

# How Glassdoor is using SQL Server Always Encrypted to protect sensitive user data

Alex Bochkov

Database Architect

[glassdoor.com](https://www.glassdoor.com)

# Alex Bochkov

**Database Architect**  
glassdoor.com



I've been working with databases for  
literally half my life.  
I joined Glassdoor almost six years ago.

<https://github.com/alex-bochkov>  
[alex.bochkov@glassdoor.com](mailto:alex.bochkov@glassdoor.com)

# Content

- Glassdoor environment and why did we choose Always Encrypted?
- How does Always Encrypted work?
- 15 challenges we've faced.
- Our solution for a key management.
- Production rollout scenarios.
- Key rotation process.
- Change management considerations.

# What is Glassdoor?

- “to help people everywhere find a job and company they love”
- Leading platform for Employer Branding
- Hundreds of millions unique visitors every year
- Decent amount of sensitive personal data: emails, names, resumes, demographics, etc.
- User anonymity is crucial for our business model

# Glassdoor Database Environment

- Major databases are using SQL Server 2019 Enterprise Edition
- On average ~50k requests/second to databases with personal data,
  - spikes up to 100k requests/second in rush hours
- 15 years of code development with a lot of legacy code
- 24/7, no downtime allowed
- Most client apps are JAVA-based, some are in Node.js, internal automation mostly in PowerShell
- Everything is in AWS

# Why Do We Need to Protect Data?

- User anonymity is important for user trust.  
We must protect user identity even from ourselves.
- GDPR and CCPA compliance
  - Doesn't require encryption as such
  - Requires reasonable approach
- Very significant data breach penalties

# What Was in Place Before Always Encrypted?

- TLS enforcement to encrypt traffic between the app and the database
- TDE to encrypt database files on Windows level
- Volume encryption on the storage level
- Database backups are encrypted with a separate certificate
- Dynamic Data Masking in some cases

We needed a way to additionally protect sensitive values and hide it from super users (DBAs, Domain Admins, etc.) and more importantly prevents accidental data exposure.

We looked at several different Java-based options:

- using existing encryption libraries and approaches
- something totally custom tailored for our environment
- Dynamic Data Masking - the security team wasn't happy about it

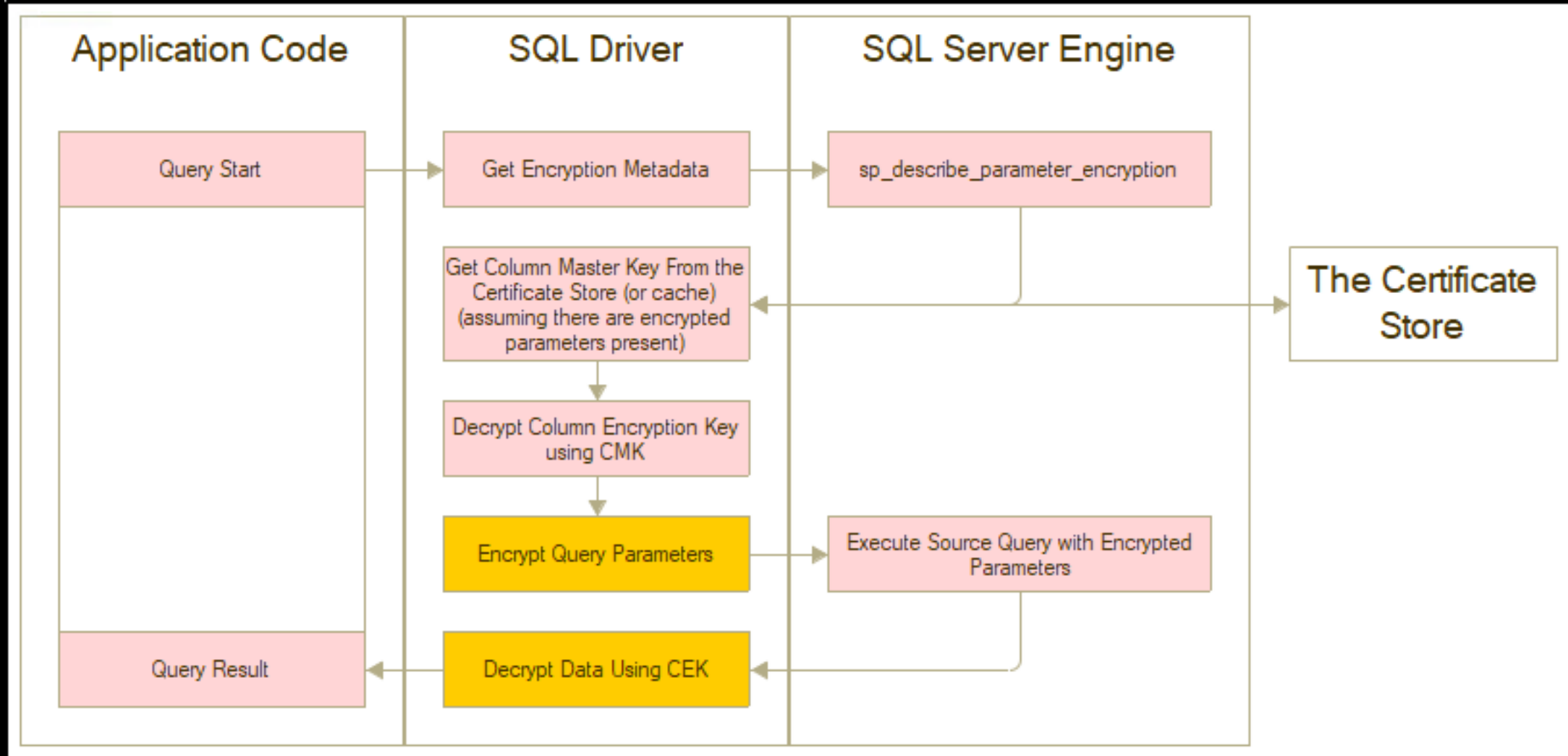
While we do have talent to create something totally custom, we were afraid that it might become a black box – cost of mistake will be significant, and we may end up losing data if there are bugs in the encryption process.

# Why Did We Choose Always Encrypted?

- We own SQL Server licenses, Always Encrypted comes with no additional costs
- Very easy to roll-out
  - driver does all the work for you behind the scene
  - very little changes on the application side
- Easy to maintain
  - Simple key generation process
  - Easy to rotate
- Flexible - each column can be encrypted separately
  - You can use separate keys for separate columns when there are different kinds of apps are working with the same db
  - You can pick between DETERMINISTIC and RANDOMIZED encryption for each column
- Harder to make a mistake in general:
  - SQL Server itself enforces correct encryption
  - Keys metadata (including the certificate thumbprint) is stored in the database
    - You can easily tell which certificate is used where



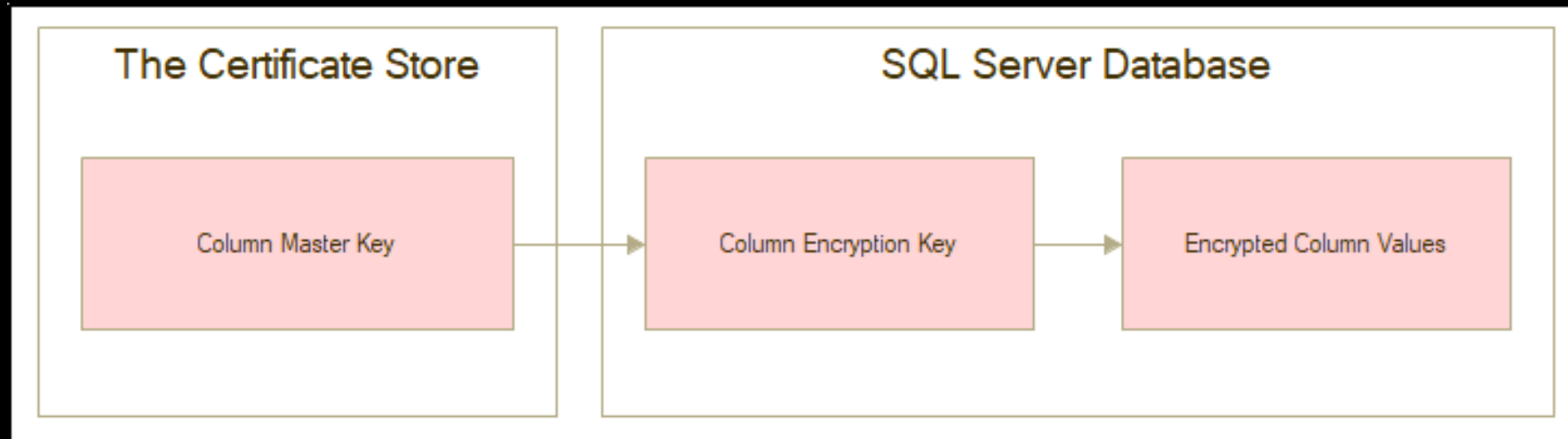
# How Does Always Encrypted Work?



# How Does Always Encrypted Work?

## Two keys approach

- **Column Master Key**
  - Stored outside of the database
    - Windows Certificate Store / Keystore File / Azure Key Vault
  - Used to encrypt the Column Encryption Key only, not involved into data encryption
- **Column Encryption Key**
  - Stored within the database itself
  - Used to encrypt actual values in the column
  - Key itself is encrypted with the Column Master key



# How Does Always Encrypted Work?

- Happens transparently to the application
- By default, enabled by adding “Column Encryption Setting=Enabled;” into the connection string
- Encryption and decryption operations are performed by the driver itself
  - Values are encrypted before parameters are sent to SQL Server
  - Values are decrypted before the result-set is returned by the driver to the app
- Supported by official drivers:

## Drivers that support Always Encrypted

Microsoft JDBC Driver for SQL Server

Microsoft SqlClient Data Provider for SQL Server (.NET)

Microsoft ODBC Driver for SQL Server

Python pyodbc community driver

## Drivers that do not support Always Encrypted

Python pymssql

jTDS (JAVA)

...

# How Does Always Encrypted Work?

- Extra call to detect the parameters encryption metadata - `sp_describe_parameter_encryption`
  - By default, when the column encryption is enabled, the driver requests the parameter encryption metadata before executing every single query
    - Analyzes the specified Transact-SQL statement and its parameters, to determine which parameters correspond to database columns that are protected by using the Always Encrypted feature. Returns encryption metadata for the parameters that correspond to encrypted columns.
    - The **sproc result contains encrypted column names and the Column Encryption Key itself (or two, when there is a rotation in process)**
- This happens for every query regardless of whether encrypted columns are included

```
1 exec sp_describe_parameter_encryption
2   @tsql = N'SELECT TOP 100 * FROM [dbo].[email] WHERE [emailAddress] = @emailAddress',
3   @params = N'@emailAddress nvarchar(256)';
```

version	column_encryption_key_encrypted_value	column_master_key_store_provider_name	column_master_key_path	column_master_key_name
1	0x016E000001630075007200720065006E00740075007300...	MSSQL_CERTIFICATE_STORE	CurrentUser/My/F39B6AA31372184BA6200720FC84A71ACDC030B2	RSA_...

parameter_ordinal	parameter_name	column_encryption_algorithm	column_encryption_type	column_encryption_key_ordinal	column_encryption_normalization_rule_version
1	@emailAddress	2	1	1	1

**COLUMN ENCRYPTION KEY value itself encrypted with CMK**

**DETERMINISTIC encryption for @emailAddress parameter**

**COLUMN MASTER KEY location**

## **[1/15] Challenges - Accept the Change**

**You need to setup your org to accept the change - things and approaches that worked before won't be possible.**

- **“No, you can’t query this column manually anymore...”**
- **“No, you can’t search or analyze data in this column anymore...”**
- **...**

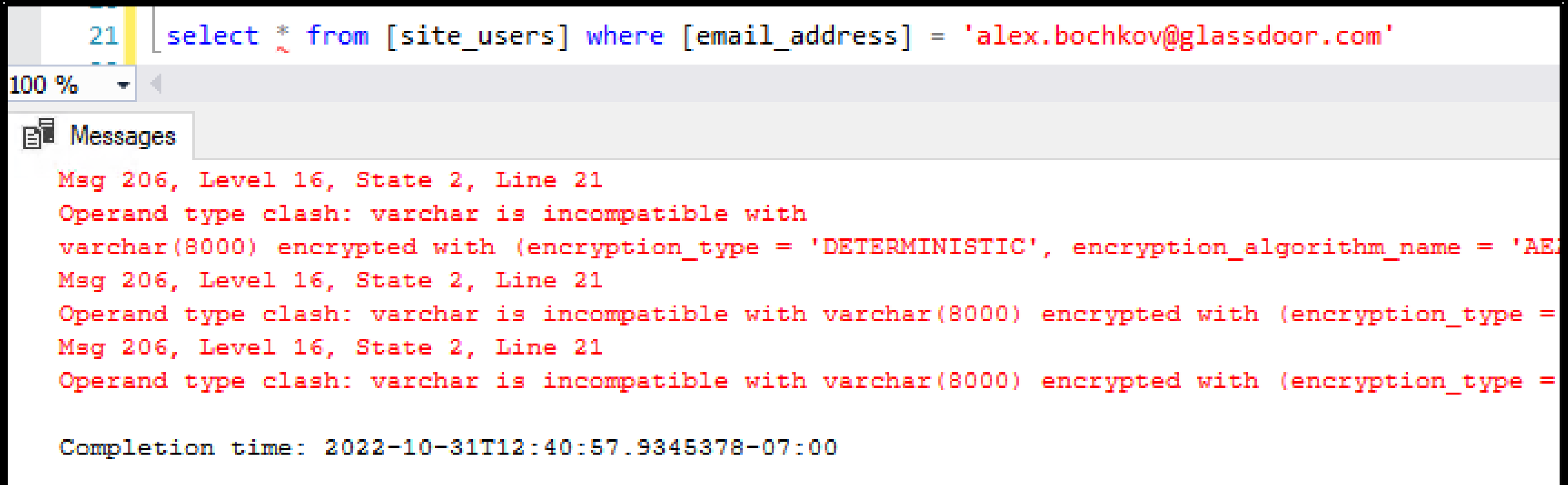
## **[2/15] Challenges - Column Master Key Management is Tricky**

- **You can store Column Master Keys in**
  - **.NET is using Windows Certificate store**
  - **JAVA is using the key-store**
  - **(easy) multi-platform use only with Azure Key Vault**
  - **(harder) Implement custom key store provider - what we ended up doing**
- **First two approaches are not quite flexible**
  - **How to distribute keys to each server safely?**
  - **How to make it work in a hybrid Windows / Linux / Mac environment?**

## [3/15] Challenges - Query Parameterization Strictly Required

All queries must be parameterized, plain-text values embedded into the query text are not supported.

- SSMS has a built-in feature to support auto-parameterization
- SQL drivers with Always Encrypted support automatically encrypt query parameters



The screenshot shows the SQL Server Enterprise Manager interface. At the top, a query is entered in the query editor: `select * from [site_users] where [email_address] = 'alex.bochkov@glassdoor.com'`. Below the query editor, the 'Messages' pane displays three error messages in red text, all occurring at 'Msg 206, Level 16, State 2, Line 21'. The messages are: 'Operand type clash: varchar is incompatible with varchar(8000) encrypted with (encryption\_type = 'DETERMINISTIC', encryption\_algorithm\_name = 'AE...', 'Operand type clash: varchar is incompatible with varchar(8000) encrypted with (encryption\_type =', and 'Operand type clash: varchar is incompatible with varchar(8000) encrypted with (encryption\_type ='. At the bottom of the messages pane, the completion time is shown as '2022-10-31T12:40:57.9345378-07:00'.

```
21 select * from [site_users] where [email_address] = 'alex.bochkov@glassdoor.com'
```

100 %

Messages

Msg 206, Level 16, State 2, Line 21  
Operand type clash: varchar is incompatible with  
varchar(8000) encrypted with (encryption\_type = 'DETERMINISTIC', encryption\_algorithm\_name = 'AE'

Msg 206, Level 16, State 2, Line 21  
Operand type clash: varchar is incompatible with varchar(8000) encrypted with (encryption\_type =

Msg 206, Level 16, State 2, Line 21  
Operand type clash: varchar is incompatible with varchar(8000) encrypted with (encryption\_type =

Completion time: 2022-10-31T12:40:57.9345378-07:00

# [3/15] Challenges - Query Parameterization Strictly Required

Examples:

```
protected MapSqlParameterSource getQueryParamsMap(UpmUser user) {  
    MapSqlParameterSource paramSource = new MapSqlParameterSource();  
    paramSource.addValue("FK_userId", user.getUserId());  
    paramSource.addValue("emailAddress", user.getEmailAddress(), Types.NVARCHAR);  
}
```

Java (Hibernate)

```
20 DECLARE @email NVARCHAR (200)= N'alex.bochkov@GLASSDOOR.COM'  
21 select * from [site_users] where [email_address] = @email
```

100 %

SSMS only

Results Messages

	user_id	email_address
1	7	alex.bochkov@GLASSDOOR.COM

```
1 $sqlCommand = New-Object System.Data.SqlClient.SqlCommand  
2 ...  
3 $emailEncrypted = $sqlCommand.Parameters.Add("email", [Data.SqlDbType]::NVarChar, 200);  
4 $emailEncrypted.Value = "alex.bochkov@glassdoor.com"  
5
```

PowerShell



## [4/15] Challenges - Performance Implications

- Significant performance impact (at least in scenarios with no encryption metadata cache)
  - Round trip for each query by default
  - Increased network traffic and CPU utilization

EventClass	SPID	StartTime	TextData
Trace Start		2022-10-31 12:55:38.560	
RPC:Starting	8040	2022-10-31 12:55:39.030	exec sp_executesql N'exec sp_describe_parameter_encryption @P0,@P1...
RPC:Starting	8040	2022-10-31 12:55:39.030	exec sp_executesql N'select userprofil0_.PK_id as pk_id1_30_, user...
RPC:Starting	8040	2022-10-31 12:55:39.267	exec sp_executesql N'exec sp_describe_parameter_encryption @P0,@P1...
RPC:Starting	8040	2022-10-31 12:55:39.267	exec sp_executesql N'select userprofil0_.PK_id as pk_id1_30_, user...
RPC:Starting	8040	2022-10-31 12:55:40.917	exec sp_executesql N'exec sp_describe_parameter_encryption @P0,@P1...
RPC:Starting	8040	2022-10-31 12:55:40.920	exec sp_executesql N'select ideallocat0_.PK_id as pk_id1_14_, idea...
RPC:Starting	8040	2022-10-31 12:55:40.920	exec sp_executesql N'exec sp_describe_parameter_encryption @P0,@P1...
RPC:Starting	8040	2022-10-31 12:55:40.920	exec sp_executesql N'select generalloc0_.PK_id as pk_id1_8_, gener...
RPC:Starting	8040	2022-10-31 12:55:41.180	exec sp_executesql N'exec sp_describe_parameter_encryption @P0,@P1...
RPC:Starting	8040	2022-10-31 12:55:41.180	exec sp_executesql N'select userprofil0_.PK_id as pk_id1_30_, user...
RPC:Starting	8040	2022-10-31 12:55:42.677	exec sp_executesql N'exec sp_describe_parameter_encryption @P0,@P1...
RPC:Starting	8040	2022-10-31 12:55:42.677	exec sp_executesql N'select resume0_.PK_id as pk_id1_20_, resume0...
RPC:Starting	8040	2022-10-31 12:55:42.680	exec sp_executesql N'exec sp_describe_parameter_encryption @P0,@P1...
RPC:Starting	8040	2022-10-31 12:55:42.680	exec sp_executesql N'select resumework0_.FK_resumeId as fk_resu22...
RPC:Starting	8040	2022-10-31 12:55:42.680	exec sp_executesql N'exec sp_describe_parameter_encryption @P0,@P1...
RPC:Starting	8040	2022-10-31 12:55:42.680	exec sp_executesql N'select resumeeduc0_.FK_resumeId as fk_resu22...
RPC:Starting	8040	2022-10-31 12:55:42.680	exec sp_executesql N'exec sp_describe_parameter_encryption @P0,@P1...

## [4/15] Challenges - Performance Implications

You can fine-tune it on the query level but that was not foreseeable in our environment

Column Encryption Setting -> Disabled

- SqlCommandColumnEncryptionSetting.Enabled for individual queries that have any parameters that need to be encrypted
- SqlCommandColumnEncryptionSetting.ResultSet for queries that do not have any parameters requiring encryption, but retrieve data from encrypted columns.

[Doc: Setting Always Encrypted at the query level](https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/develop-using-always-encrypted-with-net-framework-data-provider#setting-always-encrypted-at-the-query-level)

<https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/develop-using-always-encrypted-with-net-framework-data-provider#setting-always-encrypted-at-the-query-level>

## [5/15] Challenges - Choosing an Encryption Type

Encryption limits data manipulation options:

1. DETERMINISTIC
  - =, <>, (unique) indexes, aggregation
2. RANDOMIZED- can't really do anything

```
[email_address] NVARCHAR (200) COLLATE Latin1_General_BIN2
    ENCRYPTED WITH (
        COLUMN_ENCRYPTION_KEY = [cek],
        ENCRYPTION_TYPE = DETERMINISTIC,
        ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'
    ) NOT NULL,
[first_name] NVARCHAR (200) COLLATE Latin1_General_BIN2
    ENCRYPTED WITH (
        COLUMN_ENCRYPTION_KEY = [cek],
        ENCRYPTION_TYPE = RANDOMIZED,
        ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'
    ) NOT NULL
```

## [6/15] Challenges - Encrypted Strings are Case-sensitive

Enforce uniform shape for all strings where you need uniqueness, for example, lower-case all emails on the app side before it is inserted into the database or used as a parameter in SELECT query.

Failure to do so will result in a corrupted data-set.

```
CREATE TABLE [dbo].[site_users] (  
    [user_id] INT IDENTITY NOT NULL,  
    [email_address] NVARCHAR (200) COLLATE Latin1_General_BIN2  
        ENCRYPTED WITH (  
            COLUMN_ENCRYPTION_KEY = [cek],  
            ENCRYPTION_TYPE = DETERMINISTIC,  
            ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'  
        ) NOT NULL,  
    INDEX idx_site_users UNIQUE ([email_address])  
);
```

	user_id	email_address
1	1	alex.bochkov@glassdoor.com
2	2	alex.bochkov@glassdoor.com
3	3	Alex.Bochkov@glassdoor.com
4	4	ALEX.BOCHKOV@glassdoor.com
5	5	AlEx.BoChKoV@glassdoor.com
6	6	aLeX.bOcHkOv@glassdoor.com
7	7	alex.bochkov@GLASSDOOR.COM

## [7/15] Challenges - Strict Data Type Check

Types that are normally interchangeable are not compatible when you encrypt them:

- NVARCHAR and VARCHAR
- FLOAT, FLOAT2 and NUMERIC
- DATETIME and DATETIME2
- etc

Apps must pass parameters using exact data types as they defined in the database.

```
10
11 DECLARE @email VARCHAR (200)= 'alex.bochkov@GLASSDOOR.COM'
12 INSERT INTO [dbo].[site_users] ([email_address]) VALUES (@email)
13
```

100 %

Messages

Msg 206, Level 16, State 2, Line 12  
Operand type clash: varchar(200) encrypted with (encryption\_type = 'DETERMINISTIC', encryption\_algorithm = 'RSA', encryption\_salt = '0x00000000000000000000000000000000', encryption\_key = '0x00000000000000000000000000000000', encryption\_key\_id = 0, encryption\_protection\_level = 1, encryption\_key\_usage = 0) is incompatible with nvarchar(200) encrypted with (encryption\_type = 'DETERMINISTIC', encryption\_algorithm = 'RSA', encryption\_salt = '0x00000000000000000000000000000000', encryption\_key = '0x00000000000000000000000000000000', encryption\_key\_id = 0, encryption\_protection\_level = 1, encryption\_key\_usage = 0)

# [8/15] Challenges - Harder to Diagnose Errors

Errors are hidden on SQL Server side, columns not shown, hard to debug.

- Generic error messages in some cases
- No source query

For example, syntax errors are hard to find because it fails on the parameter encryption metadata spoc where actual query text is passed as a parameter and not visible in logs. It is impossible to say what is wrong with the query without looking into the app error log.

## Messages

Msg 206, Level 16, State 2, Line 13  
Operand type clash: nvarchar(300) encrypted with (encryption\_type = 'DETERMINISTIC',  
is incompatible with nvarchar(200) encrypted with (encryption\_type = 'DETERMINISTIC',

**String truncation**

**Invalid SQL Query**

Sev	ErrorNumber	ErrorMessage	App	SQLText
16	207	Invalid column name 'employerSchoolLevel'.	userpr...	(@P0 nvarchar(2261),@P1 nvarchar(7))exec sp_describe_parameter_encryption @P0,@P1
16	8180	Statement(s) could not be prepared.	userpr...	(@P0 nvarchar(2261),@P1 nvarchar(7))exec sp_describe_parameter_encryption @P0,@P1
16	207	Invalid column name 'nationalProviderIdentifier'.	userpr...	(@P0 nvarchar(2261),@P1 nvarchar(7))exec sp_describe_parameter_encryption @P0,@P1
16	207	Invalid column name 'employerSchoolType'.	userpr...	(@P0 nvarchar(2261),@P1 nvarchar(7))exec sp_describe_parameter_encryption @P0,@P1
16	207	Invalid column name 'employerSchoolLevel'.	userpr...	(@P0 nvarchar(2261),@P1 nvarchar(7))exec sp_describe_parameter_encryption @P0,@P1
16	8180	Statement(s) could not be prepared.	userpr...	(@P0 nvarchar(2261),@P1 nvarchar(7))exec sp_describe_parameter_encryption @P0,@P1
16	207	Invalid column name 'nationalProviderIdentifier'.	userpr...	(@P0 nvarchar(2261),@P1 nvarchar(7))exec sp_describe_parameter_encryption @P0,@P1

## [9/15] Challenges - Encrypted Values Can't Leave The Database

- temp tables don't work
- cross-database data move doesn't work - doesn't matter if the same key is present there
  - Need to move the data? The easiest option is to re-encrypt it

```
14 SELECT * INTO #t FROM [site_users];
```

100 %

Messages

Msg 33296, Level 16, State 1, Line 14  
Cannot create table '#t' since it references a column encryption key from a different database.  
Msg 33296, Level 16, State 1, Line 14  
Cannot create table '#t' since it references a column encryption key from a different database.  
Msg 33296, Level 16, State 1, Line 14  
Cannot create table '#t' since it references a column encryption key from a different database.

Completion time: 2022-10-31T10:41:00.0874395-07:00

# [10/15] Challenges - BCP Doesn't Support Always Encrypted

BCP can't decrypt data, also I wasn't able to make it move encrypted values in binary form properly

How to change binary data directly?

- This is helpful for cases when data needs to be moved between databases, but an encryption key is not provided.
- Data is not decrypted/encrypted on the client-side, so no risks of exposure.

-- allow

```
ALTER USER [dataExport] WITH ALLOW_ENCRYPTED_VALUE_MODIFICATIONS = ON;
```

-- remove permission

```
ALTER USER [dataExport] WITH ALLOW_ENCRYPTED_VALUE_MODIFICATIONS = OFF;
```

[Doc: Copying Encrypted Data using SqlBulkCopy](https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/develop-using-always-encrypted-with-net-framework-data-provider#copying-encrypted-data-using-sqlbulkcopy)

<https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/develop-using-always-encrypted-with-net-framework-data-provider#copying-encrypted-data-using-sqlbulkcopy>



## [11/15] Challenges - No Transactional Replication

“The following features don't work on encrypted columns:

- Transactional, merge, or snapshot replication”

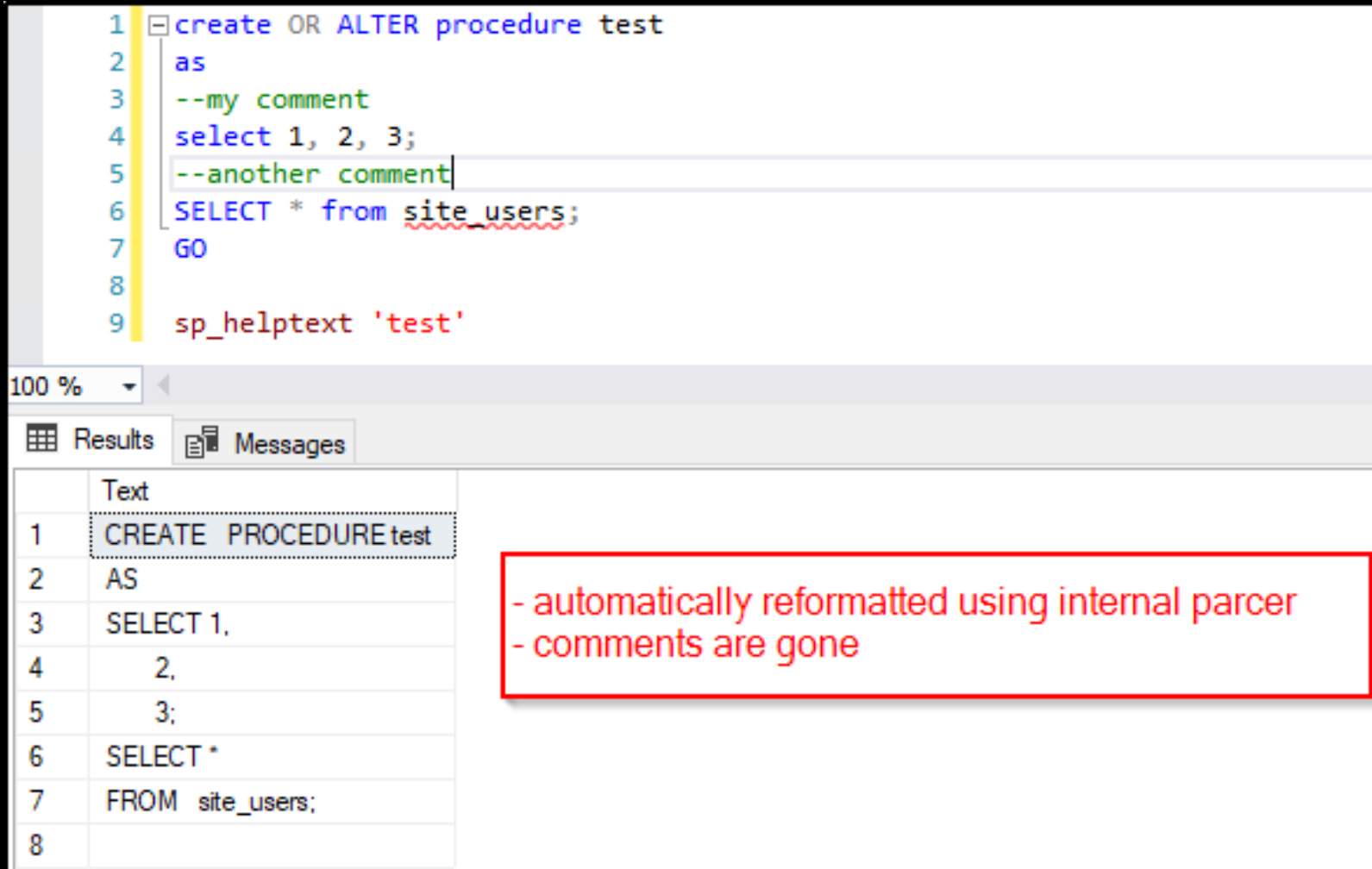
No CDC as well

[Doc: Feature details](https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine#feature-details)

<https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/always-encrypted-database-engine#feature-details>

# [12/15] Challenges - Code is Reformatted

Code is reformatted in SSMS when you push code changes while column encryption is enabled.



```
1 create OR ALTER procedure test
2 as
3 --my comment
4 select 1, 2, 3;
5 --another comment
6 SELECT * from site_users;
7 GO
8
9 sp_helptext 'test'
```

100 %

Results Messages

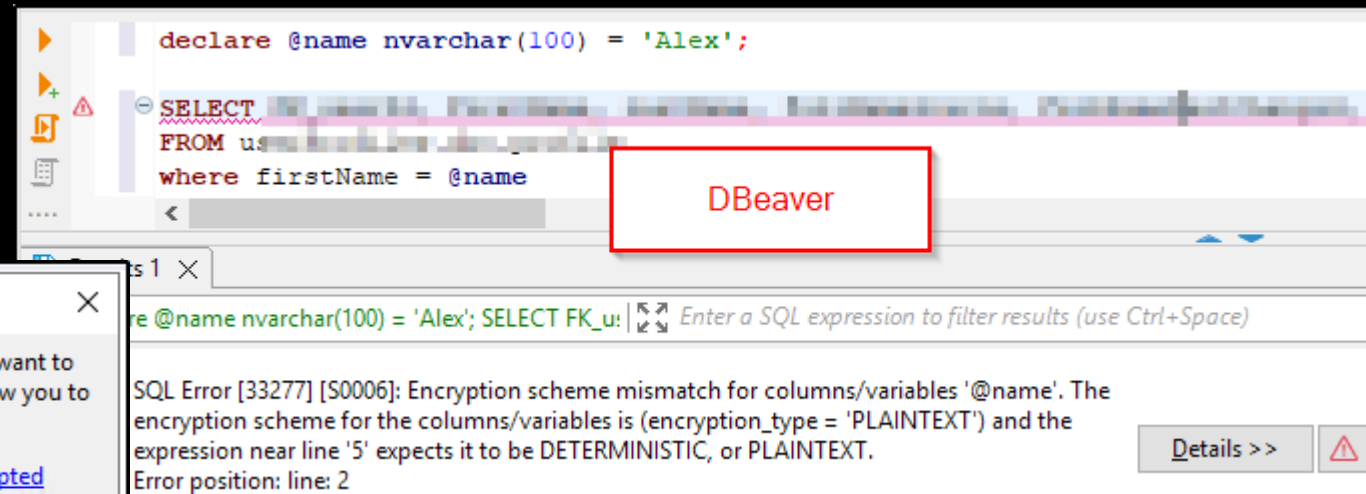
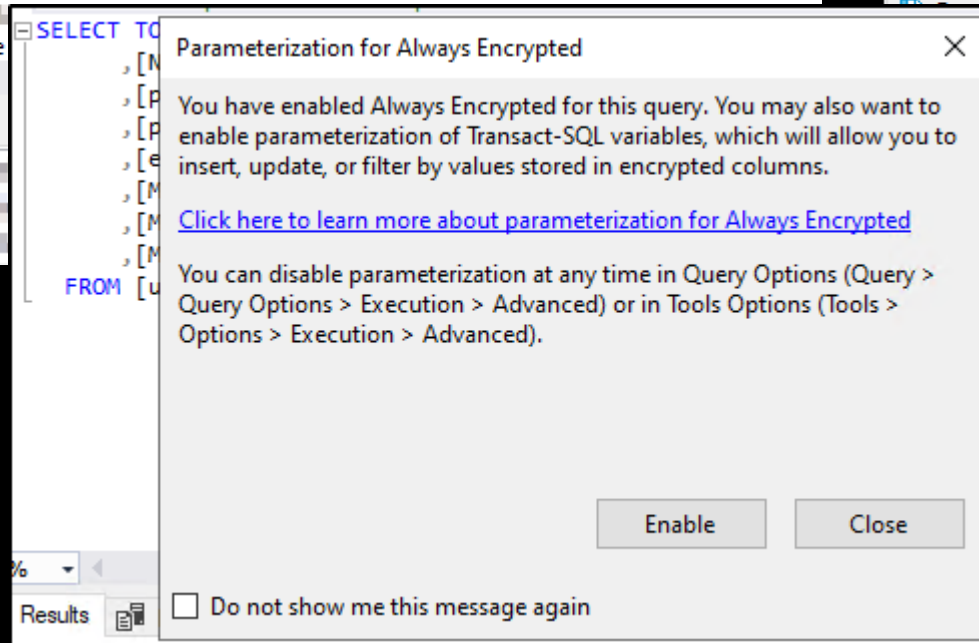
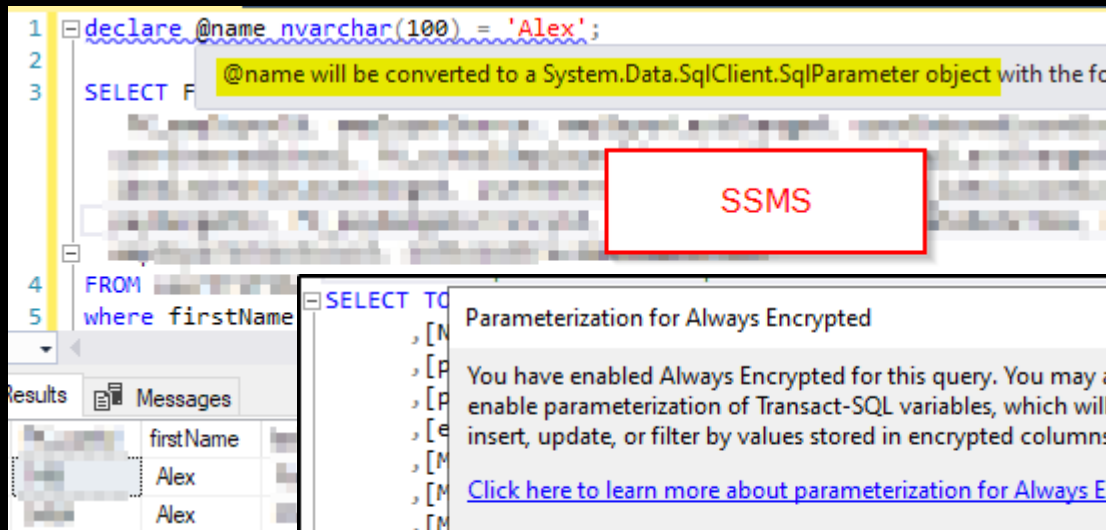
	Text
1	CREATE PROCEDURE test
2	AS
3	SELECT 1,
4	2,
5	3;
6	SELECT *
7	FROM site_users;
8	

- automatically reformatted using internal parser  
- comments are gone

# [13/15] Challenges - Limited SQL Client Options

There are limited SQL Client options that can work with encrypted columns properly.

- Most SQL Clients can decrypt data fine when it is using proper drivers and can reach the certificate store.
- Only SSMS can use encrypted query parameters because this has been built separately into SSMS app itself.



## [14/15] Challenges - Limited Data Modification Options

You can't really modify encrypted values on the database side (procs, triggers, adhoc queries), but at least you can NULL-out values and move them within the database.

No computed columns, no constraints, etc.

No FOR XML, FOR JSON.

```
8  UPDATE [dbo].[site_users]
9  -- I can wipe out the encrypted value
10 SET [email_address] = NULL
11 WHERE [user_id] = 1
12
13 UPDATE [dbo].[site_users]
14 -- I can move encrypted values around within the database
15 SET [email_address] = (SELECT a.[email_address]
16                        FROM [dbo].[site_users] a
17                        WHERE [user_id] = 10)
18 WHERE [user_id] = 1
```

# [15/15] Challenges - Data Size

Encrypted values take more space, compression is useless

```
CREATE TABLE [dbo].[site_users] (  
    [user_id] INT IDENTITY (1, 1) NOT NULL,  
    [email_address] NVARCHAR (200) COLLATE Latin1_General_BIN2  
        ENCRYPTED WITH (  
            COLUMN_ENCRYPTION_KEY = [cek],  
            ENCRYPTION_TYPE = DETERMINISTIC,  
            ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'  
        ) NOT NULL,  
    INDEX [cix_site_users] CLUSTERED ([user_id]),  
    INDEX [idx_site_users] UNIQUE ([email_address])  
);
```

1M records, email\_address is plain text

Table	IndexID	Index	Data Compr	avg_fragmentation_in_percent	page_count	index_size_mb
site_users	1	cix_site_users	NONE	0.00	7,501	58
site_users	2	idx_site_users	NONE	0.00	6,879	53

1M records, email\_address is encrypted

Table	IndexID	Index	Data Compr	avg_fragmentation_in_percent	page_count	index_size_mb
site_users	1	cix_site_users	NONE	0.00	14,778	115
site_users	2	idx_site_users	NONE	0.00	14,159	110

1M records, email\_address is encrypted, indexes compressed

Table	IndexID	Index	Data Compr	avg_fragmentation_in_percent	page_count	index_size_mb
site_users	1	cix_site_users	PAGE	0.00	14,154	110
site_users	2	idx_site_users	PAGE	0.00	14,159	110

# How We Made It Work in Our Environment?

Keys are generated in SSMS or PowerShell as usual, originally stored in the Windows Certificate store

Use custom key store provider in JAVA:

- Pretending to be the default MSSQL\_CERTIFICATE\_STORE key store provider
- Use the given key path to get actual value from our secrets store
- Benefits:
  - Works well with our-own secrets store
  - Centralized key storage and tight security around who can access it
  - Works in Hybrid environment without Azure Key Vault
  - Extra complexity makes it a bit harder to access data:
    - Glassdoor specific shared java library must be used to access encrypted data
    - Lesser chances to leak data by mistake

[Doc: Implementing a custom column master key store provider](https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/develop-using-always-encrypted-with-net-framework-data-provider#implementing-a-custom-column-master-key-store-provider)

<https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/develop-using-always-encrypted-with-net-framework-data-provider#implementing-a-custom-column-master-key-store-provider>

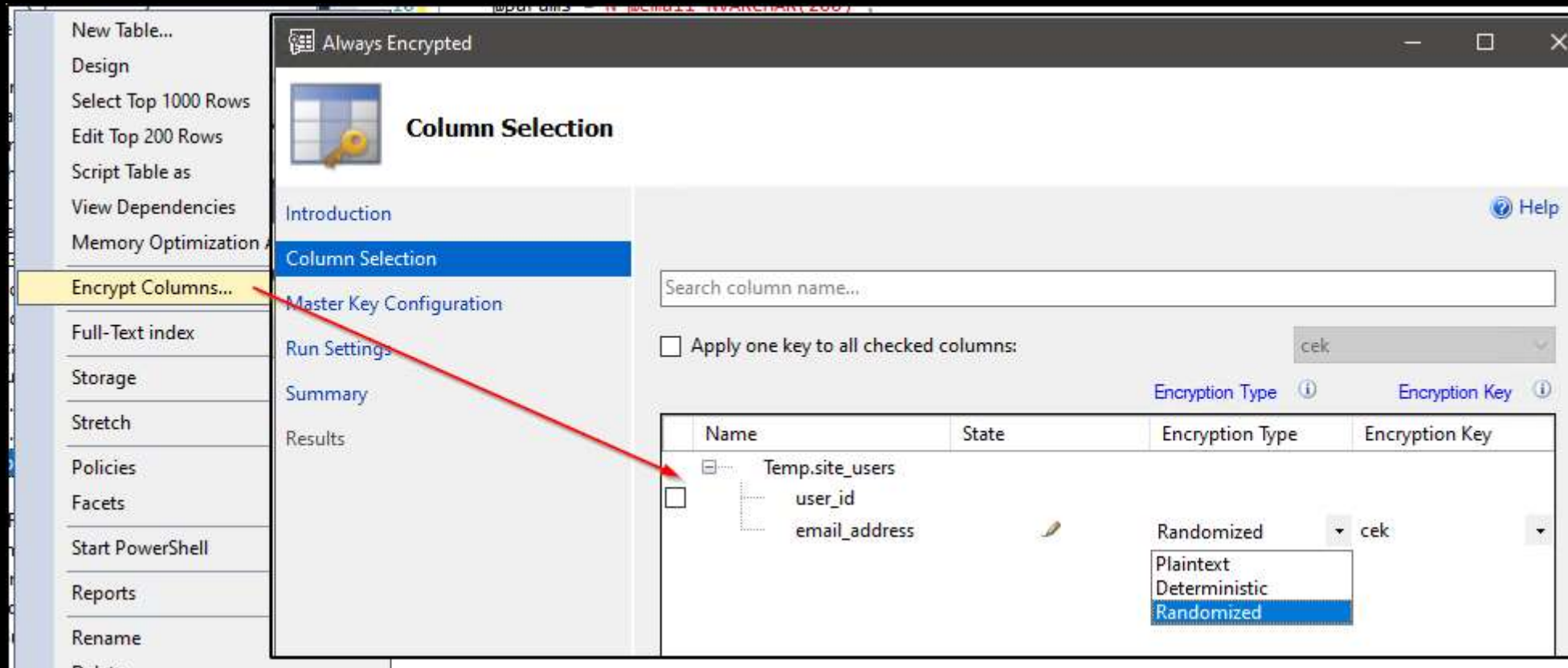
# Production Rollout - A High-Level Plan

- **Convert existing key values into uniform shape**
  - **make sure apps are reading/writing values consistently**
- **Enable encryption support and assess performance impact**
- **Generate encryption keys and make sure apps can consume it**
  - **We usually create a test table with encrypted column in each database**
  - **The app won't start if it can't read/write from/to the test table**
- **Encrypt existing data while keeping in mind roll-back plan**

# [1/4] Production Rollout Scenarios - The Easiest Case

mostly for small tables or when you can tolerate a long encryption process

- No app code changes needed\*. Data can be encrypted via SSMS.
  - the table will be blocked while encryption is in process
  - Rollback is easy – you decrypt data using the same UI





## [2/4] Production Rollout Scenarios - a Simple Case With No Downtime

For large insert-only tables (no updates)

No app code changes needed\*

- create a new encrypted column
- start copying data via PowerShell script
- pick a time and flip columns when everything is encrypted

Code Example:

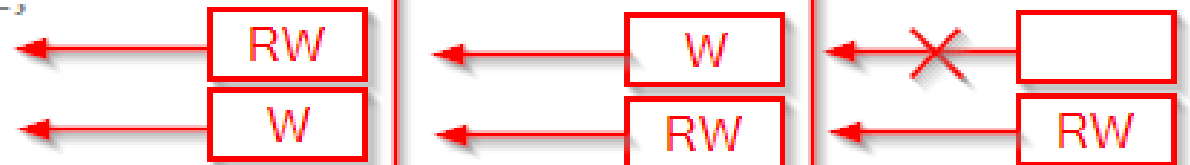
<https://github.com/alex-bochkov/pass-summit-2022-always-encrypted/tree/main/prod-rollout/option-2>

## [3/4] Production Rollout Scenarios - Leave It Up To The App Devs

Less risky but way more people involved - let application team handle it all:

- create a new encrypted column
- map that column in the application code, start writing data there
- encrypt historical records via PowerShell
- start reading from the new column
- unmap the old column from the app code, drop the old column from the database

```
CREATE TABLE [dbo].[site_users] (  
    [user_id] INT IDENTITY NOT NULL,  
    [email_address] NVARCHAR (200) NOT NULL,  
    [email_address_encrypted] NVARCHAR (200)  
        COLLATE Latin1_General_BIN2  
        ENCRYPTED WITH (  
            COLUMN_ENCRYPTION_KEY = [cek],  
            ENCRYPTION_TYPE = DETERMINISTIC,  
            ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'  
        ) NULL
```



## [4/4] Production Rollout Scenarios - a Full Example

4th approach using a trigger – a big bang but DBAs are in full control

No app code changes are needed\*

- use a trigger to detect modified records
- encrypt all historical records and newly modified values
- flip columns when everything is ready
- continue decrypting values to support instant rollback

Code Example:

<https://github.com/alex-bochkov/pass-summit-2022-always-encrypted/tree/main/prod-rollout/option-4>

```
CREATE TABLE [dbo].[site_users] (  
    [user_id],  
    [email_address] ...,  
    [email_address_encrypted] ...  
    [needsEncryption] BIT
```

Encryption  
Process

```
CREATE TABLE [dbo].[site_users] (  
    [user_id],  
    [email_address_decrypted] ...,  
    [email_address] ...  
    [needsDecryption] BIT
```

Decryption  
Process

# Key Rotation

- Column Master Key - easy
  - Generate new key and start rotation
    - Client apps will be getting both keys at the same time, may have performance impact
  - Restart all connected apps
  - Cleanup the old key
- Column Encryption Key - hard
  - Need to physically re-encrypt all data

[Doc: Rotate Always Encrypted keys using SQL Server Management Studio](https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/rotate-always-encrypted-keys-using-ssms)

<https://learn.microsoft.com/en-us/sql/relational-databases/security/encryption/rotate-always-encrypted-keys-using-ssms>

# Development and change management

- The CEK is referenced by name when you define a new encrypted column
  - Give the key the same name in all environments.
  - Key name can't be changed
- If you are building your non-prod from prod copies:
  - You may need to rotate the keys twice to persist the name (because the keys are not renamable)
  - You also need to regenerate the Column Encryption Key!  
otherwise, your prod data will be compromised even without knowing the prod Column Master Key
  - Large tables with encrypted values may need to be truncated because re-encrypting is a long process

```
CREATE TABLE [dbo].[site_users] (  
    [user_id] INT IDENTITY NOT NULL,  
    [email_address] NVARCHAR (200) COLLATE Latin1_General_BIN2  
        ENCRYPTED WITH (  
            COLUMN_ENCRYPTION_KEY = [cek],  
            ENCRYPTION_TYPE = DETERMINISTIC,  
            ALGORITHM = 'AEAD_AES_256_CBC_HMAC_SHA_256'  
        ) NOT NULL,
```

# Secure Enclaves

- Intended to solve DML limitations by operating with plain-text values in secure memory
  - Need hardware and software support (not only SQL Server involved)
- More or less easy to set up in Azure and on-prem
- Harder in AWS
- Very significant performance impact, wasn't useful for us
- You need to enable ADR to avoid long recovery times if process crashed during secure computation process

# Conclusion

- Always Encrypted is a great and reliable way to further protect sensitive data.
  - as long as you are okay with its limitations
- Microsoft is actively developing it (but mostly for secure enclaves), things are changing
  - use the most recent driver versions
- Secure Enclaves are not easy to use
  - you need good Windows Admin expertise