Azure Key Vault (AKV) integration with SQL Azure

Last updated by | Soma Jagadeesh | Jan 10, 2021 at 9:46 PM PST

Contents

TDE

Transparent Data Encryption (TDE) provides encryption-at-rest for the physical media that the data and log files are stored on (drives or backup tapes). In TDE, the data files are encrypted by a symmetric key, called the Database Encryption Key (DEK), and the DEK is encrypted by a TDE Certificate. TDE with Azure Key Vault support extends TDE to support encryption by asymmetric key stored in Azure Key Vault (AKV). This allows SQL Azure customers to assume control over the encryption key used in TDE. The data files would still be encrypted by the DEK, but instead of having a TDE Certificate protect the DEK, an asymmetric key, called the Key Encryption Key (KEK) or the TDE Protector, encrypts the DEK. This KEK will come from the customer's own Azure Key Vault, and the customer can manage key rotations, key drops, auditing/reporting of keys through Azure Key Vault.

This feature allows customers to configure their TDE setup with pre-defined keys that they control themselves.

Azure Key Vault is Microsoft's cloud key management service, and it provides customers with a central management platform for sensitive strings (called "secret" objects) and encryption keys (called "keys"). It

offers the option of storing AKV objects in hardware security modules (HSMs) at a lower price point than the customer purchasing one on their own.an

Existing service-managed TDE functionality will continue to work as is for customers who do not want to manage their own keys.

TDE with Azure Key Vault provides the following benefits:

- **Increased transparency** and granular control with the ability to manage the TDE key themselves.
- **Central management** and organization of keys and respective backups by hosting them in Azure Key Vault. All keys and secrets used across Azure can be stored in one place.
- **Separate key management from data management** within an organization, so not all DBAs have access to all encryption keys.
- **Greater trust** from customers' own clients because they can cut off SQL's access to the encryption keys at any time.
- When you rotate a key in Azure Key Vault to a new version, the rotation is applied automatically to all SQL databases under a given logical server that is using the key

TDE with Azure Key Vault is NOT meant for the following:

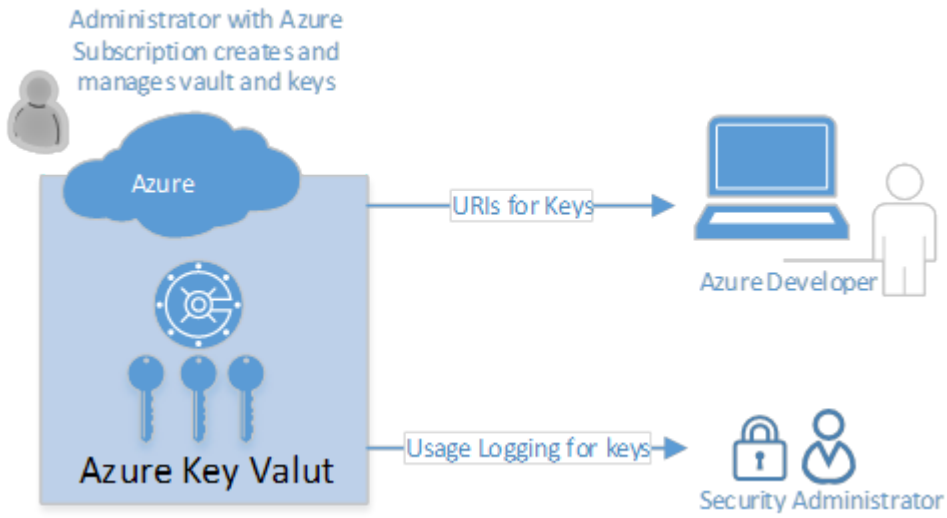
- *Automatic* key rotations for TDE will not be supported in this feature.
- Customers who like and value Microsoft handling the key rotation should use service-managed Azure SQL TDE.
- Cross tenancy between SQL and AKV and moving between tenancy is not supported. Moving SQL between tenancy must be blocked. Example:
  - Customers have asked for Server@contoso.com, AKV@boeing.com, and TDE using key vault. This is out of scope and will not work in V1. The SQL Server and AKV must be in the same tenant.
  - Block moving SQL Server from Server@contoso.com to Server@boeing.com. There is no mechanism for moving a SQL Server from server@contoso.com to Server@boeing.com with a mechanism for the customer to fix up authentication and permissions in SQL for @contoso.com accounts.
  - However, Azure SQL DB Server@constoso.com, users in @boeing.com tenancy, and AAD Auth will work. So a work-around is to keep SQL and AKV in the same tenant and if needed the SQL Users in a different tenancy.

Key Scenarios

1. Customers who use Azure Key Vault today can now extend their AKV management to also include TDE keys.
2. Customers who have clients who do not trust Microsoft with the encryption keys. Some of these customers consider the lack of customer-managed keys to be a blocker for TDE adoption.
3. Customers can cut off key access to a specific AAD user or ID whenever they want to.
4. Customers can audit key access through AKV.
5. Customers can rotate TDE keys directly through AKV by adding a new key version. The Azure SQL Server will automatically detect the new key version and rotate to the new key.

TDE with Azure Key Vault Support

Supported inAzure SQL Database (including elastic pools) and Azure SQL Data Warehouse.



There is one TDE key per logical server, so all database on the same logical server will use the same key.

TDE is a complex feature that touches upon many core functionalities within SQL, and leverages other Azure services, such as Managed Service Identity (MSI), Azure Active Directory (AAD), and Azure Key Vault (AKV).

Dependencies

- Azure Key Vault Services
- Managed Service Identity(MSI)
- Azure Active Directory
- Windows Fabric

6. Availability

Since the encryption key is managed by the customer, if the customer deletes or removes permissions from their key vault, the database will not be online until it gets access to the vault.

7. Backup and Restore

As part of backups, we will need to keep a list of AKV keys for the database so that database can be restored without any problems.

8. GeoDR

Just like our backup and restore scenarios, we will need to pass through the AKV Uri to the target servers as part of the GeoLink workflow.

9. Managed Service Identity (MSI)

In order to authenticate the user and the SQL DB service as a resource provider to the customer's AKV, we create an MSI identity for each logical SQL server and the SQL DB service. MSI is currently in preview in limited Azure regions, so our preview and GA timelines and dependent on their service being globally available.

10. SAWA v1 Migration

Our MSI integration implementation depends on Management Service migrating their primary clusters from SAWA v1 to Sterling.

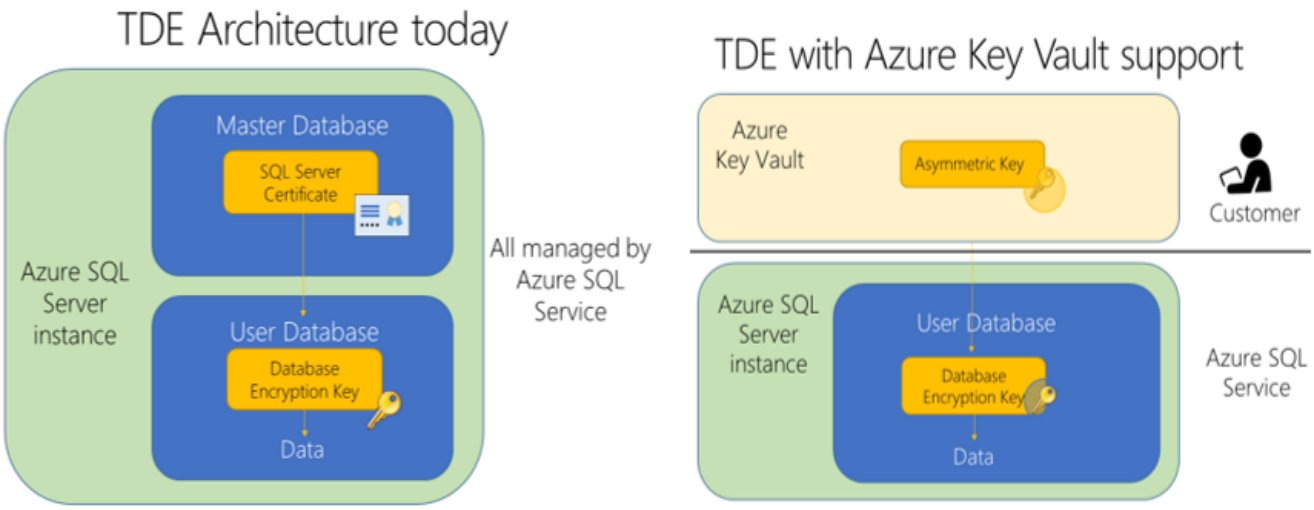
11. Azure Active Directory (AAD)

Under the covers of MSI, each MSI identity is coupled to an AAD Client ID and service principal certificate. In order for SQL DB to access the customer's Azure Key Vault, SQL DB receives an MSI identity that was generated for the MSI service through the AAD service. If the AAD service were down, we would not be able to onboard customers to TDE with AKV.

12. Azure Key Vault (AKV)

Since the TDE encryption key is now stored in AKV, we must be prepared to help the customer regain access to their encrypted database if the AKV service goes down. Currently their availability is 99.995%. Key deletions and backups will also be managed through AKV, and will need to documented clearly.

How it Works



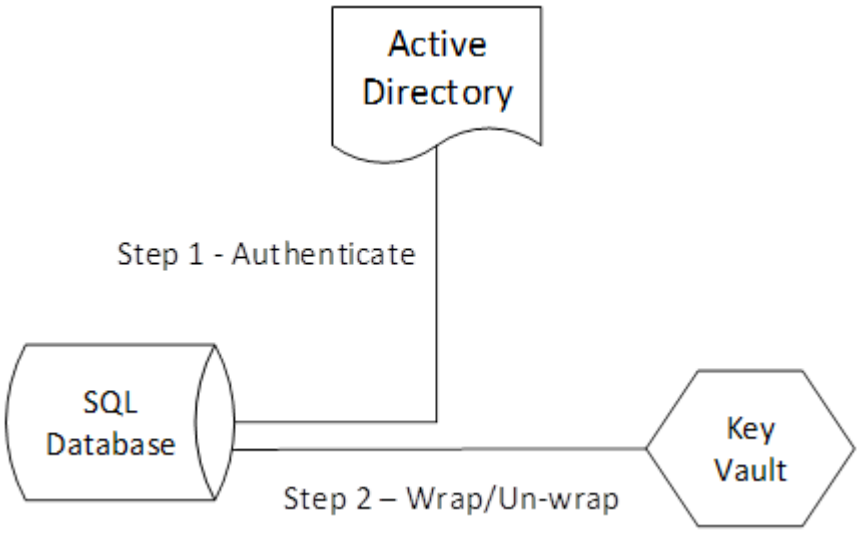
Service-managed keys (today in Azure SQL): Azure manages creation and rotation of keys and certificates involved in TDE.

User-managed keys: Your organization manages creation and rotation of the TDE Protector key through Azure Key Vault.

Key Architecture Topics

1. The TDE protector is configured at the Logical Server level. The ACLs at AKV is done at Server scope.
2. All the primary databases under one logical server will be have same type of TDE protector. If they configure to use AKV protector at Logical Server level, then all database which has TDE ON will have their DEK protected with key in AKV. Existing databases which have Service Managed Protector will be switched to AKV if AKV is configured as the Encryption Protector.
3. Geo-replication enables creating up to 4 replica (secondary) databases in different data center locations (regions). Secondary databases are available in the case of a data center outage or the inability to connect to the primary database. Standard databases can have one non-readable secondary and must use the recommended region. Premium databases can have up to four readable secondaries in any of the available regions. All servers involved in data replication should be able to access the same AKV with the same keys.

4. For backup and restore we expect customer to keep all the Key they have used for their logical server.
For all authentication purposes the SQL Database Engine is the proxy for AKV communication



An overview of how to get started:

1. Create an Azure Key Vault and an encryption key under an Azure subscription.
2. Once the key is created, save a key backup through Azure Key Vault.
3. Set up an Azure Active Directory service principal to represent an Azure SQL server.
4. Add the Key Vault key to the server, and set it as the TDE protector.
5. Enable TDE.

Note: The private preview will be using AAD client IDs and secrets for ACLing the Azure SQL Server to the Azure Key Vault. By public preview, the ACLing will be done automatically through Managed Service Identity (MSI) to improve usability.

Changes to CMS database

Schema Upgrade

The highlighted columns will be added to the existing tbl_secrets and tbl_logical_databases CMS tables

[tbl_secrets]

	Column Name	Data Type	Nullable	Description
PK	ConsumerName	Nvarchar(128)	False	Name of the consumer, could be instance or instance group
PK	ConsumerType	Int	False	An integer enum for each consumer type
PK	SecretType	Int	False	Cert, SAS key, Symmetric key, etc.
PK	GenerationId	int	False	Generation or version of the secret
	PublicBlob	Varbinary(max)	True	Public key of the blob (could be null for symmetric keys)
	PrivateBlob	Varbinary(max)	False	Private key
	IssuerName	Nvarchar(128)	False	Who issued this key – to track changes to issuer
	IssuerType	Int	False	Type of issuer
	Properties	XML	True	Customer properties for this secret (e.g. key size, expiration, policy or container name for SAS keys, etc.)
	CreationDate	Date	False	Date, the secret was created
	MarkedForDeletion	Bit	False	A bit is set if the secret should be deleted from clients
	AkvUri	nvarchar(256)	True	Azure Key Vault Uri

[tbl_logical_databases]

Update the logical_databases table to identify the kind of TDE encryption being used (Managed Vs Akv)

	Column Name	Data Type	Nullable	Description
PK	logical_server_name	Nvarchar(128)	False	
PK	logical_database_id	Int	False	
	sql_instance_name	Int	False	
	logical_database_name	int	False	
	State	Nvarchar(128)	False	
	State	Nvarchar(18)		
	Dropped_time	Datetime(7)	True	
	Edition	Nvarchar(50)	True	
	Max_size_bytes	Nvarchar(50	False	
	..	Date		
	encryption_type	Nvarhcar(128)	True	The type of encryption being used for the database. This is Service Managed vs AKV. This is set on the logical server level.
	requested_encryption_type	Nvarchar(128)	True	This is when customers changed the encryption type to be used on the database level.

[tbl_logical_servers]

Update the logical_servers table with ClientId/Application id that is generated by AAD. This will be used to authenticate against AKV.

Column Name	**DataType**	**Allow Null**	Description
logical_server_id	uniqueidentifier	FALSE	
state	nvarchar(128)	FALSE	
type	nvarchar(128)	FALSE	
sql_instance_group_name	nvarchar(128)	TRUE	
control_ring_dns_name	nvarchar(256)	TRUE	
customer_subscription_id	uniqueidentifier	FALSE	
admin_login_name	nvarchar(128)	FALSE	
admin_login_password	varbinary(1024)	FALSE	
request_id	uniqueidentifier	TRUE	
workflow_position	sys.hierarchyid	TRUE	
next_successor	int	FALSE	
create_time	datetime2(7)	FALSE	
last_update_time	datetime2(7)	FALSE	
last_state_change_time	datetime2(7)	FALSE	
....			
....			
client_id	nvarchar	TRUE	Client Id returned from AAD authentication system.

Winfab Properties

For each fabric service, a fabric property will be created which will store the AkvUri secret types.

Name	Format
AzureKeyVautUri	https://ContosoKeyVault.vault.azure.net/keys/ContosoFirstKey/cgacf4f763ar42ffb0
ClientId	8f8c4bbd-485b-45fd-98f7-ec6300b7b4ed
PartnerAkvUri (ForGeoDr)	https://ContosoKeyVault.vault.azure.net/keys/ContosoFirstKey/cgacf4f763ar42ffb0
PartnerAkvUri (ForGeoDr)	https://ContosoKeyVault.vault.azure.net/keys/ContosoFirstKey/cgacf4f763ar42ffb0
AkvSecret	Certificate Info

DDL changes

The CREATE ASYMMETRIC key DDL syntax will be extended to take the Azure Key Vault URI path. This new syntax won't be exposed to customer and will be invoked internally by NodeAgent. We will add a new provider called Internal_Provider. The provider name will differentiate different types of provider. For our case it will be "AzureKeyVault_Prov". In future, we can support other types of providers. The Key_URI will represent the URI of the key in the provider.

CREATE ASYMMETRIC KEY Asym_Key_Name

[AUTHORIZATION database_principal_name]

{

[FROM <Asym_Key_Source>]

|

WITH <key_option>

[ENCRYPTION BY <encrypting_mechanism>]

<Asym_Key_Source>::=

FILE = 'path_to_strong-name_file'

|

EXECUTABLE FILE = 'path_to_executable_file'

```
|
ASSEMBLY Assembly_Name
|
PROVIDER Provider_Name<key_option> ::=
ALGORITHM = <algorithm>
|
PROVIDER_KEY_NAME = 'key_name_in_provider'
|
CREATION_DISPOSITION = { CREATE_NEW | OPEN_EXISTING }
  |
  INTERNAL_PROVIDER Provider_Name
KEY_URI = ‘key_path_uri’
}
<algorithm> ::=
{ RSA_512 | RSA_1024 | RSA_2048 }
<encrypting_mechanism> ::=
PASSWORD = 'password'
```

Catalog view changes

A new column will be added sys.asymmetric_keys catalog view

Column name	Data type	Description
name	sysname	Name of the key. Is unique within the database.
principal_id	int	ID of the database principal that owns the key.
asymmetric_key_id	int	ID of the key. Is unique within the database.
pvt_key_encryption_type	char(2)	How the key is encrypted. NA = Not encrypted MK = Key is encrypted by the master key PW = Key is encrypted by a user-defined password SK = Key is encrypted by service master key.
pvt_key_encryption_type_desc	nvarchar(60)	Description of how the private key is encrypted. NO_PRIVATE_KEY ENCRYPTED_BY_MASTER_KEY ENCRYPTED_BY_PASSWORD ENCRYPTED_BY_SERVICE_MASTER_KEY
thumbprint	varbinary(32)	SHA-1 hash of the key. The hash is globally unique.
algorithm	char(2)	Algorithm used with the key. 1R = 512-bit RSA 2R = 1024-bit RSA 3R = 2048-bit RSA
algorithm_desc	nvarchar(60)	Description of the algorithm used with the key. RSA_512 RSA_1024

		RSA_2048
key_length	int	Bit length of the key.
sid	varbinary(85)	Login SID for this key. For Extensible Key Management keys this value will be NULL.
string_sid	nvarchar(128)	String representation of the login SID of the key. For Extensible Key Management keys this value will be NULL.
public_key	varbinary(max)	Public key.
attested_by	nvarchar(260)	System use only.
provider_type	nvarchar(120)	Type of cryptographic provider: CRYPTOGRAPHIC PROVIDER = Extensible Key Management keys NULL = Non-Extensible Key Management keys INTERNAL PROVIDER
cryptographic_provider_guid	uniqueidentifier	GUID for the cryptographic provider. For non-Extensible Key Management keys this value will be NULL. For AzureKeyVault_Prov it would be A5863D19-01D3-4B17-9B7B-3FD7CB69DFB0
cryptographic_provider_algid	sql_variant	Algorithm ID for the cryptographic provider. For non-Extensible Key Management keys this value will be NULL.
key_path	**nvarchar(4000)**	**A provider specific path of the key. For Azure SQL DB, the only valid provider will be Azure Key Vault, so this must be the path to an Azure Key Vault Key.** **Includes Key Version in AKV case.**

Import/Export

TDE is ignored during Import/Export.

Powershell Cmdlets

For reference, the current TDE cmdlets are:

PS C:\> **Set-AzureRMSqlDatabaseTransparentDataEncryption** -ServerName "myserver" -ResourceGroupName "Default-SQL-WestUS" -DatabaseName "database1" -State "Enabled"

PS C:\> **Get-AzureRMSqlDatabaseTransparentDataEncryption** -ServerName "myserver" -ResourceGroupName "Default-SQL-WestUS" -DatabaseName "database1"

PS C:\> **Get-AzureRMSqlDatabaseTransparentDataEncryptionActivity** -ServerName "myserver" -ResourceGroupName "Default-SQL-WestUS" -DatabaseName "database1"

Best Practices / Customer Compliance Responsibilities

Recommended key vault organization:

1. Keep all application-related keys for a single application within the same Key Vault
a. Alternatively, user can create a Security Group (an AAD group) and add applications into the Security Group. Then ACL the security group to AKV
1. 1 Key per data type or resource
2. 1 Resource group could govern multiple departments. Each department owns multiple key vaults

Encryption at Rest recommendations:

3. Store root key for separate encryption features into separate key vaults to prevent different entities from accessing the same key vault (higher protection)
a. Ex. Store Key1 in AKV1 for VM Disk encryption; Key2 in AKV2 for TDE

Recommended key creation & maintenance workflow:

1. Create an HSM-backed key locally
2. Import the key to AKV
a. Note: Do not add an expiration date to the key when importing it to Azure key Vault. It will not be accepted by the Azure SQL service, because an expired key will cause the SQL resource to become unavailable. If an expiration date must be set, please set it to "12/31/9999."
1. Before using it first-time, take a key backup
2. Any time the user makes key metadata changes (add ACLs, add tags, change exp date, attributes), take another backup
a. Note: taking a backup in Azure Key Vault is a key transaction which returns a backup file. This backup file can be saved anywhere.

Customer is responsible for defining rotation policies and enforcing them. Customer is highly encouraged NOT to delete previous rolled over keys, since old backups and LTR backups rely on those keys.

Customer is responsible for configuring GeoDR of both Azure SQL DB databases. Target servers must have access to a copy of the source server's TDE protector in order to read in the secondary database. Azure Key Vault in any region will work with Azure SQL DB database in any other region while allowing Azure SQL DB to maintain its availability SLAs.

Classification

Root cause Tree - Security/User Request/How-to/advisory

How good have you found this content?