'merche@gmail.com', EncryptByKey(Key_GUID('STASH_ADMI  
N_KEY'), '25702985'))  
INSERT INTO CLIENTE VALUES
```

```
(6, 'Sabrina', 'Prieto', 'sabriprieto@gmail.com', EncryptByKey(Key_GUID('STASH_ADMIN_KEY'), '75201098'))
```

GO

Al realizar la búsqueda, observamos que los datos de teléfono se encuentran encriptados.

```
SELECT * FROM CLIENTE
```

GO

110 %

Results Messages

| | ID_CLIENTE | NOMBRE | APELLIDO | EMAIL | TELEFONO |
|---|------------|----------|----------|-----------------------|---|
| 1 | 4 | Enrique | Gomez | kikegomez@gmail.com | 0x004BE678251435468CB98D5CCF0EE50102000000E84F69... |
| 2 | 5 | Mercedes | Lopez | merche@gmail.com | 0x004BE678251435468CB98D5CCF0EE50102000000DF0FF8... |
| 3 | 6 | Sabrina | Prieto | sabriprieto@gmail.com | 0x004BE678251435468CB98D5CCF0EE50102000000CD0674... |

Ahora procederemos a desencriptar los mismos.

```
SELECT ID_CLIENTE, NOMBRE + ' ' + APELLIDO AS [NOMBRE DEL CLIENTE],  
CONVERT(VARCHAR, DecryptByKey(TELEFONO)) as 'TELEFONO DEL CLIENTE'  
FROM CLIENTE
```

GO

```
SELECT ID_CLIENTE, NOMBRE + ' ' + APELLIDO AS [NOMBRE DEL CLIENTE],  
CONVERT(VARCHAR, DecryptByKey(TELEFONO)) as 'TELEFONO DEL CLIENTE'  
FROM CLIENTE
```

GO

110 %

Results Messages

| | ID_CLIENTE | NOMBRE DEL CLIENTE | TELEFONO DEL CLIENTE |
|---|------------|--------------------|----------------------|
| 1 | 4 | Enrique Gomez | 82 |
| 2 | 5 | Mercedes Lopez | 25702985 |
| 3 | 6 | Sabrina Prieto | 75201098 |

4.BACKUP ENCRYPTADO

[Volver al índice →](#)

4.1 GENERANDO BACKUP DATABASE

Un backup encriptado es una copia de seguridad de una base de datos que ha sido protegida mediante un algoritmo de cifrado, lo que significa que los datos contenidos en la copia de seguridad no son legibles a menos que se utilice la clave o contraseña correcta para descifrarlos.

La encriptación de la copia de seguridad es importante para proteger la información confidencial de la empresa, ya que en el caso de que la copia de seguridad caiga en manos equivocadas, los datos no podrán ser accedidos sin la clave de descifrado.

```
--BACKUP ENCRYPTADO--
```

```
--VERIFICAMOS LA EXISTENCIA DE LA BASE DE DATOS--
```

```
DROP DATABASE IF EXISTS STASH_BACKUP_ENCRYPTED  
GO
```

```
--CREAMOS LA BASE DE DATOS--
```

```
CREATE DATABASE STASH_BACKUP_ENCRYPTED  
GO
```

```
USE STASH_BACKUP_ENCRYPTED  
GO
```

```
CREATE TABLE EMPLEADO (  
    ID_EMPLEADO INT IDENTITY (1,1),  
    NOMBRE VARCHAR(50),  
    APELLIDO VARCHAR (50),  
    FECHA_INGRESO DATE,  
    EMAIL VARCHAR (50),  
    TELEFONO VARCHAR (20)  
)  
GO
```

```
--CREAMOS LA LLAVE--
```

```
--NOS ASEGURAMOS DE ENCONTRARNOS EN MASTER--
```

```
USE MASTER  
GO
```

```
CREATE MASTER KEY ENCRYPTION BY PASSWORD='STASH1234.'  
GO
```

```
-- CREAMOS UN CERTIFICADO--
```

```
CREATE CERTIFICATE STASH_MOTOR_CERT  
WITH SUBJECT = 'Certificado para Backup'  
GO
```

```
--VERIFICAMOS LOS CERTIFICADOS--
```

```
SELECT TOP 2 NAME, pvt_key_encryption_type, issuer_name FROM  
SYS.certificates  
ORDER BY NAME DESC  
GO
```

```
--VERIFICAMOS LOS CERTIFICADOS--  
SELECT TOP 2 NAME, pvt_key_encryption_type, issuer_name FROM SYS.certificates  
ORDER BY NAME DESC  
GO
```

10 %

Results Messages

| | NAME | pvt_key_encryption_type | issuer_name |
|---|---------------------------|-------------------------|-------------------------------|
| 1 | TDE_STASHMOTORCERTIFICADO | MK | TRIGGERDB DATABASE ENCRYPTION |
| 2 | STASH_MOTOR_CERT | MK | Certificado para Backup |


```
--VOLVEMOS A LA BASE DE DATOS--

USE STASH_BACKUP_ENCRYPTED
GO

CREATE DATABASE ENCRYPTION KEY
WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE STASH_MOTOR_CERT;
GO
```

0 %

Messages

Warning: The certificate used for encrypting the database encryption key

Completion time: 2023-03-11T03:21:36.3251415+01:00

--ACTIVAMOS LA ENCRPTION--

```
ALTER DATABASE STASH_BACKUP_ENCRYPTED
SET ENCRYPTION ON;
GO
```

```
USE master
GO
```

--INDICAMOS LA RUTA DONDE DESEAMOS REALIZAR EL BACKUP Y SU VEZ--
--DEBEMOS INDICAR EL CERTIFICADO--

```
BACKUP DATABASE STASH_BACKUP_ENCRYPTED
TO DISK = 'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\Seguridad\BACKUP_ENCRPTADO\STASH_BACKUP_
ENCRYPTED.bak'
WITH
ENCRYPTION
(
ALGORITHM = AES_256,
SERVER CERTIFICATE = STASH_MOTOR_CERT
),
STATS = 10,INIT
GO
```

Messages

Warning: The certificate used for encrypting the database encryption key has not been backed up.
Msg 3201, Level 16, State 1, Line 70
Cannot open backup device 'C:\Users\AndresPrieto\Documents\SQL Server Management Studio\SEGURIDA
Msg 3013, Level 16, State 1, Line 70
BACKUP DATABASE is terminating abnormally.

Completion time: 2023-03-11T03:26:09.5743827+01:00

Para solucionar el error hemos creado un backup del certificado

--PARA PODER SOLUCIONAR ESTO, DEBEMOS REALIZAR UNA COPIA DEL CERTIFICADO--

```
BACKUP CERTIFICATE STASH_MOTOR_CERT
TO FILE = 'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\Seguridad\STASH_MOTOR_CERT'
WITH PRIVATE KEY (FILE = 'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\Seguridad\STASH_MOTOR_CERTKEY.pvk',
ENCRYPTION BY PASSWORD = 'STASH1234.')
GO
```

Este equipo > Disco local (C:) > Archivos de programa > Microsoft SQ

| Nombre | Fecha de modificac |
|----------------------------|--------------------|
| BACKUPTDE | 10/03/2023 3:05 |
| STASH_MOTOR_CERT | 11/03/2023 3:39 |
| STASH_MOTOR_CERTKEY.pvk | 11/03/2023 3:39 |
| STASHTRIGGERDB_CERT | 10/03/2023 2:42 |
| STASHTRIGGERDB_CERTKEY.pvk | 10/03/2023 2:42 |

Ejecutamos nuevamente el script para generar el backup

```
--EJECUTAMOS NUEVAMENTE EL SCRIPT--

BACKUP DATABASE STASH_BACKUP_ENCRYPTED
TO DISK = 'C:\Program Files\Microsoft SQL Server\MSSQL16.MSSQLSERVER\MSSQL\Seguridad\BACKUP_ENCRYPTED'
WITH
ENCRYPTION
(
ALGORITHM = AES_256,
SERVER CERTIFICATE = STASH_MOTOR_CERT
),
STATS = 10,INIT
GO
```

10 %

Messages

```
10 percent processed.
21 percent processed.
30 percent processed.
41 percent processed.
51 percent processed.
60 percent processed.
71 percent processed.
80 percent processed.
90 percent processed.
100 percent processed.
Processed 384 pages for database 'STASH_BACKUP_ENCRYPTED', file 'STASH_BACKUP_ENCRYPTED' on file 1.
Processed 1 pages for database 'STASH_BACKUP_ENCRYPTED', file 'STASH_BACKUP_ENCRYPTED_log' on file 1.
BACKUP DATABASE successfully processed 385 pages in 0.348 seconds (8.624 MB/sec).

Completion time: 2023-03-11T03:43:32.3533999+01:00
```

| | | | |
|--|----------------------------|-----------------------|-------------|
| Este equipo > Disco local (C:) > Archivos de programa > Microsoft SQL Server > MSSQL16.MSS | | | |
| | Nombre | Fecha de modificación | Tipo |
| ido | STASH_BACKUP_ENCRYPTED.bak | 11/03/2023 3:43 | Archivo BAK |

4.2 RESTORE BACKUP DATABASE

[Volver al índice →](#)

--RESTAURANDO BACKUP--

```
RESTORE DATABASE STASH_BACKUP_ENCRYPTED
FROM DISK = 'C:\Program Files\Microsoft SQL
Server\MSSQL16.MSSQLSERVER\MSSQL\Seguridad\BACKUP_ENCRIPTADO\STASH_BACKUP_
ENCRYPTED.bak'
WITH FILE = 1, NOUNLOAD, STATS = 5;
GO
```

```
Messages
80 percent processed.
86 percent processed.
90 percent processed.
95 percent processed.
100 percent processed.
Processed 384 pages for database 'STASH_BACKUP_ENCRYPTED', file 'STASH_BACKUP_ENCRYPTED' on file 1.
Processed 1 pages for database 'STASH_BACKUP_ENCRYPTED', file 'STASH_BACKUP_ENCRYPTED_log' on file 1.
RESTORE DATABASE successfully processed 385 pages in 0.100 seconds (30.014 MB/sec).

Completion time: 2023-03-11T03:52:22.8692703+01:00
```

5.DATA MASKING

[Volver al índice →](#)

Es una técnica de seguridad que consiste en ocultar, enmascarar o modificar información sensible en una base de datos, con la finalidad de protegerla ante accesos no autorizados.

--VERIFICAMOS LA EXISTENCIA DE LA BASE DE DATOS--

```
DROP DATABASE IF EXISTS STASH_DATA_MASKING
GO
```

-- CREAMOS LA BASE DE DATOS --

```
CREATE DATABASE STASH_DATA_MASKING
GO
```

```
USE STASH_DATA_MASKING
GO
```

-- CREMOS LA TABLA TARJETA -

```
CREATE TABLE TARJETA (
    ID_TARJETA INT IDENTITY (1,1) PRIMARY KEY,
    NUMERO bigint,
    CVV bigint ,
    FECHA_CADUCIDAD DATE )
GO
```

--INSERTAMOS VALORES DENTRO DE LA TABLA--

```
INSERT INTO TARJETA (NUMERO, CVV, FECHA_CADUCIDAD) VALUES
    (2514639852148749 , 422 , '2025-06-08'),
    (6987542685743694, 688 , '2026-07-17'),
    (5201487423145896, 356 , '2027-04-30'),
    (5847362871874536, 598, '2026-05-24'),
    (3627145820305897, 102, '2024-02-25')
GO
```

```
SELECT * FROM TARJETA
GO
```

110 %

Results Messages

| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
|---|------------|------------------|-----|-----------------|
| 1 | 1 | 2514639852148749 | 422 | 2025-06-08 |
| 2 | 2 | 6987542685743694 | 688 | 2026-07-17 |
| 3 | 3 | 5201487423145896 | 356 | 2027-04-30 |
| 4 | 4 | 5847362871874536 | 598 | 2026-05-24 |
| 5 | 5 | 3627145820305897 | 102 | 2024-02-25 |

--CREAMOS EL USUARIO CONTABILIDAD, EL CUAL TENDRÁ PERMISO DE SELECT--

```
DROP USER IF EXISTS CONTABILIDAD
GO
```

```
CREATE USER CONTABILIDAD WITHOUT LOGIN
GO
```

--OTORGAMOS PERMISOS DE SELECT AL USUARIO--

```
GRANT SELECT ON TARJETA TO CONTABILIDAD
GO
```

--CONSULTA DEL CATÁLOGO DE VISTAS DEL SISTEMA--

--PARA OBTENER INFORMACION SOBRE LAS COLUMNAS ENMASCARADAS--

```
SELECT c.name, tbl.name as table_name, c.is_masked, c.masking_function
      FROM sys.masked_columns AS c
      JOIN sys.tables AS tbl
      ON c.[object_id] = tbl.[object_id]
      WHERE is_masked = 1;
GO
```

--PARA SIMPLIFICAR LAS SIGUIENTES DEMOSTRACIONES--

--CREAMOS UN PROCEDIMIENTO ALMACENADO, CON LA FINALIDAD DE FACILITAR LAS DEMOSTRACIONES--

```
CREATE OR ALTER PROCEDURE MASK_STATUS
AS
BEGIN
      SELECT c.name, tbl.name as table_name, c.is_masked,
c.masking_function
      FROM sys.masked_columns AS c
      JOIN sys.tables AS tbl
      ON c.[object_id] = tbl.[object_id]
```

```
WHERE is_masked = 1;
END
GO

--PROBAMOS AL PROCEDIMIENTO ALMACENADO--

EXEC MASK_STATUS
GO
```

TIPOS DE ENMASCARAMIENTO

[Volver al índice →](#)

Para la elaboración y demostración de los tipos de enmascaramiento que se efectuarán a continuación, hemos creado en la base de datos un **usuario "CONTABILIDAD"** al cual solo se le han otorgado permisos de SELECT. Por consiguiente, dicho usuario a diferencia de **"DBO"** al realizar consultas sobre la tabla, verá los campos con sus respectivos campos enmascarados.

5.1 FUNCIÓN DEFAULT

[Volver al índice →](#)

Consiste en sustituir los valores reales de la tabla, por una cadena de caracteres proporcionales a la longitud del contenido (xxx)

En este primer caso, trabajaremos con el campo "CVV" de la tabla TARJETA

--FUNCION DEFAULT--

```
ALTER TABLE TARJETA
ALTER COLUMN CVV VARCHAR(20) MASKED WITH (FUNCTION = 'default()');
GO
```

Usuario dbo

| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
|---|------------|------------------|-----|-----------------|
| 1 | 1 | 2514639852148749 | 422 | 2025-06-08 |
| 2 | 2 | 6987542685743694 | 688 | 2026-07-17 |
| 3 | 3 | 5201487423145896 | 356 | 2027-04-30 |
| 4 | 4 | 5847362871874536 | 598 | 2026-05-24 |
| 5 | 5 | 3627145820305897 | 102 | 2024-02-25 |

Usuario Contabilidad

110 %

| Results | | Messages | | |
|---------|------------|------------------|------|-----------------|
| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
| 1 | 1 | 2514639852148749 | xxxx | 2025-06-08 |
| 2 | 2 | 6987542685743694 | xxxx | 2026-07-17 |
| 3 | 3 | 5201487423145896 | xxxx | 2027-04-30 |
| 4 | 4 | 5847362871874536 | xxxx | 2026-05-24 |
| 5 | 5 | 3627145820305897 | xxxx | 2024-02-25 |

| Results | | Messages | | |
|---------|------|------------|-----------|------------------|
| | name | table_name | is_masked | masking_function |
| 1 | CVV | TARJETA | 1 | default() |

5.2 PARTIAL DATAMASKING

[Volver al indice →](#)

Es un tipo de técnica de enmascaramiento de datos que oculta sólo una parte de los datos.

En este caso trabajaremos sobre el campo NÚMERO, enmascarando los primeros 12 dígitos de la tarjeta como es habitualmente, solo permitiendo ver los últimos 4 de la misma.

--FUNCTION PARTIAL--
--REALIZAMOS EL ENMASCARAMIENTO DE LOS PRIMEROS 12 DIGITOS DEL CAMPO
TARJETA--

```
ALTER TABLE TARJETA  
    ALTER COLUMN [NUMERO] VARCHAR(50) MASKED WITH (FUNCTION =  
'partial(0,"XXXXXXXXXXXX",4)')  
GO
```

Usuario dbo

| Results | | Messages | | |
|---------|------------|------------------|-----|-----------------|
| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
| 1 | 1 | 2514639852148749 | 422 | 2025-06-08 |
| 2 | 2 | 6987542685743694 | 688 | 2026-07-17 |
| 3 | 3 | 5201487423145896 | 356 | 2027-04-30 |
| 4 | 4 | 5847362871874536 | 598 | 2026-05-24 |
| 5 | 5 | 3627145820305897 | 102 | 2024-02-25 |

Usuario Contabilidad

| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
|---|------------|------------------|------|-----------------|
| 1 | 1 | XXXXXXXXXXXX8749 | xxxx | 2025-06-08 |
| 2 | 2 | XXXXXXXXXXXX3694 | xxxx | 2026-07-17 |
| 3 | 3 | XXXXXXXXXXXX5896 | xxxx | 2027-04-30 |
| 4 | 4 | XXXXXXXXXXXX4536 | xxxx | 2026-05-24 |
| 5 | 5 | XXXXXXXXXXXX5897 | xxxx | 2024-02-25 |

| | name | table_name | is_masked | masking_function |
|---|--------|------------|-----------|-------------------------------|
| 1 | NUMERO | TARJETA | 1 | partial(0, "XXXXXXXXXXXX", 4) |
| 2 | CVV | TARJETA | 1 | default() |

5.3 RANDOM MASK

[Volver al indice →](#)

Este tipo de técnica genera valores aleatorios como método de enmascaramiento.

Lo aplicaremos nuevamente sobre el campo CVV, el cual puede tener un mayor beneficio de seguridad, puesto que los número al generarse aleatoriamente el sistema, en caso de que la base de datos quede expuesta, si los campos se ven encriptados con xxx como el ejemplo anterior esto genera ruido, en cambio si tiene dígitos pueden llegar a suponer que esto ha sido un descuido y se encuentran expuestos.

```
--FUNCTION RANDOM--
--EN ESTE CASO APLICAMOS EL RANDOM MASK SOBRE EL CAMPO CVV-
--CON LA FINALIDAD DE QUE GENERE 3 DIGITOS ALEATORIOS--
--PARA DICHO CAMPO--

ALTER TABLE TARJETA
ALTER COLUMN CVV int MASKED WITH (FUNCTION = 'random(100, 999)')
GO
```

Usuario dbo

| Results | | Messages | | |
|---------|------------|------------------|-----|-----------------|
| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
| 1 | 1 | 2514639852148749 | 422 | 2025-06-08 |
| 2 | 2 | 6987542685743694 | 688 | 2026-07-17 |
| 3 | 3 | 5201487423145896 | 356 | 2027-04-30 |
| 4 | 4 | 5847362871874536 | 598 | 2026-05-24 |
| 5 | 5 | 3627145820305897 | 102 | 2024-02-25 |

Usuario Contabilidad

| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
|---|------------|------------------|-----|-----------------|
| 1 | 1 | XXXXXXXXXXXX8749 | 927 | 2025-06-08 |
| 2 | 2 | XXXXXXXXXXXX3694 | 998 | 2026-07-17 |
| 3 | 3 | XXXXXXXXXXXX5896 | 892 | 2027-04-30 |
| 4 | 4 | XXXXXXXXXXXX4536 | 246 | 2026-05-24 |
| 5 | 5 | XXXXXXXXXXXX5897 | 712 | 2024-02-25 |

| | name | table_name | is_masked | masking_function |
|---|--------|------------|-----------|-------------------------------|
| 1 | NUMERO | TARJETA | 1 | partial(0, "XXXXXXXXXXXX", 4) |
| 2 | CVV | TARJETA | 1 | random(100, 999) |

5.4 CUSTOM STRING DYNAMIC DATA

[Volver al indice →](#)

Este tipo de técnica genera una sustitución en una columna determinada , por una cadena de caracteres “string” definidas por el dbo.

Aplicaremos dicha técnica sobre el campo NÚMERO con la finalidad de sustituir los primeros 12 dígitos de la tarjeta por un mensaje “ACCESO RESTRINGIDO”

```
--FUNCTION STRING DYNAMIC DATA MASK--  
  
--EN ESTE CASO APLICAMOS EL RANDOM MASK SOBRE EL CAMPO NUMERO--  
--CON LA FINALIDAD DE SUSTITUIR LOS PRIMEROS 12 DIGITOS--  
--POR EL MENSAJE ACCESO RESTRINGIDO--
```

```
ALTER TABLE TARJETA  
    ALTER COLUMN [NUMERO] ADD MASKED WITH (FUNCTION =  
'partial(0,"ACCESO-RESTRINGIDO-",4)')  
GO
```

Usuario dbo

| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
|---|------------|------------------|-----|-----------------|
| 1 | 1 | 2514639852148749 | 422 | 2025-06-08 |
| 2 | 2 | 6987542685743694 | 688 | 2026-07-17 |
| 3 | 3 | 5201487423145896 | 356 | 2027-04-30 |
| 4 | 4 | 5847362871874536 | 598 | 2026-05-24 |
| 5 | 5 | 3627145820305897 | 102 | 2024-02-25 |

Usuario Contabilidad

| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
|---|------------|-------------------------|------|-----------------|
| 1 | 1 | ACCESO-RESTRINGIDO-8749 | xxxx | 2025-06-08 |
| 2 | 2 | ACCESO-RESTRINGIDO-3694 | xxxx | 2026-07-17 |
| 3 | 3 | ACCESO-RESTRINGIDO-5896 | xxxx | 2027-04-30 |
| 4 | 4 | ACCESO-RESTRINGIDO-4536 | xxxx | 2026-05-24 |
| 5 | 5 | ACCESO-RESTRINGIDO-5897 | xxxx | 2024-02-25 |

| | name | table_name | is_masked | masking_function |
|---|--------|------------|-----------|--------------------------------------|
| 1 | NUMERO | TARJETA | 1 | partial(0, "ACCESO-RESTRINGIDO-", 4) |
| 2 | CVV | TARJETA | 1 | default() |

5.5 DATE TIME (SQL SERVER 2022 (16.x))

[Volver al indice →](#)

Esta función permite ocultar uno o todos los campos de tipo date, correspondiente a una columna. Aplicaremos dicha técnica sobre el campo FECHA, con la finalidad de modificar el año del vencimiento de la misma, también podríamos hacerlo con los meses o los días de la misma.

```
--FUNCTION DATE TIME-- DISPONIBLE PARA LA VERSION SQL SERVER 2022 (16.x)
--LO APLICAREMOS AL CAMPO FECHA, MODIFICANDO EL CAMPO AÑO--
--PARA MODIFICAR EL AÑO--
ALTER TABLE TARJETA
ALTER COLUMN FECHA_CADUCIDAD ADD MASKED WITH (FUNCTION = 'datetime("Y")')
GO
--PARA MODIFICAR EL MES--
ALTER TABLE TARJETA
ALTER COLUMN FECHA_CADUCIDAD ADD MASKED WITH (FUNCTION = 'datetime("M")')
GO
--PARA MODIFICAR EL MES--
ALTER TABLE TARJETA
ALTER COLUMN FECHA_CADUCIDAD ADD MASKED WITH (FUNCTION = 'datetime("M")')
GO
```

Usuario dbo

| Results Messages | | | | |
|------------------|------------|------------------|-----|-----------------|
| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
| 1 | 1 | 2514639852148749 | 422 | 2025-06-08 |
| 2 | 2 | 6987542685743694 | 688 | 2026-07-17 |
| 3 | 3 | 5201487423145896 | 356 | 2027-04-30 |
| 4 | 4 | 5847362871874536 | 598 | 2026-05-24 |
| 5 | 5 | 3627145820305897 | 102 | 2024-02-25 |

Usuario Contabilidad

| Results Messages | | | | |
|------------------|------------|-------------------------|------|-----------------|
| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
| 1 | 1 | ACCESO-RESTRINGIDO-8749 | xxxx | 2000-06-08 |
| 2 | 2 | ACCESO-RESTRINGIDO-3694 | xxxx | 2000-07-17 |
| 3 | 3 | ACCESO-RESTRINGIDO-5896 | xxxx | 2000-04-30 |
| 4 | 4 | ACCESO-RESTRINGIDO-4536 | xxxx | 2000-05-24 |
| 5 | 5 | ACCESO-RESTRINGIDO-5897 | xxxx | 2000-02-25 |

| | name | table_name | is_masked | masking_function |
|---|-----------------|------------|-----------|--------------------------------------|
| 1 | NUMERO | TARJETA | 1 | partial(0, "ACCESO-RESTRINGIDO-", 4) |
| 2 | CVV | TARJETA | 1 | default() |
| 3 | FECHA_CADUCIDAD | TARJETA | 1 | datetime("Y") |

5.6 UNMASK

[Volver al indice →](#)

Se refiere a una instrucción de SQL que se utiliza para quitar la máscara de datos aplicada a una columna en una tabla.

Al aplicar la función de UNMASK, permitimos que se tenga información sobre una tabla en concreto.


Para demostrarlo, aplicaremos dicha función sobre el usuario CONTABILIDAD, pudiendo este ver todo el contenido de la misma.

--FUNCIÓN UNMASK--

--LA APLICAREMOS SOBRE EL USUARIO CONTABILIDAD--

--CON LA FINALIDAD DE QUE SEA CAPAZ DE VISUALIZAR TODO EL CONTENIDO DE LA TABLA--

```
GRANT UNMASK TO CONTABILIDAD
GO
```

|  Messages | | | | |
|--|------------|------------------|-----|-----------------|
| CONTABILIDAD | | | | |
| Completion time: 2023-03-12T01:22:16.2525894+01:00 | | | | |
| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
| 1 | 1 | 2514639852148749 | 422 | 2025-06-08 |
| 2 | 2 | 6987542685743694 | 688 | 2026-07-17 |
| 3 | 3 | 5201487423145896 | 356 | 2027-04-30 |
| 4 | 4 | 5847362871874536 | 598 | 2026-05-24 |
| 5 | 5 | 3627145820305897 | 102 | 2024-02-25 |

Ya el usuario CONTABILIDAD tiene acceso full al contenido de la misma.

Para revocar el permiso anteriormente concedido

--PARA REVOCAR EL PERMISO ANTERIORMENTE CONCEDIDO--

```
REVOKE UNMASK TO CONTABILIDAD
GO
```

5.7 PERMISOS GRANULARES SQL SERVER 2022 (16.x)

[Volver al índice →](#)

A partir de SQL Server 2022 (16.x), se puede evitar el acceso no autorizado a datos confidenciales y obtener control mediante el enmascaramiento a un usuario no autorizado en distintos niveles de la base de datos. Puede conceder o revocar el permiso UNMASK en el nivel de base de datos, en el nivel de esquema, en el nivel de tabla o en el nivel de columna a un rol de usuario o base de datos. Esta mejora proporciona una manera más granular de controlar y limitar el acceso no autorizado a los datos almacenados en la base de datos, y de mejorar la administración de la seguridad de los datos.

Para su demostración, continuando con los ejemplos anteriores, el usuario CONTABILIDAD al realizar una consulta sobre la tabla TARJETA, este es el resultado.

| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
|---|------------|-------------------------|------|-----------------|
| 1 | 1 | ACCESO-RESTRINGIDO-8749 | xxxx | 2000-06-08 |
| 2 | 2 | ACCESO-RESTRINGIDO-3694 | xxxx | 2000-07-17 |
| 3 | 3 | ACCESO-RESTRINGIDO-5896 | xxxx | 2000-04-30 |
| 4 | 4 | ACCESO-RESTRINGIDO-4536 | xxxx | 2000-05-24 |
| 5 | 5 | ACCESO-RESTRINGIDO-5897 | xxxx | 2000-02-25 |

Todos los campos de la tabla se encuentran enmascarados con distintas técnicas que se muestran a continuación.

| | name | table_name | is_masked | masking_function |
|---|-----------------|------------|-----------|--------------------------------------|
| 1 | NUMERO | TARJETA | 1 | partial(0, "ACCESO-RESTRINGIDO-", 4) |
| 2 | CVV | TARJETA | 1 | default() |
| 3 | FECHA_CADUCIDAD | TARJETA | 1 | datetime("Y") |

Para aplicar los permisos UNMASK de manera granular, con la finalidad de permitir solo acceso a una de las columnas y no a toda la tabla lo haremos de la siguiente manera.

```
--PERMISOS GRANULARES--
```

```
--ESTO NOS PERMITE ASIGNAR EL UNMASK SOBRE UN CAMPO EN CONCRETO--
```

```
--Y NO SOBRE LA TABLA EN GENERAL--
```

```
--PERMISO SOBRE EL CAMPO NUMERO--
```

```
GRANT UNMASK ON TARJETA(NUMERO) TO CONTABILIDAD;
```

```
--IMPERSONAMOS AL USUARIO CONTABILIDAD--
```

```
EXECUTE AS USER = 'CONTABILIDAD';
```

```
GO
```

```
CONTABILIDAD
```

```
Completion time: 2023-03-12T01:39:40.1850740+01:00
```

```
SELECT * FROM TARJETA
```

```
GO
```

| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
|---|------------|------------------|------|-----------------|
| 1 | 1 | 2514639852148749 | xxxx | 2000-06-08 |
| 2 | 2 | 6987542685743694 | xxxx | 2000-07-17 |
| 3 | 3 | 5201487423145896 | xxxx | 2000-04-30 |
| 4 | 4 | 5847362871874536 | xxxx | 2000-05-24 |
| 5 | 5 | 3627145820305897 | xxxx | 2000-02-25 |

Probemos con el campo CVV

| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
|---|------------|-------------------------|-----|-----------------|
| 1 | 1 | ACCESO-RESTRINGIDO-8749 | 422 | 2000-06-08 |
| 2 | 2 | ACCESO-RESTRINGIDO-3694 | 688 | 2000-07-17 |
| 3 | 3 | ACCESO-RESTRINGIDO-5896 | 356 | 2000-04-30 |
| 4 | 4 | ACCESO-RESTRINGIDO-4536 | 598 | 2000-05-24 |
| 5 | 5 | ACCESO-RESTRINGIDO-5897 | 102 | 2000-02-25 |

Para poder revocar dichos permisos debemos indicar tanto la tabla como el campo de la misma, a diferencia de un unmask convencional.

```
REVOKE UNMASK ON TARJETA(NUMERO) TO CONTABILIDAD
GO
```

6.0 ROW ENCRYPTION (ROW-LEVEL SECURITY)

[Volver al índice →](#)

Row-Level Security (RLS) es una característica de SQL Server que permite a los administradores de bases de datos restringir el acceso a filas específicas en una tabla para usuarios y roles específicos.

En lugar de permitir que todos los usuarios accedan a todas las filas de una tabla, RLS permite definir una política de seguridad en la que se especifican las condiciones que deben cumplir los usuarios para acceder a las filas de la tabla. Las políticas de seguridad se definen mediante una función que evalúa la condición de seguridad en cada fila.

Los predicados de filtros permiten que los datos cifrados se busquen y se recuperen de una tabla de base de datos mediante la aplicación de una condición o expresión de filtro en una o más columnas cifradas de la tabla.

Los predicados de filtros son importantes en el cifrado de filas, ya que permiten que los datos cifrados se puedan buscar y filtrar sin tener que descifrarlos primero. Esto ayuda a proteger la privacidad y la confidencialidad de los datos al evitar la exposición de información sensible en texto claro.

```
--ROW ENCRYPTION--
```

```
--VERIFICAMOS LA EXISTENCIA DE LA BASE DE DATOS--
```

```
DROP DATABASE IF EXISTS STASH_RLS  
GO
```

```
--CREAMOS LA BASE DE DATOS--
```

```
CREATE DATABASE STASH_RLS  
GO
```

```
--INICIALMENTE CREAREMOS DISTINTOS VENDEDORES PARA NUESTRO EJEMPLO--
```

```
CREATE USER VENDEDOR_PRO WITHOUT LOGIN  
GO
```

```
CREATE DATABASE STASH_RLS  
GO
```

```
--INICIALMENTE CREAREMOS DISTINTOS VENDEDORES PARA NUESTRO EJEMPLO--
```

```
CREATE USER VENDEDOR_PRO WITHOUT LOGIN  
GO
```

```
CREATE USER VENDEDOR_INTERMEDIO WITHOUT LOGIN  
GO
```

```
CREATE USER VENDEDOR_INICIAL WITHOUT LOGIN  
GO
```

```
--CREAMOS LA TABLA VENTA--
```

```
CREATE TABLE VENTA (  
    ID_VENTA INT IDENTITY (1,1) PRIMARY KEY,  
    ID_PRODUCTO INT NOT NULL,  
    CANTIDAD INT NOT NULL,
```

```
FECHA_VENTA DATE NOT NULL,
VENDEDOR VARCHAR(50) NOT NULL,
);
```

```
--INSERTAMOS VALORES--
```

```
INSERT INTO VENTA VALUES
(1, 6, '2023-01-06', 'VENDEDOR_PRO'),
(3, 26, '2023-01-08', 'VENDEDOR_INICIAL'),
(5, 16, '2023-01-08', 'VENDEDOR_PRO'),
(14, 56, '2023-01-09', 'VENDEDOR_INTERMEDIO'),
(7, 4, '2023-01-10', 'VENDEDOR_INICIAL'),
(15, 116, '2023-01-11', 'VENDEDOR_PRO'),
(4, 86, '2023-01-12', 'VENDEDOR_PRO'),
(1, 26, '2023-01-13', 'VENDEDOR_INICIAL'),
(23, 326, '2023-01-16', 'VENDEDOR_INTERMEDIO'),
(20, 8, '2023-01-20', 'VENDEDOR_INICIAL'),
(35, 416, '2023-01-22', 'VENDEDOR_PRO'),
(23, 148, '2023-01-25', 'VENDEDOR_INTERMEDIO'),
(45, 38, '2023-01-28', 'VENDEDOR_INICIAL');
GO
```

```
SELECT * FROM VENTA
GO
```

| | ID_VENTA | ID_PRODUCTO | CANTIDAD | FECHA_VENTA | VENDEDOR |
|----|----------|-------------|----------|-------------|---------------------|
| 1 | 1 | 1 | 6 | 2023-01-06 | VENDEDOR_PRO |
| 2 | 2 | 3 | 26 | 2023-01-08 | VENDEDOR_INICIAL |
| 3 | 3 | 5 | 16 | 2023-01-08 | VENDEDOR_PRO |
| 4 | 4 | 14 | 56 | 2023-01-09 | VENDEDOR_INTERMEDIO |
| 5 | 5 | 7 | 4 | 2023-01-10 | VENDEDOR_INICIAL |
| 6 | 6 | 15 | 116 | 2023-01-11 | VENDEDOR_PRO |
| 7 | 7 | 4 | 86 | 2023-01-12 | VENDEDOR_PRO |
| 8 | 8 | 1 | 26 | 2023-01-13 | VENDEDOR_INICIAL |
| 9 | 9 | 23 | 326 | 2023-01-16 | VENDEDOR_INTERMEDIO |
| 10 | 10 | 20 | 8 | 2023-01-20 | VENDEDOR_INICIAL |
| 11 | 11 | 35 | 416 | 2023-01-22 | VENDEDOR_PRO |
| 12 | 12 | 23 | 148 | 2023-01-25 | VENDEDOR_INTERMEDIO |
| 13 | 13 | 45 | 38 | 2023-01-28 | VENDEDOR_INICIAL |


```
--EL VENDEDOR PRO PUEDE VISUALIZAR TODAS LAS VENTAS--
--EL VENDEDOR INTERMEDIO VE FILAS DONDE VENDEDOR SERA IGUAL A "VENDEDOR
INTERMEDIO" O "VENDEDOR PRO" PERMITIENDO QUE SE CREEN OTROS USUARIOS CON
LOW-RIGHTS EN EL FUTURO--
--EL VENDEDOR INICIAL SOLO VISUALIZA DONDE VENDEDOR SEA IGUAL A "VENDEDOR
INICIAL"

CREATE FUNCTION dbo.administradoporusuario$predicadodeseguridad
(@administradoporusuario AS sysname)
RETURNS TABLE
WITH SCHEMABINDING
AS
RETURN (SELECT 1 AS administradoporusuario$predicadodeseguridad
WHERE @administradoporusuario = USER_NAME() --(SI)administradoporusuario =
NOMBRE DE USUARIO DE LA BASE DE DATOS
OR (USER_NAME() = 'VENDEDOR_INTERMEDIO' and @administradoporusuario <>
'VENDEDOR_PRO') --(SI) el usuario es VENDEDOR_INTERMEDIO Y VENDEDOR_PRO NO
ADMINISTRA LA FILA--
OR (USER_NAME() = 'VENDEDOR_PRO')); --(SI) SI EL USUARIO ES VENDEDOR_PRO
PUEDE VER TODO
GO

--CREAMOS LA POLITICA DE SEGURIDAD--

CREATE SECURITY POLICY AdministradoPorUsuarioPolicy
ADD FILTER PREDICATE
dbo.administradoporusuario$predicadodeseguridad(VENDEDOR) ON dbo.VENTA
GO

--ACTIVAMOS LA POLITICA DE SEGURIDAD--

ALTER SECURITY POLICY AdministradoPorUsuarioPolicy WITH (STATE = ON)

--IMPERSONAMOS CON LOS DISTINTOS USUARIOS--

GRANT SELECT ON VENTA TO VENDEDOR_INICIAL

EXECUTE AS USER='VENDEDOR_INICIAL'

SELECT * FROM VENTA
GO

REVERT

-----
```

```
GRANT SELECT ON VENTA TO VENDEDOR_INTERMEDIO
```

```
EXECUTE AS USER='VENDEDOR_INTERMEDIO'
```

```
SELECT * FROM VENTA
```

```
GO
```

```
REVERT
```

```
GRANT SELECT ON VENTA TO VENDEDOR_PRO
```

```
EXECUTE AS USER='VENDEDOR_PRO'
```

```
SELECT * FROM VENTA
```

```
GO
```

```
REVERT
```

Ejecutamos la SELECT con cada uno de los usuarios anteriormente creados, los cuales solo tendrán acceso proporcional a la política de seguridad anteriormente creada

```
--IMPERSONAMOS CON LOS DISTINTOS USUARIOS--  
  
GRANT SELECT ON VENTA TO VENDEDOR_INICIAL  
  
EXECUTE AS USER='VENDEDOR_INICIAL'  
  
SELECT * FROM VENTA  
GO  
  
REVERT
```

110 %

Results Messages

| | ID_VENTA | ID_PRODUCTO | CANTIDAD | FECHA_VENTA | VENDEDOR |
|---|----------|-------------|----------|-------------|------------------|
| 1 | 2 | 3 | 26 | 2023-01-08 | VENDEDOR_INICIAL |
| 2 | 5 | 7 | 4 | 2023-01-10 | VENDEDOR_INICIAL |
| 3 | 8 | 1 | 26 | 2023-01-13 | VENDEDOR_INICIAL |
| 4 | 10 | 20 | 8 | 2023-01-20 | VENDEDOR_INICIAL |
| 5 | 13 | 45 | 38 | 2023-01-28 | VENDEDOR_INICIAL |

```

GRANT SELECT ON VENTA TO VENDEDOR_INTERMEDIO

EXECUTE AS USER='VENDEDOR_INTERMEDIO'

SELECT * FROM VENTA
GO

REVERT

```

110 %

| | ID_VENTA | ID_PRODUCTO | CANTIDAD | FECHA_VENTA | VENDEDOR |
|---|----------|-------------|----------|-------------|---------------------|
| 1 | 2 | 3 | 26 | 2023-01-08 | VENDEDOR_INICIAL |
| 2 | 4 | 14 | 56 | 2023-01-09 | VENDEDOR_INTERMEDIO |
| 3 | 5 | 7 | 4 | 2023-01-10 | VENDEDOR_INICIAL |
| 4 | 8 | 1 | 26 | 2023-01-13 | VENDEDOR_INICIAL |
| 5 | 9 | 23 | 326 | 2023-01-16 | VENDEDOR_INTERMEDIO |
| 6 | 10 | 20 | 8 | 2023-01-20 | VENDEDOR_INICIAL |
| 7 | 12 | 23 | 148 | 2023-01-25 | VENDEDOR_INTERMEDIO |
| 8 | 13 | 45 | 38 | 2023-01-28 | VENDEDOR_INICIAL |

```

GRANT SELECT ON VENTA TO VENDEDOR_PRO

EXECUTE AS USER='VENDEDOR_PRO'

SELECT * FROM VENTA
GO

```

110 %

| | ID_VENTA | ID_PRODUCTO | CANTIDAD | FECHA_VENTA | VENDEDOR |
|----|----------|-------------|----------|-------------|---------------------|
| 1 | 1 | 1 | 6 | 2023-01-06 | VENDEDOR_PRO |
| 2 | 2 | 3 | 26 | 2023-01-08 | VENDEDOR_INICIAL |
| 3 | 3 | 5 | 16 | 2023-01-08 | VENDEDOR_PRO |
| 4 | 4 | 14 | 56 | 2023-01-09 | VENDEDOR_INTERMEDIO |
| 5 | 5 | 7 | 4 | 2023-01-10 | VENDEDOR_INICIAL |
| 6 | 6 | 15 | 116 | 2023-01-11 | VENDEDOR_PRO |
| 7 | 7 | 4 | 86 | 2023-01-12 | VENDEDOR_PRO |
| 8 | 8 | 1 | 26 | 2023-01-13 | VENDEDOR_INICIAL |
| 9 | 9 | 23 | 326 | 2023-01-16 | VENDEDOR_INTERMEDIO |
| 10 | 10 | 20 | 8 | 2023-01-20 | VENDEDOR_INICIAL |
| 11 | 11 | 35 | 416 | 2023-01-22 | VENDEDOR_PRO |
| 12 | 12 | 23 | 148 | 2023-01-25 | VENDEDOR_INTERMEDIO |
| 13 | 13 | 45 | 38 | 2023-01-28 | VENDEDOR_INICIAL |

7.ALWAYS ENCRYPTED

[Volver al índice →](#)

Always Encrypted es una característica en Microsoft SQL Server que proporciona cifrado de datos sensibles en una base de datos, tanto en reposo como en tránsito, para evitar el acceso no autorizado a los datos. Permite el cifrado de datos a nivel de columna, de modo que sólo los usuarios o aplicaciones autorizadas puedan acceder a los datos.

Con Always Encrypted, los datos se cifran antes de que salgan de la aplicación cliente y solo se descifran cuando son accedidos por un usuario o aplicación autorizados. Esto significa que incluso si alguien obtiene acceso no autorizado a la base de datos, no podrá leer los datos cifrados sin las claves de cifrado necesarias.

```
--ALWAYS ENCRYPTED--
```

```
--VERIFICAMOS LA EXISTENCIA DE LA BASE DE DATOS--
```

```
DROP DATABASE IF EXISTS STASH_ALWAYS_ENCRYPTED  
GO
```

```
-- CREAMOS LA BASE DE DATOS --
```

```
CREATE DATABASE STASH_ALWAYS_ENCRYPTED  
GO
```

```
USE STASH_ALWAYS_ENCRYPTED  
GO
```

```
-- CREMOS LA TABLA TARJETA --
```

```
CREATE TABLE TARJETA (  
    ID_TARJETA INT IDENTITY (1,1) PRIMARY KEY,  
    NUMERO bigint,  
    CVV bigint ,  
    FECHA_CADUCIDAD DATE )  
GO
```

```
--INSERTAMOS VALORES DENTRO DE LA TABLA--
```

```
INSERT INTO TARJETA (NUMERO, CVV, FECHA_CADUCIDAD) VALUES  
    (2514639852148749 , 422 , '2025-06-08'),  
    (6987542685743694, 688 , '2026-07-17'),  
    (5201487423145896, 356 , '2027-04-30')  
GO
```

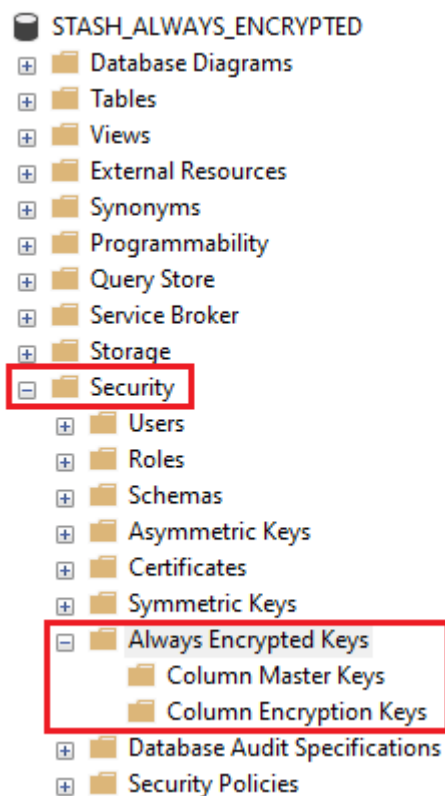
```
SELECT * FROM TARJETA  
GO
```

| Results | | Messages | | |
|---------|------------|------------------|-----|-----------------|
| | ID_TARJETA | NUMERO | CVV | FECHA_CADUCIDAD |
| 1 | 1 | 2514639852148749 | 422 | 2025-06-08 |
| 2 | 2 | 6987542685743694 | 688 | 2026-07-17 |
| 3 | 3 | 5201487423145896 | 356 | 2027-04-30 |

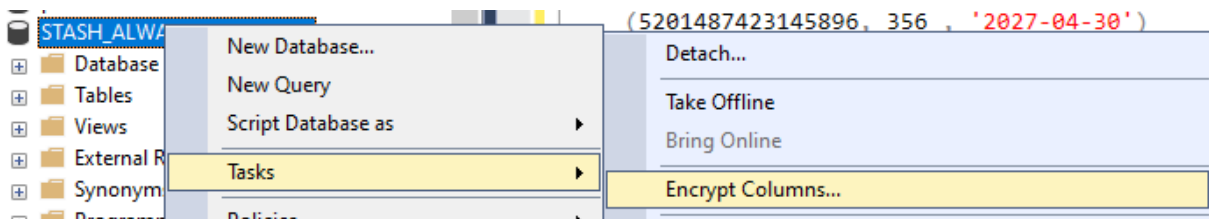
Ahora configuremos SSMS para habilitar Always Encrypted.

Lo podemos hacer desde el entorno gráfico de la siguiente manera:

En primer lugar verificamos en el apartado de seguridad de la base de datos, la existencia de las claves de encriptación, y podemos observar que se encuentran vacías.



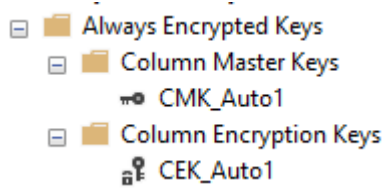
Ahora procedemos a habilitar always encrypted.



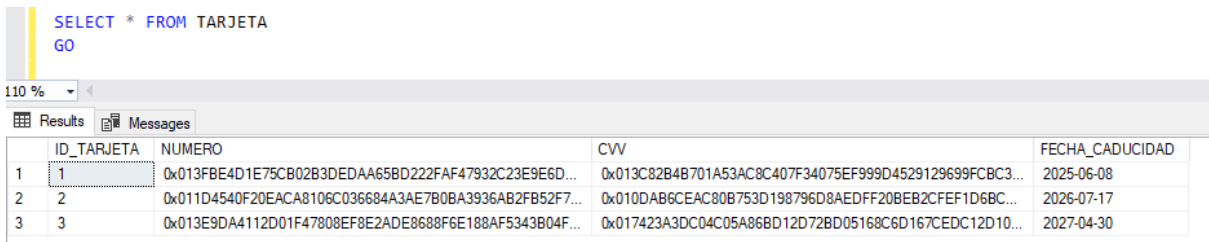
Una vez que se despliega el Wizard, elegimos la columna que deseamos encriptar, en este caso la columna “NÚMERO” tipo Randomized, “CVV” tipo “Deterministic”

| Name | State | Encryption Type | Encryption Key |
|-------------|-------|-----------------|----------------|
| dbo.TARJETA | | | |
| ID_TARJETA | | | |
| NUMERO | 🔒 | Randomized | CEK_Auto1 |
| CVV | 🔒 | Deterministic | CEK_Auto1 |
| FECHA_CA... | | | |

| Task | Details |
|---|------------------------|
| Generate new column master key CMK_Auto1 in Windows certificate store Cu... | Passed |
| Generate new column encryption key CEK_Auto1 | Passed |
| Generate PowerShell script C:\Users\AndresPrieto\Desktop\Always Encrypted.... | Passed |



Ejecutamos nuevamente una consulta sobre la tabla y observamos que efectivamente fue encriptado.



8.AUDITORIA

[Volver al índice →](#)

Proceso que permite monitorear y rastrear actividades, para garantizar la seguridad e integridad de los datos, permitiendo controlar registros de auditoría, restricciones de acceso, pruebas de penetración, entre otras.

Para la demostración de su funcionamiento haremos pruebas de auditoría donde nos interese guardar registros realizados por usuarios a nivel de Instancia y a nivel de Base de datos.

8.1 A NIVEL DE INSTANCIA

8.1.1 APPLICATION LOG

```
--LOG--
```

```
USE MASTER
```

```
GO
```

```
--CREAMOS LA AUDITORIA--
```

```
-- APPLICATION LOG--
```

```
CREATE SERVER AUDIT [STASH_AUDIT]
```

```
    TO    application_log
```

```
WITH
```

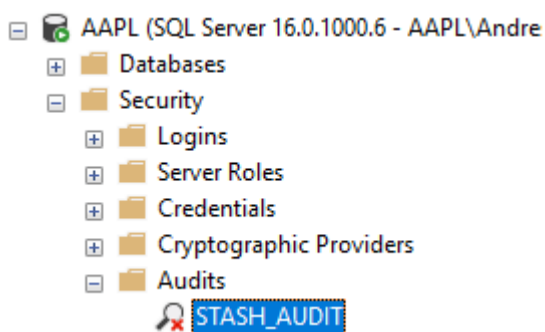
```
    ( QUEUE_DELAY = 1000,
```

```
      ON_FAILURE = FAIL_OPERATION
```

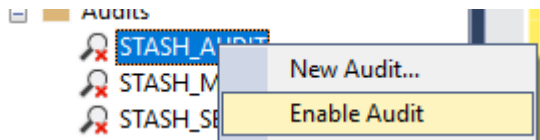
```
)
```

```
GO
```

Verificamos su creación desde el entorno gráfico

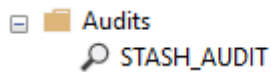


Las auditorías por defecto vienen deshabilitadas , para ello podemos habilitarlas tanto desde el entorno gráfico como de la línea de comandos.



--HABILITAMOS LA AUDITORÍA--

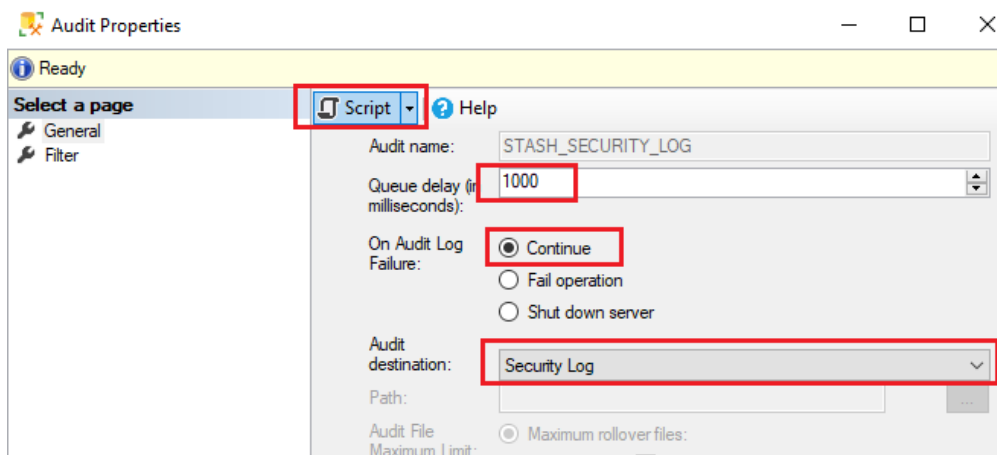
```
ALTER SERVER AUDIT STASH_AUDIT WITH (STATE = ON)
GO
```



7.1.2 SECURITY LOG

[Volver al índice →](#)

Podemos generar el script desde el entorno gráfico de la siguiente manera:



--SECURITY LOG--

```
USE master
GO
```

--SCRIPT GENERADO DESDE EL ENTORNO GRAFICO--

```
CREATE SERVER AUDIT [STASH_SECURITY_LOG]
TO SECURITY_LOG WITH (QUEUE_DELAY = 1000, ON_FAILURE = CONTINUE)
GO
```


7.1.3 FILE LOG

[Volver al índice →](#)

Esta vez creamos una auditoría donde almacenaremos todos los cambios que se produzcan como backups, dbcc, entre otros.

--FILE LOG--

```
CREATE SERVER AUDIT [STASH_MOTOR_AUDIT]
TO FILE
    (FILEPATH = 'C:\Auditoria',
    maxsize = 0 mb,
    max_rollover_files = 2147483647,
    reserve_disk_space = off
)
with
(Queue_DELAY = 1000, ON_FAILURE = CONTINUE)
GO
```

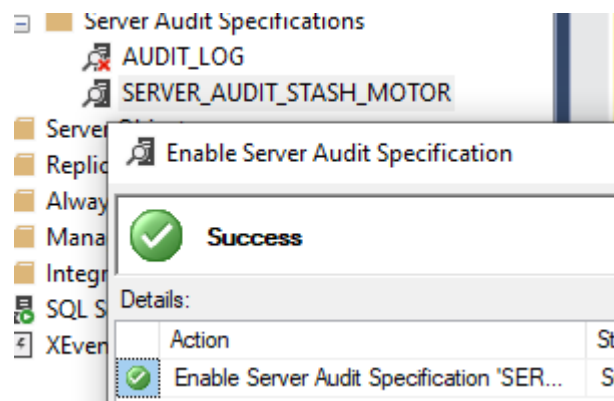
The screenshot shows the 'New Audit' dialog box in SQL Server Enterprise Manager. The 'Audit name' field contains 'STASH_MOTOR_AUDIT'. The 'Queue delay (in milliseconds)' spinner is set to 1000. Under 'On Audit Log Failure', the 'Continue' radio button is selected. The 'Audit destination' dropdown is set to 'File'. The 'Path' text box shows 'C:\Auditoria\' and has a browse button (...). Under 'Audit File Maximum Limit', the 'Maximum rollover files' radio button is selected, and the 'Unlimited' checkbox is checked. The 'Maximum file size' section shows '0' in the spinner, with 'MB' selected and 'Unlimited' checked. The 'Reserve disk space' checkbox is unchecked.

Una vez creada la auditoria procedemos a configurarla, esta vez lo haremos desde el entorno gráfico.

| | Audit Action Type | Object Class |
|-----|---------------------------|--------------|
| 1 | BACKUP_RESTORE_GROUP | |
| 2 | DBCC_GROUP | |
| 3 | SERVER_STATE_CHANGE_GROUP | |
| ▶ 4 | FAILED_LOGIN_GROUP | |
| * 5 | | |

Para la demostración, una de las sucesos que hemos indicado es que se guarde un registro cuando se realice un backup.

Previamente debemos asegurarnos de tenerla activada.



--HABILITAMOS LA AUDITORIA--

--GENERAMOS UN BACKUP PARA PROBAR LA AUDITORIA-- --HACER DBCC--

```
BACKUP DATABASE STASHMOTOR
    TO DISK = 'C:\Auditoria\STASHMOTOR.bak'
    with init;
go
```

--CONTROLAMOS LAS OPERACIONES O BIEN DESDE EL ENTORNO GRAFICO O DESDE TRANSACT--

```
SELECT event_time, action_id, class_type, server_instance_name,
server_principal_name , [database_name], [statement]
FROM sys.fn_get_audit_file ('C:\Auditoria\*', default, default)
GO
```

| | event_time | action_id | class_type | server_instance_name | server_principal_name | database_name | statement |
|---|-----------------------------|-----------|------------|----------------------|-----------------------|---------------|---|
| 1 | 2023-03-14 11:14:55.8565950 | AUSC | A | AAPL | AAPL\AndresPrieto | | |
| 2 | 2023-03-14 11:20:57.0167731 | BA | DB | AAPL | AAPL\AndresPrieto | STASHMOTOR | BACKUP DATABASE STASHMOTOR TO DISK = 'C:\Audit... |
| 3 | 2023-03-14 11:22:14.3548691 | BA | DB | AAPL | AAPL\AndresPrieto | STASHMOTOR | BACKUP DATABASE STASHMOTOR TO DISK = 'C:\Audit... |

8.2 A NIVEL DE SERVIDOR

[Volver al indice →](#)

La auditoría de base de datos es una técnica que se utiliza para registrar y monitorear los eventos que ocurren en una base de datos. El objetivo de la auditoría de base de datos es mejorar la seguridad y el cumplimiento normativo, así como detectar y prevenir actividades maliciosas. [Volver al indice →](#)

--AUDITORIA SOBRE BASES DE DATOS--

```
CREATE DATABASE STASH_AUDIT
GO
```

```
USE STASH_AUDIT
GO
```

Inicialmente creamos la auditoría, debemos indicar los siguientes parámetros:

ACCIÓN QUE DESEAMOS AUDITAR:

LA CLASE DEL OBJETO QUE DESEAMOS AUDITAR:

LA TABLA SOBRE LA QUE CREAREMOS LA AUDITORÍA:

USUARIO SOBRE EL QUE CREAMOS LA AUDITORÍA:

| Name: | STASHMOTOR_DATABASE_AUDIT | | | | | |
|----------|---------------------------|--------------|---------------|-------------|----------------|-------|
| Audit: | STASH_MOTOR_AUDIT | | | | | |
| Actions: | | | | | | |
| | Audit Action Type | Object Class | Object Schema | Object Name | Principal Name | |
| 1 | INSERT | OBJECT | dbo | CLIENTE | ... | dbo |
| 2 | SELECT | OBJECT | dbo | CLIENTE | ... | ADMIN |
| 3 | DELETE | OBJECT | dbo | CLIENTE | ... | dbo |
| 4 | UPDATE | OBJECT | dbo | CLIENTE | ... | dbo |
| 5 | | | | | | |

También hemos creado un usuario ADMIN para facilitar la demostración

```
--CREAMOS UN USUARIO SIN LOGIN --  
CREATE USER ADMIN WITHOUT LOGIN  
GO
```

```
--CONCEDEMOS PERMISOS DE LOGIN--  
GRANT SELECT ON CLIENTE TO ADMIN  
GO
```

```
EXECUTE AS USER = 'ADMIN'  
GO
```

EJECUTAMOS LAS ACCIONES PROPIAS DE LA AUDITORÍA

Para ello creamos una tabla cliente donde ejecutaremos una serie de acciones

```
CREATE TABLE CLIENTE (  
    ID_Cliente int IDENTITY (1,1),  
    Nombre VARCHAR(20),  
    Apellido VARCHAR (20)  
)  
GO
```

INSERT [Volver al índice →](#)

dbo

Completion time: 2023-03-14T12:46:21.5057710+01:00

```
INSERT INTO CLIENTE VALUES (  
    'Sairelys','Verde'  
)  
GO
```

SELECT [Volver al índice →](#)

Impersonamos con el usuario ADMIN

```
EXECUTE AS USER = 'ADMIN'  
GO
```

ADMIN

Completion time: 2023-03-14T12:49:05.6956677+01:00

```
SELECT * FROM CLIENTE
GO
```

UPDATE [Volver al índice →](#)

dbo

Completion time: 2023-03-14T12:46:21.5057710+01:00

```
--UPDATE SOBRE LA TABLA CON DBO--

UPDATE CLIENTE
SET Apellido = 'Verde'
WHERE ID_Cliente = 1;
```

.10 %

Messages

(1 row affected)

Completion time: 2023-03-14T12:51:24.9555143+01:00

Comprobamos la auditoría [Volver al índice →](#)

```
SELECT event_time, action_id, class_type, server_instance_name,
server_principal_name , [database_name], [statement] ,
database_principal_name
FROM sys.fn_get_audit_file ('C:\Auditoria\* ', default, default)
GO
```

| | event_time | action_id | class_type | server_instance_name | server_principal_name |
|---|-----------------------------|-----------|------------|----------------------|--|
| 1 | 2023-03-14 11:14:55.8565950 | AUSC | A | AAPL | AAPL\AndresPrieto |
| 2 | 2023-03-14 11:20:57.0167731 | BA | DB | AAPL | AAPL\AndresPrieto |
| 3 | 2023-03-14 11:22:14.3548691 | BA | DB | AAPL | AAPL\AndresPrieto |
| 4 | 2023-03-14 11:49:01.0091633 | SL | U | AAPL | S-1-9-3-1495909179-1083534289-1683240882-732003712 |
| 5 | 2023-03-14 11:51:24.9402135 | UP | U | AAPL | AAPL\AndresPrieto |
| 6 | 2023-03-14 11:53:29.4252094 | IN | U | AAPL | AAPL\AndresPrieto |

| | database_name | statement | database_principal_name |
|----|---------------|--|-------------------------|
| | STASHMOTOR | BACKUP DATABASE STASHMOTOR TO DISK = 'C:\Audit... | dbo |
| | STASHMOTOR | BACKUP DATABASE STASHMOTOR TO DISK = 'C:\Audit... | dbo |
| 12 | STASH_AUDIT | SELECT * FROM CLIENTE | ADMIN |
| | STASH_AUDIT | UPDATE CLIENTE SET Apellido = 'Verde' WHERE ID_Clie... | dbo |
| | STASH_AUDIT | INSERT INTO CLIENTE VALUES ('Sairelys','Verde') | dbo |

