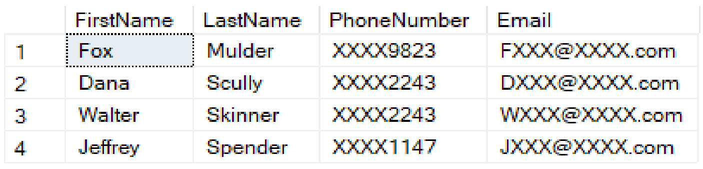
# Data Protection features in SQL Server

At Skyline we have a moral (and oftentimes legal) responsibility to build software and data solutions that can properly protect our client's confidential data. It can be detrimental to an organization if this data falls into the wrong hands. As part of Skyline's Solutions Protection program, we use many different tools to help keep our client's confidential information secure. One of the tools we leverage is the data protection features within SQL Server to protect confidential data and make it available to only those authorized to see it. This blog will cover the practical applications of these features as well as a few pros and cons of each.

## 1. Dynamic Data Masking

Adding a dynamic data mask to a column in SQL Server blocks out part of the information column.

This is useful if an employee needs to see only part of some sort of ID number or even part of a phone number for verification purposes.



| | FirstName | LastName | PhoneNumber | Email |
|---|---|---|---|---|
| 1 | Fox | Mulder | XXXX9823 | FXXX@XXXX.com |
| 2 | Dana | Scully | XXXX2243 | DXXX@XXXX.com |
| 3 | Walter | Skinner | XXXX2243 | WXXX@XXXX.com |
| 4 | Jeffrey | Spender | XXXX1147 | JXXX@XXXX.com |

Pros:

- It's easy to add dynamic data masking to a column so end users will just see the masked data.

Cons:

- It doesn't really protect the underlying data, and if enough data is shown malicious attackers can try enough combinations to guess at the data (this is called brute forcing the data).

## 2. Different Database Schemas

Schemas are a way to restrict access to tables within a database. For instance, if we want to restrict access to tables containing an employee's salary information, we can create a new schema called SensitiveEmployeeInformation and place the table containing the sensitive employee information into that schema. This way only those with access to the SensitiveEmployeeInformation schema would be able to see the table containing the employee salaries.

Pros:

- Schemas are a lightweight way to restrict access to sensitive information that have negligible performance downsides when querying the data.

Cons:

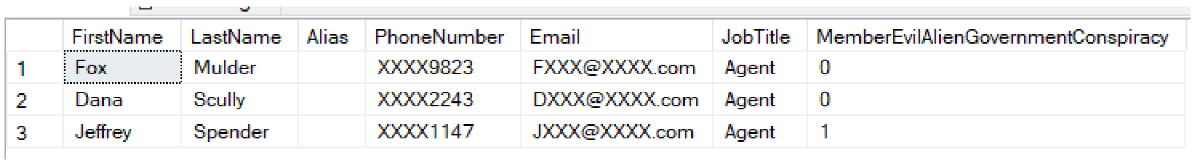
- Placing the tables in a different schema does not encrypt the underlying data (if that is a regulatory requirement).
- Great care must be taken to ensure only authorized users are added to the groups that can access the schema.

## 3. Row level Security

Row level security allows for the same users to access the same table, but SQL Server will automatically filter out rows the user is not allowed to see.

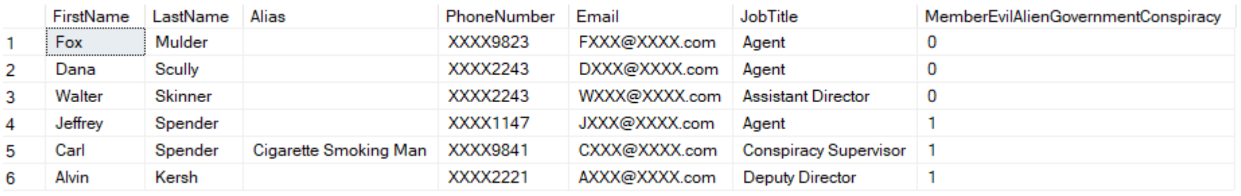
For instance, if we only want Agents to see their fellow agents but not anyone on the director level, a filtering predicate would be created to filter out anyone higher than the agent level.

Any agent querying the data would see a subset:



| | FirstName | LastName | Alias | PhoneNumber | Email | JobTitle | MemberEvilAlienGovernmentConspiracy |
|---|---|---|---|---|---|---|---|
| 1 | Fox | Mulder | | XXXX9823 | FXXX@XXXX.com | Agent | 0 |
| 2 | Dana | Scully | | XXXX2243 | DXXX@XXXX.com | Agent | 0 |
| 3 | Jeffrey | Spender | | XXXX1147 | JXXX@XXXX.com | Agent | 1 |

Anyone in a director level or higher position would see the full data set:



| | FirstName | LastName | Alias | PhoneNumber | Email | JobTitle | MemberEvilAlienGovernmentConspiracy |
|---|---|---|---|---|---|---|---|
| 1 | Fox | Mulder | | XXXX9823 | FXXX@XXXX.com | Agent | 0 |
| 2 | Dana | Scully | | XXXX2243 | DXXX@XXXX.com | Agent | 0 |
| 3 | Walter | Skinner | | XXXX2243 | WXXX@XXXX.com | Assistant Director | 0 |
| 4 | Jeffrey | Spender | | XXXX1147 | JXXX@XXXX.com | Agent | 1 |
| 5 | Carl | Spender | Cigarette Smoking Man | XXXX9841 | CXXX@XXXX.com | Conspiracy Supervisor | 1 |
| 6 | Alvin | Kersh | | XXXX2221 | AXXX@XXXX.com | Deputy Director | 1 |

Pros:

- There is no need to put higher scarcity data in different schemas or tables. The data follows the same pathways for all users

Cons:

- There are extra steps in adding or removing columns on a table with row level security.
- Any of the columns that are referenced in row level security can't be included in an index.

The next three options involve enabling encryption on the database. This involves creating a database master key and database encryption keys (among other steps). This will add additional steps if a database needs to be restored from a backup. It is **highly recommended** that before enabling these next features a risk assignment is done on where the keys are stored and how to access them if a database restore is needed.

## 4. Transparent Data Encryption

Transparent data encryption is a feature that encrypts any data that is being saved to the hard drive/disk. So, if any data is updated in a table, that data becomes transparently encrypted immediately upon save. When the data is pulled back, SQL Server will unencrypt it for you.

Pros:

- This satisfies the regulatory requirement that any data "at rest" be encrypted. All data within the database is encrypted.
- In the past a popular hacking technique has been to steal the database backups and restore the database to a server controlled by the malicious attacker. Without access to the database key, this is impossible.

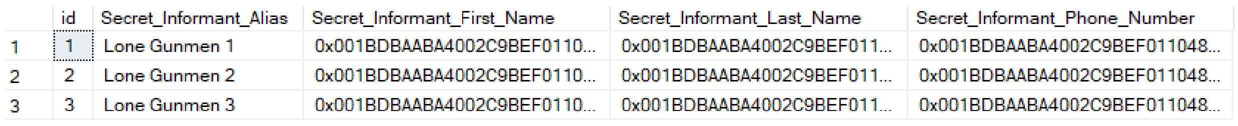
Cons:

- Since all data saved to disk is encrypted, the CPU has more work to do by encrypting and decrypting the data in the background. Be sure to plan for extra CPU capacity to account for this.
- While all at rest data is encrypted, transparent data encryption does not set any access controls regarding who can see the data once the data is back "in motion".

## 5. Encrypted Columns

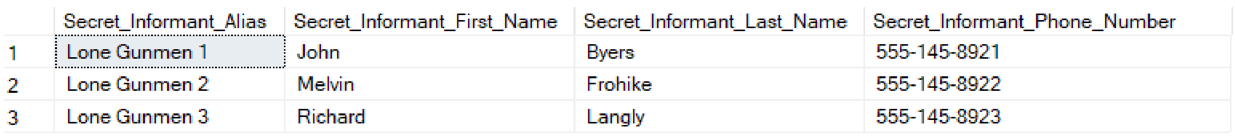
Encrypted Columns in SQL Server are columns within a table that have been encrypted to hide whatever sensitive data that the column contains. This is a good way to both hide sensitive data like a social security number or a date of birth and have the data encrypted "at rest". To read the data, special permissions are needed to access the necessary keys.

To anyone who doesn't have the access to decrypt the columns, the data would look like this:



| | id | Secret_Informant_Alias | Secret_Informant_First_Name | Secret_Informant_Last_Name | Secret_Informant_Phone_Number |
|---|---|---|---|---|---|
| 1 | 1 | Lone Gunmen 1 | 0x001BDBAABA4002C9BEF0110... | 0x001BDBAABA4002C9BEF011... | 0x001BDBAABA4002C9BEF011048... |
| 2 | 2 | Lone Gunmen 2 | 0x001BDBAABA4002C9BEF0110... | 0x001BDBAABA4002C9BEF011... | 0x001BDBAABA4002C9BEF011048... |
| 3 | 3 | Lone Gunmen 3 | 0x001BDBAABA4002C9BEF0110... | 0x001BDBAABA4002C9BEF011... | 0x001BDBAABA4002C9BEF011048... |

To those with access to decrypt the columns, the data would look like this:



| | Secret_Informant_Alias | Secret_Informant_First_Name | Secret_Informant_Last_Name | Secret_Informant_Phone_Number |
|---|---|---|---|---|
| 1 | Lone Gunmen 1 | John | Byers | 555-145-8921 |
| 2 | Lone Gunmen 2 | Melvin | Frohike | 555-145-8922 |
| 3 | Lone Gunmen 3 | Richard | Langly | 555-145-8923 |

Pros:

- The data is encrypted so this satisfies any sort of regulatory requirement of "encrypting data at rest".
- Since not all the columns are encrypted, this does not have the same CPU requirements as Transparent Data Encryption.

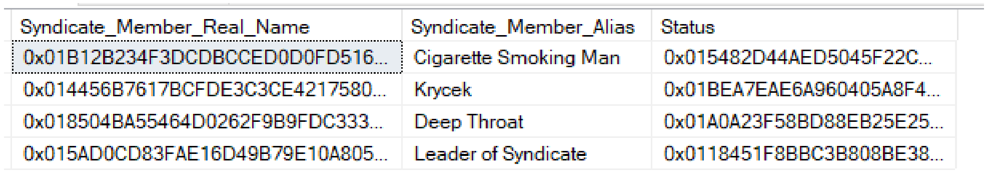
Cons:

- Access controls must still be implemented to determine who can read the data and who cannot.

## 6. Always Encrypted

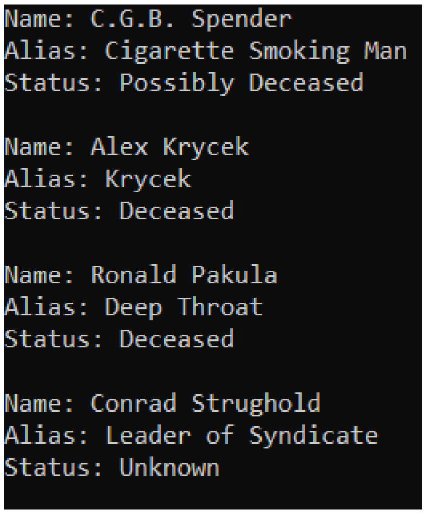
The Always Encrypted feature even prevents those who manage the database from accessing or decrypting sensitive data, while still allowing end users to read and interact with the same data. In this case, a web or desktop application is set up to encrypt or decrypt the data without the SQL Server being able to read the data.

For instance, if a Database Administrator looks in this table all he or she would see is:



| Syndicate_Member_Real_Name | Syndicate_Member_Alias | Status |
|---|---|---|
| 0x01B12B234F3DCDBCCED0D0FD516... | Cigarette Smoking Man | 0x015482D44AED5045F22C... |
| 0x014456B7617BCFDE3C3CE4217580... | Krycek | 0x01BEA7EAE6A960405A8F4... |
| 0x018504BA55464D0262F9B9FDC333... | Deep Throat | 0x01A0A23F58BD88EB25E25... |
| 0x015AD0CD83FAE16D49B79E10A805... | Leader of Syndicate | 0x0118451F8BBC3B808BE38... |

However, an end user using an application outside of SQL Server would see the following:



```
Name: C.G.B. Spender
Alias: Cigarette Smoking Man
Status: Possibly Deceased

Name: Alex Krycek
Alias: Krycek
Status: Deceased

Name: Ronald Pakula
Alias: Deep Throat
Status: Deceased

Name: Conrad Strughold
Alias: Leader of Syndicate
Status: Unknown
```

Pros:

- Always Encrypted is useful if the people working in the database are not always authorized to see the data inside the database (dates of birth, SSN's, salaries)

Cons:

- Depending on how much of the data is encrypted, any data that needs to be manipulated (added, updated, deleted) must happen in a custom-built third-party application outside of SQL Server. Thoroughly determine whether the third-party application can handle the data manipulation requirements for production scenarios.

In summary, we have looked at some of the possible ways that SQL Server can be used to protect an organization's sensitive data. Each method has its own specific use case that need to be weighed against the security and compliance needs of your organization. If you need help in unraveling the sometimes-complicated knot of which security tools to use, give Skyline a call. Our Solutions Protection program can help secure your organization and create a path forward to meet your security and compliance goals.