Data Vault design in Azure SQL - Utilizing Customer keys to read PII information in databases.

Problem Statement:

In today's modern era, data is referred as the new oil or the new diamond. Data is the key - the power to build more accurate ML/AI models for better analysis on customers, or to build/upgrade the products.

This increase in data consumption in turn leads to many data security issues. Over past couple of years, the data security attacks have increased significantly.

Personally Identifiable Information (PII) uses data to confirm an individual's identity, so protecting it is essential for personal privacy.

With multiple clients migrating over to cloud, cloud security and protecting the PII data becomes of paramount importance to prevent any data hacks.

One can protect the data in cloud by ensuring private connections, strict Firewall rules and Identity authorization etc but the data management team having access to the data storage layer can access and/or compromise the PII data.

One way to ensure the data safety is enabling Dynamic data masking (DDM) thereby limiting sensitive data exposure by masking it to nonprivileged users.

But are there any Client-side encryption methods to ensure safety of data without compromising on the data ingestion process even for the ones having admin access on the data architecture.

Prerequisites:

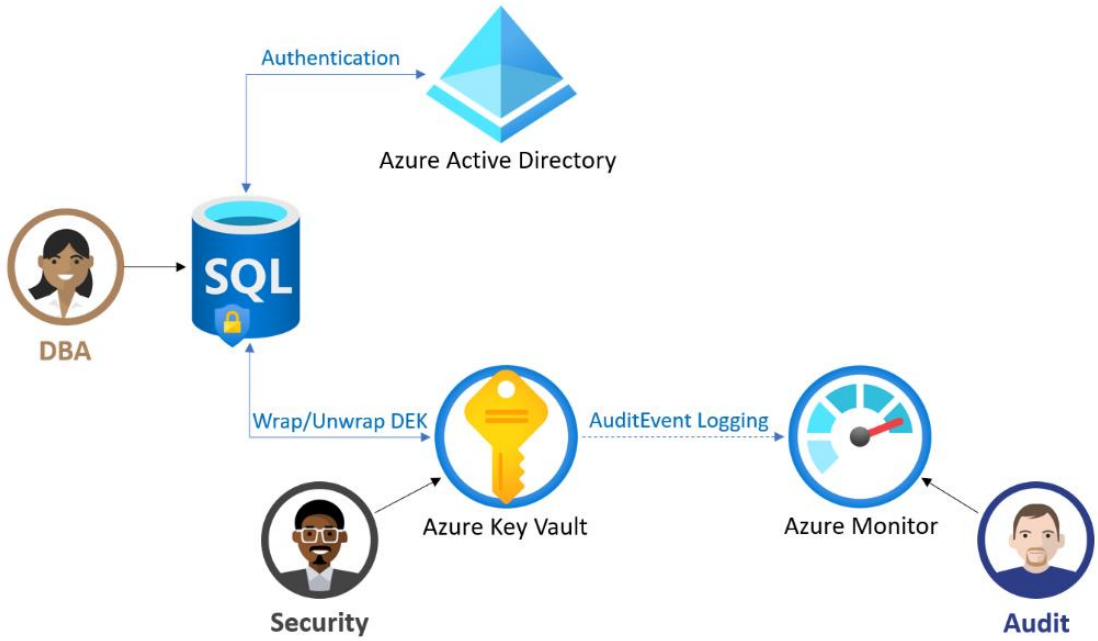
1) Azure SQL Database
2) Azure Key Vault
3) SSMS

Solution:

In Azure SQL Database, there is server-side encryption like Transparent data encryption to protect data at rest. But a DAB having access to the database can still access the entire data in plain text despite Server-side encryption enabled. Hence, there needs to be Client-side encryption in combination with Server-side encryption to ensure PII data is always protected and only accessible to data owners governing the encryption keys to decrypt the data. For others including the DBA will only be able to see data in encrypted state and not the underlying plain text data.

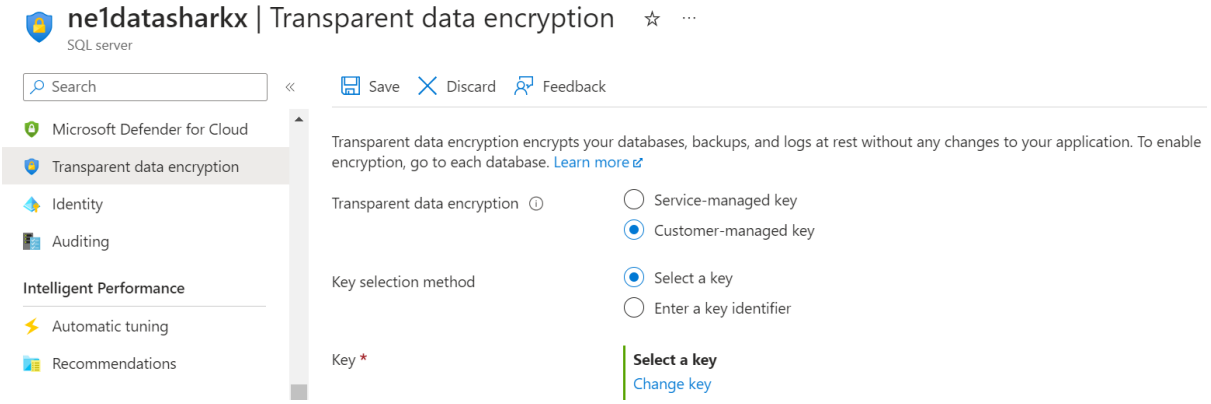
Azure SQL database provides an out of box functionality of Always encrypted to accomplish Client-side encryption. To protect the data in memory from data theft, one can use Always encrypted which encrypts sensitive data in memory or in use during computations.

Overview of customer managed TDE work (reference: Microsoft Doc):

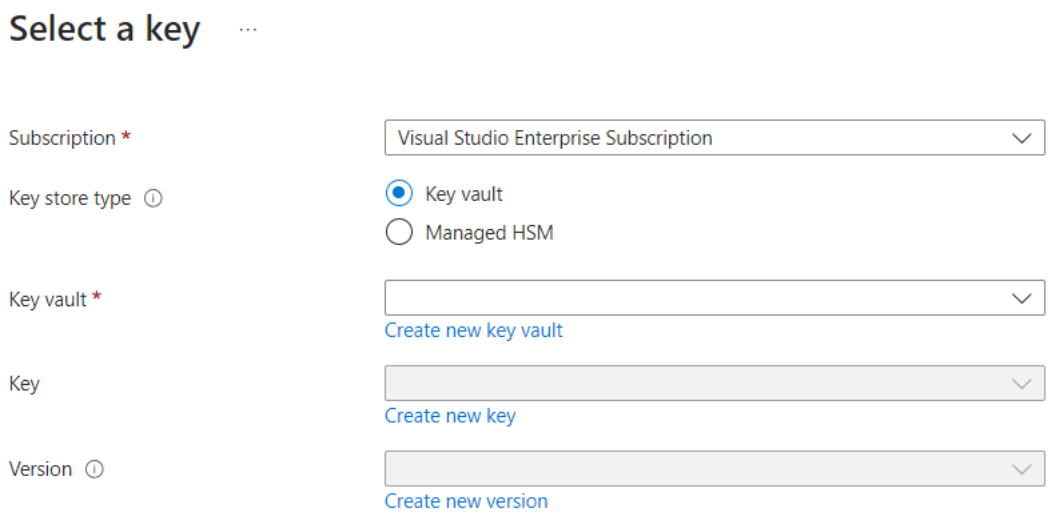


Implementation:

1) In Azure Portal, navigate to the Azure SQL server configuration and proceed to the Transparent data encryption security setting.



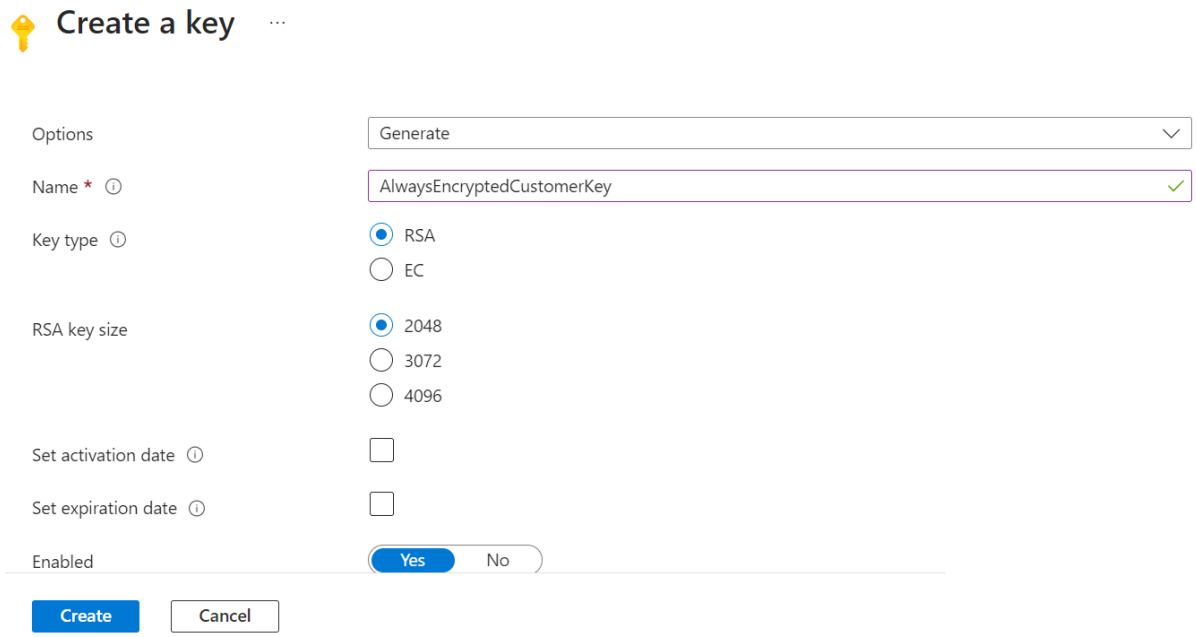
2) To use Bring your own key (BYOK), Click on Customer-managed key; then navigate to “Select a Key” and finally Click on “Change key”.
On the next page, you need to create a new Azure Key Vault (AKV) or select an existing AKV.



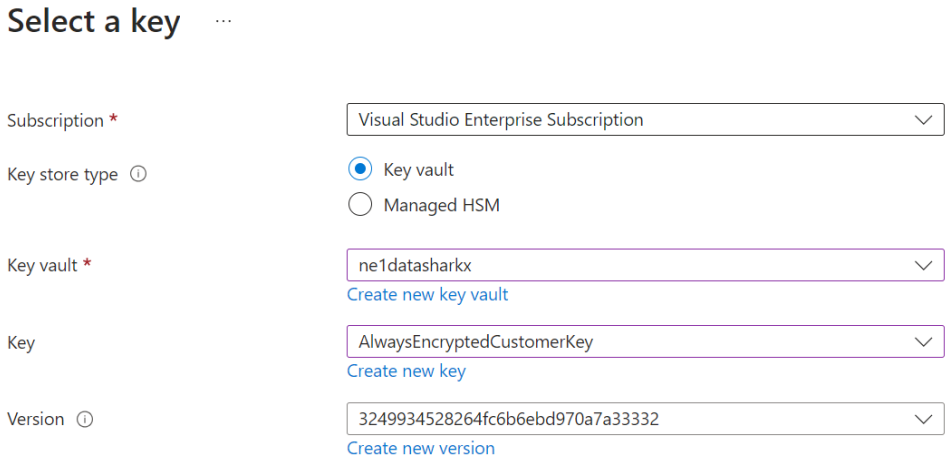
Steps to Create a new AKV (reference: [Microsoft doc](#)).

In case if there is already an AKV, select it from the drop down.

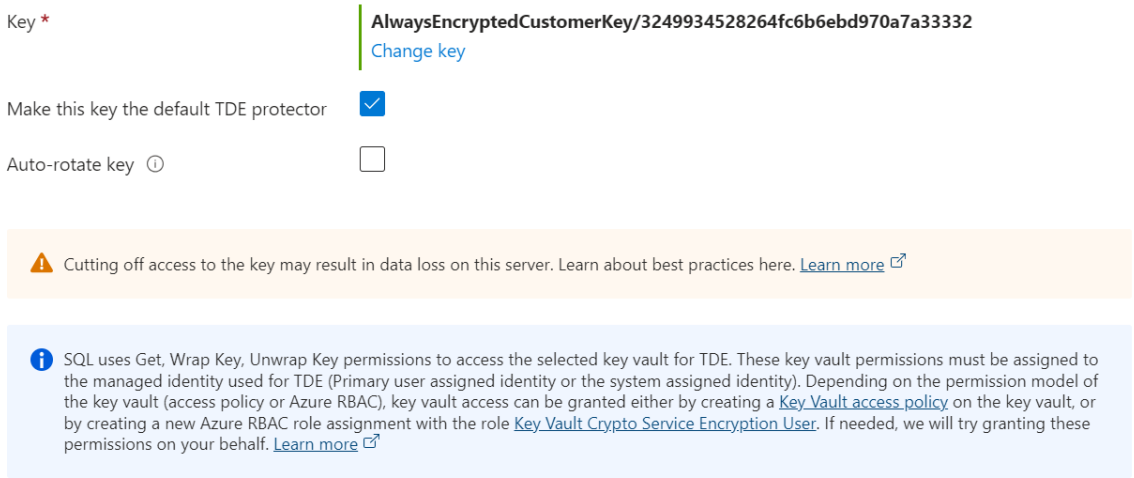
In the key field, click on Create new key. It redirects you to a page, as shown below. On this page, enter the key name, select key type as RSA and use the default RSA key size as 2048.



In the final step, Select the version of the key.

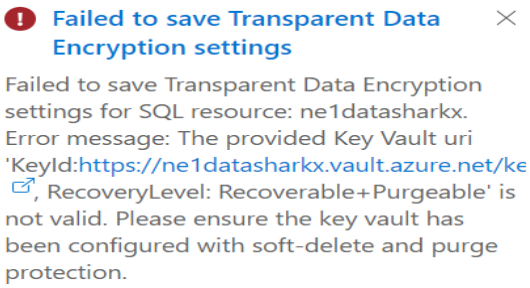


Click on Select, and you can view the key configurations in the customer-managed key selection.



Note: At the bottom, it gives a message, "SQL uses Get, Wrap Key, Unwrap Key permissions to access the selected key vault. These permissions are only used to access the key vault for TDE."

Also, the AKV must have Soft-delete and Purge protection enabled else there would be the below error:

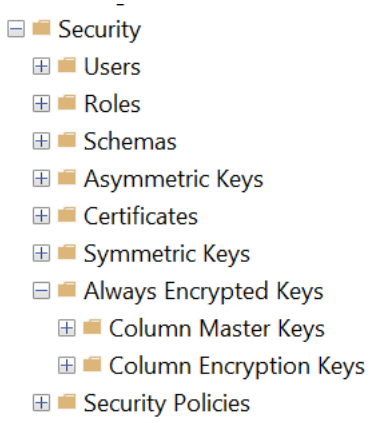


3) Create the Encryption Keys

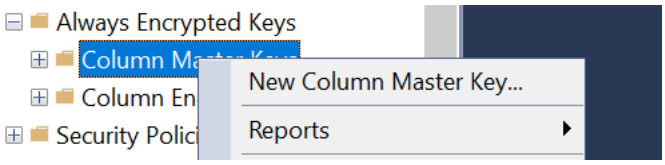
I) Column Master Key

a. Login to the Azure SQL database in which we need to enabled encryption via an AD account via SSMS.

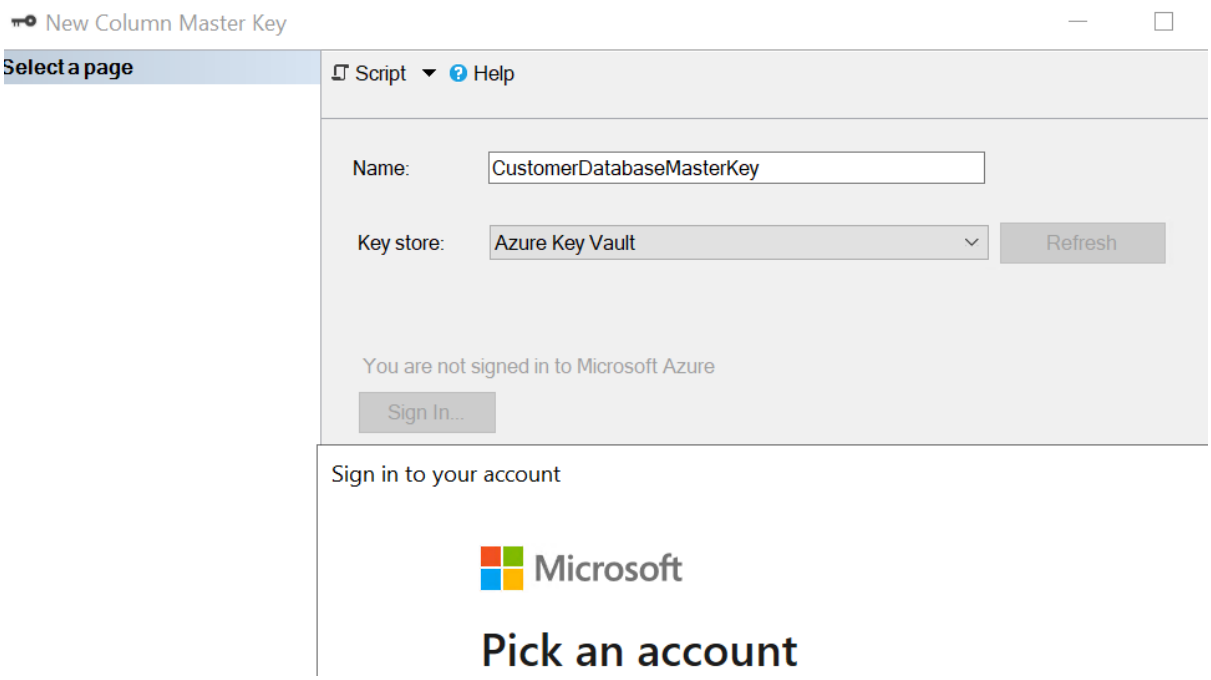
b. Expand the Database, expand the Security and finally expand the Always Encrypted Keys.



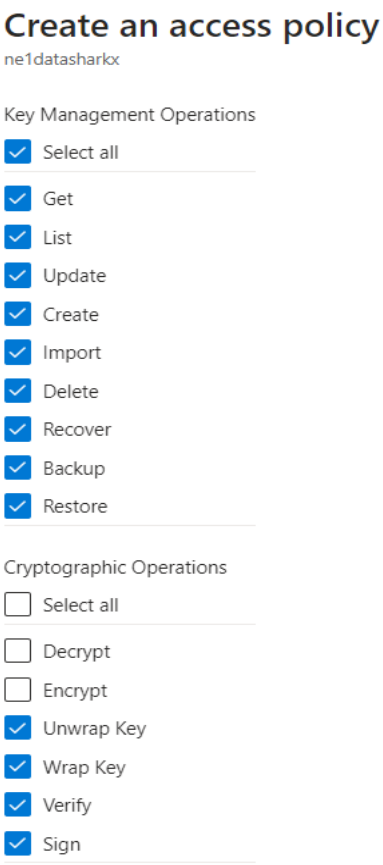
c. Right-click on the Column Master Keys folder and select New Column Master Key to launch the New Column Master Key wizard.



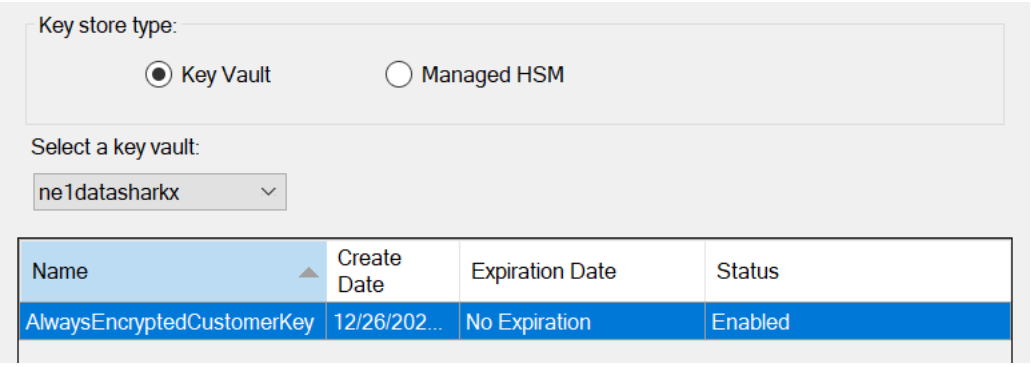
d. Provide the name for the key and Select AKV from the key store drop down. Click on the Sign In and authenticate with the necessary credentials.



Note: The Credential should have **create, get, list, sign, verify, wrap** and **unwrap** permissions within Access Policies of the AKV



e. Upon successful authentication, select the correct subscription, AKV and key from the options presented. Click OK to close the wizard.

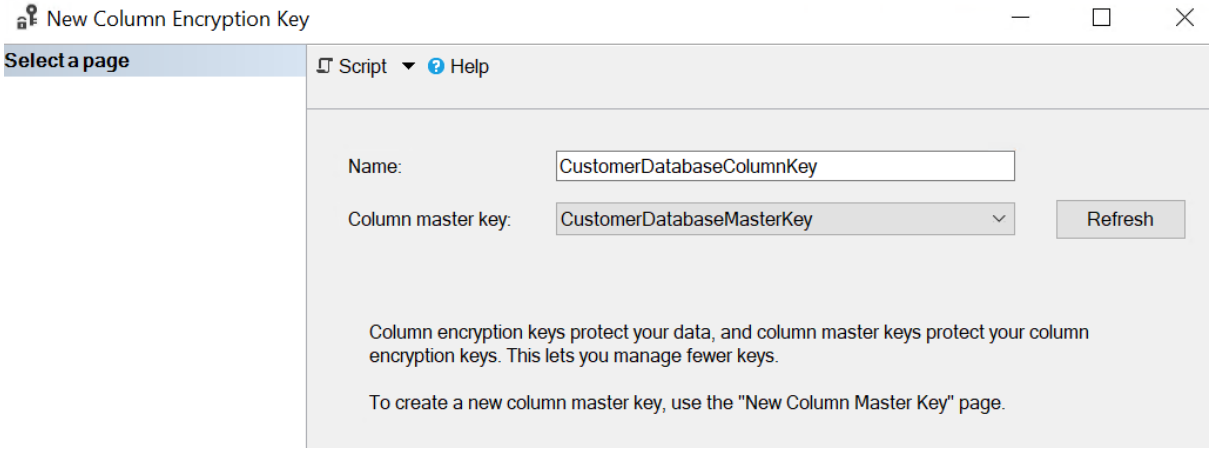


Script version:

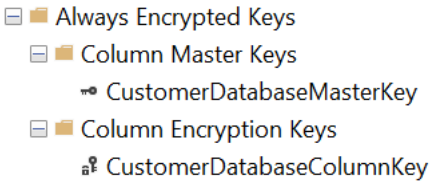
```
CREATE COLUMN MASTER KEY [CustomerDatabaseMasterKey]
WITH
(
    KEY_STORE_PROVIDER_NAME = N'AZURE_KEY_VAULT',
    KEY_PATH = N'https://ne1datasharkx.vault.azure.net/keys/AlwaysEncryptedCustomerKey/b2d24046035d41cf88d82212a9f4f505'
)
GO
```

II) Column Encryption Key

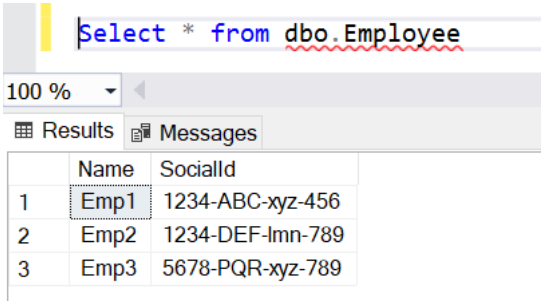
a. Similarly, right-click on the Column Encryption Keys folder and select New Column Encryption Key to launch the New Column Encryption Key wizard. Provide a name for the key and select the appropriate column master key to protect the column encryption key. Click OK to close the wizard.



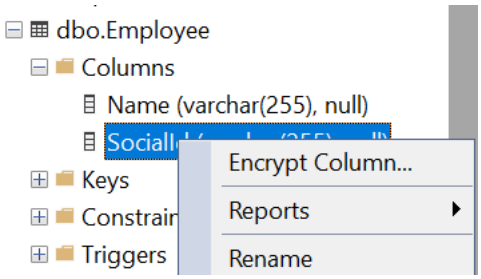
Both Keys:



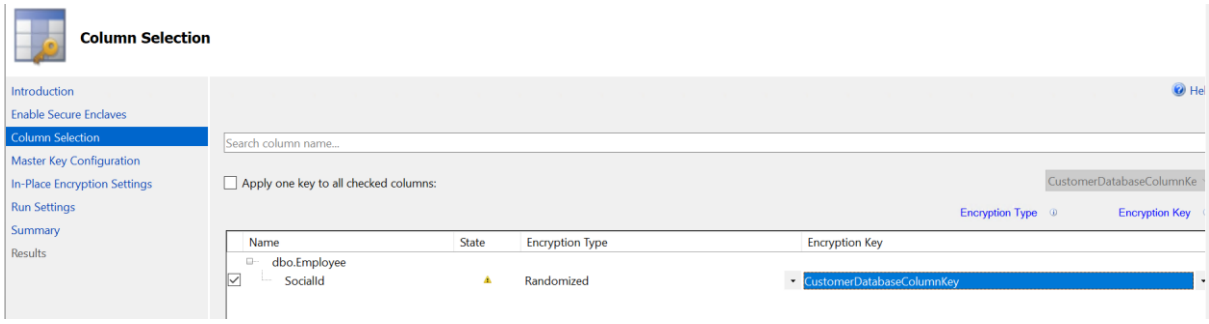
4) Apply Encryption on Tables
   a. In our use case, the sample data consists of Employee details consisting of Name and SocialId (PII : which should be encrypted)



   b. Expand the Database, expand the Tables and finally expand the Columns of the table selected.
   Right-click the column and chose Encrypt Column. The Always Encrypted wizard would open.

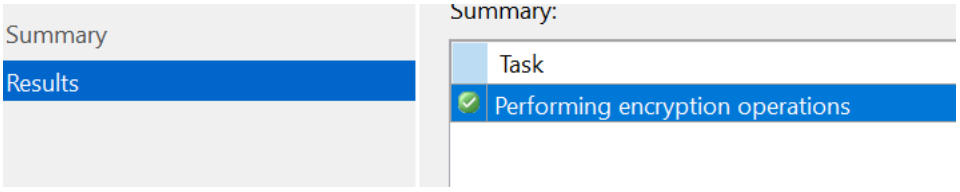


   c. On the Column Selection page click the check box next to the column(s) to be encrypted and choose either deterministic or randomized in the Encryption Type and select the column encryption key created earlier in the Encryption Key column.



d. Click Next and Proceed to Finish Now.

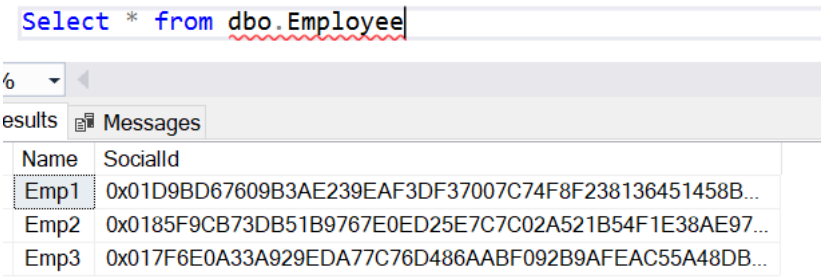
Note: If the table has huge amount of data or is actively being written(transactional), it is better to change the option to Generate PowerShell script to run later and schedule the encryption to run during an off hour's maintenance time.



e. Updates table definition

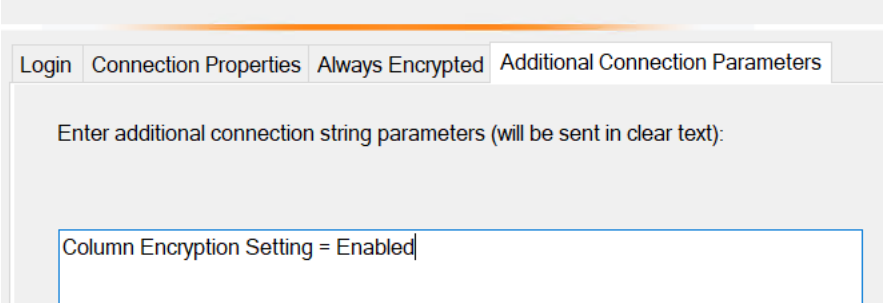
```
CREATE TABLE [dbo].[Employee](
    [Name] [varchar](255) NULL,
    [SocialId] [varchar](255) COLLATE Latin1_General_BIN2 ENCRYPTED WITH (COLUMN_ENCRYPTION_KEY = [CustomerDatabaseColumnKey], ENCRYPTION_TYPE = Randomized, ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256') NULL
) ON [PRIMARY]
GO
```

Select Query execution on the Employee table with SocialId column being encrypted.

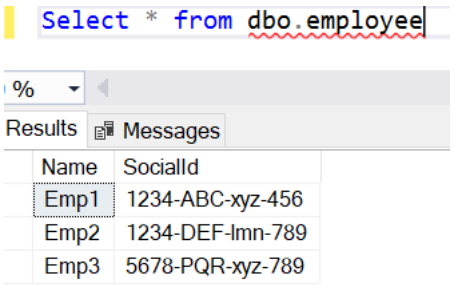


5) View Decrypted version of data.

a. Create a new connection to the database via SSMS with the below Config. When connection to the server expand the Options on the Connect to Server dialog, switch to the Additional Connection Parameters page and enter the text Column Encryption Setting = Enabled and click Connect.



b. When running queries in SSMS against a table with encrypted data you may be prompted to authenticate to Azure before being shown the decrypted data.



Challenges in implementing the solution:

1) Any misstep with the customer managed key or accidental Azure Key Vault deletion would lead to loss of database access or database being in inaccessible state.
2) Unlike service managed key, where Azure will take care of key management and rotation seamlessly with no user intervention required; in customer managed key, user needs to manage the backup and recovery of the keys which adds an additional overhead.

Business Benefit:

1) Customer-managed key allows separation of duties between management of keys and data to help meet compliance with organizations security policies.
2) Key Vault administrator can revoke key access policy permissions to make revoke the access on database.
3) TDE with customer-managed keys improves on service-managed keys by enabling central management of keys in Azure Key Vault, giving customers full and granular control over usage and management of the TDE protector.
4) Allows central management of Keys in Azure key vault.