LISTING 1-1 Implementing column-level encryption using a password

```
USE tempdb;
GO

-- Create sample table
CREATE TABLE Employees (
    EmployeeID INT PRIMARY KEY,
    EmployeeName VARCHAR(300),
    Position VARCHAR(100),
    Salary VARBINARY(128)
);
GO
-- Create SMK
CREATE SYMMETRIC KEY SMK_Emp
WITH ALGORITHM = AES_256 ENCRYPTION BY PASSWORD = 'Pa$$w0rd';
GO
-- Open SMK
OPEN SYMMETRIC KEY SMK_Emp DECRYPTION BY PASSWORD = 'Pa$$w0rd';
GO
-- Verify open keys
SELECT * FROM sys.openkeys;
GO
-- Insert data
INSERT Employees VALUES (1, 'Marcus', 'CTO', ENCRYPTBYKEY(KEY_GUID('SMK_Emp'),
'$100000'));
INSERT Employees VALUES (2, 'Christopher', 'CIO', ENCRYPTBYKEY(KEY_GUID('SMK_Emp'),
'$200000'));
INSERT Employees VALUES (3, 'Isabelle', 'CEO', ENCRYPTBYKEY(KEY_GUID('SMK_Emp'),
'$300000'));
GO
-- Query table with encrypted values
SELECT * FROM Employees;
GO
-- Query table with decrypted values
SELECT *, CONVERT(VARCHAR, DECRYPTBYKEY(Salary)) AS DecryptedSalary
FROM Employees;
GO
-- Close SMK
CLOSE SYMMETRIC KEY SMK_Emp
GO
-- Query table with decrypted values after key SMK is closed
SELECT *, CONVERT(VARCHAR, DECRYPTBYKEY(Salary)) AS DecryptedSalary
FROM Employees;
GO
-- Clever CTO updates their salary to match CEO's salary
UPDATE Employees
SET Salary = (SELECT Salary FROM Employees WHERE Position = 'CEO')
WHERE EmployeeName = 'Marcus';
GO
-- Open SMK and query table with decrypted values
OPEN SYMMETRIC KEY SMK_Emp DECRYPTION BY PASSWORD = 'Pa$$w0rd';
SELECT *, CONVERT(VARCHAR, DECRYPTBYKEY(Salary)) AS DecryptedSalary
FROM Employees;
GO
-- Cleanup
DROP TABLE Employees;
DROP SYMMETRIC KEY SMK_Emp;
GO
```

LISTING 1-2 Implementing column-level encryption using a certificate

```
USE WideWorldImporters;
GO
-- Create database master key
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'GoodLuckWithExam!'
-- Create certificate
CREATE CERTIFICATE Cert_BAN
   WITH SUBJECT = 'Bank Account Number';
GO
-- Create SMK
CREATE SYMMETRIC KEY Key_BAN
    WITH ALGORITHM = AES_256
    ENCRYPTION BY CERTIFICATE Cert_BAN;
GO
-- Create a column to store encrypted data
ALTER TABLE Purchasing.Suppliers
    ADD EncryptedBankAccountNumber varbinary(128);
GO
-- Open the SMK to encrypt data
OPEN SYMMETRIC KEY Key_BAN
   DECRYPTION BY CERTIFICATE Cert_BAN;
GO
-- Encrypt Bank Account Number
UPDATE Purchasing.Suppliers
SET EncryptedBankAccountNumber = EncryptByKey(Key_GUID('Key_BAN'), BankAccountNumber);
GO
-- Close SMK
CLOSE SYMMETRIC KEY Key_BAN
GO
/*
Verify encryption was successful
*/
-- Query 1: Check encryption has worked
SELECT TOP 5 SupplierID, SupplierName, BankAccountNumber, EncryptedBankAccountNumber,
    CONVERT(NVARCHAR(50), DecryptByKey(EncryptedBankAccountNumber)) AS
DecryptedBankAccountNumber
FROM Purchasing.Suppliers
GO
-- Query 2: Open the SMK
OPEN SYMMETRIC KEY Key_BAN
   DECRYPTION BY CERTIFICATE Cert_BAN;
GO
-- Query with decryption function
SELECT NationalIDNumber, EncryptedNationalIDNumber
    AS 'Encrypted ID Number',
    CONVERT(nvarchar, DecryptByKey(EncryptedNationalIDNumber))
    AS 'Decrypted ID Number'
    FROM HumanResources.Employee;
-- Results can be seen in Figure 1-3
GO
-- Close SMK
CLOSE SYMMETRIC KEY Key_BAN;
GO
```

LISTING 1-3 Implementing column-level encryption using a certificate

```
USE tempdb;
GO
-- Create table
CREATE TABLE dbo.Customers(
    CustomerID INT ,
    Name NVARCHAR(50) NULL,
    City NVARCHAR(50) NULL,
    BirthDate DATE NOT NULL
);
  GO
-- Insert sample data
INSERT Customers VALUES (1, 'Victor', 'Sydney', '19800909');
INSERT Customers VALUES (2, 'Sofia', 'Stockholm', '19800909');
INSERT Customers VALUES (3, 'Marcus', 'Sydney', '19900808');
INSERT Customers VALUES (4, 'Christopher', 'Sydney', '19800808');
INSERT Customers VALUES (5, 'Isabelle', 'Sydney', '20000909');
GO
-- Query unencrypted data
SELECT * FROM Customers;
GO
```

LISTING 1-4 Implementing Always Encrypted

```
-- Create CMK
CREATE COLUMN MASTER KEY [CMK_Auto1]
WITH
(
    KEY_STORE_PROVIDER_NAME = N'MSSQL_CERTIFICATE_STORE',
    KEY_PATH = N'CurrentUser/my/21CC13CA4E733072106BF516CB7BF51939C397A6'
);
GO
-- Create CEK
CREATE COLUMN ENCRYPTION KEY [CEK_Auto1]
WITH VALUES
(
    COLUMN_MASTER_KEY = [CMK_Auto1],
    ALGORITHM = 'RSA_OAEP',
    ENCRYPTED_VALUE = 0x016E000001630075007200720065006E0074007
    5007300650072002F006D0079002F003200310063006300310033006300
    …
    6100340065003700330033003000370032003100300030600620066003501
    1E60B9B4D7E6EB28F3A834FD8435A84421A80F36C14D2B371ED55C6D0AB
    37117FCE4444E64A9C6D4B1CCC8053C0FFE
)
GO
CREATE TABLE [dbo].[Customers](
    [CustomerID] [int] NULL,
    [Name] [nvarchar](50) NULL,
    [City] [nvarchar](50) COLLATE Latin1_General_BIN2
        ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = [CEK_Auto1],
        ENCRYPTION_TYPE = Deterministic,
        ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256') NULL,
    [BirthDate] [date]
        ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = [CEK_Auto1],
        ENCRYPTION_TYPE = Randomized,
        ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256') NOT NULL
)
GO
```

LISTING 1-5 Always Encrypted Powershell script

```
-- Create CMK
Import-Module SqlServer
# Set up connection and database SMO objects

$sqlConnectionString = "Data Source=DBA;Initial Catalog=tempdb;Integrated
Security=True;MultipleActiveResultSets=False;Connect
Timeout=30;Encrypt=False;TrustServerCertificate=True;Packet Size=4096;Application
Name=`"Microsoft SQL Server Management Studio`""
$smoDatabase = Get-SqlDatabase -ConnectionString $sqlConnectionString

# If your encryption changes involve keys in Azure Key Vault, uncomment one of the lines
below in order to authenticate:
#   * Prompt for a username and password:
#Add-SqlAzureAuthenticationContext -Interactive

#   * Enter a Client ID, Secret, and Tenant ID:
#Add-SqlAzureAuthenticationContext -ClientID '<Client ID>' -Secret '<Secret>' -Tenant
'<Tenant ID>'

# Change encryption schema

$encryptionChanges = @()

# Add changes for table [dbo].[Customers]
$encryptionChanges += New-SqlColumnEncryptionSettings -ColumnName dbo.Customers.City
-EncryptionType Deterministic -EncryptionKey "CEK_Auto1"
$encryptionChanges += New-SqlColumnEncryptionSettings -ColumnName dbo.Customers.
BirthDate -EncryptionType Randomized -EncryptionKey "CEK_Auto1"

Set-SqlColumnEncryption -ColumnEncryptionSettings $encryptionChanges -InputObject
$smoDatabase
GO
```

LISTING 1-6 Implementing transparent database encryption

```
USE master;
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'TOp$3cr3t';
GO
CREATE CERTIFICATE TDECertificate WITH SUBJECT = 'TDE self-signed certificate';
GO
USE WorldWideImporters;
GO
CREATE DATABASE ENCRYPTION KEY WITH ALGORITHM = AES_256
ENCRYPTION BY SERVER CERTIFICATE TDECertificate;
GO
ALTER DATABASE WorldWideImporters SET ENCRYPTION ON;
GO
/*
-- Use the following command to disable TDE
ALTER DATABASE WorldWideImporters SET ENCRYPTION OFF;
*/
GO
```

LISTING 1-7 Backing up TDE certificates and keys

```
USE master;
GO
-- Backup SMK
BACKUP SERVICE MASTER KEY
TO FILE = 'S:\SecureLocation\ServerMasterKey.key'
BY PASSWORD = 'TOp$3cr3t';
GO
-- Backup DMK
BACKUP MASTER KEY
TO FILE = 'S:\SecureLocation\DatabaseMasterKey.key'
BY PASSWORD = 'TOp$3cr3t';
GO
-- Backup TDECertificate
BACKUP CERTIFICATE TDECertificate
TO FILE = 'S:\SecureLocation\TDECertificate.cer'
WITH PRIVATE KEY(
    FILE = 'S:\SecureLocation\TDECertificate.key',
    ENCRYPTION BY PASSWORD = 'TOp$3cr3t'
);
```

LISTING 1-8 Implementing database backup encryption

```
USE master;
GO
CREATE MASTER KEY ENCRYPTION BY PASSWORD = '1@IT1xl@t$Ov@lFf3V3i3ntldr';
GO
CREATE CERTIFICATE BackupCertificate
    WITH SUBJECT = 'Backup self-signed certificate';
GO
BACKUP DATABASE WorldWideImporters
    TO DISK = N'S:\SQLBackup\WorldWideImporters.bak'
    WITH ENCRYPTION (
        ALGORITHM = AES_256,
        SERVER CERTIFICATE = BackupCertificate
    );
GO
```

LISTING 1-9 Creating logins

```
USE master;
-- Create Windows login
CREATE LOGIN [SQL\Marcus] FROM WINDOWS
GO
-- Create SQL login
CREATE LOGIN Isabelle
    WITH PASSWORD = 'A2c3456$#',
    CHECK_EXPIRATION = ON,
    CHECK_POLICY = ON;
GO
-- Create login from a certificate
CREATE CERTIFICATE ChristopherCertificate
    WITH SUBJECT = 'Christopher certificate in master database',
    EXPIRY_DATE = '30/01/2114';
GO
CREATE LOGIN Christopher FROM CERTIFICATE ChristopherCertificate;
GO
```

LISTING 1-10 Creating and maintaining roles

```
-- Add Windows login to fixed server role
USE master;
GO
ALTER SERVER ROLE sysadmin ADD MEMBER [SQL\Marcus]
GO
-- Add database user
USE [WideWorldImporters]
GO
CREATE USER [Isabelle] FOR LOGIN [Isabelle]
GO
-- Add database user to fixed database roles
ALTER ROLE [db_datareader] ADD MEMBER [Isabelle]
GO
ALTER ROLE [db_datawriter] ADD MEMBER [Isabelle]
GO
```

LISTING 1-11 Creating and maintaining user-defined server roles

```
-- Create user-defined server role
USE master;
GO
CREATE SERVER ROLE CustomerServeRole
GO
-- Add members to user-defined server role
ALTER SERVER ROLE CustomerServeRole ADD MEMBER Christopher
ALTER SERVER ROLE processadmin ADD MEMBER CustomerServeRole
ALTER SERVER ROLE securityadmin ADD MEMBER CustomerServeRole
--
GRANT SHUTDOWN TO CustomerServeRole
GRANT VIEW SERVER STATE TO CustomerServeRole
GO
-- Deny control to logins
DENY CONTROL ON LOGIN::[NT SERVICE\SQLSERVERAGENT] TO CustomerServeRole
DENY CONTROL ON LOGIN::sa TO CustomerServeRole
DENY CONTROL ON LOGIN::[NT SERVICE\MSSQLSERVER] TO CustomerServeRole
DENY CONTROL ON LOGIN::[SQL\Administrator] TO CustomerServeRole
GO
```

LISTING 1-12 Managing object level permissions

```
USE WideWorldImporters;
GO
GRANT SELECT ON Sales.Orders TO Isabelle;
GO
DENY DELETE ON Sales.Orders TO Isabelle;
GO
GRANT UPDATE ON Sales.Orders (InternalComments) TO Isabelle;
GO
GRANT UPDATE ON Sales.Orders (DeliveryInstructions) TO Isabelle;
GO
GRANT UPDATE ON Sales.Orders (Comments) TO Isabelle;
GO
```

LISTING 1-13 Implementing row level security

```
CREATE DATABASE Hospital;
GO
USE Hospital;
-- Create database schema
CREATE TABLE Patients (
    PatientID INT PRIMARY KEY,
    PatientName NVARCHAR(256),
    Room INT,
    WardID INT,
    StartTime DATETIME,
    EndTime DATETIME
);
CREATE TABLE Staff (
    StaffID INT PRIMARY KEY,
    StaffName NVARCHAR(256),
    DatabasePrincipalID INT
);
CREATE TABLE StaffDuties (
    StaffID INT,
    WardID INT,
    StartTime DATETIME,
   EndTime DATETIME
);
CREATE TABLE Wards (
    WardID INT PRIMARY KEY,
   Ward NVARCHAR(128)
);
GO
-- Create roles for nurses and doctors
CREATE ROLE Nurse;
CREATE ROLE Doctor;
-- Grant permissions to nurses and doctors
GRANT SELECT, UPDATE ON Patients to Nurse;
GRANT SELECT, UPDATE ON Patients to Doctor;
GO
-- Create a user for each doctor and nurse
CREATE USER NurseMarcus WITHOUT LOGIN;
ALTER ROLE Nurse ADD MEMBER NurseMarcus;
INSERT Staff VALUES ( 100, N'Nurse Marcus', DATABASE_PRINCIPAL_ID('NurseMarcus'));
GO
CREATE USER NurseIsabelle WITHOUT LOGIN;
ALTER ROLE Nurse ADD MEMBER NurseIsabelle;
INSERT Staff VALUES ( 101, N'Nurse Isabelle', DATABASE_PRINCIPAL_ID('NurseIsabelle') );
GO
CREATE USER DoctorChristopher WITHOUT LOGIN
ALTER ROLE Doctor ADD MEMBER DoctorChristopher
INSERT Staff VALUES ( 200, 'Doctor Christopher', DATABASE_PRINCIPAL_
ID('DoctorChristopher'));
GO
CREATE USER DoctorSofia WITHOUT LOGIN
ALTER ROLE Doctor ADD MEMBER DoctorSofia
INSERT Staff VALUES ( 201, N'Doctor Sofia', DATABASE_PRINCIPAL_ID('DoctorSofia'));
GO
-- Insert ward data
```

```
INSERT Wards VALUES( 1, N'Emergency');
INSERT Wards VALUES( 2, N'Maternity');
INSERT Wards VALUES( 3, N'Pediatrics');
GO
-- Insert patient data
INSERT Patients VALUES ( 1001, N'Victor', 101, 1, '20171217',  '20180326')
INSERT Patients VALUES ( 1002, N'Maria', 102, 1, '20171027',  '20180527')
INSERT Patients VALUES ( 1003, N'Nick', 107, 1, '20170507',  '20170611')
INSERT Patients VALUES ( 1004, N'Nina', 203, 2, '20170308',  '20171214')
INSERT Patients VALUES ( 1005, N'Larissa', 205, 2, '20170127',  '20170512')
INSERT Patients VALUES ( 1006, N'Marc', 301, 3, '20170131',  NULL)
INSERT Patients VALUES ( 1007, N'Sofia', 308, 3, '20170615',  '20170904')
GO
-- Inset nurses' duties
INSERT StaffDuties VALUES ( 101, 1, '20170101', '20171231')
INSERT StaffDuties VALUES ( 101, 2, '20180101', '20181231')
INSERT StaffDuties VALUES ( 102, 1, '20170101', '20170630')
INSERT StaffDuties VALUES ( 102, 2, '20170701', '20171231')
INSERT StaffDuties VALUES ( 102, 3, '20180101', '20181231')
-- Insert doctors' duties
INSERT StaffDuties VALUES ( 200, 1, '20170101', '20171231')
INSERT StaffDuties VALUES ( 200, 3, '20180101', '20181231')
INSERT StaffDuties VALUES ( 201, 1, '20170101', '20181231')
GO
-- Query patients
SELECT * FROM patients;
-- Query assignments
SELECT d.StaffID, StaffName, USER_NAME(DatabasePrincipalID) as DatabaseUser, WardID,
StartTime, EndTime
FROM StaffDuties d
INNER JOIN Staff s ON (s.StaffID = d.StaffID)
ORDER BY StaffID;
GO
-- Implement row level security
CREATE SCHEMA RLS;
GO
-- RLS predicate allows access to rows based on a user's role and assigned staff duties.
-- Because users have both SELECT and UPDATE permissions, we will use this function as a
-- filter predicate (filter which rows are accessible by SELECT and UPDATE queries) and
-- a block predicate after update (prevent user from updating rows to be outside of
-- visible range).
-- RLS predicate allows data access based on role and staff duties.
CREATE FUNCTION RLS.AccessPredicate(@Ward INT, @StartTime DATETIME, @EndTime DATETIME)
    RETURNS TABLE
    WITH SCHEMABINDING
AS
RETURN SELECT 1 AS Access
FROM dbo.StaffDuties AS d JOIN dbo.Staff AS s ON d.StaffId = s.StaffId
WHERE ( -- Nurses can only see patients who overlap with their wing assignments
    IS_MEMBER('Nurse') = 1
    AND s.DatabasePrincipalId = DATABASE_PRINCIPAL_ID()
    AND @Ward = d.WardID
    AND (d.EndTime >= @StartTime AND d.StartTime <= ISNULL(@EndTime, GETDATE()))
)
OR ( -- Doctors can see all patients
```

```
    IS_MEMBER('Doctor') = 1
);
GO
-- RLS filter predicate filters which data is seen by SELECT and UPDATE queries
-- RLS block predicate after update prevents updating data outside of visible range
CREATE SECURITY POLICY RLS.PatientsSecurityPolicy
ADD FILTER PREDICATE RLS.AccessPredicate(WardID, StartTime, EndTime) ON dbo.Patients,
ADD BLOCK PREDICATE RLS.AccessPredicate(WardID, StartTime, EndTime) ON dbo.Patients
AFTER UPDATE;
GO
-- Test RLS
-- Impersonate a nurse
EXECUTE ('SELECT * FROM patients;') AS USER = 'NurseIsabelle';
-- Only 3 patient records seen
GO
-- Impersonate a doctor
EXECUTE ('SELECT * FROM patients;') AS USER = 'DoctorChristopher';
-- All 7 patient records returned
GO
-- Attempt by nurse to move patient to another ward
EXECUTE ('UPDATE patients SET WardID = 1 WHERE patientId = 1006;') AS USER =
'NurseIsabelle'
-- Filtered, consequently 0 rows affected
EXECUTE ('UPDATE patients SET WardID = 3 WHERE patientId = 1001;') AS USER =
'NurseIsabelle'
-- Blocked from changing wing, with following error:
/*
Msg 33504, Level 16, State 1, Line 156
The attempted operation failed because the target object 'Hospital.dbo.Patients' has
a block predicate that conflicts with this operation. If the operation is performed
on a view, the block predicate might be enforced on the underlying table. Modify the
operation to target only the rows that are allowed by the block predicate.
The statement has been terminated.
*/
```

LISTING 1-14 Implementing DDM

```
USE WideWorldImporters
GO
-- Create non-privileged user
CREATE USER NonPrivilegedUser WITHOUT LOGIN
GO
ALTER ROLE db_datareader ADD MEMBER NonPrivilegedUser
GO
-- Mask email address
ALTER TABLE Application.People
ALTER COLUMN EmailAddress ADD MASKED WITH (FUNCTION = 'EMAIL()')
-- Mask phone number
ALTER TABLE Application.People
ALTER COLUMN PhoneNumber ADD MASKED WITH (FUNCTION = 'PARTIAL(0,"XXX-XXX-",4)')
GO
-- Query table as dbo
SELECT TOP 5 FullName, PhoneNumber, FaxNumber, EmailAddress FROM Application.People;
-- Query table as non-privileged user
EXECUTE AS USER = 'NonPrivilegedUser';
SELECT TOP 5 FullName, PhoneNumber, FaxNumber, EmailAddress FROM Application.People;
REVERT;
GO
```

LISTING 1-15 Implementing auditing

```
USE [master]
GO
-- Create audit
CREATE SERVER AUDIT [SQLAudit]
TO FILE (
    FILEPATH = N'C:\SQLAudit\',
    MAXSIZE = 4096 MB,
    MAX_ROLLOVER_FILES = 2147483647,
    RESERVE_DISK_SPACE = OFF
)
WITH (
    QUEUE_DELAY = 1000,
    ON_FAILURE = CONTINUE,
AUDIT_GUID = '4ab952ef-97c4-4e02-8a9e-1b4324499705'
)
ALTER SERVER AUDIT [SQLAudit] WITH (STATE = ON)
GO
-- Create server audit specification
CREATE SERVER AUDIT SPECIFICATION [ServerAuditSpecification-Logins]

FOR SERVER AUDIT [SQLAudit]
ADD (FAILED_LOGIN_GROUP),
ADD (SUCCESSFUL_LOGIN_GROUP),
ADD (LOGOUT_GROUP)
WITH (STATE = ON)
GO
-- Create database audit specification
USE [WideWorldImporters]
GO
CREATE DATABASE AUDIT SPECIFICATION [DatabaseAuditSpecification-Governance]
FOR SERVER AUDIT [SQLAudit]
ADD (BACKUP_RESTORE_GROUP),
ADD (AUDIT_CHANGE_GROUP),
ADD (DATABASE_PERMISSION_CHANGE_GROUP),
ADD (USER_CHANGE_PASSWORD_GROUP),
ADD (UPDATE ON OBJECT::[Sales].[OrderLines] BY [public]),
ADD (SELECT ON OBJECT::[Application].[People] BY [Isabelle])
WITH (STATE = ON)
GO
```