개인정보보호법 시행에 따른 SQL Server 데이터 보안에 대하여

㈜엑셈 컨설팅본부/ SQL Server팀 양 동환

최근 DB업계 가장 큰 화두는 보안!

최근 몇 년 사이에 포탈사이트나 금융권을 비롯한 여러 업체에서 해킹으로 인한 개인정보유출이 매스컴을 통해 알려지면서 큰 화제가 되었다. 그 영향인지 정부에서는 개인정보보호법을 제정해 시행함으로써 DB 업계에서도 보안이 큰 화두로 떠오르게 됨을 물론 보안업체들은 연일 주가가 상승하고 있다. 그렇다면 SQL Server 를 사용하는 고객들을 위해 SQL Server 에서 사용 가능한 보안에 대해 설명하고자 한다. 그 전에 먼저 개인정보보호법에 대해 간략히 설명하겠다.

개인정보보호법이란?

목적 : 개인정보의 수집, 유출, 오용, 남용으로부터 사생활의 비밀 등을 보호함으로써 국민의 권리와 이익을 증진하고, 나아가 개인의 존엄과 가치를 구현하기 위하여 개인정보 처리에 관한 사항을 규정함을 목적

대상 : 업무를 목적으로 개인정보파일을 운용하기 위하여 스스로 또는 다른 사람을 통해 개인정보를 처리하는 공공기관, 법인, 단체 및 개인 등을 말한다.

정보통신망 이용촉진 및 정보보호 등에 관련 법률이란?

목적 : 정보통신망의 이용을 촉진하고 정보통신서비스를 이용하는 자의 개인정보를 보호함과 아울러 정보통신망을 건전하고 안전하게 이용할 수 있는 환경을 조성하여 국민생활의 향상과 공공복리의 증진에 이바지함을 목적

대상 : [전기통신사업법] 제 2 조 제 8 호에 따른 전기통신사업자와 영리를 목적으로 전기통신사업자와 전기통신역무를 이용하여 정보를 제공하거나 정보의 제공을 매개하는 자

SQL Server 의 보안 설정은 인증모드를 설정하여 SQL Server 에 연결할 수 있는 권한을 부여하는 것부터 시작해서, 데이터베이스에 접근할 수 있는 권한을 부여하는 과정, 스키마 또는 특정 개체에 액세스 권한을 부여하는 과정으로 진행한다. 쉽게 말해 보안의 시작은 애초에 꼭 필요한 사람만 SQL Server 에 접근하도록 하고, 꼭 필요한 권한만을 부여하여야 한다는 점이다. 물론 관리자에게는 불편하고 귀찮은 일이 될 수도 있겠지만 보안에 있어서는 '설마~'라는 생각은 하지 않는 것이 좋다.

필자는 이 글에서 DB 데이터 암호화에 대한 방법을 중점적으로 설명하고자 한다. 아래 표를 DB 데이터 암호화 방법 및 장단점에 대해 알아보겠다.

구분	암호화 기술	내용
칼럼 단위	Plug-in 방식	**`암/복호화 모듈이 DBMS 안에서 실행** **`원래의 테이블과 동일한 이름의 뷰가 생성** **`실제 테이블을 변경하기 위해 instead of trigger 생성** **`ORACLE, SQL Server, DB2 사용하는 환경에 적합**
	API 방식	**`암/복호화 모듈이 어플리케이션에서 실행** **`저장/변경 시 암호화되고, 조회 시 복호화하는 API 함수 호출** **`뷰나 트리거 등이 생성되지 않음**
블록 단위	TDE	**`DB 커널에서 암호화된 테이블 스페이스를 생성하고, 암호화 대상 테이블을 해당 테이블 스페이스로 이동** **`DBMS 커널에서 DB 의 블록 단위로 자동으로 암/복호화 수행** **`HSM(Hard Security Machine) 어플라이언스와 연동하여 키 관**

		리
	File 암호화	`암호화 파일을 사용하여 테이블 스페이스를 생성하고, 암호화 대상 테이블을 해당 테이블 스페이스로 이동 `**OS** 커널에서 **DB** 의 블록 단위로 자동으로 암/복호화 수행 `**HSM(Hard Security Machine)** 어플라이언스와 연동하여 키 관리

표 1. DB 암호화 방식

암호화 기술	장점	단점
Plug-in	`기존 소스 변경 최소화 `기존의 쿼리 변경 없음 `구축에 소요되는 기간 짧음 `최소비용	`**DBMS** 서버에 부하 `암호화 대상 칼럼 증가 시 추가 암호화 작업 필요 `**SQL** 튜닝 필요 `별도 인덱스 기능 필요 `일부 어플리케이션 변경 필요
API	`**DB** 서버에 부하가 없음 `속도 빠름	`다수의 프로그램 수정 `장기간 소요 `프로그램 수정, 테스트를 위한 비용이 크게 발생 `별도의 인덱스 기능 필요 `대량의 데이터 암/복호화 수행 시 **Network Jam**, 암/복호화 성능 저하

TDE	`어플리케이션 변경 없음 `성능 우수 `모든 데이터 타입 지원	`DB 에 직접 접속 시, 보안 성 떨어짐 `SQL Server 2008 EE 이상 업그레이드 필요
File 암호화	`DBMS 자체 인덱스 모두 사용 가능	`DB 에 직접 접속 시 보안 성 떨어짐 `일부 OS, Volume Manager 의 경우 지원 불가

표 2. DB 암호화 방식의 장단점

그렇다면 DB 암호화 방식 중 Plug-in 방식과 TDE 방식을 SQL Server 에서 간단한 TEST 와 함께 설명하도록 하겠다.

Plug-In & TDE TEST

칼럼 암호화(Plug-In)

칼럼 암호화를 사용하기 위해서는 우선 다음과 같이 몇 가지 고려 사항을 생각해봐야 한다.

- 암호화 된 열의 사이즈 증가
- 암호화에 사용된 암호 또는 키 관리 중요
- 암호화 된 데이터에 대한 인덱스 생성 및 정렬 불가능
- 암호화 및 복호화를 위한 CPU 사용량 증가
- 암호화 및 복호화 함수에 대한 정확한 이해

그 다음 암호화 할 칼럼을 선정 한 후에 암호화 방법을 선택한다. 암호화 방법은 아래 표를 통해 설명하겠다.

암호화 방법	사용 가능한 암호화 알고리즘
대칭키를 사용한 암호화 및 복호화	**SEED, ARIA-128/192/256, AES-128/192/256, Blowfish, Camelia128/192/256, MISTY1, KASUMI 등**
비대칭키를 사용한 암호화 및 복호화	**RSA, KCDSA(전자 서명용), RSAES-OAEP, RSAES-PKCS1 등**
인증서를 사용한 암호화 및 복호화	**-**
HASHBYTES 함수를 사용한 암호화 (복호화 할 수 없는 단 방향 암호화)	**MD2/4/5, SHA-224/256/384/512, Whirlpool 등**

표3 암호화 방법 및 사용 가능한 암호화 알고리즘

그럼 칼럼 암호화 테스트에 앞서 각 암호화 방법들을 구현하기 위해 사용되는 구문들을 표를 통해 간단히 설명하겠다.

암호화 방법	구문	암/복호화 함수
대칭키	**CREATE SYMMETRIC KEY key_name** **WITH <key_option>** **ENCRYPTION BY <encryption_mechanism>**	**EncryptByKey()** **DecryptByKey()**
비대칭키	**CREATE ASYMMETRIC KEY asym_key_name** **WITH <key_option>** **ENCRYPTION BY**	**EncryptByAsymKey()** **DecryptByAsymKey()**

	<encrypting_mechanism>	
인증서	**CREATE CERTIFICATE certificate_name** **WITH SUBJECT =** **'certificate_subject_name'**	**EncryptByCert()** **DecryptByCert()**
HASHBYTES	**HASHBYTES('<algorithm>', {@input |** **'input'})**	**HASHBYTES()** **없음**

표4 암호화 방법에 따른 구문 및 암/복호화 함수

테스트로는 간단히 대칭키를 사용한 암호화 방법을 대표로 진행해 보도록 하겠다. 테스트에 앞서 다음과 같이 'TEST_COL'이라는 테이블을 임시로 생성해 주었다.

```
SELECT * FROM TEST_COL
```

결과 | 메시지

	ID	생년월일	이름
1	1	870122	김태희
2	2	880122	한지민
3	3	890122	이연희
4	4	900122	김희선
5	5	910122	손나은

먼저 대칭키를 사용한 칼럼 암호화를 보겠다.

```
-- 대칭 키 만들기
CREATE SYMMETRIC KEY symkey_test
    WITH ALGORITHM = AES_256
    ENCRYPTION BY PASSWORD = 'Pa$$w0rd'
GO
```

```
-- 대칭 키 열기(생성 직후에는 열려있으나 그 외에는 열어줘야함)
OPEN SYMMETRIC KEY symkey_test
    DECRYPTION BY PASSWORD = 'Pa$$w0rd'
GO
```

```
-- 암호화 열 추가
ALTER TABLE TEST_COL
    ADD 암호_생년월일 VARBINARY(256)
GO
```

```
-- 추가한 열에 암호화 한 데이터 저장
UPDATE TEST_COL
    SET 암호_생년월일 = ENCRYPTBYKEY(KEY_GUID('symkey_test'),생년월일)
GO
```

```
-- 기존 암호화 되지 않은 열 제거
ALTER TABLE TEST_COL
    DROP COLUMN 생년월일
```

이렇게 암호화를 적용하고 단순히 SELECT 를 하게 되면 아래와 같이 암호화된 열에 이상한 값들이 보이게 된다.

```
SELECT * FROM TEST_COL
```

결과 | 메시지

	ID	이름	암호_생년월일
1	1	김태희	0x007F949E86C73F4796DE572EE2EDB1BB0100000080215AD1C9B139066CFD797788157A7ECA8A61E701861FC4D12EE6316BC10F39
2	2	한지민	0x007F949E86C73F4796DE572EE2EDB1BB01000000012F556F20AEFF0D70FD562E27DCF5A9FA62B29CD83387FC7852543C2CECD732
3	3	이연희	0x007F949E86C73F4796DE572EE2EDB1BB01000000437AA1DBBA8757E86F810C7F775DBFD4C153CD1F557028A64FB7DD84F9CB8951
4	4	김희선	0x007F949E86C73F4796DE572EE2EDB1BB010000000D17177A68806D9CFB290EB0A74B2829C3CDDA0EB632481DF5A88218EFC87E3A9
5	5	손나은	0x007F949E86C73F4796DE572EE2EDB1BB01000000C528E38284299225792B99F1A6EF2A8BD75C38645147DAA741CB09023F7C8319

그럴 땐 SELECT 구문을 아래와 같이 수정해 주어야 한다.

```
SELECT ID, 이름, CONVERT(VARCHAR, 암호_생년월일) AS '생년월일'
    FROM TEST_COL
```

결과 메시지

	ID	이름	생년월일
1	1	김태희	
2	2	한지민	
3	3	이연희	
4	4	김희선	
5	5	손나은	

암호화 한 열을 복호화 하여 조회하고 싶을 때에는 아래와 같이 SELECT 구문을 수정해 주면 된다. 복호화 하기 전 항상 대칭 키가 먼저 열려있는지 확인해야 한다. 대칭 키를 열어주지 않으면 복호화 되지 않는다.

```
-- 대칭 키 열기(생성 직후에는 열려있으나 그 외에는 열어줘야함)
OPEN SYMMETRIC KEY symkey_test
    DECRYPTION BY PASSWORD = 'Pa$$w0rd'
GO

SELECT ID, 이름, CONVERT(VARCHAR, DECRYPTBYKEY(암호_생년월일)) AS '생년월일'
    FROM TEST_COL
```

결과 메시지

	ID	이름	생년월일
1	1	김태희	870122
2	2	한지민	880122
3	3	이연희	890122
4	4	김희선	900122
5	5	손나은	910122

다음은 암호화 된 열에 INSERT 를 할 경우에 대한 구문이다.

```
-- 암호화 된 열에 행 추가
INSERT INTO TEST_COL
VALUES ('아이유', ENCRYPTBYKEY(KEY_GUID('symkey_test'),'920122'))
```

테스트를 모두 완료하면 테이블을 삭제한 후 대칭키를 삭제하는 구문이다.

```
--대칭키 제거
DROP SYMMETRIC KEY symkey_test
```

대칭키에 대한 테스트는 이것으로 마치겠다.

TDE(Transparent Data Encryption)

앞에서 살펴 본 칼럼 암호화 방법들은 암호화 되지 않은 칼럼에 대해서는 쉽게 조회가 가능하고 암호화 및 복호화 함수를 사용하도록 해당 쿼리 문들을 모두 수정해야 한다는 불편함을 가지고 있다. 이에 SQL Server 에서는 칼럼 수준이 아닌 데이터베이스 수준의 암호화 방식인 TDE 를 제공한다. TDE 는 데이터베이스 암호화뿐만 아니라 그 데이터베이스의 백업도 암호화 되어 저장된다. 그러므로 데이터베이스 혹은 백업이 외부로 유출되더라도 복원 및 Attach 가 불가능하다. 그리고 TDE 는 데이터베이스 엔진에서 암/복호화를 수행하기 때문에 쿼리 문 수정이 필요하지 않다는 장점이 있다. 하지만 엔진에서 암/복호화를 수행함으로써 엔진 서비스가 시작하고 있을 때는 쉽게 해당 데이터에 대한 조회가 가능하고 CPU 사용량도 늘어난다는 단점도 분명히 존재한다. 유의 할 점으로는 TDE 는 Enterprise Edition 에서만 제공되는 기능이다(SQL Server 2008 이상 버전). 참고로 <개인정보 암호화 조치 안내서>를 보면 TDE 방식에서 안전한 암호 알고리즘을 사용하여 암호화하면 개인정보보호법에 위반되지 않는다고 나와있다.

TDE 암호화 수행과정 및 구문은 다음과 같다.

수행과정(아래방향으로 진행)	구문
마스터 키 생성 (master DB 에서 수행)	**CREATE MASTER KEY ENCRYPTION** **BY PASSWORD = 'password'**
인증서 만들기 (master DB 에서 수행)	**CREATE CERTIFICATE certificate_name** **WITH SUBJECT = 'certificate_description'**
데이터베이스 암호화 키 만들기 (암호화 대상 DB 에서 수행)	**CREATE DATABASE ENCRYPTION KEY** **WITH ALGORITHM = ' algorithm_name'** **ENCRYPTION BY SERVER{** **CERTIFICATE encryption_name | ASYMMETRIC**

	KEY encryptor_name }
데이터베이스 암호화 속성 설정 (암호화 대상 DB 에서 수행)	**CREATE DATABASE ENCRYPTION KEY** **WITH ALGORITHM = algorithm_name** **ENCRYPTION BY SERVER CERTIFICATE certificate_name**

표5 TDE 수행과정에 따른 구문

그럼 간단한 예제를 통해 TDE 를 구성하는 테스트를 진행해보겠다.

```
-- 마스터 키 생성
USE master
GO

CREATE MASTER KEY ENCRYPTION
    BY PASSWORD = 'Pa$$w0rd'
GO
```

```
-- 인증서 만들기
USE master
GO

CREATE CERTIFICATE CERTofTDE
    WITH SUBJECT = 'TDEtest'
GO
```

```
--데이터베이스 암호화 키 만들기
USE TEST
GO

CREATE DATABASE ENCRYPTION KEY
    WITH ALGORITHM = AES_256
    ENCRYPTION BY SERVER CERTIFICATE CERTofTDE
GO
```

위의 구문을 실행하면 메시지 창에 아래와 같은 경고가 발생한다.

> 경고: 데이터베이스암호화키를암호화하는데사용된인증서가백업되지않았습니다. 인증서와인증서에연결된개인키를즉시백업해야합니다. 인증서를사용할수없게되거나다른 서버에서데이터베이스를복원하거나연결해야할경우인증서와개인키의백업본이있어야합니다. 그렇지않으면데이터베이스를열수없습니다.

TDE 를 사용하는 경우에는 경고메시지에 나온 것처럼 관련된 인증서를 필히 백업을 받아야 한다.

```
-- 인증서 백업(개인키 포함)
BACKUP CERTIFICATE CERTofTDE
    TO FILE = 'CERTofTDE.cer'
    WITH PRIVATE KEY (FILE = 'E:₩BACKUP₩CERTofTDE.PVK',
        ENCRYPTION BY PASSWORD = 'Pa$$w0rd')
GO
```

다음은 TDE 로 암호화된 데이터베이스에 대한 백업&복원을 테스트를 해보았다.

```
-- TDE 적용된 데이터베이스 백업
BACKUP DATABASE TEST
    TO DISK = 'E:₩BACKUP₩TEST.bak'
    WITH INIT
GO
```

```
RESTORE DATABASE TEST
    FROM DISK = 'E:₩BACKUP₩TEST.bak'
    WITH MOVE 'TEST' TO 'D:₩DATA₩TEST.mdf',
         MOVE 'TEST_LOG' TO 'D:₩DATA₩TEST_LOG.ldf'
GO
```

```
-- TDE 적용된 데이터베이스 다른 서버에서 복원
USE master
GO

CREATE MASTER KEY ENCRYPTION
    BY PASSWORD = 'Pa$$w0rd'
GO

CREATE CERTIFICATE CERTofTDE
    FROM FILE = 'E:₩BACKUP₩CERTofTDE.cer'
    WITH PRIVATE KEY (FILE = 'E:₩BACKUP₩CERTofTDE.pvk',
        DECRYPTION BY PASSWORD = 'Pa$$w0rd')
GO
```

TDE 가 적용된 데이터베이스를 다른 서버에 복원하려면 위의 그림과 같이 새로운 서버에 마스터 키를 생성한 후 백업 받아둔 인증서를 먼저 복원한 후 데이터베이스를 복원해야 오류 없이 복원이 가능하다.

다음은 TDE 비활성화 및 TDE 관련된 것들의 제거에 대한 테스트를 해 보았다.

```
--TDE 비활성화
USE master
GO

ALTER DATABASE TEST
    SET ENCRYPTION OFF
GO
```

TDE 관련된 것들을 제거하려면 TDE 를 적용하기 위해 진행했던 순서의 반대로 진행해야 한다.

데이터베이스 암호화 키를 먼저 제거한 후 인증서 제거, 마스터 키 제거 순으로 진행하면 된다.

```
--TDE 제거
USE TEST
GO

DROP DATABASE ENCRYPTION KEY
GO

USE master
GO

DROP CERTIFICATE CERTofTDE
GO

DROP MASTER KEY
GO
```

그리고 TDE 적용된 시점에 남아있는 데이터베이스 로그도 삭제하기 위한 과정도 필요하다.

복구모델을 단순으로 변경한 후 강제로 CHECKPOINT 를 2 번정도 발생시켜 로그를 지우는 과정이다.

```
-- TDE 적용시점의 데이터베이스 로그 삭제
ALTER DATABASE TEST
    SET RECOVERY simple
GO

USE TEST
GO

CHECKPOINT      -- 2번정도 실행
GO

ALTER DATABASE TEST
    SET RECOVERY full
GO
```

보안은 더 이상 옵션이 아닌 필수!

보안은 시스템을 공격하는 데 드는 노력보다 그 데이터로부터 얻는 이득이 많은 시스템에 충분한 장벽을 만드는 활동이다. 시간이 지날수록 데이터의 중요성이 점점 커지는 이 때 그 데이터를 지키기 위한 보안은 더 이상 옵션이 아닌 필수사항이라고 볼 수 있다. 예전과 달리 이젠 데이터 보호가 법으로도 제정된 만큼 보안에 더욱 신경을 써야 한다. 다행히 TDE 를 사용한 데이터베이스 암호화 방식이 법으로도 문제가 없는 만큼 따로 비용을 지불해서 3rd Party Tool 을 사용하지 않고도 SQL Server 자체 기능으로 충분히 암호화를 할 수 있다는 건 크나큰 비용절감을 가져온 것이라고 본다. 물론 TDE 에도 그에 따른 단점이 분명히 존재하지만 장점이 더욱 많은 기능이라고 생각한다. 앞으로 SQL Server 신 버전이 나올수록 보안에 대한 기능들은 업그레이드 될 것이라고 판단하며 이 글을 마친다.