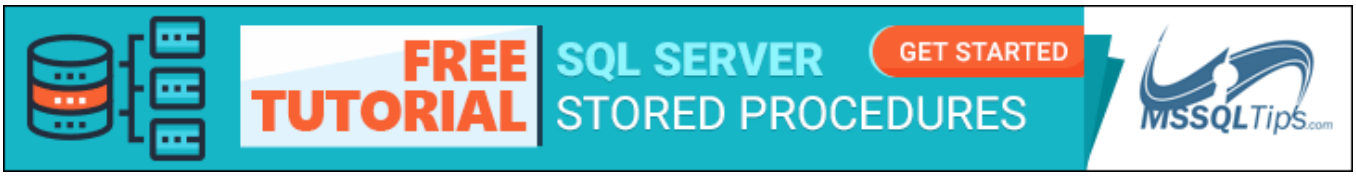
MSSQLTips Free SQL Server Conference Registration + Travel Expenses click to learn more

SQL Server 2016 Always Encrypted



Free Webcast - Simplify SQL Server Administration

Problem

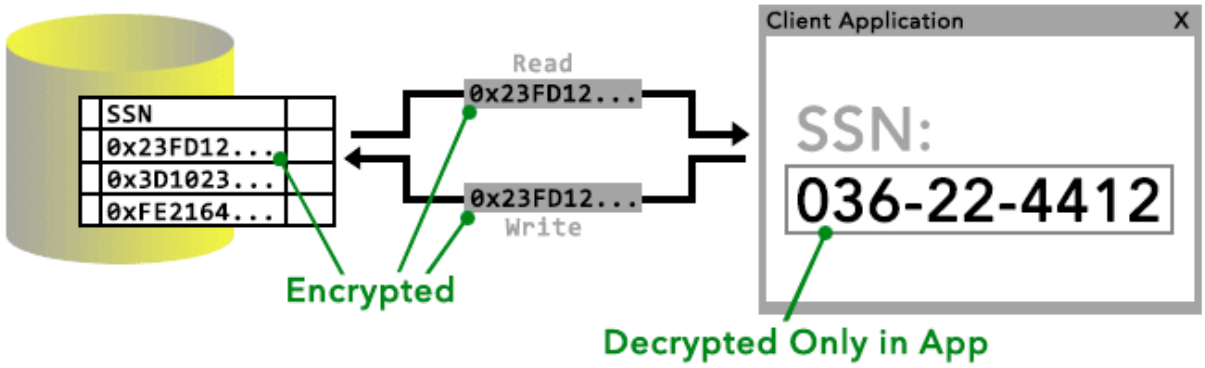
SQL Server has had ways to encrypt data in the past - for example, Transparent Data Encryption (TDE). This technology protects the data at rest and when the database files or backups are compromised. However it is accessible to the database itself and to anyone who can own it and obtain the key/certificate/password (system administrators, hackers, what have you).

Solution

Always Encrypted is a new feature in SQL Server 2016, which encrypts the data both at rest *and* in motion (and keeps it encrypted in memory). So this protects the data from rogue administrators, backup thieves, and man-in-the-middle attacks. Unlike TDE, as well, Always Encrypted allows you to encrypt only certain columns, rather than the entire database.

The client library ensures that plaintext is only revealed within the application or middle tier, and nowhere in between the application and the database. In the following illustration, I attempt to show that the data is

simply ciphertext both in the database and in both directions between the application and the database:



And this brings about the first limitation of Always Encrypted: It is not supported by all client libraries at this moment. In fact the only provider that currently works with Always Encrypted is the ADO.NET 4.6, so you will need to ensure .NET Framework 4.6 is installed on any machine that will run a client application that interfaces with Always Encrypted data.

This tip walks through basic Always Encrypted configuration, shows some examples, and explains limitations, all based on the most recent build at the time of writing (CTP 2.2). You can play with this feature by downloading the latest SQL Server 2016 CTP.

SQL Server 2016 Always Encrypted Concepts

There are several core concepts used in Always Encrypted:

- **Column Master Key** - this is an encryption key that protects the column encryption key(s). You must have at least one master key before encrypting any columns.
- **Column Encryption Key** - this is the encryption key that actually protects that encrypted columns.
- **Column-level encryption setting** - a column must be set to encrypted, with a specific column encryption key, the algorithm (currently only one algorithm is supported), and the type of encryption to be used:
  - **Deterministic** - always encrypts to the same ciphertext, so can be used for certain operations (point lookups, distinct, group by) and can be indexed.
  - **Randomized** - more secure, but cannot be evaluated or used in any operations (write/display only) and cannot be indexed.

- **Connection string** - in order for the client driver to understand that column encryption is in use, the connection string must have the following attribute:

```
Column Encryption Setting = enabled;
```

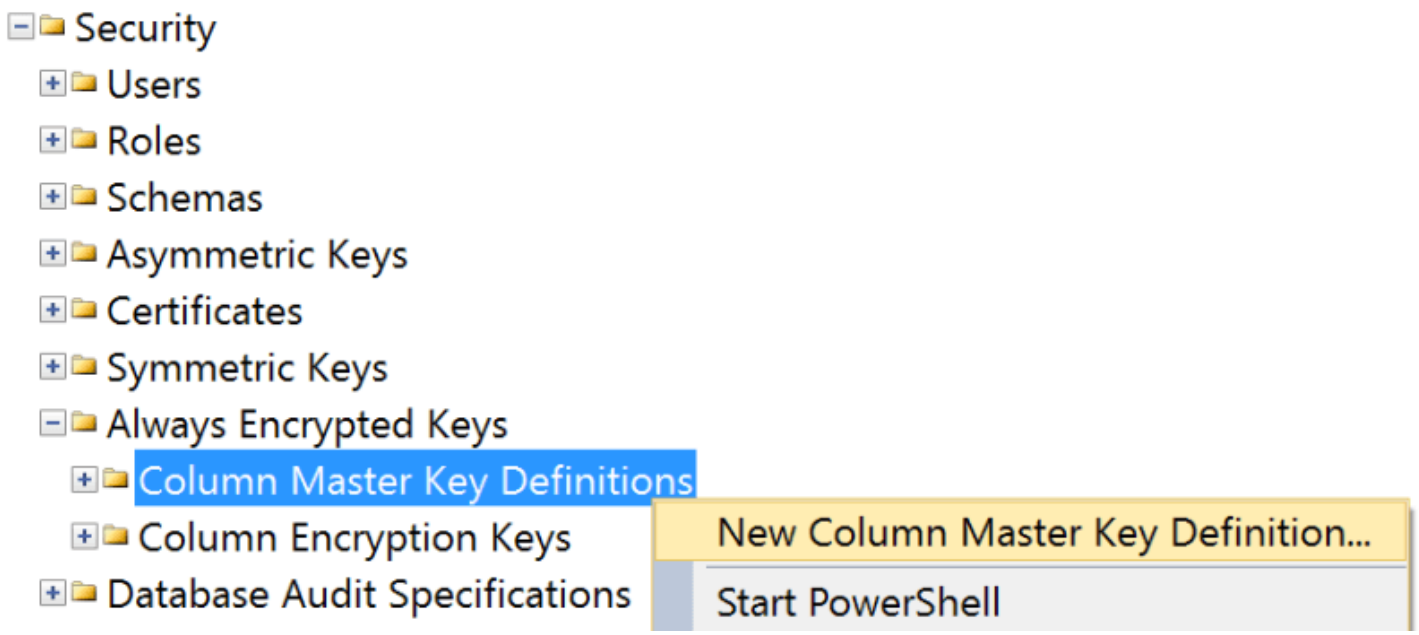
The application code itself, aside from the setting in the connection string, does not have to change at all, since it doesn't need to know which columns are actually encrypted.

SQL Server 2016 Always Encrypted Example

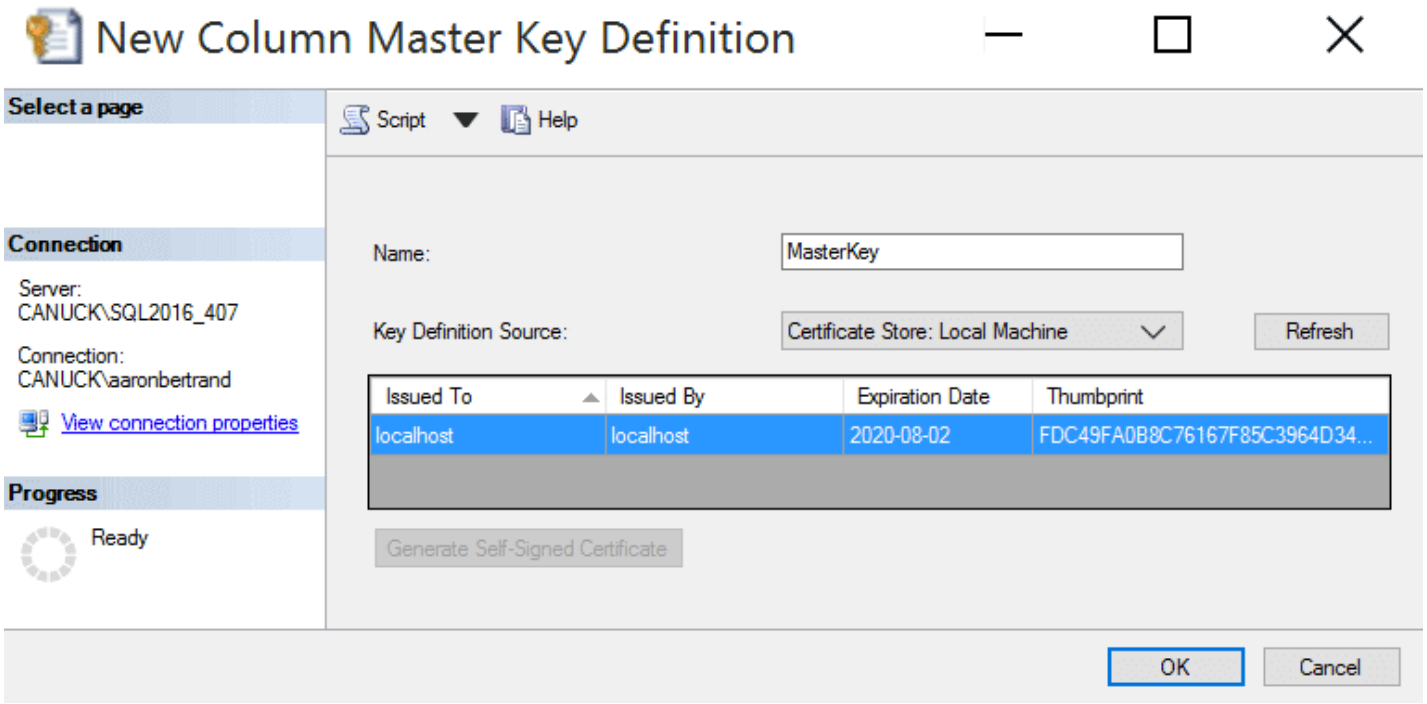
To keep things simple, I'm going to demonstrate with an example on a single, local machine (to avoid the complications of having keys distributed to multiple machines, an exercise I'll leave for the reader). First, let's create a database:

```
CREATE DATABASE AEDemo;
```

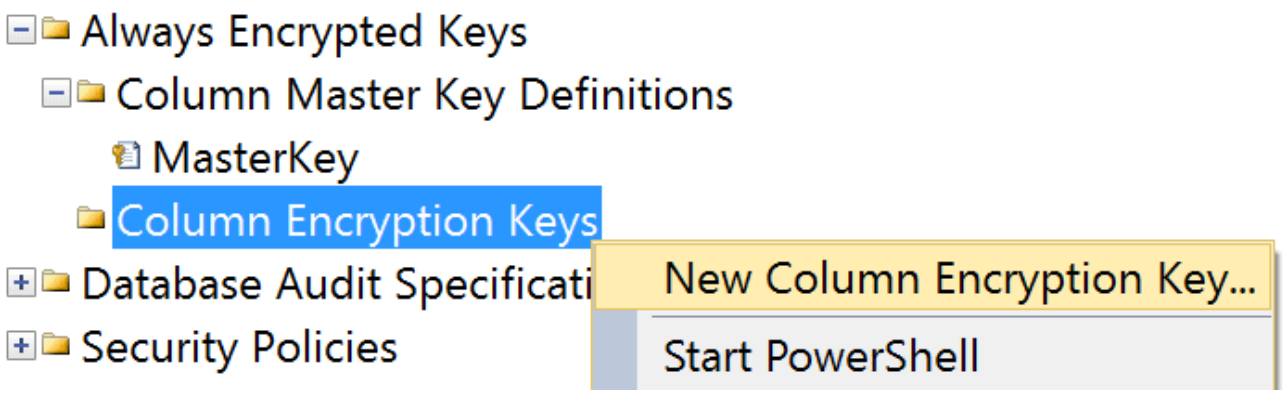
Now, I'm going to create a master key and a column key. In Object Explorer, expand the database, expand Security, and expand Always Encrypted Keys. You'll see two nodes there, and you can right-click the first to create a master key:



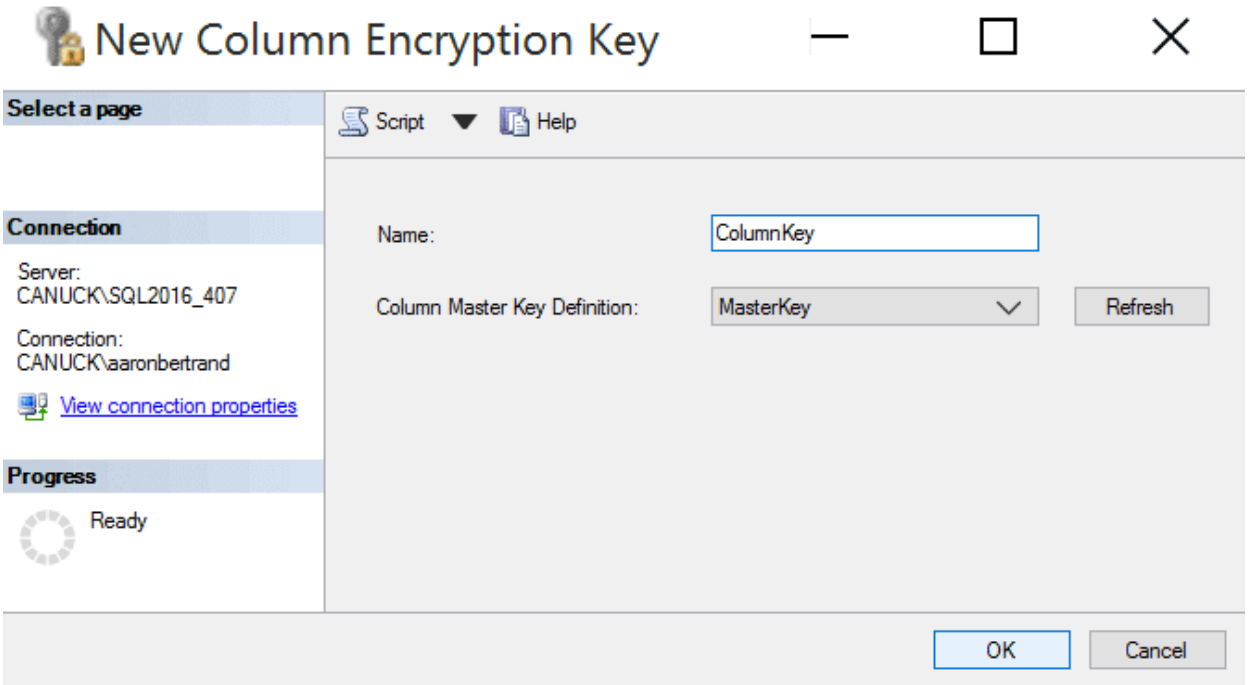
The dialog doesn't give you many options here; provide a name, and pick the key source. (I chose Local Machine but if you pick Current User, you may need to generate a self-signed certificate.) Note that you can create multiple master keys (to support key rotation).



Next, create a column encryption key using the other node:



Similarly, this dialog just lets you assign a name, and pick the master key it is associated with:



On my machine, this would generate the following CREATE statements (but please don't try to copy these and run them on your own machine):

```
CREATE COLUMN MASTER KEY DEFINITION [MasterKey]
WITH
(
        KEY_STORE_PROVIDER_NAME = N'MSSQL_CERTIFICATE_STORE',
        KEY_PATH = N'LocalMachine/My/FDC49FA0B8C76167F85C3964D349CB09D97E43B7'
);

CREATE COLUMN ENCRYPTION KEY [ColumnKey]
WITH VALUES
(
        COLUMN MASTER KEY DEFINITION = [MasterKey],
        ALGORITHM = 'RSA_OAEP',
        ENCRYPTED_VALUE = 0x01700000016C...
        -- really long value ----^^^^^^^^^^
);
```

Now with the keys created, we can create a table that uses them. Let's say we have an Employees table, and we want to encrypt LastName and Salary. We want to use deterministic encryption for LastName, because we're likely to look up an employee that way, but we can use randomized encryption on Salary, because we're highly unlikely to ever want to look up an employee because they are making $74,208 (and we know we can't perform range queries in any case).

The syntax for specifying encryption on a column is a bit cumbersome. As I mentioned earlier, only one encryption algorithm is supported, and it's quote a mouthful: AEAD_AES_256_CBC_HMAC_SHA_256. Also, any columns using string data types that use deterministic encryption must use one of the BIN2 collations.

```
CREATE TABLE dbo.EncryptedTable
(
```

```
  ID INT IDENTITY(1,1) PRIMARY KEY,

  LastName NVARCHAR(32) COLLATE Latin1_General_BIN2
    ENCRYPTED WITH
    (
       ENCRYPTION_TYPE = DETERMINISTIC,
       ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256',
       COLUMN_ENCRYPTION_KEY = ColumnKey
    ) NOT NULL,

  Salary INT
    ENCRYPTED WITH
    (
       ENCRYPTION_TYPE = RANDOMIZED,
       ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256',
       COLUMN_ENCRYPTION_KEY = ColumnKey
    ) NOT NULL
);
GO

CREATE PROCEDURE dbo.AddPerson
  @LastName NVARCHAR(32),
  @Salary   INT
AS
BEGIN
  INSERT dbo.EncryptedTable(LastName,Salary) SELECT @LastName, @Salary;
END
GO

CREATE PROCEDURE dbo.GetPeopleByLastName
  @LastName NVARCHAR(32)
AS
BEGIN
  SELECT ID, LastName, Salary
  FROM dbo.EncryptedTable
  WHERE LastName = @LastName COLLATE Latin1_General_BIN2;
END
GO
```

Now, I have yet to figure out how to insert a row into this table from Management Studio, and I suppose that is partly the point:

```
INSERT dbo.EncryptedTable(LastName,Salary) SELECT N'Bertrand',720000;

-- Result:
Msg 206, Level 16, State 2
Operand type clash: nvarchar is incompatible with nvarchar(4000) encrypted with
 (encryption_type = 'DETERMINISTIC', encryption_algorithm_name
= 'AEAD_AES_256_CBC_HMAC_SHA_256', column_encryption_key_name = 'ColumnKey',
 column_encryption_key_database_name = 'AEDemo')
```

If I use correctly typed parameters instead of ad hoc SQL, I get a different error:

```
DECLARE @LastName NVARCHAR(32) = N'Bertrand', @Salary INT = 720000;
INSERT dbo.EncryptedTable(LastName,Salary) SELECT @LastName, @Salary;
```

```
-- Result:
Msg 33299, Level 16, State 6
Encryption scheme mismatch for columns/variables '@LastName'. The encryption scheme
 for the columns/variables is (encryption_type = 'PLAINTEXT') and the expression near
  line '2' expects it to be (encryption_type = 'DETERMINISTIC', encryption_algorithm_name
  = 'AEAD_AES_256_CBC_HMAC_SHA_256', column_encryption_key_name = 'ColumnKey',
  column_encryption_key_database_name = 'AEDemo') (or weaker).
```

I get the same error if I create call the AddPerson procedure with matching parameters:

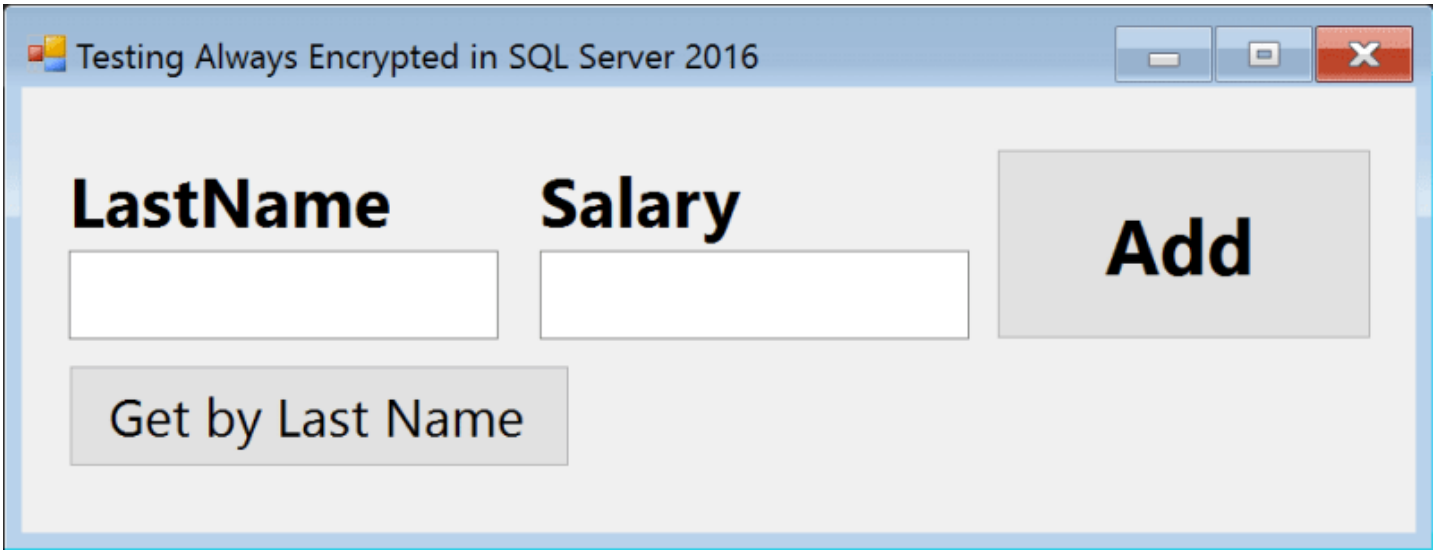
```
DECLARE @LastName NVARCHAR(32) = N'Bertrand', @Salary INT = 720000;
EXEC dbo.AddPerson @LastName, @Salary;
```

So, dusting off Visual Studio, I'm going to make a very simple Windows Forms application that will allow me to populate and query this table. I'm not going to include all of the code here, just the relevant bits. First, the connection string needs to include the Column Encryption Setting attribute indicated above, so my App.Config has this:

```
<connectionStrings>
  <add name="AEDB" connectionString="Data Source=.;Initial Catalog=AEDemo;
    Integrated Security=True;Column Encryption Setting=Enabled"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

I add two textboxes and two buttons to my form, to allow me to (a) enter a last name and a salary, and insert a row; or (b) enter a last name, and retrieve and display the salary. (This is rather simplistic because, of course, in most companies there will be multiple people with the same last name. But this is just a demo.)

The form ultimately looks like this:



And the code behind those two buttons:

```
private void button1_Click(object sender, EventArgs e)
{
```

```
  using (SqlConnection con = new SqlConnection())
  {
    con.ConnectionString = ConfigurationManager.ConnectionStrings["AEDB"].ToString();
    con.Open();
    using (SqlCommand cmd = new SqlCommand("dbo.AddPerson", con))
    {
      cmd.CommandType = CommandType.StoredProcedure;
      SqlParameter ln = new SqlParameter("@LastName", SqlDbType.NVarChar, 32);
      ln.Value = textBox1.Text;
      SqlParameter sal = new SqlParameter("@Salary", SqlDbType.Int);
      sal.Value = Convert.ToInt32(textBox2.Text);
      cmd.Parameters.Add(ln);
      cmd.Parameters.Add(sal);
      cmd.ExecuteNonQuery();
      MessageBox.Show("Person added.");
      textBox1.Clear(); textBox2.Clear();
    }
  }
}

private void button2_Click(object sender, EventArgs e)
{
  using (SqlConnection con = new SqlConnection())
  {
    con.ConnectionString = ConfigurationManager.ConnectionStrings["AEDB"].ToString();
    con.Open();
    using (SqlCommand cmd = new SqlCommand("dbo.GetPeopleByLastName", con))
    {
      cmd.CommandType = CommandType.StoredProcedure;
      SqlParameter ln = new SqlParameter("@LastName", SqlDbType.NVarChar, 32);
      ln.Value = textBox1.Text;
      cmd.Parameters.Add(ln);
      SqlDataReader rdr = cmd.ExecuteReader();
      while (rdr.Read())
      {
        textBox2.Text = rdr["Salary"].ToString();
      }
    }
  }
}
```

Quite crude and rudimentary, but it gets the job done: When you enter a last name and a salary and press the Add button, it adds it to the database, and then clears the form. If you add just a last name and press the other button, it populates the salary field with that person's salary.

If we watch the instance with Profiler, we can see calls like this go across the wire, demonstrating that the values were converted from plaintext to ciphertext long before they arrived at SQL Server:

```
exec dbo.AddPerson
 @LastName = 0x01A3F81E529328147EA31036CD6A628118E74ED653D2A6DE51...,
 @Salary = 0x01AFB0BAEA355EDC5EEE8E815AF247F333CD2180285A403A63DF...
```

If we go back to Management Studio, we can now at least LOOK at the data:

```
SELECT ID, LastName, Salary FROM dbo.EncryptedTable;

-- Result:

1   0x01A3F81E529328147EA3103...   0x01AFB0BAEA355EDC5EEE8E8...
2   0x01D4F9DF70F408909651C1B...   0x01835D094FA4285A90282E2...
3   0x01492D5727DB04D5693C565...   0x01608DC314883FAB89F2EB9...
```

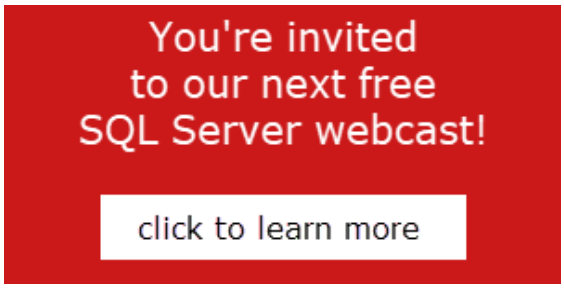
Summary

Always Encrypted provides a secure way to protect data in a particular column, in a way that is both safer and more efficient than Transparent Data Encryption. There are some limitations, though, and I talk about some of them in more depth in a recent T-SQL Tuesday post here. For a C# demo of SQL Server 2015 Always Encrypted, click here to download the project and begin experimenting.

Next Steps

- Download the latest SQL Server 2016 CTP.
- Try out Always Encrypted where you have columns you want to protect.
- See these related tips and other resources:
  - SQL Server Column Level Encryption Example using Symmetric Keys
  - Natively Encrypting Social Security Numbers in SQL Server 2005
  - Implementing Transparent Data Encryption in SQL Server 2008
  - Always Encrypted (MSDN)
  - Always Encrypted (client development) (MSDN)
  - Getting Started With Always Encrypted (SQL Server Security Blog)
  - All Encryption Tips
  - All Security Tips

Last Update: 2015-09-15



About the author

Aaron Bertrand is a Senior Consultant at SQL Sentry, Inc., and has been contributing to the community for **about two decades**, first earning the Microsoft MVP award in 1997.

View all my tips

Related Resources

- More SQL Server DBA Tips...

Post a comment or let the author know this tip helped.

All comments are reviewed, so stay on subject or we may delete your comment. Note: your email address is not published. Required fields are marked with an asterisk (*).

*Name *Email Notify for updates

*** NOTE *** - If you want to include code from SQL Server Management Studio (SSMS) in your post, please copy the code from SSMS and paste the code into a text editor like NotePad before copying the code below to remove the SSMS formatting.

p

Send me SQL tips: