Microsoft

Manage Security for Azure SQL Database

Module 5

Learning Units covered in this Module

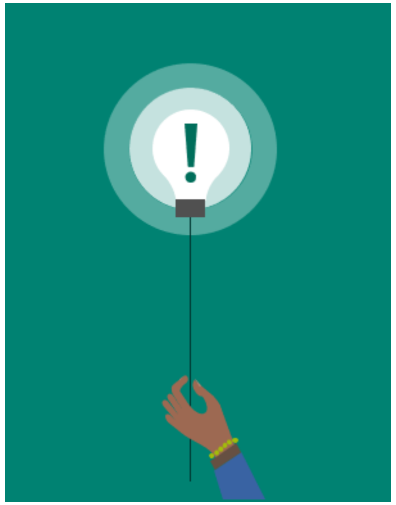
- Lesson 1: Implement Data Encryption In Transit
- Lesson 2: Implement Transparent Data Encryption
- Lesson 3: Implement Always Encrypted
- Lesson 4: Implement Row Level Security
- Lesson 5: Implement Dynamic Data Masking

Lesson 1: Implement Data Encryption In Transit

Objectives

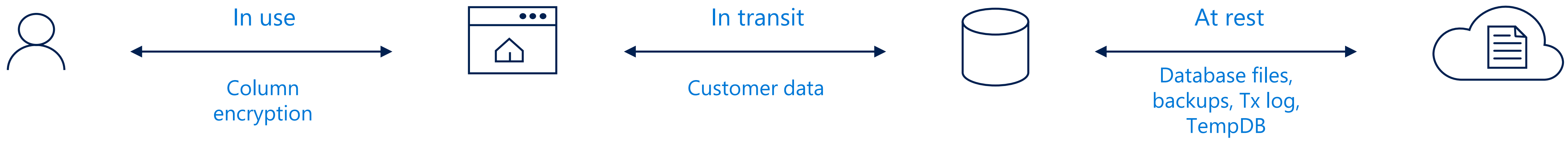
After completing this learning, you will be able to:

- Know the types of data encryption for an Azure SQL Database.
- Implementing Transport Layer Security (TLS) to protect data in transit.

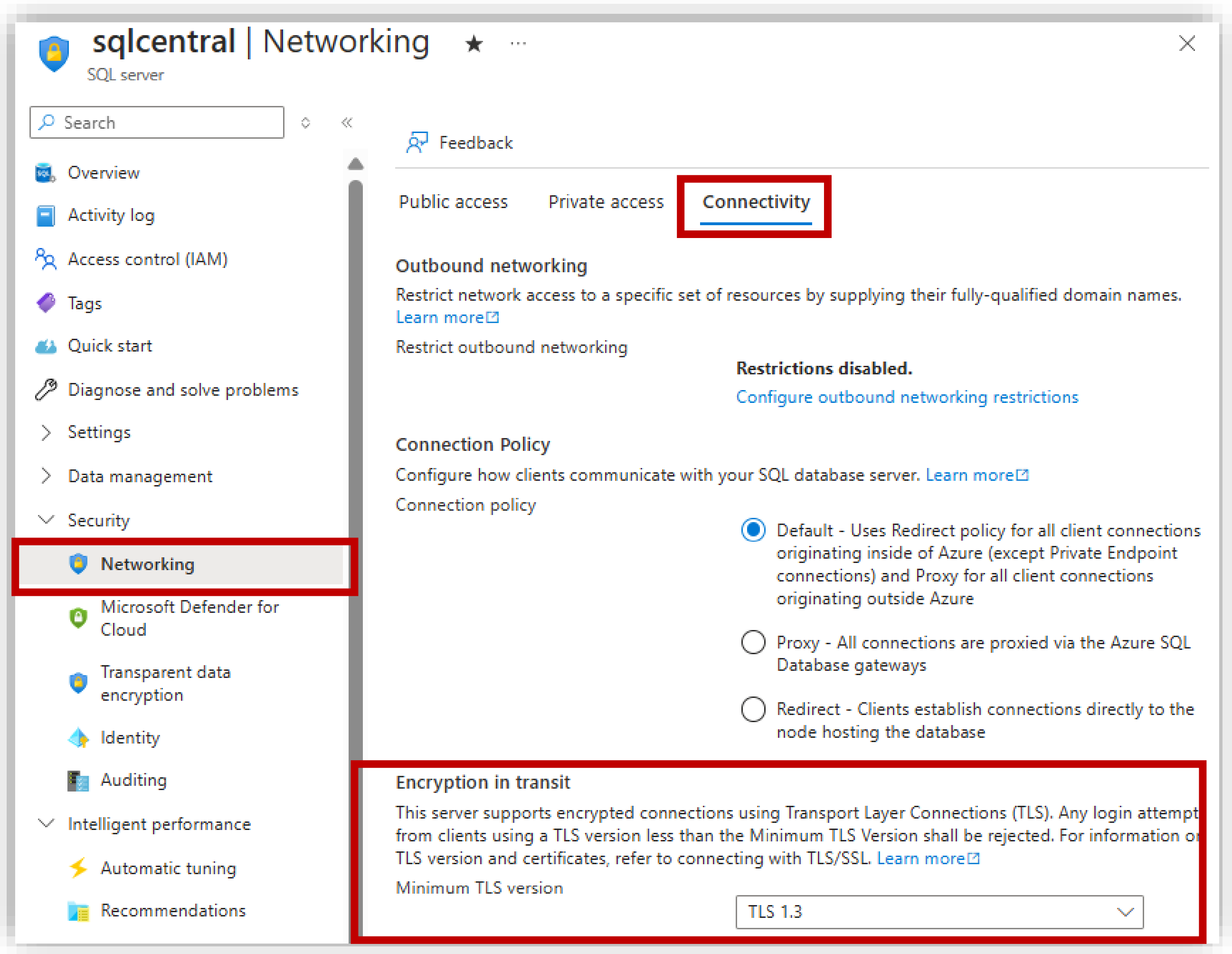


Types of data encryption

Data encryption	Encryption technology	Customer value
In transit	Transport Layer Security (TLS) from the client to the server.	Protects data between client and server against snooping and man-in-the-middle attacks. NOTE: Azure SQL Database is phasing out Secure Sockets Layer (SSL) 3.0 and TLS 1.0 in favor of TLS 1.2.
At rest	Transparent Data Encryption (TDE) for Azure SQL Database.	Protects data on the disk. Key management is done by Azure, which makes it easier to obtain compliance.
In use (end-to-end)	Always Encrypted for client-side column encryption.	Data is protected end-to-end, but the application is aware of encrypted columns. This is used in the absence of data masking and TDE for compliance-related scenarios.



Data encryption – In Transit



Outbound networking

Restrict network access to a specific set of resources by supplying their fully-qualified domain names. Learn more

Restrict outbound networking

Restrictions disabled.

Configure outbound networking restrictions

Connection Policy

Configure how clients communicate with your SQL database server. Learn more

Connection policy

● Default - Uses Redirect policy for all client connections originating inside of Azure (except Private Endpoint connections) and Proxy for all client connections originating outside Azure

○ Proxy - All connections are proxied via the Azure SQL Database gateways

○ Redirect - Clients establish connections directly to the node hosting the database

Encryption in transit

This server supports encrypted connections using Transport Layer Connections (TLS). Any login attempt from clients using a TLS version less than the Minimum TLS Version shall be rejected. For information o TLS version and certificates, refer to connecting with TLS/SSL. Learn more

Minimum TLS version: TLS 1.3

Microsoft gives customers the ability to use the Transport Layer Security (TLS) protocol to protect data when it's traveling between the cloud services and customers.

TLS provides strong authentication, message privacy, and integrity (enabling detection of message tampering, interception, and forgery)

You should only use TLS 1.2 or higher as the earlier protocols are vulnerable and have been deprecated.

Questions?

Knowledge Check

What is the minimum version of TLS that should be used?

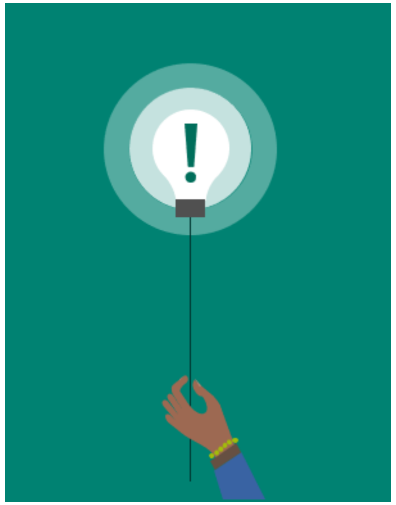
Name the feature to encrypt the data both at rest and motion.

Lesson 2: Implement Transparent Data Encryption

Objectives

After completing this learning, you will be able to:

- Know how to secure data at rest using Transparent Data Encryption.



Understanding TDE Functionality

Data is encrypted at rest.

Encryption keys are managed by Azure.

Performs real-time I/O encryption and decryption of the data at the page level.

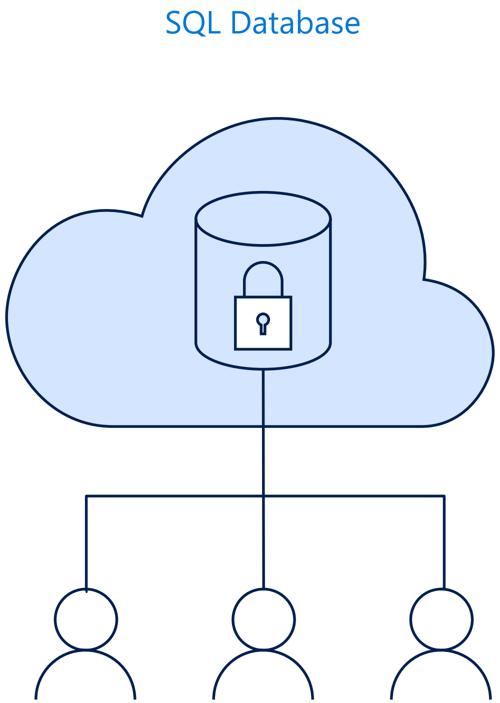
Each page is decrypted when it's read into memory and then encrypted before being written to disk.

TDE is enabled for all newly deployed Azure SQL Databases.

No need for application change.

Support for equality operations (including joins) on encrypted data.

Bring You Own Key (BYOK) supported.



Encryption Keys

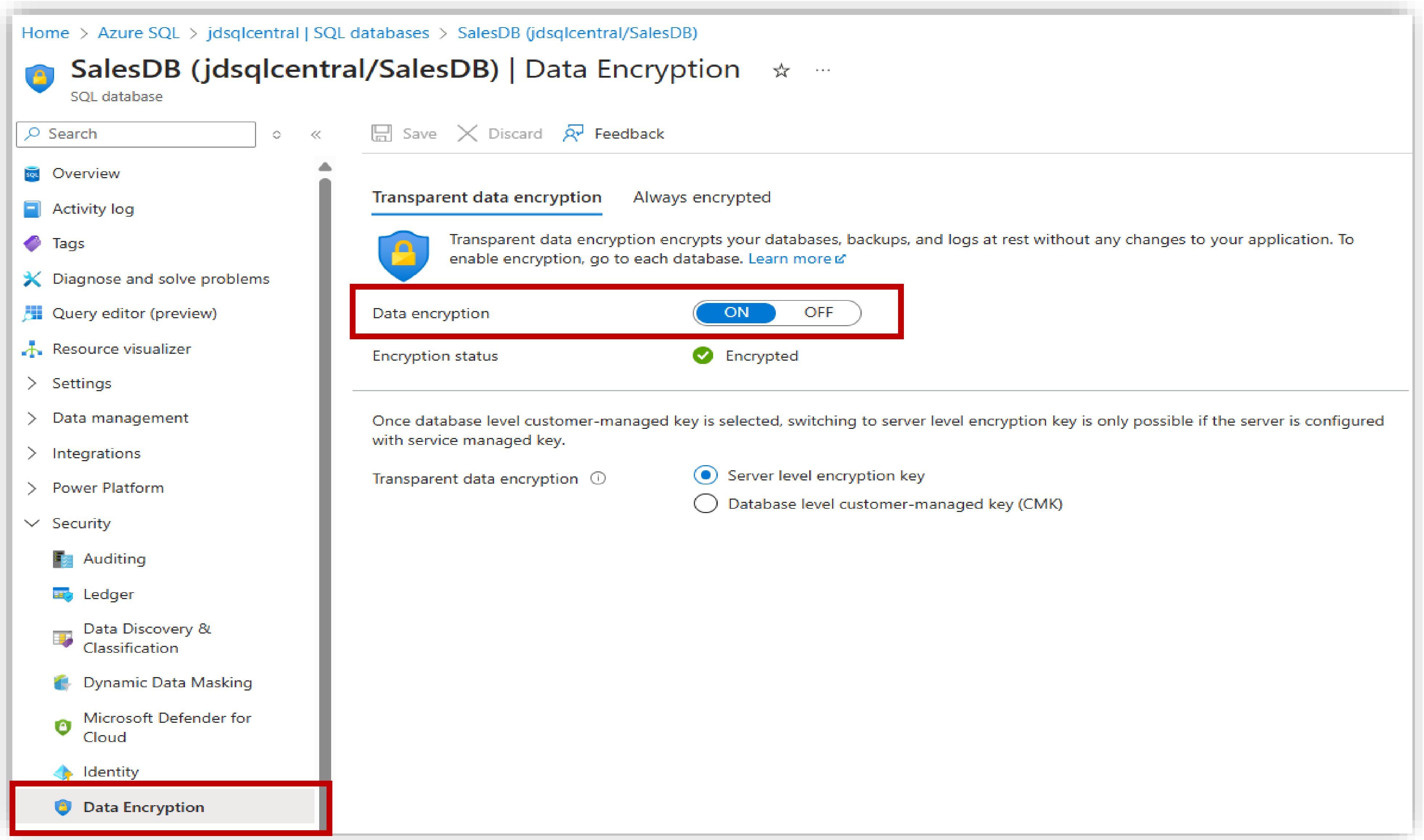
Service-managed transparent data encryption

- The database encryption key is protected by a built-in server certificate.
- Unique for each server.
- Primary and geo-secondary database are protected by the primary database's parent server key.
- Microsoft automatically rotates these certificates at least every 90 days.

Bring Your Own Key

- Take control over your transparent data encryption keys and control who can access them and when.
- Azure Key Vault.
- You set the asymmetric key at the server level, and all databases under that server inherit it.
- You can control key management tasks such as key rotations and key vault permissions.

Enable TDE Using Azure Portal



Home > Azure SQL > jdsqlcentral | SQL databases > SalesDB (jdsqlcentral/SalesDB)

SalesDB (jdsqlcentral/SalesDB) | Data Encryption ☆ ···

SQL database

Transparent data encryption encrypts your databases, backups, and logs at rest without any changes to your application. To enable encryption, go to each database. Learn more

Data encryption [✓ ON] OFF

Encryption status ✓ Encrypted

Once database level customer-managed key is selected, switching to server level encryption key is only possible if the server is configured with service managed key.

Transparent data encryption ⓘ

● Server level encryption key

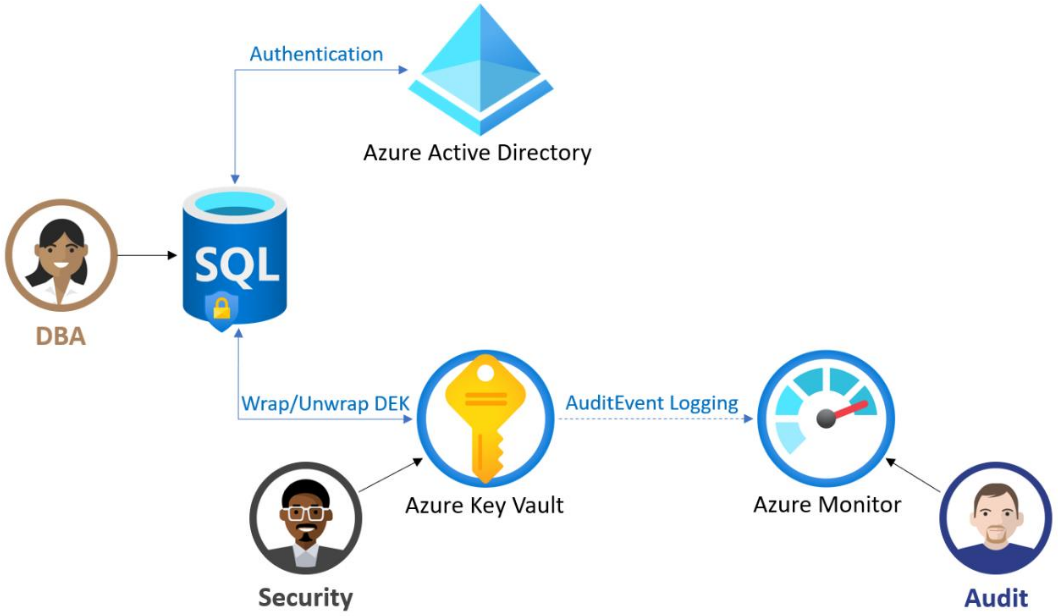
○ Database level customer-managed key (CMK)

TDE with customer-managed key (BYOK)

You are responsible for and in a full control of a key lifecycle management (key creation, upload, rotation, deletion), key usage permissions, and auditing of operations on keys.

The key used for encryption of the Database Encryption Key (DEK), called TDE protector, is a customer-managed asymmetric key stored in a customer-owned and customer-managed Azure Key Vault (AKV), a cloud-based external key management system.

TDE protector is set at the logical server level and is inherited by all encrypted databases associated with that server.



Demonstration

Implement TDE using Azure Portal and T-SQL Code

- Enable TDE With Bring Your Own Key using Azure Portal.

Questions?

Knowledge Check

Does TDE encrypt the data in motion?

What kind of application changes are required to use TDE?

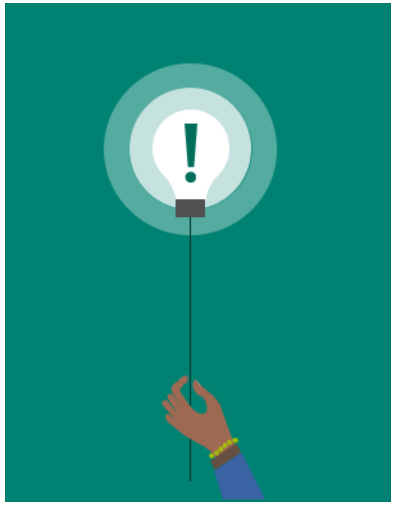
Which 2 types of Encryption Keys can be used for TDE?

Lesson 3: Implement Always Encrypted

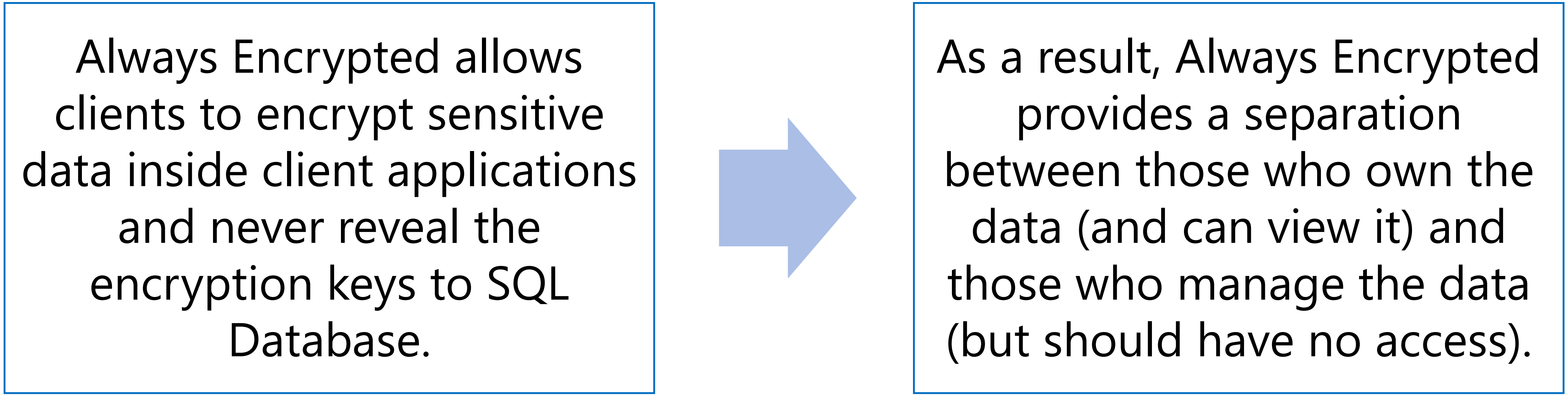
Objectives

After completing this learning, you will be able to:

- Know how to secure data at rest and in motion using Always encrypted.



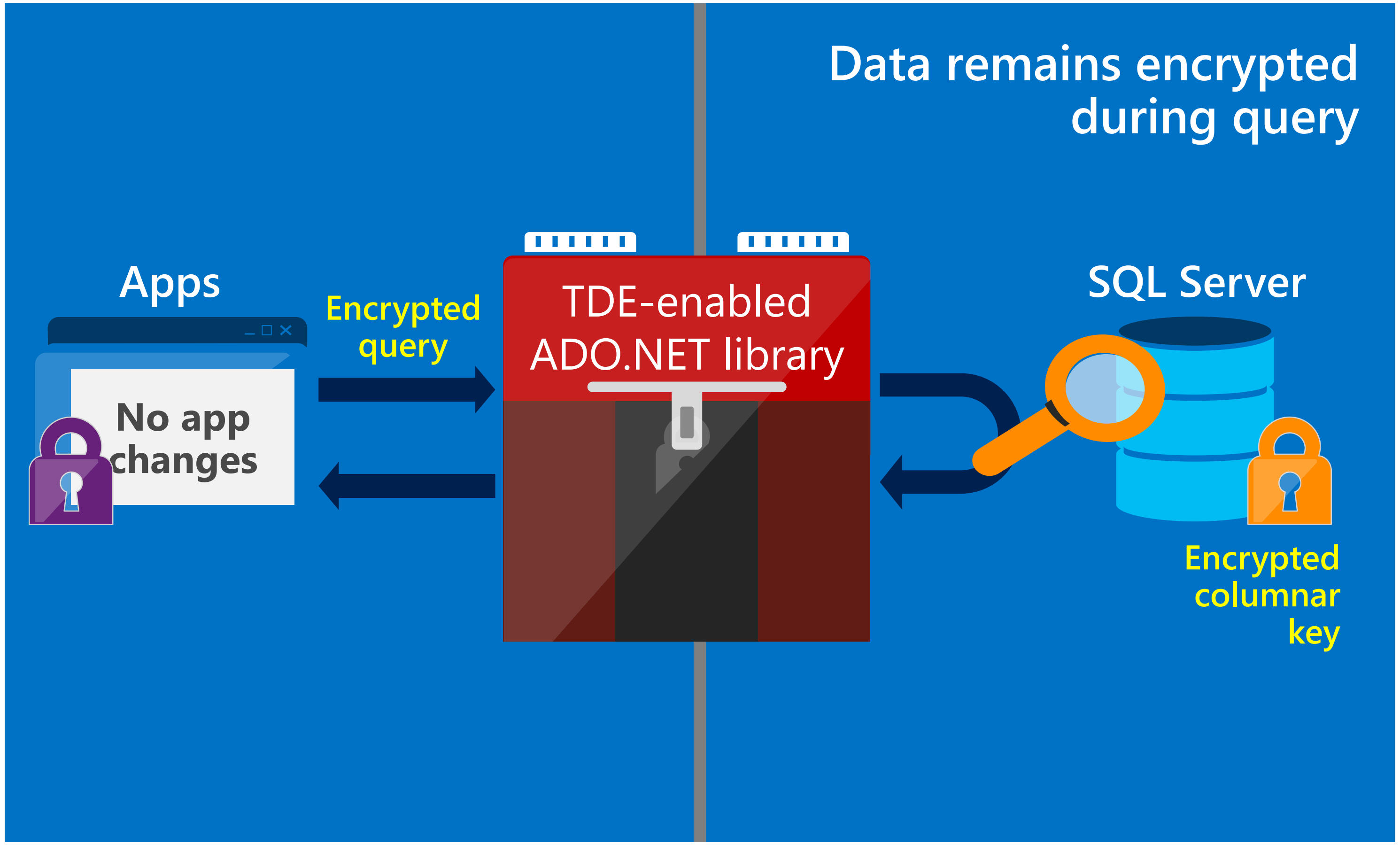
Always Encrypted



Always Encrypted allows clients to encrypt sensitive data inside client applications and never reveal the encryption keys to SQL Database.

As a result, Always Encrypted provides a separation between those who own the data (and can view it) and those who manage the data (but should have no access).

Understanding Always Encrypted Functionality



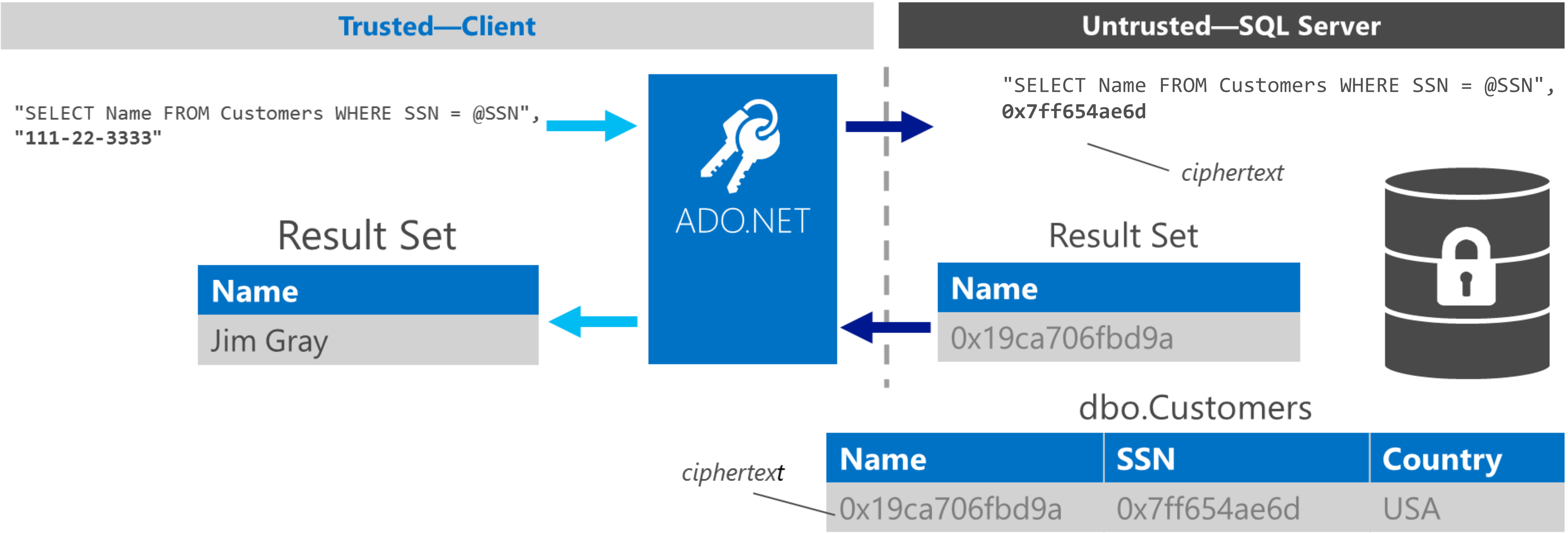
Capability

- Transparent client-side encryption, while SQL Server executes T-SQL queries on encrypted data.

Benefits

- Sensitive data remains encrypted and query-able at all times.
- Unauthorized users never have access to data or keys.
- No changes to applications are necessary.

Understanding Always Encrypted Functionality (Contd.)



```
"SELECT Name FROM Customers WHERE SSN = @SSN",
"111-22-3333"
```

```
"SELECT Name FROM Customers WHERE SSN = @SSN",
0x7ff654ae6d
```

Result Set

Name
Jim Gray

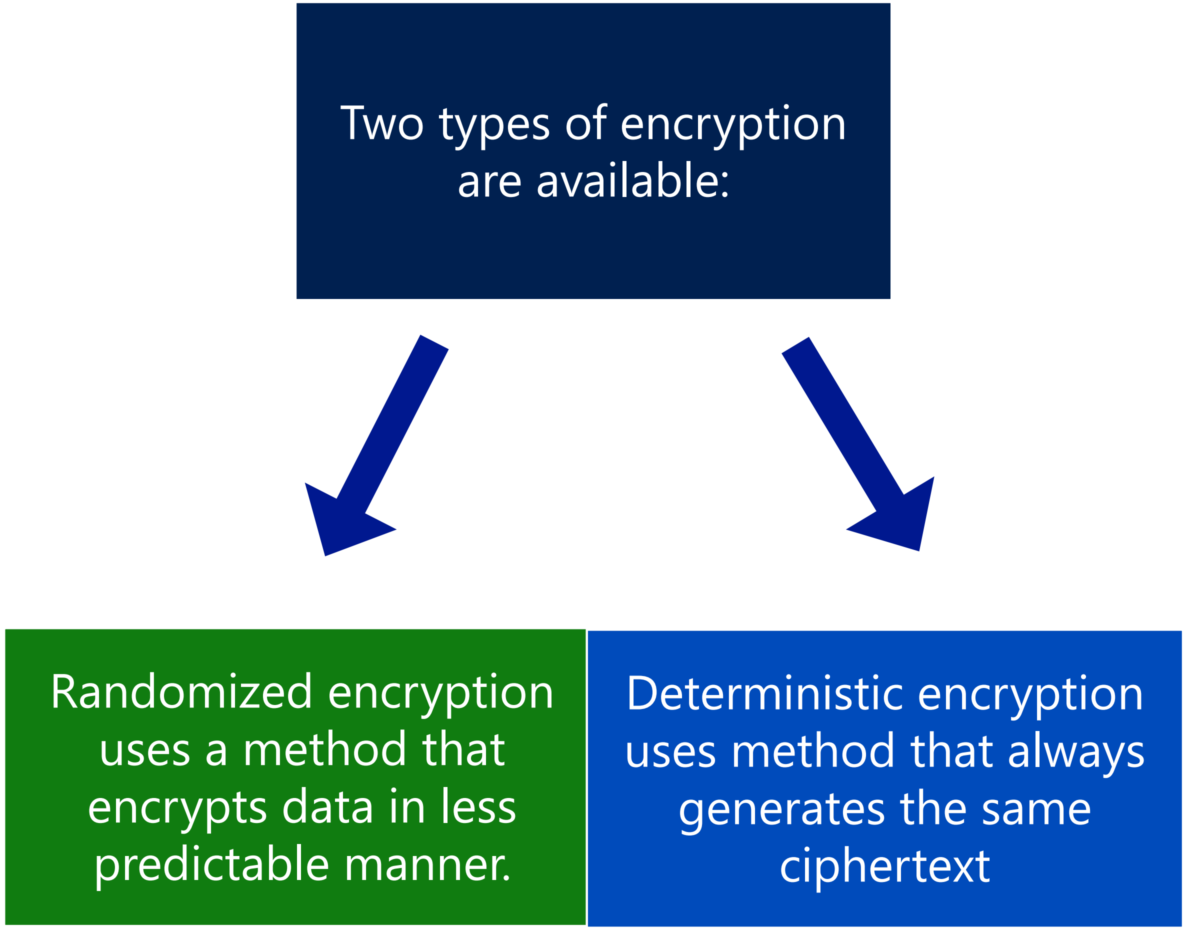
Result Set

Name
0x19ca706fbd9a

dbo.Customers

Name	SSN	Country
0x19ca706fbd9a	0x7ff654ae6d	USA

Encryption Methodologies



Two types of encryption are available:

Randomized encryption uses a method that encrypts data in less predictable manner.

Deterministic encryption uses method that always generates the same ciphertext

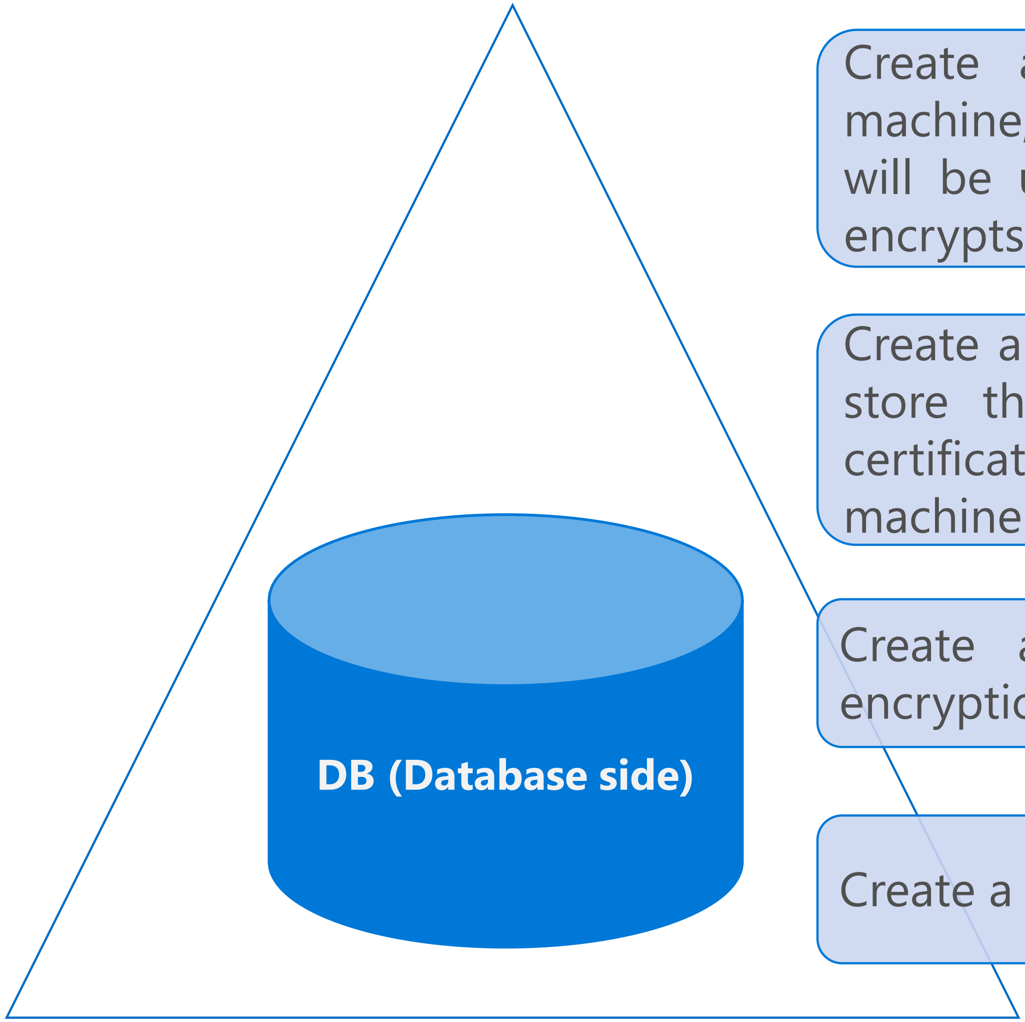
Randomized encryption

- Encrypt('123-45-6789') = 0x17cfd50a
- Repeat: Encrypt('123-45-6789') = 0x9b1fcf32
- Allows for transparent retrieval of encrypted data **but no operations.**
- More secure

Deterministic encryption

- Encrypt('123-45-6789') = 0x85a55d3f
- Repeat: Encrypt('123-45-6789') = 0x85a55d3f
- Allows for transparent retrieval of encrypted data **and quality.**
- **Comparison** (for example, in WHERE clauses and joins, distinct, group by).

Enabling Always Encrypted on Azure SQL DB



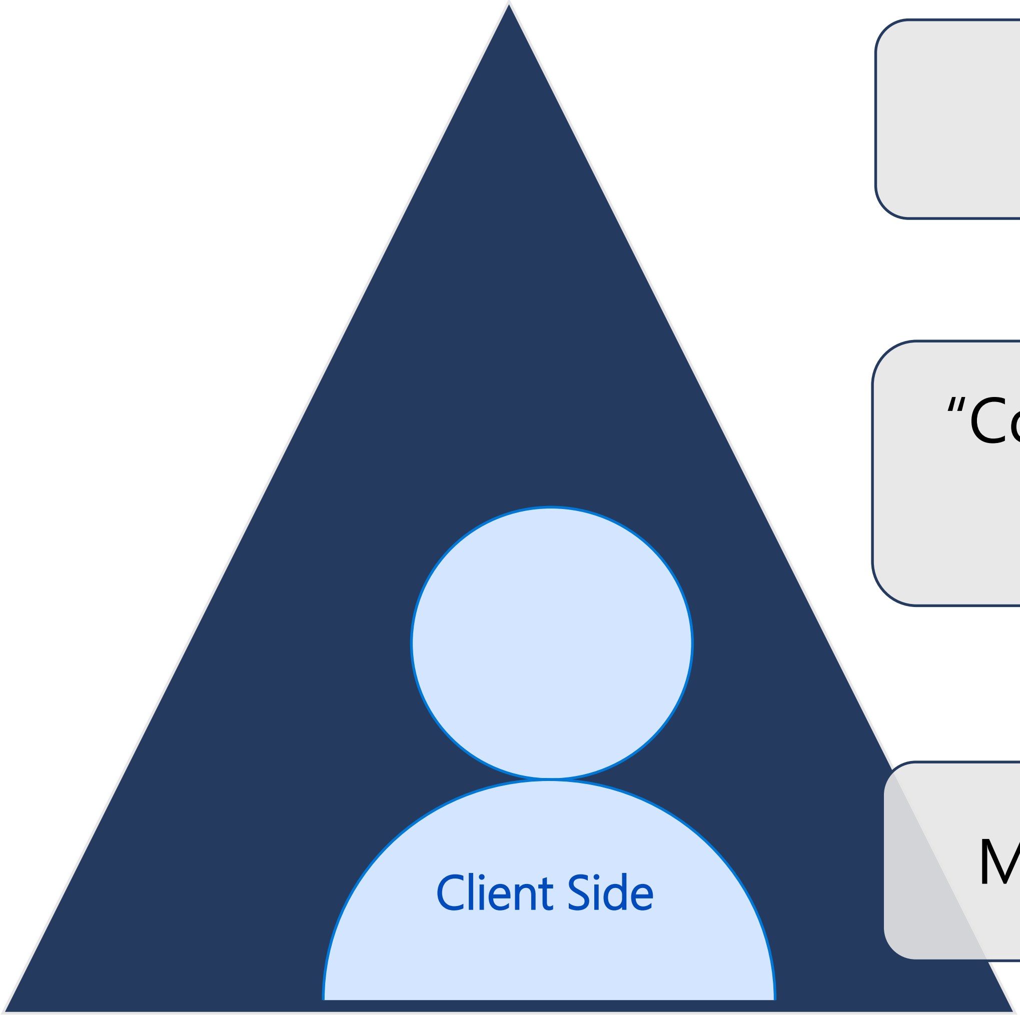
Create a local, self-signed certificate on the development machine, which will act as a Column Master Key (CMK). The CMK will be used to protect Column Encryption Keys (CEK), which encrypts the sensitive data.

Create a CMK store definition object in the database, which will store the information about the location of the CMK. The certificate will never be copied to the database or SQL Server machine.

Create a Column Encryption Key (CEK). You use column encryption keys to encrypt data in database columns.

Create a table with encrypted columns.

Enabling Always Encrypted on Azure SQL DB (contd.)



Same local stored certificate.

“Column Encryption Setting=Enabled;” in connection string.

Make sure you use the correct driver.

Demonstration

Enable Always Encrypted

- Enable Always Encrypted
- Select data through Application.

Implement Always Encrypted

- **Exercise 1:** Implement Always Encrypted on Azure SQL Database.
- **Exercise 2:** Use the .Net Client App to Test Always Encrypted.

Questions?

Knowledge Check

Can a DBA see Always Encrypted data?

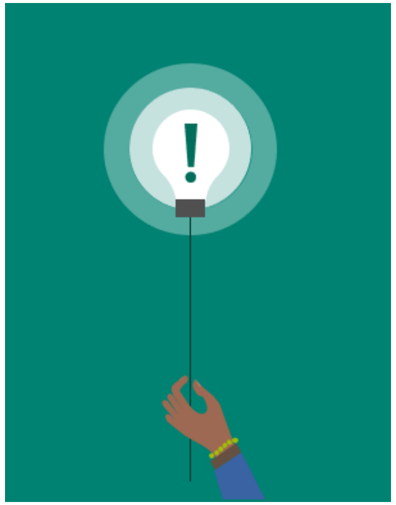
What are the 4 steps that you need to perform to enable Always Encrypted?

Lesson 4: Implement Row Level Security

Objectives

After completing this learning, you will be able to:

- Know how to control access to the data using Row Level Security (RLS).



Row Level Security (RLS)

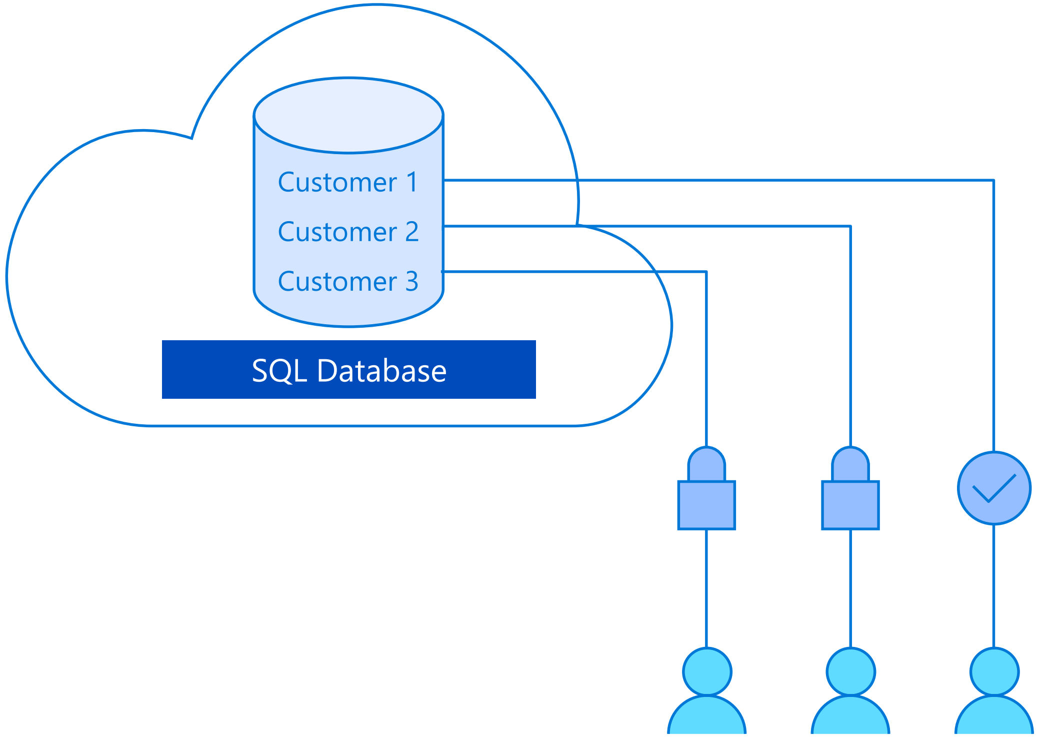
Row-Level Security enables customers to control access to rows in a database table based on the characteristics of the user executing a query.

Understanding RLS Functionality

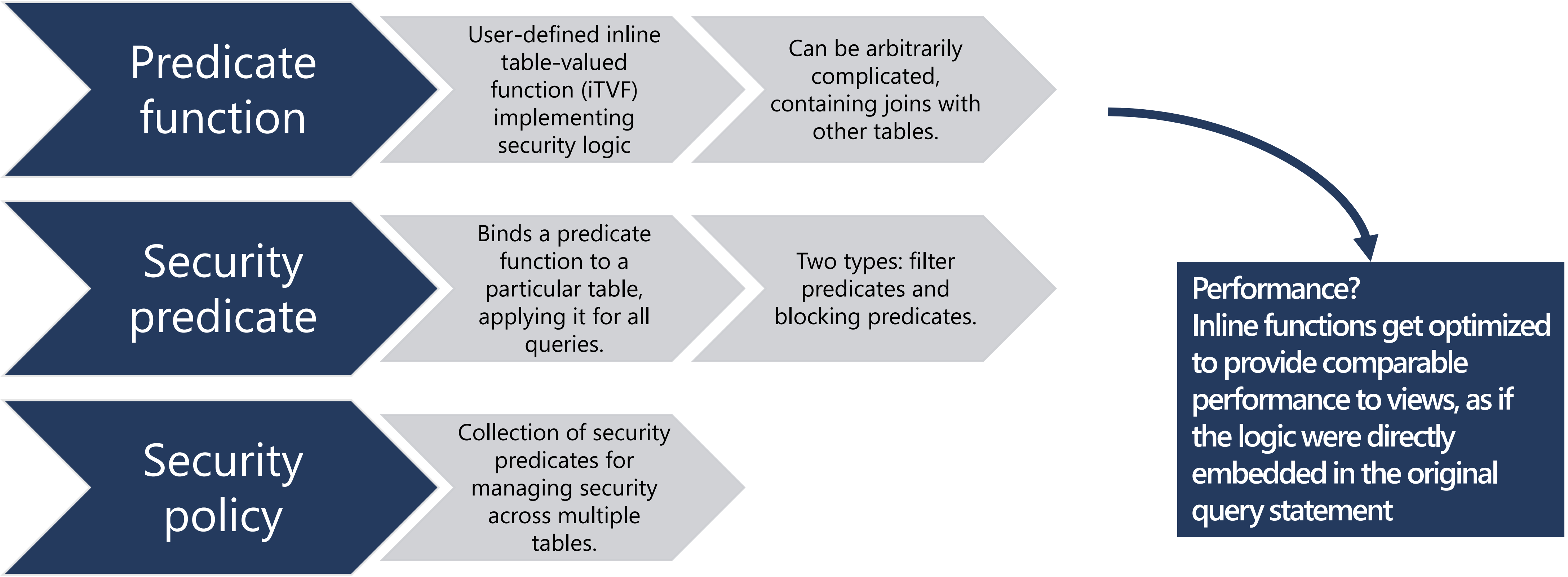
Fine-grained access control over specific rows in a database table.

Helps to prevent unauthorized access when multiple users share the same tables, or to implement connection filtering in multitenant applications.

Enforcement logic inside the database and schema is bound to the table.



RLS Implementation details



Predicate function

User-defined inline table-valued function (iTVF) implementing security logic

Can be arbitrarily complicated, containing joins with other tables.

Security predicate

Binds a predicate function to a particular table, applying it for all queries.

Two types: filter predicates and blocking predicates.

Security policy

Collection of security predicates for managing security across multiple tables.

Performance?
Inline functions get optimized to provide comparable performance to views, as if the logic were directly embedded in the original query statement

```
CREATE SECURITY POLICY mySecurityPolicy
ADD FILTER PREDICATE dbo.fn_securitypredicate(wing, startTime, endTime)
ON   dbo.patients
```

Demonstration

Implement RLS using T-SQL Code

- Enable RLS using T-SQL.

Implement Row Level Security

- **Exercise 1**: Implement Row Level Security on Azure SQL Database.

Questions?

Knowledge Check

What is the purpose of the predicate function?

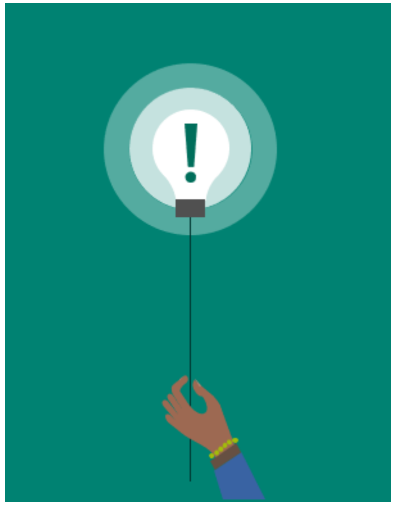
Why do we need row level security (RLS)?

Lesson 5: Implement Dynamic Data Masking

Objectives

After completing this learning, you will be able to:

- Know how to mask the critical data using Dynamic Data Masking.



Dynamic Data Masking

Dynamic Data Masking is a policy-based security feature that helps to limit the exposure of data in a database by returning masked data to non-privileged users who run queries over designated database fields.

Understanding Dynamic Data Masking Functionality

- Prevent abuse of sensitive data by hiding it from users.
- Easy configuration in new Azure Portal.
- Policy-driven at table and column level, for a defined set of users.
- Data masking applied in real-time to query results based on policy.
- Multiple masking functions available, such as full or partial, for various sensitive data categories (credit card numbers, SSN, etc.).

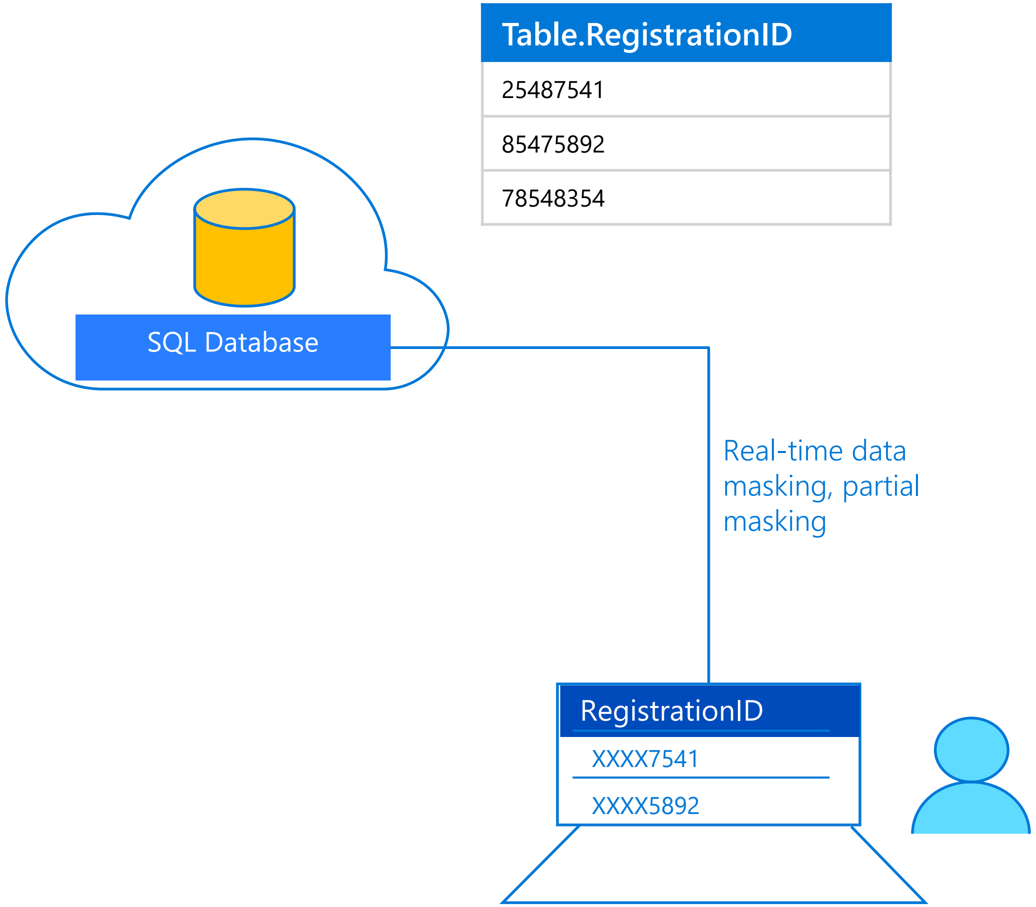


Table.RegistrationID
25487541
85475892
78548354

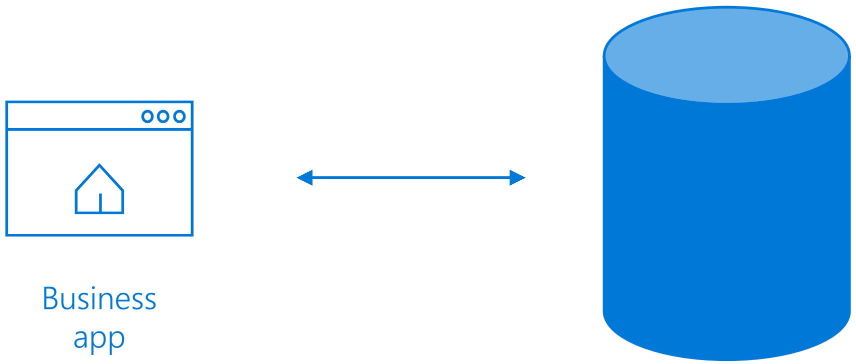
RegistrationID
XXXX7541
XXXX5892

Enable Dynamic Data Masking on Azure SQL DB

Security officer defines dynamic data masking policy in T-SQL over sensitive data in the Employee table.

The app user selects from the Employee table.

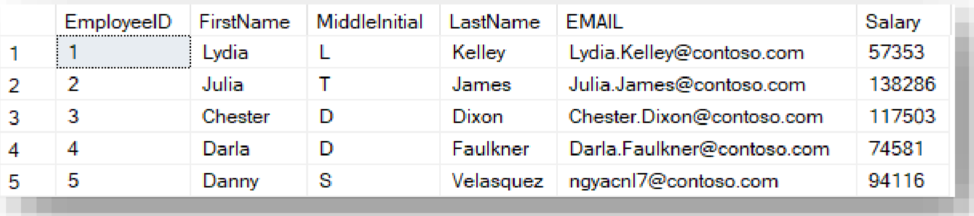
The dynamic data masking policy obfuscates the sensitive data in the query results.



```
ALTER TABLE dbo.Employee
ALTER COLUMN [FirstName] ADD MASKED WITH (FUNCTION = 'partial(1,
"xxx", 2)')

ALTER TABLE dbo.Employee
ALTER COLUMN [EMAIL] ADD MASKED WITH (FUNCTION = 'email()')

ALTER TABLE dbo.Employee
ALTER COLUMN [Salary] ADD MASKED WITH (FUNCTION =
'random(2000,20000)’)

GRANT UNMASK to admin1
```

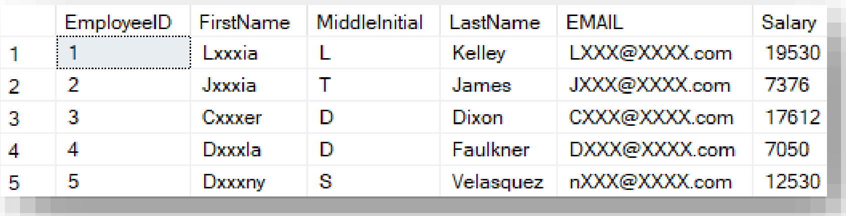
```
SELECT EmployeeID,
FirstName, MiddleInitial,
LastName, EMAIL, Salary
FROM [Employee]
```

Other Login



	EmployeeID	FirstName	MiddleInitial	LastName	EMAIL	Salary
1	1	Lydia	L	Kelley	Lydia.Kelley@contoso.com	57353
2	2	Julia	T	James	Julia.James@contoso.com	138286
3	3	Chester	D	Dixon	Chester.Dixon@contoso.com	117503
4	4	Darla	D	Faulkner	Darla.Faulkner@contoso.com	74581
5	5	Danny	S	Velasquez	ngyacnl7@contoso.com	94116

Admin Login



	EmployeeID	FirstName	MiddleInitial	LastName	EMAIL	Salary
1	1	Lxxxia	L	Kelley	LXXX@XXXX.com	19530
2	2	Jxxxia	T	James	JXXX@XXXX.com	7376
3	3	Cxxxer	D	Dixon	CXXX@XXXX.com	17612
4	4	Dxxxla	D	Faulkner	DXXX@XXXX.com	7050
5	5	Dxxxny	S	Velasquez	nXXX@XXXX.com	12530

Dynamic Data Masking Functions

DEFAULT

RANDOM

CUSTOM

EMAIL

DATETIME (2022)

CREDIT CARD (AZURE)

Demonstration

Implement Dynamic Data masking T-SQL Code

- Enable Dynamic Data masking using T-SQL.

Questions?

Knowledge Check

What’s the purpose of Dynamic Data Masking?

List two different masking rules in Dynamic Data Masking?

Module Summary

- Implement Data Encryption in Transit
- Implement Transparent Data Encryption
- Implement Always Encrypted
- Implement Row Level Security
- Implement Dynamic Data Masking

Microsoft