KAPITOLA

7

Šifrování v systému SQL server

Témata kapitoly:

- **Šifrovací klíče**
- **Transparentní šifrování dat**
- **Povolení transparentního šifrování dat**
- **Šifrování bez klíčů**
- **Katalogové pohledy zabezpečení**
- **Efektivita dotazů**

Vysoce výkonné počítače a levná úložiště umožnila firmám skladovat nebývalé množství osobních dat o svých zákaznících. Výsledkem je velký tlak na zabezpečení těchto důvěrných informací. Dokonce i státy a vlády se zapojily do hry tím, že vydaly množství zákonů o ochraně spotřebitelů a zákonů a nařízení o ochraně osobních údajů. Pokud vaše organizace spadá do oblasti různých regulací a zákonů o ochraně dat, jakými jsou Health Insurance Portability and Accountability Act (HIPAA) nebo Securities and Exchange Commission's Fair a Accurate Credit Transactions Act (FACTA), pak vás může těšit, že systém SQL Server 2008 obsahuje několik rozšíření jazyka Transact-SQL (T-SQL) a zabudovaných funkcí, aby bylo zabezpečení osobních dat v databázi jednodušší než kdykoli předtím.

V této kapitole budeme rozebírat nástroje pro šifrování zabudované do systému SQL Server 2008 a budeme se zabývat následujícími tématy:

- Model šifrování systému SQL Server včetně hierarchie zašifrování v systému SQL Server a nově představený koncept certifikátů serveru a šifrovacích klíčů databáze.
- Transparentní šifrování dat, které vám umožňuje transparentně zašifrovat celou databázi najednou, aniž byste zasáhli klientské aplikace.
- Správa rozšiřitelných klíčů (EKM – Extensible Key Management), která vám umožňuje použít moduly hardwarového zabezpečení (HSM – hardware security modules) třetích stran pro externí správu podnikových šifrovacích klíčů.
- Funkce pro symetrické a asymetrické šifrování.
- Generování jednosměrných hodnot otisků dat a použití certifikátů k digitálnímu podepisování vašich dat.
- Použití katalogových pohledů zabezpečení pro přístup k metadatům o zabezpečení.
- Výkon při dotazování se na zašifrovaná data.

POZNÁMKA

Zašifrování systému SQL Server, kterému se věnujeme v této kapitole, je jenom malou částí vaší celkové strategie zabezpečení. Zašifrování na úrovni databáze ochrání vaše data, když jsou „v klidu", a je vaší poslední obrannou linií v celkové strategii. Při vývoji vaší celkové strategie zabezpečení byste měli vzít v potaz několik faktorů, včetně fyzického zabezpečení, síťového zabezpečení, zabezpečení klientských počítačů a zabezpečení aplikací. Lidé, kteří si myslí, že zašifrování jejich dat v klidu na jejich serverech je úplnou strategií zabezpečení, způsobují mnoho zmatků. Ujistěte se, že uvažujete o všech rozdílných částech této skládačky a nepředpokládejte, že samotné zašifrování na úrovni databáze je úplnou strategií zabezpečení.

Šifrovací klíče

Model šifrování systému SQL Server obsahuje zabudovanou správu šifrovacích klíčů ve formátu v souladu se standardem ANSI X9.17. Tento standard definuje několik vrstev šifrovacích klíčů, které se používají k zašifrování dalších klíčů, které se následně používají k zašifrování vlastních dat. Vrstvy šifrovacích klíčů definovaných standardem ANSI X9.17 zhruba odpovídající vrstvám klíčů definovaným systémem SQL Server a jsou vypsány v tabulce 7.1.

Tabulka 7.1: Porovnání vrstev šifrovacích klíčů systému SQL Server a standardu ANSI X9.17

Vrstva systému SQL Server	Vrstva standardu ANSI X9.17	Popis
Hlavní klíč služby (SMK – Service master key)	Hlavní klíč (KKM – Master Key)	Zašifruje hlavní klíč databáze
Hlavní klíč databáze (DMK – Database master key), asymetrické klíče, certifikáty, symetrické klíče	Klíč k šifrování klíčů (KEK – Key Encrypting Key)	Zašifruje symetrické klíče
Symetrické klíče	Datový klíč (KD – Data Key)	Šifruje data

POZNÁMKA

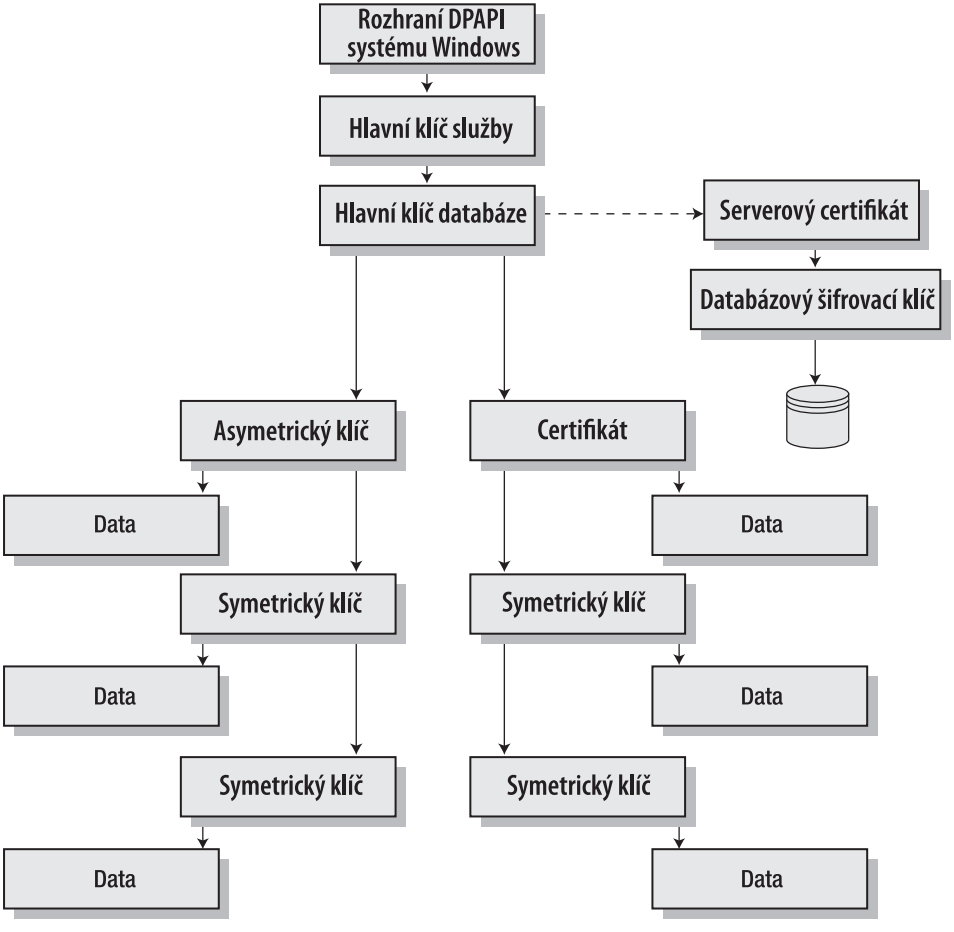
Standard ANSI X9.17 je standard pro „Správu klíčů finančních institucí". Tento standard definuje metody pro zabezpečené uchovávání a přenášení šifrovacích klíčů, což je oblast šifrování, která byla předmětem debat v oboru kryptografie po několik desetiletí. Správa šifrovacích klíčů není jednoduchá. Nakonec moderní teorie šifrování má jen jednu primární směrnici: Zabezpečení systému je dáno šifrovacími klíči. Bez ohledu na to, jaký algoritmus použijete k zašifrování vašich dat, pokud není váš klíč řádně zabezpečen, mohou být vaše data jednoduše zneužita. Správa šifrovacích klíčů v systému SQL Server je postavena na aplikačním rozhraní DPAPI (Data Protection API), díky čemuž je tak bezpečný, jak jen může být na počítači, který provozuje systém Windows.

Hlavní klíč služby (SMK) je klíč nejvyšší úrovně, základ všech klíčů v systému SQL Server. Existuje jediný hlavní klíč služby definovaný pro každou instanci systému SQL Server 2008. Hlavní klíč služby je zabezpečen pomocí rozhraní DPAPI systému Windows (Windows Data Protection API) a je vydán pro zašifrování další vrstvy klíčů, hlavních klíčů databází (DMK – Database master key). Hlavní klíč služby se vytvoří automaticky při první příležitosti, když je potřeba.

Hlavní klíče databáze se používají pro zašifrování symetrických klíčů, asymetrických klíčů a certifikátů. Každá databáze může mít jediný, pro ni definovaný hlavní klíč databáze.

Další vrstva klíčů obsahuje symetrické klíče, asymetrické klíče a certifikáty. Symetrické klíče jsou primární prostředky pro zašifrování dat v databázi. Zatímco asymetrické klíče a certifikáty je možné použít k zašifrování dat (s trochou úsilí, kterému se budeme věnovat v sekci „Asymetrické klíče" dále v této kapitole), Microsoft doporučuje, abyste šifrovali data výhradně s pomocí symetrických klíčů.

Ke všem klíčům a certifikátům představeným v systému SQL Server 2005 systém SQL Server 2008 navíc představuje koncept serverových certifikátů a databázových šifrovacích klíčů pro podporu nové funkce transparentního zašifrování dat. Serverový certifikát je prostě certifikát vytvořený v databázi `master`. Databázový šifrovací klíč je speciální symetrický klíč, který se používá k zašifrování celé databáze najednou. Serverovému certifikátu a databázovému šifrovacímu klíči se budeme podrobně věnovat v sekci „Transparentní šifrování dat" dále v této kapitole. Obrázek 7.1 zobrazuje hierarchii šifrovacích klíčů v systému SQL Server 2008.



Obrázek 7.1: Hierarchie šifrovacích klíčů v systému SQL Server

Hlavní klíč služby (SMK)

Systém SQL Server 2008 obsahuje následující příkazy jazyka T-SQL pro změnu, zálohu a smazání hlavních klíčů služby:

- `ALTER SERVICE MASTER KEY`: Umožňuje vám změnit nebo obnovit hlavní klíč služby. Tento příkaz je možné použít pro změnu hlavního klíče služby a k automatickému dešifrování a opětovnému zašifrování celé hierarchie šifrovacích klíčů.
- `BACKUP SERVICE MASTER KEY`: Zazálohuje váš hlavní klíč služby do souboru. Hlavní klíč služby je zašifrován před zazálohováním a uložen v zašifrovaném formátu. Musíte poskytnout heslo, které se použije pro zašifrování hlavního klíče služby do souboru.
- `RESTORE SERVICE MASTER KEY`: Obnoví váš hlavní klíč služby ze souboru. Příkaz `RESTORE` vyžaduje, abyste použili stejné heslo, které jste použili při zálohování hlavního klíče služby. Podobně jako příkaz `ALTER SERVICE MASTER KEY`, příkaz `RESTORE SERVICE MASTER KEY` obnovuje hierarchii šifrovacích klíčů.

Po instalaci nové instalace systému SQL Server 2008 byste měli okamžitě zazálohovat hlavní klíč služby a uložit jej na bezpečném místě. Příkaz `BACKUP SERVICE MASTER KEY` má následující podobu:

```
BACKUP SERVICE MASTER KEY TO FILE = 'c:\MK\backup_master_key.dat'
    ENCRYPTION BY PASSWORD = 'p@$$w0rD';
```

V tomto případě se hlavní klíč služby zazálohuje do souboru `c:\MK\backup_master_key.dat` pomocí hesla `p@$$w0rD`. Šifrovací heslo bude vyžadováno, pokud budete potřebovat obnovit tento hlavní klíč služby.

Pokud potřebujete změnit, obnovit ze zálohy nebo regenerovat váš hlavní klíč služby, systém SQL Server se pokusí dešifrovat a znovu zašifrovat všechny klíče v hierarchii šifrovacích klíčů. Pokud kterékoliv z těchto dešifrování selže, selže celý proces. Pokud se tak stane, můžete použít volbu `FORCE` pro příkazy `ALTER SERVICE MASTER KEY` a `RESTORE SERVICE MASTER KEY`. Avšak buďte si vědomi toho, že pokud musíte použít volbu `FORCE`, můžete počítat se ztrátou dat.

Hlavní klíče databáze

Jak jsme zmínili dříve, hierarchie šifrovacích klíčů systému SQL Server obsahuje jeden hlavní klíč databáze pro každou databázi. Hlavní klíč databáze přímo šifruje asymetrické klíče a certifikáty, které je možné použít k zašifrování symetrických klíčů. Symetrické klíče se následně používají k zašifrování jiných symetrických klíčů a dat.

Na rozdíl od hlavního klíče služby, který se vygeneruje automaticky při první příležitosti, kdy je potřeba, hlavní klíč databáze musí být explicitně vytvořen pomocí příkazu `CREATE MASTER KEY`. Systém SQL Server obsahuje následující příkazy jazyka T-SQL pro správu hlavních klíčů databáze:

- `CREATE MASTER KEY`: Vytvoří hlavní klíč databáze uvnitř databáze. Musíte zadat heslo pro zašifrování hlavního klíče databáze při jeho vytvoření.
- `ALTER MASTER KEY`: Umožní vám regenerovat váš hlavní klíč databáze nebo změnit způsob zabezpečení hlavního klíče databáze přidáním nebo odebráním šifrování pomocí hesla nebo hlavního klíče služby. Pokud zregenerujete váš hlavní klíč databáze, všechny klíče, které ochraňuje, se dešifrují a opětovně zašifrují.

- DROP MASTER KEY: Smaže hlavní klíč databáze z aktuální databáze. Pokud jsou jakékoli privátní klíče chráněny pomocí hlavního klíče databáze, příkaz DROP selže.
- BACKUP MASTER KEY: Zálohuje váš hlavní klíč databáze do souboru. Musíte zadat heslo, které se použije pro zašifrování hlavního klíče databáze do souboru.
- RESTORE MASTER KEY: Obnoví váš hlavní klíč databáze ze souboru. Pro úspěšné provedení operace RESTORE musíte zadat stejné heslo, které ještě použili při zálohování hlavního klíče databáze. Musíte také zadat druhé heslo pro zašifrování hlavního klíče databáze po jeho obnovení. V průběhu procesu obnovování se systém SQL Server pokusí dešifrovat a opětovně zašifrovat všechny klíče chráněné hlavním klíčem databáze.
- OPEN MASTER KEY: Otevře hlavní klíč databáze, aby se mohl použít k šifrování a dešifrování. Hlavní klíč databáze musí být otevřen pro úspěšné provedení operací šifrovaní a dešifrování, přestože systém SQL Server může implicitně otevřít váš hlavní klíč databáze, pokud je chráněn pomocí hlavního klíče služby, což krátce rozebereme.
- CLOSE MASTER KEY: Zavře hlavní klíč databáze, který byl explicitně otevřen použitím příkazu OPEN MASTER KEY potom, co jste ukončili jeho používání pro šifrování a dešifrování.

Příkazy ALTER MASTER KEY a RESTORE MASTER KEY se pokusí regenerovat hierarchii šifrování klíčů, které hlavní klíč databáze chrání. To znamená, že tyto příkazy se automaticky pokusí dešifrovat a opětovně zašifrovat všechny šifrovací klíče v hierarchii pod hlavním klíčem databáze. Pokud jakékoli z těchto dešifrování selže, selže celý příkaz ALTER nebo RESTORE. Příkazy ALTER nebo RESTORE donutíte dokončit bez ohledu na chyby pomocí volby FORCE. Avšak pozor: Volbu FORCE byste měli použít jako poslední možnost, protože vždy vede ke ztrátě dat.

Všechny příkazy pro správu hlavních klíčů databáze vyžadují oprávnění CONTROL na danou databázi a vždy musí být spuštěny v kontextu aktuální databáze.

Následující příkaz vytváří hlavní klíč databáze v databázi AdventureWorks:

```
USE AdventureWorks;
GO
CREATE MASTER KEY
 ENCRYPTION BY PASSWORD = N'Avx3$5*!';
```

Měli byste si zazálohovat všechny vaše hlavní klíče databáze a uložit je na bezpečném místě, hned jak je vytvoříte. Hlavní klíč databáze můžete zazálohovat pomocí podobných příkazů, jako je následující:

```
BACKUP MASTER KEY TO FILE = N'c:\MK\AwMasterKeyBackup.bak'
 ENCRYPTION BY PASSWORD = N'#%e3)Fr';
GO
```

Pokud budete kdykoli potřebovat obnovit hlavní klíč databáze, použijte následující příkaz RESTORE MASTER KEY:

```
RESTORE MASTER KEY FROM FILE = 'c:\MK\AwMasterKeyBackup.bak'
 DECRYPTION BY PASSWORD = N'#%e3)Fr'
 ENCRYPTION BY PASSWORD = N'Avx3$5*!';
GO
```

Při obnovení hlavního klíče databáze musíte v podmínce `DECRYPTION BY PASSWORD` zadat stejné heslo, jako jste použili při provádění operace `BACKUP`.

Systém SQL Server 2008 poskytuje dvě metody zabezpečení hlavních klíčů databáze. První metoda vyžaduje, abyste explicitně zadali heslo, když budete vytvářet, měnit nebo obnovovat váš hlavní klíč databáze. Toto heslo se použije pro zašifrování hlavního klíče databáze při jeho uložení do databáze. Pokud zašifrujete váš hlavní klíč databáze pomocí hesla, musíte zadat stejné heslo pokaždé, když musíte přistoupit ke klíčům, které hlavní klíč databáze chrání. To také znamená že musíte použít příkazy `OPEN MASTER KEY` a `CLOSE MASTER KEY` k explicitnímu otevření a zavření hlavního klíče databáze.

Ve výchozím nastavení systém SQL Server také nabízí druhou metodu zabezpečení vašich hlavních klíčů databáze. Když vytvoříte hlavní klíč databáze, automaticky se zašifruje pomocí hlavního klíče služby a trojitého algoritmu DES a jeho kopie se uloží současně do aktuální databáze a databáze master. To umožní systému SQL Server automaticky otevírat a zavírat váš hlavní klíč databáze, když je to potřeba, aniž byste zadávali heslo.

Výhodou je, že toto automatické zabezpečení založené na hlavním klíči služby zjednodušuje vývoj, protože nemusíte explicitně používat příkazy `OPEN MASTER KEY` a `CLOSE MASTER KEY` k otevření a zavření hlavního klíče databáze pokaždé, když zašifrujete svoje data. Nemusíte se také starat o správu, uchovávání a/nebo přenos hesla do systému SQL Server pokaždé, když chcete provést operaci zašifrování nebo dešifrování. Stinnou stránkou této metody (a vy jste věděli, že tato metoda bude nějakou mít) je, že každý uživatel skupiny `sysadmin` může dešifrovat hlavní klíč databáze. V mnoha firmách může být toto rozhodující faktor proti použití této metody.

Automatické zašifrování vašeho hlavního klíče databáze pomocí hlavního klíče služby můžete vypnout pomocí příkazu `ALTER MASTER KEY`, jak můžete vidět v následujícím kódu v jazyce T-SQL:

```
USE AdventureWorks;
GO
ALTER MASTER KEY
 DROP ENCRYPTION BY SERVICE MASTER KEY;
```

Smazání hlavního klíče databáze je stejně jednoduché jako příkaz `DROP`:

```
DROP MASTER KEY;
```

Příkazy `OPEN MASTER KEY` a `CLOSE MASTER KEY` se používají k otevření a zavření hlavního klíče databáze tak, aby jej bylo možné použít k šifrování a dešifrování dalších klíčů a certifikátů, které chrání. Tuto klíče je pak možné použít k šifrování a dešifrování symetrických klíčů a dat. Jak jsme poznamenali, systém SQL Server může implicitně otevírat a zavírat váš hlavní klíč databáze, pokud je zašifrován hlavním klíčem služby. V sekci „Symetrické klíče“ dále v této kapitole popíšeme, jak používat hlavní klíč databáze s automatickým zašifrováním hlavním klíčem služby i bez něj.

Asymetrické klíče

Šifrování systému SQL Server obsahuje podporu pro asymetrické klíče, které se ve skutečnosti skládají ze dvou šifrovacích klíčů: veřejného klíče a privátního klíče. Privátní klíč může mít klíč o délce 512, 1 024 nebo 2 048 bitů, Systém SQL Server poskytuje následující příkazy pro správu asymetrických klíčů:

- `CREATE ASYMMETRIC KEY` Umožňuje vám generovat novou dvojici veřejný a privátní klíč asymetrického klíče, importovat tuto dvojici ze souboru nebo imporotovat veřejný klíč ze sestavení platformy .NET. Tento příkaz vyžaduje oprávnění `CREATE ASYMMETRIC KEY` na danou databázi.
- `ALTER ASYMMETRIC KEY` Umožňuje vám měnit vlastnosti existujícího asymetrického klíče. Pomocí tohoto příkazu můžete odstranit privátní klíč z dvojice veřejný klíč/privátní klíč nebo změnit heslo, které se používá k zašifrování privátního klíče ve dvojici veřejný a privátní klíč. Tento příkaz vyžaduje oprávnění `CONTROL` na asymetrický klíč, pokud z něj odstraňujete privátní klíč.
- `DROP ASYMMETRIC KEY` Smaže asymetrický klíč z databáze. Tento příkaz vyžaduje oprávnění `CONTROL` na asymetrický klíč.

Identifikátory algoritmu a délky klíče, které poskytuje systém SQL Server pro použití v podmínce `WITH ALGORITHM` v příkazech `CREATE` a `ALTER ASYMMETRIC KEY`, jsou vypsány v tabulce 7.2.

Tabulka 7.2: Identifikátory algoritmu a délky klíče pro asymetrické klíče

Identifikátor	Popis
`RSA_512`	512, bitový privátní klíč pro šifrovací algoritmus RSA
`RSA_1024`	1 024, bitový privátní klíč pro šifrovací algoritmus RSA
`RSA_2048`	2 048, bitový privátní klíč pro šifrovací algoritmus RSA

Když vytvoříte asymetrický klíč, jeho privátní klíč je ve výchozím nastavení chráněn pomocí hlavního klíče databáze. Pokud hlavní klíč databáze neexistuje, musíte zadat heslo pro zašifrování privátního klíče v okamžiku jeho vytvoření. Avšak pokud hlavní klíč databáze existuje, je volba `ENCRYPTION BY PASSWORD` v příkazu `CREATE` jen volitelná.

POZNÁMKA

Neexistují žádné příkazy `BACKUP` nebo `RESTORE` pro asymetrické klíče v systému SQL Server 2008. Pokud importujete asymetrický klíč z externího souboru nebo sestavení, nemůže v tom být problém. Avšak pokud používáte systém SQL Server ke generování párů veřejných a privátních klíčů pro asymetrické zašifrování v databázi, doporučujeme vám nahrávat asymetrické klíče z externích zdrojů nebo zvažovat použití certifikátů. Certifikáty, které jsou popsané v následující sekci, mají v jazyce T-SQL příkazy `BACKUP` a `RESTORE`.

Když budete měnit pár asymetrického klíče pomocí příkazu `ALTER ASYMMETRIC KEY`, platí následující pravidla:

- Pokud měníte heslo, které se používá k zašifrování privátního klíče, nebo pokud je privátní klíč chráněn hlavním klíčem databáze a chcete jej změnit tak, aby byl chráněn heslem, je povinná podmínka `ENCRYPTION BY PASSWORD`.
- Pokud je privátní klíč chráněn heslem a vy jej chcete změnit tak, aby byl chráněn hlavním klíčem databáze, nebo měníte heslo, které se používá k zašifrování privátního klíče, je povinná podmínka `DECRYPTION BY PASSWORD`.

Systém SQL Server poskytuje zabudované funkce jazyka T-SQL `EncryptByAsymKey` a `DecryptByAsymKey` k šifrování a dešifrování dat pomocí asymetrických klíčů. Funkce `EncryptByAsymKey`

vyžaduje, abyste zadali identifikátor páru asymetrického klíče, který jste obdrželi, pomocí funkce `AsymKey_ID`. Funkce `AsymKey_ID` přijímá jako parametr název asymetrického klíče a vrací jako výsledek hodnotu identifikátoru v datovém typu `integer`. Dále funkce `EncryptByAsymKey` přijímá prostý text pro zašifrování ve formátu konstanty datového typu `char`, `nchar`, `varchar`, `nvarchar`, `binary` nebo `varbinary`, výrazu, proměnné nebo názvu sloupce (v příkazu jazyka DML). Funkce `EncryptByAsymKey` vrátí výsledek jako typ `varbinary` bez ohledu na to, v jakém datovém typu zadáte prostý text.

Funkce `DecryptByAsymKey` dešifruje data, která jste dříve zašifrovali pomocí funkce `EncryptByAsymKey`. Funkce `DecryptByAsymKey` přijímá identifikátor páru asymetrického klíče, stejně jako funkce `EncryptByAsymKey`. Také přijímá zašifrovaný text typu `varbinary` a volitelné heslo asymetrického klíče, které je vyžadováno, pokud je asymetrický klíč zašifrován heslem. Heslo asymetrického klíče je možné vynechat, pokud je asymetrický klíč zabezpečen pomocí hlavního klíče databáze, ale pokud je použito, musí být typu `nvarchar`.

Zde je příklad použití šifrování a dešifrování pomocí asymetrického klíče:

```
CREATE ASYMMETRIC KEY SampleAsymKey
WITH ALGORITHM = RSA_2048
ENCRYPTION BY PASSWORD = N'B&^19!{f!5h';

DECLARE @plaintext NVARCHAR(58);
DECLARE @ciphertext VARBINARY(256);
-- inicializace prostého textu
SET @plaintext = N'Toto je vzorek textu ve formátu prostého textu';
PRINT @plaintext;
-- šifrovat prostý text
SET @ciphertext = EncryptByAsymKey (AsymKey_ID (N'SampleAsymKey'), @plain-
text);
PRINT @ciphertext;
-- dešifrovat zašifrovanou informaci
SET @plaintext = DecryptByAsymKey (AsymKey_ID (N'SampleAsymKey'), @ciphertext,
 N'B&^19!{f!5h');
PRINT CAST(@plaintext AS NVARCHAR(MAX));

DROP ASYMMETRIC KEY SampleAsymKey;
GO
```

Přestože systém SQL Server 2008 poskytuje šifrovací funkce `EncryptByAsymKey` a `DecryptByAsymKey`, Microsoft doporučuje, abyste použili asymetrické klíče jen k zašifrování symetrických klíčů a tyto symetrické klíče k zašifrování vašich dat. Jedním důvodem pro to je rychlost. Symetrické zašifrování je značně rychlejší než asymetrické zašifrování. Dalším důvodem pro zašifrování dat pomocí symetrického zašifrování je omezení na velikost dat, se kterou si poradí asymetrické zašifrování. Tabulka 7.3 ukazuje omezení asymetrických algoritmů založených na délkách privátních klíčů implementovaných v systému SQL Server 2008.

Tabulka 7.3: Asymetrické algoritmy, délky klíčů a omezení

Algoritmus	Privátní klíč	Prostý text	Zašifrovaný text
RSA_512	512 bitů	53 bajtů	64 bajtů
RSA_1024	1 024 bitů	117 bajtů	128 bajtů
RSA_2048	2 048 bitů	245 bajtů	256 bajtů

Pro asymetrický klíč s privátním klíčem o délce například 1 024 bitů algoritmus `RSA_1024` zašifruje hodnotu typu `varchar` jen o maximální délce 117 znaků, nebo hodnotu typu `nvarchar` s maximální délkou 58 znaků. Díky tomuto omezení je asymetrické šifrování chabou volbou pro data o značné délce. Avšak pokud budete muset zašifrovat dlouhá data asymetricky (pravděpodobně kvůli obchodním požadavkům), můžete toto omezení obejít, například pomocí uživatelem definované funkce, které ukazujeme v následujícím příkladu:

```
USE AdventureWorks;
GO

CREATE FUNCTION dbo.BigAsymEncrypt (@asym_key_name NVARCHAR(128),
@plain_text NVARCHAR(MAX))
RETURNS VARBINARY(MAX)
AS
BEGIN

 -- Spočítej velikost kousku prostého textu
 DECLARE @chunk VARBINARY(512);
 DECLARE @chunksize INT;
 SELECT @chunksize = (key_length / 16) - 11
 FROM sys.asymmetric_keys
 WHERE name = @asym_key_name;

 -- Smaž výsledek šifrování
 DECLARE @result VARBINARY(MAX);
 SET @result = CAST('' AS VARBINARY(MAX));

 -- Procházej prostým textem a zašifruj jej po kouscích
 DECLARE @i INT;
 SET @i = 1;

 WHILE @i < LEN(@plain_text)
 BEGIN
 SET @chunk = EncryptByAsymKey(AsymKey_ID(@asym_key_name),
 SUBSTRING(@plain_text, @i, @chunksize - 11));

 -- Připoj kousky zašifrovaného textu k výsledku
 SET @result = @result + CAST(@chunk AS VARBINARY(MAX));

 -- Posuň pozici čítače o velikost kousku minus 11
```

```
 SET @i = @i + @chunksize - 11;
 END;

 -- Vrať výsledek
 RETURN @result;
END
GO

CREATE FUNCTION dbo.BigAsymDecrypt (@asym_key_name NVARCHAR(128),
 @cipher_text VARBINARY(MAX),
 @password NVARCHAR(256))
RETURNS NVARCHAR(MAX)
AS
BEGIN

 -- Spočítej velikost kousku zašifrovaného textu
 DECLARE @chunksize INT;
 DECLARE @chunk VARBINARY(512);
 SELECT @chunksize = (key_length / 8)
 FROM sys.asymmetric_keys
 WHERE name = @asym_key_name;

 -- Inicializuj výsledek prostého textu
 DECLARE @result NVARCHAR(MAX);
 SET @result = N'';

 -- Procházej zašifrovaným textem a dešifruj jej po kouscích
 DECLARE @i INT;
 SELECT @i = 1;
 WHILE @i < DATALENGTH(@cipher_text)
 BEGIN

 -- Dešifruj zašifrovaný text
 SELECT @chunk = DecryptByAsymKey (AsymKey_ID (@asym_key_name),
 SUBSTRING(@cipher_text, @i, @chunksize), @password);
 -- Připoj prostý kousky prostého k výsledku
 SELECT @result = @result + CAST(@chunk AS NVARCHAR(MAX));

 -- Posuň ukazatel na kousek
 SET @i = @i + @chunksize;
 END;

 -- Vrať výsledek
 RETURN @result;
END
GO
```

Funkce `BigAsymEncrypt` v tomto příkladu rozdělí prostý text typu `nvarchar(max)`, který jí byl předán, a zašifruje jej po kouscích. Velikost kousků prostého textu odpovídá počtu bitů v privátním klíči asymetrického klíče děleno 16 (pokud by byl prostý text typu `varchar` místo `nvarchar`, pak by se dělilo místo toho 8) minus 11 bajtů. 11 bajtů navíc použije nástroj Microsoft Enhanced Cryptographic Provider pro odsazení standardu PKCS #1. Uživatelem definovaná funkce (UDF) provádí smyčku a po každé iteraci zvýší počítadlo o velikost každého kousku. Funkce `BigAsymDecrypt` rozdělí zašifrovaný text a dešifruje jej po kouscích a připojí dešifrované kousky prostého textu k výsledku typu `nvarchar`. Velikost kousku zašifrovaného textu typu `varbinary` se spočítá jako délka privátního klíče asymetrického klíče dělená 8. Zde je příklad těchto uživatelem definovaných funkcí v akci:

```
USE AdventureWorks;
GO

-- Tento kód testuje asymetrické šifrovací funkce
DECLARE @testplain NVARCHAR(MAX);
DECLARE @testcipher VARBINARY(MAX);

-- Definuje prostý text k testování
SET @testplain = N'"Lidská bytost je součástí celku, který my nazýváme' +
 N'''vesmír'', část omezenou v čase a prostoru. On prožívá ' +
 N'své myšlenky a pocity jako něco, co je oddělené od zbytku... ' +
 N'jako druh optického klamu svého svědomí. Tento klam je pro nás ' +
 N'určitým druhem vězení, jenž nás omezuje na naše osobní touhy ' +
 N'a náklonnost k několika nám nejbližším osobám. Naším cílem je ' +
 N'vymanit se z toho vězení tak, že rozšíříme okruh našeho soucitu, ' +
 N'aby zahrnoval všechny živé bytosti a přírodu v celé své ' +
 N'kráse." - Albert Einstein';

-- Vytvoří testovací asymetrický klíč
CREATE ASYMMETRIC KEY SampleAsymKey
WITH ALGORITHM = RSA_2048
ENCRYPTION BY PASSWORD = N'B&^19!{f!5h';

-- Otestuje funkce BigAsymEncrypt a BigAsymDecrypt
PRINT @testplain
SET @testcipher = dbo.BigAsymEncrypt (N'SampleAsymKey', @testplain);
PRINT @testcipher
PRINT dbo.BigAsymDecrypt (N'SampleAsymKey', @testcipher, N'B&^19!{f!5h');

-- Smaže testovací asymetrický klíč
DROP ASYMMETRIC KEY SampleAsymKey;
GO
```

Tento příklad používá asymetrické šifrovací funkce pro zašifrování a dešifrování dlouhých řetězců typu `nvarchar(max)`. Přestože můžete použít tyto metody pro obejití omezení délky textu pro asymetrické zašifrování, symetrické zašifrování je značně rychlejší a obecně by se mělo používat pro zašifrování vašich dat.

Certifikáty

Certifikáty jsou dalším nástrojem, který systém SQL Server poskytuje pro asymetrické šifrování. *Certifikát* je v podstatě pár veřejný a privátní klíč asymetrického klíče, který obsahuje další data popisující daný certifikát.

Další data obsahují počáteční datum, datum vypršení a předmět certifikátu. Na rozdíl od asymetrických klíčů systému SQL Server mohou být certifikáty zálohovány do souborů a také z nich obnoveny. Pokud potřebujete, aby váš systém SQL Server generoval páry veřejného a privátního klíče pro asymetrické šifrování, pak díky schopnosti vytvářet zálohy jsou certifikáty lepší volbou než asymetrické klíče.

Certifikáty jsou podepsané certifikační autoritou, kterou je často důvěryhodná třetí strana, přestože systém SQL Server může generovat certifikáty podepsané sebou samým. Systém SQL Server podporuje certifikáty, které se řídí standardem ITU-T X.509 (ITU-T – International Telecommunication Union Telecommunication Standardization Sector) (standard je dostupný na adrese *http://www.itu.int/ITU-T/index.phtml*).

Systém SQL Server poskytuje následující rozšíření jazyka T-SQL pro správu certifikátů:

- `CREATE CERTIFICATE` Umožňuje vám v systému SQL Server generovat certifikáty podepsané sebou samým, nahrávat certifikáty ze souborů zašifrovaných standardem DER (DER – Distinguished Encoding Rules) nebo vytvářet certifikáty ze souborů dynamicky linkovaných knihoven (DLL – dynamic link libraries) podepsaných certifikáty. Pokud je vynechaná podmínka `ENCRYPTION BY PASSWORD`, systém SQL Server ve výchozím nastavení použije hlavní klíč databáze pro zabezpečení certifikátu.
- `BACKUP CERTIFICATE`: Umožňuje vám exportovat certifikát do souboru. Exportovaný privátní klíč je zašifrován heslem, které zadáte v podmínce `ENCRYPTION BY PASSWORD`. Neexistuje žádný příkaz `RESTORE CERTIFICATE`, pro obnovení zazálohovaného certifikátu použijte příkaz `CREATE CERTIFICATE`.
- `ALTER CERTIFICATE` Umožní vám přidat nebo odebrat privátní klíč z certifikátu, změnit privátní klíč certifikátu nebo zpřístupnit certifikát pro dialogy služby Service Broker.
- `DROP CERTIFICATE`: Smaže existující certifikát. Nelze smazat certifikát, který se aktuálně používá k ochraně symetrických klíčů.

POZNÁMKA

Pro privátní klíče importované ze souborů zašifrovaných standardem DER nebo knihoven DLL podepsaných certifikátem podporuje systém SQL Server certifikáty s délkou privátního klíče od 384 do 3 456 bitů v násobcích 64 bitů. Privátní klíče certifikátů generovaných systémem SQL Server mají délku 1 024 bitů.

Následující příklad demonstruje, jak generovat a zálohovat certifikáty systému SQL Server podepsané sebou samým a jejich privátní klíče (které se zálohují do samostatného souboru).

```
USE AdventureWorks;
GO

CREATE CERTIFICATE SampleCert
 ENCRYPTION BY PASSWORD = N'p$@1k-#tZ'
 WITH SUBJECT = N'Sample Certificate',
```

```
 EXPIRY_DATE = N'10/31/2026';

BACKUP CERTIFICATE SampleCert
 TO FILE = N'c:\MK\BackupSampleCert.cer'
 WITH PRIVATE KEY (
 FILE = N'c:\MK\BackupSampleCert.pvk' ,
 ENCRYPTION BY PASSWORD = N'p@$$w0rd',
 DECRYPTION BY PASSWORD = N'p$@1k-#tZ'
 );

DROP CERTIFICATE SampleCert;
GO
```

Pro obnovení zazálohovaného certifikátu a jeho privátního klíče můžete spustit příkaz `CREATE CERTIFICATE` jako v následujícím kódu:

```
CREATE CERTIFICATE SampleCert
 FROM FILE = N'c:\MK\BackupSampleCert.cer'
 WITH PRIVATE KEY (
 FILE = N'c:\MK\BackupSampleCert.pvk',
 DECRYPTION BY PASSWORD = N'p@$$w0rd',
 ENCRYPTION BY PASSWORD = N'p$@1k-#tZ'
 );
GO
```

Microsoft doporučuje, abyste certifikáty, stejně jako asymetrické klíče, použili pro zašifrování vašich symetrických klíčů a tyto symetrické klíče použili k zašifrování vašich dat. Avšak jazyk T-SQL poskytuje pro zašifrování dat pomocí certifikátu funkce `EncryptByCert` a `DecryptByCert`.

Zašifrování pomocí certifikátů má stejná omezení délky jako asymetrické zašifrování. Maximální délka, prostého textu, jaký můžete předat funkci `EncryptByCert`, je možné spočítat pomocí tohoto vzorce:

Maximální_délka_prostého_textu_v_bajtech = (délka_privátního_klíče_v_bitech/ 8) – 11. Délka zašifrovaného textu, který funkce vrátí, je možné spočíst pomocí následujícího vzorce: *délka_zašifrovaného_textu_v_bajtech = (délka_privátního_klíče_v_bitech/8).*

Funkce `EncryptByCert` i `DecryptByCert` vyžadují identifikátor certifikátu, což je pro daný certifikát číslo v datovém typu `integer`. K získání identifikátoru certifikátu z jeho jména je možné použít funkci `Cert_ID`. Pokud chcete použít funkci `Cert_ID`, předejte jí název certifikátu jako typ `nvarchar` nebo `varchar`.

Funkce `EncryptByCert` přijímá prostý text, který si přejete zašifrovat pomocí certifikátu. Funkce `DecryptByCert` přijímá text, který jste dříve zašifrovali a přejete si jej dešifrovat. Funkce `DecryptByCert` obsahuje třetí (volitelný) parametr, kterým je heslo certifikátu, což je stejné heslo, jaké jste zadali při vytváření certifikátu. Pokud je certifikát zabezpečen pomocí hlavního klíče databáze, tento parametr by měl být při volání funkce `DecryptByCert` vypuštěn.

Následující příklad ukazuje, jak použít funkce `EncryptByCert` a `DecryptByCert` za předpokladu, že certifikát `SampleCert`, který jste vytvořili dříve v této sekci, skutečně existuje v databázi AdventureWorks.

```
USE AdventureWorks;
GO

-- Inicializuj prostý text
DECLARE @plain_text NVARCHAR(58);
SET @plain_text = N'Toto je test!';
PRINT @plain_text;

-- Zašifruj prostý text pomocí certifikátu
DECLARE @cipher_text VARBINARY(128);
SET @cipher_text = EncryptByCert(Cert_ID(N'SampleCert'), @plain_text);
PRINT @cipher_text;

-- Dešifruj prostý text pomocí certifikátu
SET @plain_text = CAST(DecryptByCert(Cert_ID(N'SampleCert'),
@cipher_text, N'p$@1k-#tZ') AS NVARCHAR(58));
PRINT @plain_text;
GO
```

Symetrické klíče

Symetrické klíče jsou vespod hierarchie šifrovacích klíčů systému SQL Server. Symetrický klíč se používá k zašifrování jiných symetrických klíčů nebo dat. Protože zašifrování symetrickými klíči je o mnoho rychlejší než asymetrické zašifrování a netrpí omezením velikosti dat jako implementace asymetrického zašifrování systému SQL Server, Microsoft doporučuje šifrovat data výhradně pomocí symetrických klíčů.

Zatímco asymetrické šifrování vyžaduje dva klíče (pár veřejný a privátní klíč), symetrické zašifrování vyžaduje jediný klíč současně pro zašifrování i dešifrování vašich dat. Symetrické šifrování se provádí pomocí blokových šifrovacích algoritmů, které šifrují vaše data po blocích o konstantní velikosti, a proudových šifrovacích algoritmů, které šifrují vaše data v průběžném proudu. Blokové šifrovací algoritmy mají nastavenou velikost šifrovacího klíče a velikost šifrovaného bloku, jak vidíte v Tabulce 7.4.

Tabulka 7.4: Podporované algoritmy v systému SQL 2008

Algoritmus	Délka uloženého klíče	Skutečná délka klíče	Velikost bloku	Poznámka
`DES`	64 bitů	56 bitů	64 bitů	
`Triple_DES`	128 bitů	112 bitů	64 bitů	
`Triple_DES_3KEY`	128 bitů	112 bitů	64 bitů	Tato volba je dostupná jen pro databázové šifrovací klíče.
`DESX`	192 bitů	184 bitů	64 bitů	

Algoritmus	Délka uloženého klíče	Skutečná délka klíče	Velikost bloku	Poznámka
RC2	128 bitů	128 bitů	64 bitů	
RC4	40 bitů	40 bitů	N/A	Microsoft doporučuje nepoužívat algoritmus RC4 pro zašifrování vašich dat. Algoritmus RC4 je proudová šifra, položka velikost bloku se tak na ni nevztahuje.
RC4_128	128 bitů	128 bitů	N/A	Microsoft doporučuje nepoužívat algoritmus RC4_128 pro zašifrování vašich dat. Algoritmus RC4_128 je proudová šifra, položka velikost bloku se tak na ni nevztahuje.
AES_128	128 bitů	128 bitů	128 bitů	
AES_192	192 bitů	192 bitů	128 bitů	

Může spočíst délku zašifrovaného textu v závislosti na délce prostého textu pomocí jednoho z následujících vzorců:

Pro 8bajtové blokové šifry, jakými je rodina algoritmů DES (Data Encryption Standard), použijte vzorec *délka zašifrovaného textu = 8 * ((délka prostého textu + 8) / 8) + 36*.

Pro 16bajtové blokové šifry, jakými je rodina algoritmů AES (Advanced Encryption Standard), použijte vzorec *délka zašifrovaného textu = 16 * ((délka prostého textu + 16) / 16) + 44*.

Pokud používáte ověřovatele, přidejte v obou případech 20 bajtů k celkové délce.

RODINA ALGORITMŮ DES

Jak trojitý algoritmus DES, tak i DESX jsou vylepšením původního algoritmu DES, který byl autorizován pro použití na všechna neklasifikovaná data Národním bezpečnostním úřadem (NSA – National Security Agency) v roce 1976. Jak trojitý algoritmus DES, tak DESX byly představeny, aby s minimálními změnami posílily algoritmus DES, protože do roku 1994 byla nalezena nejrůznější slabá místa.

Algoritmus DES používá 56bitový klíč (ačkoli systém SQL Server používá k jeho uložení 64 bitů).

Trojitý algoritmus DES používá dva 56bitové klíče pro klíč o celkové délce 112 bitů (systém SQL Server používá k jeho ukládání 128 bitů). Šifrovací algoritmus Triple DES šifruje data pomocí jednoho 56bitového klíče, provede operaci dešifrování na datech pomocí druhého klíče a opět zašifruje data pomocí prvního klíče. Celkové efektivní zabezpečení trojitého algoritmu DES je změřeno asi na 80 bitů.

Algoritmus DESX provede exkluzivní operaci OR (XOR) na úrovni bitů s dalšími 64 bity klíče navíc před provedením zašifrování pomocí algoritmu DES a provede navíc další bitovou operaci XOR s dalšími 64 bity klíče navíc po zašifrování. Díky tomu je celková délka klíče pro algoritmus DESX 184 bitů, přestože zabezpečení poskytované algoritmem DESX je zhruba ekvivalentní klíči s délkou jen 118 bitů. Systém SQL Server používá 192 bitů k uložení 184 bitů materiálu klíče.

Systém SQL Server poskytuje následující příkazy pro správu symetrických klíčů:

- `CREATE SYMMETRIC KEY` Vytvoří symetrický klíč pro použití při zašifrování. Symetrické klíče mohou být zašifrovány pomocí certifikátů, asymetrických klíčů, hesel, nebo dokonce dalších symetrických klíčů.
- `ALTER SYMMETRIC KEY` Umožňuje vám změnit metodu zabezpečení vašich symetrických klíčů.
- `DROP SYMMETRIC KEY` Smaže symetrický klíč z databáze. Symetrické klíče nelze smazat, dokud jsou otevřené.
- `OPEN SYMMETRIC KEY` Otevře a dešifruje symetrický klíč, aby jej bylo možné použít.
- `CLOSE SYMMETRIC KEY` Zavře symetrický klíč, který byl dříve otevřen.
- `CLOSE ALL SYMMETRIC KEYS` Zavře všechny současně otevřené symetrické klíče v aktuálním připojení.

Systém SQL Server neobsahuje příkazy k zálohování nebo obnovení pro symetrické klíče. Protože symetrické klíče jsou uloženy v aktuální databázi, zálohují se v průběhu normálního procesu zálohování databáze. Můžete také znovu vytvořit symetrický klíč z ničeho pomocí příkazu `CREATE SYMMETRIC KEY`. Pokud chcete znovu vytvořit symetrický klíč z ničeho, musíte zadat hodnoty `KEY_SOURCE` a `IDENTITY_VALUE`. Hodnotu `KEY_SOURCE` systém SQL Server zpracuje algoritmem hash a provede na ní bitové manipulace, aby vygeneroval symetrický šifrovací klíč. Pokud není zadána, systém SQL Server náhodně vygeneruje hodnotu `KEY_SOURCE`. Hodnotu `IDENTITY_VALUE` systém SQL Server použije pro vygenerování identifikátoru GUID klíče. Kopie identifikátoru GUID se uloží s daty, na jejichž zašifrování se klíč použije. Pokud chcete znovu vytvořit symetrický klíč, musíte zadat stejné hodnoty `KEY_SOURCE` a `IDENTITY_VALUE`, jaké jste zadali původně při vytváření klíče. Systém SQL Server garantuje, že při poskytnutí stejných hodnot `IDENTITY_VALUE` a `KEY_SOURCE` se vygeneruje identický klíč.

Následující příklad vytvoří symetrický klíč a pak jej smaže:

```
CREATE SYMMETRIC KEY SymTest
WITH ALGORITHM = Triple_DES
ENCRYPTION BY PASSWORD = '$#ad%61*(;dsPSlk';

DROP SYMMETRIC KEY SymTest;
```

TIP

Můžete také vytvořit dočasné symetrické klíče tak, že před název symetrického klíče umístíte znak # v příkazu `CREATE SYMMETRIC KEY`. Dočasný symetrický klíč je dostupný jen pro aktuální připojení a automaticky se smaže, když se připojení ukončí.

Samozřejmě není moc užitečné vytvářet symetrický klíč, pokud jej nemůžete použít k zašifrování informací. Ale jak jsme uvedli, symetrické klíče v systému SQL Server je možné použít k ochraně dalších symetrických klíčů nebo dat. Symetrický klíč ochráníte jiným symetrickým klíčem tak, že použijete podmínku `ENCRYPTION BY SYMMETRIC KEY` v příkazu `CREATE SYMMETRIC KEY`. Pro šifrování a dešifrování dat použijte funkce `EncryptByKey` a `DecryptByKey`. Následující příklad vytváří symetrický klíč, který se použije k zašifrování dalšího symetrického klíče, který následně použijí funkce `EncryptByKey` a `DecryptByKey` k zašifrování a dešifrování nějakých dat.

```
USE AdventureWorks;
GO

-- Vytvoří symetrický klíč pro zašifrování symetrického klíče
CREATE SYMMETRIC KEY SymKey
WITH ALGORITHM = Triple_DES
ENCRYPTION BY PASSWORD = '$#ad%61*(;dsPSlk';

-- Otevře klíč pro zašifrovaní jiného klíče
OPEN SYMMETRIC KEY SymKey
DECRYPTION BY PASSWORD = '$#ad%61*(;dsPSlk';

-- Vytvoří symetrický klíč pro zašifrování dat
CREATE SYMMETRIC KEY SymData
WITH ALGORITHM = Triple_DES
ENCRYPTION BY SYMMETRIC KEY SymKey;

-- Otevře klíč pro zašifrování dat
OPEN SYMMETRIC KEY SymData
DECRYPTION BY SYMMETRIC KEY SymKey;

-- Inicializuje prostý text
DECLARE @plain_text NVARCHAR(512);
SET @plain_text = N'"Ti, kdo se ve jménu bezpečnosti vzdávají svobody, ' +
N'nezaslouží si ani svobodu ani bezpečnost." - Benjamin Franklin'
PRINT @plain_text;

-- Zašifruje data
DECLARE @cipher_text VARBINARY(1024);
SET @cipher_text = EncryptByKey(Key_GUID(N'SymData'), @plain_text);
PRINT @cipher_text;

-- Dešifruje data
SET @plain_text = CAST(DecryptByKey(@cipher_text) AS NVARCHAR(512));
PRINT @plain_text;

-- Zavře klíč pro zašifrování dat
CLOSE SYMMETRIC KEY SymData;

-- Zavře klíč zašifrovaný jiným klíčem
CLOSE SYMMETRIC KEY SymKey;

-- Smaže symetrické klíče
DROP SYMMETRIC KEY SymData;
DROP SYMMETRIC KEY SymKey;
```

POZNÁMKA

Nejprve musíte otevřít váš symetrický klíč a teprve poté jím můžete ochránit jiný klíč nebo s jeho pomocí zašifrovat data.

Funkce `EncryptByKey` vyžaduje identifikátor GUID symetrického klíče, který používáte k zašifrování vašich dat. Identifikátor GUID symetrického klíče je možné získat tak, že předáte název klíče funkci `Key_GUID`. Proměnná prostý_text , která se předává této funkci, je typu `char`, `varchar`, `nchar`, `nvarchar`, `binary` nebo `varbinary`. Návratová hodnota funkce `EncryptByKey` je typu `varbinary(8000)`. Blokové šifry v systému SQL Server, jako například trojitý algoritmus DES a algoritmus AES, automaticky používají šifrovací režim známý jako řetězení šifrovacích bloků (CBC – Cipher Block Chaining) a náhodné iniciační vektory (IV – initialization vectors), aby dále zatemnil vaše data.

Funkce `EncryptByKey` navíc také přijímá volitelný parametr authenticator value (hodnota ověřitele), aby pomohl zabránit substituci celých zašifrovaných hodnot ve vašich datech. Hodnota ověřitele je hodnota typu `sysname`, který je synonymem datového typu `nvarchar(128)`. Pokud poskytnete hodnotu ověřitele, zašifruje se společně s prostým textem a tím ještě více zatemní vaše data. Hodnotu ověřitele je možné použít k „připoutání" vašich zašifrovaných dat k určitému záznamu. Pokud použijete nějakého ověřitele, pak musíte nastavit parametr `add_authenticator` pro funkci `EncryptByKey` na hodnotu 1.

Funkce `DecryptByKey` přijímá zašifrovaná data jako typ `varbinary(8000)` a vrací dešifrovaný prostý text jako typ `varbinary(8000)`. Pokud byla vaše data původně typu `varchar` nebo `nvarchar`, pak k převedení výsledku do původních datových typů budete potřebovat příkazy `CAST` nebo `CONVERT`. Pokud jste použili hodnotu ověřitele, když jste šifrovali prostý text, pak musíte zadat stejnou hodnotu ověřitele pro dešifrování vašeho zašifrovaného textu. Všimněte si, že nemusíte zadávat parametr `Key_GUID`, když voláte funkci `DecryptByKey`. To je proto, že systém SQL Server uchovává identifikátor GUID se zašifrovanými daty v průběhu procesu dešifrování.

TECHNOLOGIE CBC, VÝPLNĚ A NÁHODNÝ INICIALIZAČNÍ VEKTOR: VŠE DOHROMADY

Systém SQL Server poskytuje určité šifrovací funkce, které nadále zatemňují váš zašifrovaný text. Technologie CBC maskuje každý blok prostého textu s dříve zašifrovaným blokem zašifrovaného textu před tím, než jej zašifruje. Díky tomu závisí všechny bloky zašifrovaného textu na všech předchozích blocích, vaše data jsou tak ještě více citlivá na nejjemnější změny nebo poškození a ještě více dochází k mixování vašeho zašifrovaného textu.

Protože první blok textu nemá svůj „předchozí blok", se kterým by mohl být maskován, systém SQL Server vygeneruje náhodný inicializační vektor. Náhodný inicializační vektor se použije pro maskování prvního bloku textu před zašifrováním. Náhodný inicializační vektor pomáhá dále zatemnit váš zašifrovaný text, ale také vám zabraňuje, abyste vytvořili použitelný index na vašich zašifrovaných datech. Protože se inicializační vektor generuje náhodně, zašifrování stejných dat podruhé se stejným klíčem nevygeneruje stejný výsledek. Zatímco náhodný inicializační vektor pomáhá lépe ochránit vaše data, výrazně snižuje účinnost vyhledávání v datech.

Blokové šifrovací algoritmy vyprodukují zašifrovaný text o délce, která je násobkem délky velikosti bloku. Pro 64bitové blokové algoritmy, jako například algoritmus DES, zašifrovaný text musí být násobkem 8 bajtů. 128bitové algoritmy, jakým je algoritmus AES, produkují zašifrovaný text, jehož délka je násobkem 16 bajtů. Problémem je, že ne každý text, který si přejete zašifrovat, bude mít velikost v násobcích délky šifrovacího bloku. Systém SQL Server automaticky doplní váš prostý text blokem cifer před tím, než jej zašifruje, čímž zajistí, že délka šifrovaného textu vždycky bude násobkem velikosti zašifrovaného bloku.

Navíc kromě doplnění vašeho prostého textu před jeho zašifrováním systém SQL Server k němu uloží další metadata, jako jsou následující data:

- Prvních 16 bajtů je identifikátor GUID symetrického klíče, který byl použit pro zašifrování daných dat.
- Další 4 bajty představují číslo verze. Systém SQL Server 2008 má napevno zadáno 01000000.
- Dalších 8 bajtů pro 64bitové šifry jako algoritmus DES (16 bajtů pro šifry jako algoritmus AES) jsou náhodně vygenerované inicializační vektory.
- Dalších 8 bajtů obsahuje různé použité volby pro zašifrování dat. Pokud bylo použito nastavení ověřitele, pak je přidána také 20bajtová hodnota algoritmu SHA-1.
- Zbytek zašifrovaného textu jsou samotná zašifrovaná data, která jsou doplněna tak, aby odpovídala velikosti bloků šifry.

Když použijete funkce `EncryptByKey` a `DecryptByKey` a symetrické klíče, které používáte pro zašifrování a dešifrování dat jsou chráněny dalším klíčem, musíte explicitně otevřít symetrický klíč pomocí příkazu `OPEN SYMMETRIC KEY`. Systém SQL Server poskytuje další funkce, které automaticky otevírají a dešifrují váš symetrický klíč před tím, než dešifrujete vaše data:

- `DecryptByKeyAutoAsymKey` Dešifruje vaše data pomocí symetrického klíče, který je chráněn asymetrickým klíčem. Tato funkce automaticky otevře a dešifruje váš symetrický klíč jeho přiřazeným asymetrickým klíčem.
- `DecryptByKeyAutoCert` Dešifruje vaše data pomocí symetrického klíče, který je chráněn certifikátem. Tato funkce automaticky otevře a dešifruje váš symetrický klíč jeho přiřazeným certifikátem.

Klíče jsou k dispozici všem uživatelům, kterým byl udělen přístup k systému SQL Server v daném čase (na rozdíl od dočasných symetrických klíčů, které jsme zmínili dříve v této sekci). Každé uživatelské připojení může otevírat a zavírat klíče nezávisle na všech dalších připojeních. Pokud například uživatelé Joe a Lisa otevřeli připojení k systému SQL Server ve stejnou dobu, oba mohou současně používat stejný symetrický klíč. Pokud uživatel Joe zavře symetrický klíč ve svém připojení, nebude to mít žádný dopad na symetrický klíč v připojení uživatele Lisa.

POZNÁMKA

Všechny otevřené klíče v připojení se po jeho ukončení automaticky zavřou.

Transparentní šifrování dat

Systém SQL Server 2008 představuje novou možnost šifrování známou jako transparentní šifrování dat (TDE – Transparent Data Encryption). Transparentní šifrování dat šifruje každou stránku vaší celé databáze a automaticky dle potřeby dešifruje každou stránku během přístupu. Tato funkce vám umožňuje zabezpečit celou vaši databázi, aniž byste se starali o podrobnosti zašifrování na úrovni sloupců. Výhoda této funkce je v tom, že vám umožní zabezpečit transparentně vaši databázi bez jakýchkoli změn v koncových aplikacích. Transparentní zašifrování dat nevyžaduje žádné místo navíc a může generovat daleko efektivnější plány dotazů než dotazy na zašifrovaná data, protože transparentní zašifrování dat umožňuje systému SQL Server používat řádné indexy. Slabou stránkou transparentního zašifrování dat je skutečnost, že vyžaduje

další režii, protože systém SQL Server musí dešifrovat každou stránku dat při každém dotazu. Účinky zašifrování na výkon vyhledávání se budeme zabývat v sekci „účinnost dotazů“ později v této kapitole.

Povolení transparentního šifrování dat

Pokud chcete povolit transparentní šifrování dat a zašifrovat databázi, musíte nejprve vytvořit hlavní klíč databáze a certifikát serveru v databázi `master` dle následujícího postupu:

```
USE master;
GO

CREATE MASTER KEY
 ENCRYPTION BY PASSWORD = 'p@$$w0rd';
GO

CREATE CERTIFICATE AdvWorksCert
 WITH SUBJECT = 'Certificate to Encrypt AdventureWorks DB',
 EXPIRY_DATE = '2022-12-31';
GO
```

Všimněte si, že příkazy pro vytvoření hlavního klíče a certifikátu jsou stejné jako v jakékoli jiné instanci, kromě toho, že jsou vytvořeny v databázi master.

UPOZORNĚNÍ

Jakmile vytvoříte certifikát serveru, je kritické jej zazálohovat a zálohu uložit na bezpečném místě. Pokud budete někdy potřebovat obnovit databázi zašifrovanou pomocí transparentního zašifrování dat a ztratíte certifikát serveru, utrpíte ztrátu dat. Mít zálohu certifikátu serveru je kritické, abyste zabránili ztrátě dat a přerušení obchodních operací!

Dalším krokem je vytvoření šifrovacího klíče databáze a povolení šifrování v databázi, kterou chcete zabezpečit. Následující příklad povoluje transparentní šifrování dat v databázi AdventureWorks.

```
USE AdventureWorks;
GO

CREATE DATABASE ENCRYPTION KEY
 WITH ALGORITHM = TRIPLE_DES_3KEY
 ENCRYPTION BY SERVER CERTIFICATE AdvWorksCert;
GO

ALTER DATABASE AdventureWorks
 SET ENCRYPTION ON;
GO
```

Nový příkaz `CREATE DATABASE ENCRYPTION KEY` vyváří šifrovací klíč databáze pro databázi AdventureWorks. Pomocí tohoto příkazu můžete určit, že šifrovací klíč databáze je zabezpečen pomocí certifikátu serveru `AdvWorksCert`, který jsme vytvořili dříve a určili jsme, že se

použije trojitý algoritmus DES se třemi klíči. Algoritmy dostupné pro příkaz `CREATE DATABASE ENCRYPTION KEY` jsou omezené na `TRIPLE_DES_3KEY`, `AES_128`, `AES_192` a `AES_256`.

Navíc kromě příkazu `CREATE DATABASE ENCRYPTION KEY` jazyk T-SQL poskytuje ekvivalentní příkazy `DROP` a `ALTER` pro správu šifrovacích klíčů databáze. Neexistuje žádný příkaz pro zálohu a systém SQL Server neposkytuje žádný způsob, jak exportovat šifrovací klíč databáze z vaší databáze.

Abyste správně zabezpečili vaši databázi, musí transparentní šifrování dat provést některé další kroky hned po zapnutí. Jeden z kroků, který transparentní zašifrování dat provádí, je vynulování a zašifrování protokolu transakcí. Tímto způsobem nebude hacker s programem pro čtení protokolu schopen zrekonstruovat vaše citlivá data z těchto protokolů. Transparentní zašifrování dat také zašifruje databázi tempdb. To zabraňuje hackerům v získání přístupu k citlivým informacím uloženým dočasně během zpracování. Skutečnost, že transparentní šifrování dat zašifruje i databázi tempdb, může zpomalit zpracování v jiných databázích na tomtéž serveru, můžete však vylepšit výkon tím, že umístíte zašifrované databáze v jiné instanci systému SQL Server odděleně od nešifrovaných databází.

Volba mezi transparentním šifrováním dat a šifrováním na úrovni sloupců

Transparentní šifrování dat nabízí výhodu, že je neviditelné pro vaše klientské aplikace a provádí dokonalou práci při zabezpečení dat. Nevýhodou transparentního šifrování dat je to, že s sebou přináší režii v průběhu každého přístupu k databázi, protože kompletně celé stránky dat je potřeba dešifrovat při každém přístupu k databázi. Transparentní šifrování dat také šifruje databázi tempdb, což může negativně ovlivnit výkon každé další databáze na stejné instanci systému SQL Server.

Zašifrování na úrovni sloupců má další výhodu v tom, že poskytuje naprostou přesnost a další flexibilitu při zabezpečení vašich dat. Hlavními nevýhodami šifrování na úrovni sloupců je to, že vyžaduje další programování, možná budete tedy muset změnit existující klientské aplikace a nelze efektivně využívat indexy na zašifrovaných datech pro optimalizaci dotazů (viz sekce „Efektivita dotazů“ dále v této kapitole).

Pokud máte tedy dostupné tyto možnosti zašifrování dat systému SQL Server, pro kterou z nich se rozhodnete a kdy? Většina skutečných scénářů pokrývá širokou škálu možností: Následuje několik možných případů a souvisejících úvah při rozhodování, kterou metodu použít:

- Jen relativně malá část vašich dat vyžaduje zašifrování. V tomto případě je smysluplné vyhnout se režii způsobené transparentním šifrováním dat a dát přednost více cílenému přístupu v podobě šifrování na úrovni sloupců. Avšak pokud se má zašifrovat většina vašich dat, je vhodná metoda transparentního šifrování dat.
- Musíte vyhledávat v zašifrovaných sloupcích. Mějte na paměti, že účinnost vyhledávání a zabezpečení jsou protikladné cíle a měli byste provést cokoli, abyste se vyhnuli návrhu systému, který musí vyhledávat v zašifrovaných sloupcích. Pokud absolutně musíte vyhledávat v zašifrovaných sloupcích, transparentní šifrování dat poskytne daleko vyšší účinnost, protože zašifrování dat na úrovni sloupců vždy vyústí v sekvenční procházení záznamů celé tabulky.
- Regulační požadavky diktují podmínky. V některých případech zákony nebo podzákonné předpisy diktují, jaká část vašich dat musí být zašifrována. Obvykle se tato nařízení vztahují na důvěrné informace o zákazníkovi, kreditních kartách a zdravotní informace. Můžete mít také jiné požadavky v závislosti na průmyslovém oboru, které

nařizují další ochranu citlivých dat. Avšak častěji tato nařízení budou uvádět, že „důvěrná data" musí být zašifrována, a ponechají na vás, abyste rozhodli, na jak velké množství dat se tyto požadavky vztahují.

- Povinnosti plynoucí ze smlouvy uvádějí požadavky. Když uzavíráte smlouvu s kreditní společností, kreditními kancelářemi a vydavateli jakýchkoli druhů kreditních karet, požadavky na zabezpečení, které musí vaše společnost implementovat, jsou často uvedeny ve smlouvě až do n-tého stupně. Tyto smouvy mohou určovat několik klíčových položek, jako například která data se musí zašifrovat, který algoritmus se musí použít pro zašifrování a dešifrování dat a požadavky na protokolování změn dat nebo dotazů.
- Podporujete klientskou aplikaci, kterou nelze změnit. Není neobvyklé, když manažeři v oboru informačních technologií a správci databází podporují zděděné starší databázové a klientské aplikace. Pokud byl zdrojový kód této apliakce dávno ztracen nebo nemáte čas nebo prostředky pro modifikaci existující aplikace, má smysl použít transparentní šifrování dat, protože šifrování, které poskytuje, je transparentní pro klientské aplikace.
- Potřebujete další flexibilitu v průběhu šifrování. V některých případech budete potřebovat další flexibilitu, jako například když budete mít pevně dané požadavky na zašifrování vašich symetrických klíčů pomocí asymetrických klíčů místo certifikátu nebo potřebujete zadat ověřitele v průběhu šifrování, nebo dokonce budete potřebovat zašifrovat data pomocí přístupového hesla místo klíče. V těchto případech možná budete muset použít zašifrování na úrovni sloupců místo transparentního šifrování dat. Tento případ není převědčivý jako většina ostatních a pravděpodobně uděláte lépe, pokud si znovu projdete zadání, pokud je to určující faktor.

Jakmile se správcové databází, vývojáři a obzvláště manažeři dozví o transparentním zašifrování dat, očekáváme, že bude mnoho lidí, kteří se vydají snadnou cestou „zašifruj celou databázi a zapomeň". To je nešťastné, protože v mnoha situacích bude tento plán značně přestřelený a bude nutit server k vykonávání mnoha zbytečných zašifrování a dešifrování, což bude vázat prostředky serveru, které by mohly být využity jinde. Velmi doporučujeme, abyste pořádně zvážili klady a zápory transparentního šifrování dat a zašifrování na úrovni sloupců dříve, než se rozhodnete, jakou strategii šifrování databáze použijete pro vaši databázi.

Rozšiřitelná správa klíčů

Systém SQL Server 2008 navíc poskytuje kromě transparentního šifrování dat také několik funkcí známých jako rozšiřitelná správa klíčů (EKM – Extensible Key Management). Rozšiřitelná správa klíčů vám umožňuje použít aplikační rozhraní Microsoft Cryptographic API (CryptoAPI) pro zašifrování a generování klíčů.

Podpora rozšiřitelné správy klíčů je navržená tak, aby umožňovala prodejcům třetích stran poskytovat hardware pro generování šifrovacích klíčů a další nástroje známé jako moduly hardwarového zabezpečení (HSM – Hardware Security Module). Modul hardwarového zabezpečení může nabídnout mnoho výhod oproti standardním zabudovaným šifrovacím funkcím, jako je hardwarem urychlené šifrování a dešifrování nebo další správu klíčů. Modul hardwarového zabezpečení může být karta smart card, klíčenka USB, zařízení flash nebo specializovaný externí modul.

Příkazy jazyka T-SQL pro šifrování v DML (Data Manipulation Language), jako například příkaz `CREATE SYMMETRIC KEY`, nyní obsahují podmínku `FROM PROVIDER` a volbu `CREATE_DISPOSITION`, aby bylo možné poskytnout podporu pro rozšiřitelnou správu klíčů.

Funkce rozšiřitelné správy klíčů a modulů hardwarového zabezpečení jsou závislé na výrobci, a váš výrobce modulu hardwarového zabezpečení vám poskytne specifické instrukce při implementaci řešení EKM/HSM ve vašem prostředí.

Šifrování bez klíčů

Kromě použití certifikátů, asymetrických klíčů a symetrických klíčů můžete zašifrovat vaše data pomocí heslových frází. *Heslová fráze* je řetězec nebo binární hodnota, ze které může systém SQL Server odvodit symetrický klíč pro zašifrování vašich dat.

Funkce `EncryptByPassPhrase` a `DecryptByPassPhrase` tento typ zašifrování umožňují, jak ukazuje následující příklad:

```
DECLARE @plain_text nvarchar(1000),
  @enc_text varbinary(2000);
SET @plain_text = N'Kdo chvíli stál, již stojí opodál...';
SET @enc_text = EncryptByPassPhrase(N'E Pluribus Unum', @plain_text);
SELECT 'Originální prostý text = ', @plain_text;
SELECT 'Zašifrovaný text = ', @enc_text;
SELECT 'Dešifrovaný prostý text = ',
  CAST(DecryptByPassPhrase(N'E Pluribus Unum', @enc_text) AS nvarchar(1000));
```

Funkce `EncryptByPassPhrase` přijímá prostý text, který chcete zašifrovat. Na druhé straně funkce `DecryptByPassPhrase` přijímá dříve zašifrovaný text, který dešifruje. Pro obě funkce můžete přidat hodnotu ověřitele, abyste dále zatemnili váš zašifrovaný text, jak ukazuje následující kód:

```
SET @enc_text = EncryptByPassPhrase(N'E Pluribus Unum', @plain_text,
1, N'Authentic');
```

Obě funkce vrací hodnotu datového typu `varbinary(8000)`. Po použití funkce `DecryptByPassPhrase` budete muset převést výsledek zpět do datového typu `varchar` nebo `nvarchar`, jak bylo uvedeno v předchozím příkladě.

POZNÁMKA

`EncryptByPassPhrase` a `DecryptByPassPhrase` používají trojitý algoritmus DES pro šifrování a dešifrování dat. V těchto funkcích nelze vybrat jiný algoritmus pro šifrování a dešifrování.

Otisky dat a podpisy

Před systémem SQL Server 2005 obsahoval jazyk T-SQL jen několik velmi jednoduchých základních hashovacích funkcí: `CHECKSUM` a `BINARY_CHECKSUM`. Žádná z těchto funkcí není bez kolizí a obě vrací 32bitovou hodnotu otisku, která je o hodně kratší než minimální délka pro zabezpečené aplikace doporučené kryptografickými experty.

Funkce `HashBytes`, která byla představena v systému SQL Server 2005, přijímá název algoritmu hashování a vlastní vstupní řetězec:

```
SELECT HashBytes ('SHA1', 'Jak lvové bijem o mříže, jak lvové v kleci jatí...');
```

Algoritmus použitý v tomto případě je algoritmus SHA-1. Pro tento parametr můžete použít algoritmy MD2, MD4, MD5, SHA nebo SHA-1. První tři jsou algoritmy Message Digest, které ze vstupu generují 128bitové hodnoty otisku. Poslední dvě hodnoty jsou algoritmy Secure Hash Algorithm, které generují 160bitový otisk vstupu.

Vstupní hodnoty pro funkci `HashBytes` jsou typu `varchar`, `nvarchar` nebo `varbinary`. Výsledek funkce `HashBytes` je vždy hodnota typu `varbinary` s maximální délkou 8 000 bajtů.

UPOZORNĚNÍ

Kryptografové doporučují vyhnout se algoritmům hashování MD2, MD4 a MD5 pro zabezpečené aplikace, protože v nich byla objevena slabá místa, která mohou odkrýt vaše zabezpečená data

Systém SQL Server také poskytuje funkce pro podepsání dat pomocí certifikátů a asymetrických klíčů a pro ověření těchto podpisů. To je užitečné pro ochranu integrity citlivých dat, protože jakákoli drobná změna dat ovlivní podpis. Funkce `SignByCert` a `SignByAsymKey` podepisují vaše data pomocí certifikátu nebo asymetrického klíče a vrací podpis jako hodnotu typu `varbinary`. Délka podpisu závisí na délce certifikátu nebo privátního klíče v asymetrickém klíči. 2 048bitový privátní klíč generuje 256bajtový podpis, 1 024bitový privátní klíč generuje 128bajtový podpis a tak dále. Syntaxe pro funkce `SignByCert` a `SignByAsymKey` jsou následující:

```
SignByCert ( ID_certifikátu, prostý_text, heslo)
SignByAsymKey ( ID_asymetrického_klíče, prostý_text, heslo )
```

Funkce `SignByCert` přijímá hodnotu identifikátoru certifikátu, kterou je možné obdržet pomocí funkce `Cert_ID`. Funkce `SignByAsymKey` přijímá identifikátor asymetrického klíče, který se získá pomocí funkce `AsymKey_ID`. Parametr prostý_text je pro obě funkce prostý text, který je třeba podepsat — hodnota typu `char`, `varchar`, `nchar` nebo `nvarchar`. Parametr heslo je heslo požadované k dešifrování certifikátu nebo asymetrického klíče, pokud je chráněn heslem.

Pomocí funkcí `VerifySignedByCert` a `VerifySignedByAsymKey` můžete ověřovat dříve podepsaná data, tyto funkce mají následující syntaxi:

```
VerifySignedByCert ( ID_certifikátu, prostý_text, podpis )
VerifySignedByAsymKey ( ID_asymetrického_klíče, prostý_text, podpis )
```

Funkce `VerifySignedByCert` a `VerifySignedByAsymKey` přijímají identifikátor certifikátu, respektive asymetrického klíče. Parametr prostý_text u obou funkcí je prostý text, který byl dříve podepsán, a parametr podpis je podpis typu `varbinary`, který byl vygenerován. Tyto dvě funkce vygenerují podpis pro hodnotu prostý_text a porovnají ji s hodnotou podpis, kterou jste jim předali. Obě funkce vrátí hodnotu 1, pokud data odpovídají hodnotě podpis, nebo hodnotu 0, pokud neodpovídají.

Katalogové pohledy zabezpečení

Systém SQL Server 2008 poskytuje několik katalogových pohledů zabezpečení a pohledů dynamické správy, které je možné použít k získání informací o šifrovacích funkcích. V systému SQL Server 2008 jsou dostupné následující pohledy:

- `sys.asymmetric_keys` Tento katalogový pohled vrací informace o nainstalovaných párech asymetrických klíčů v aktuální databázi. Informace vrácená tímto pohledem obsahuje název a identifikátor asymetrického klíče, typ zašifrování privátního klíče, použitý šifrovací algoritmus, veřejný klíč a další informace o každém nainstalovaném páru asymetrických klíčů.
- `sys.certificates` Tento katalogový pohled vrací informace o nainstalovaných certfikátech v aktuální databázi. Informace vrácená tímto pohledem je podobná té, kterou vrátí pohled `sys.asymmetric_keys`. Obsahuje název certifikátu, identifikátor certifikátu, typ zašifrování privátního klíče, název vydavatele certifikátu, sériové číslo certifikátu a další specifické informace o certifikátu (jako třeba předmět, počáteční datum a datum vypršení).
- `sys.crypt_properties` Tento katalogový pohled vrátí záznam pro každou kryptografickou vlastnost spojenou se zabezpečeným objektem v databázi. Informace vrácená o každém zabezpečeném objektu obsahuje třídu zabezpečeného objektu, identifikátor zabezpečeného objektu, použitý typ šifrování a hodnotu otisku SHA-1 certifikátu nebo asymetrického klíče, který byl použit pro zašifrování zabezpečeného objektu.

POZNÁMKA

Zabezpečené objekty v systému SQL Server 2008 jsou prostředky a objekty, pro které databázový stroj systému SQL Server reguluje oprávnění a přístup. Zabezpečené objekty se dělí do tří oborů, ke kterým může systém SQL Server regulovat přístup: server, databáze a schéma. Obor serveru obsahuje zabezpečené objekty jako koncové body, přihlašovací jména a databáze. Obor databáze obsahuje uživatele, role, certifikáty, páry asymetrických klíčů, symetrické klíče, schémata a další zabezpečené objekty v oboru databáze. Obor schémat obsahuje tabulky, pohledy, funkce, omezení a další objekty. Ne všechny zabezpečené objekty mají kryptografické vlastnosti, ale katalogový pohled zabezpečení sys.crypt_properties vrátí informace o těch, které je mají.

- `sys.dm_database_encryption_keys` Pohled dynamické správy vrátí informace o stavu transparentního šifrování databáze a šifrovací klíče použité v dané databázi. Vrácené hodnoty ve sloupci `encryption_state` jsou 0, pokud není přítomno žádné zašifrování, 1, pokud není databáze zašifrovaná, 3, pokud je databáze zašifrovaná, nebo další hodnoty, které indikují, že šifrování nebo dešifrování databáze v současnosti probíhá.
- `sys.key_encryptions` Tento katalogový pohled vrátí záznam pro každé zašifrování klíče, jak je určeno podmínkou `ENCRYPTION BY` v příkazu `CREATE SYMMETRIC KEY`. Vrácená informace obsahuje identifikátor zašifrovaného klíče, typ zašifrování, miniaturu certifikátu nebo symetrického klíče, který byl použitý k zašifrování klíče. Miniatura ve smyslu katalogových pohledů zabezpečení systému SQL Server 2008 je hodnota otisku SHA-1 certifikátu nebo asymetrického klíče, nebo identifikátor GUID symetrického klíče. Několik katalogových pohledů zabezpečení vrací miniaturu certifikátů, asymetrických klíčů nebo symetrických klíčů.
- `sys.master_key_passwords`: Tento katalogový pohled vrací záznam pro každé heslo hlavního klíče databáze, které bylo přidáno pomocí uložené systémové procedury `sp_control_dbmasterkey_password`. Každý záznam vrací identifikátor pověření, ke kterému heslo náleží, a identifikátor GUID původní databáze v čase vytvoření. Systém SQL Server používá identifikátor GUID pro identifikaci pověření, které mohou obsaho-

vat hesla, které chrání hlavní klíč databáze v případě, že selže automatické dešifrování. Hesla použitá k ochraně hlavních klíčů databáze jsou uložena v umístění pověření.

- `sys.openkeys` Tento katalogový pohled vrátí informace o všech otevřených šifrovacích klíčích v aktuálním uživatelském připojení. Vrácené informace obsahují identifikátor a název databáze, která obsahuje daný klíč, identifikátor, název a GUID každého otevřeného klíče a datum a čas, kdy byl klíč otevřen.
- `sys.symmetric_keys` Tento katalogový pohled vrátí záznam pro každý symetrický klíč v databázi. Vrácená informace obsahuje název, identifikátor, GUID, délku a algoritmus symetrického klíče.Vrací se také identifikátor principu, který vlastní daný klíč a data, kdy byl symetrický klíč poprvé vytvořen a naposledy modifikován.

Efektivita dotazů

Jak jsme se zmínili dříve v této kapitole, systém SQL Server automaticky generuje náhodný inicializační vektor, aby zabránil statistickým útokům na sloupce s daty. Potřeba eliminovat vzory ze zašifrovaných dat je v rozporu s potřebou indexovat a rychle vyhledávat ve stejných datech. Indexování využívá těchto vzorů k organizaci dat pro účinné hledání a získávání.

Hacker, který zná relativní frekvenci, se kterou se určitý kousek zašifrovaných dat objevuje v daném sloupci, by mohl tuto informaci použít k vyvození ještě dalších informací. Například z podnikové databáze obsahující informace o zaměstnancích v tabulce zašifrované bez použití náhodného inicializačního vektoru by mohly uniknout další informace na základě poskytnutého vzoru. Uvažujme tabulku `HumanResources.Employee` v databázi AdventureWorks. Většina řídicích a manažerských pozic se objevuje pouze jednou, zatímco pozice na nižších úrovních se mohou vyskytovat mnohokrát. Hacker může být schopen odvodit další informace z tohoto vzoru, například informaci o tom, kteří zaměstnanci jsou nejlépe placeni. Hacker může použít znalosti, jako je tato, k tomu, aby lépe zacílil svůj útok. Náhodný inicializační vektor systému SQL Server pomáhá eliminovat tyto vzory ze zašifrovaných dat. To má dvě hlavní implikace pro vývojáře v jazyce T-SQL:

- Stejný inicializační vektor, který byl použit při zašifrování, je třeba použít i při dešifrování.
- Šifrovací funkce nejsou deterministické, což znamená, že zašifrování stejného prostého textu vícekrát stejným klíčem nevygeneruje stejný zašifrovaný text.

Díky nedeterministické povaze šifrovacích funkcí systému SQL 2008 je zbytečné indexovat zašifrované sloupce přímo. Prohledávání zašifrovaných sloupců vyžaduje dešifrovat každou hodnotu ve sloupci a porovnat je jednu po druhé. To je velmi neefektivní a může se to stát úzkým hrdlem ve vašich aplikacích, pokud jsou vaše tabulky veliké.

Byly navrženy určité metody pro zvýšení efektivity prohledávání zašifrovaných dat. Tyto metody obvykle zahrnují uchovávání hodnoty otisku zašifrovaných dat pro indexování. Hlavní problém s těmito metodami je takový, že znovu zavádějí statistické vzory, které eliminoval náhodný inicializační vektor.

Můžete použít několik přístupů k dosažení rovnováhy mezi zabezpečením dat a účinností hledání. Nejdůležitějším doporučením je *nešifrovat* sloupce, které často používáte ve vašich vyhledávacích kritériích (podmínky `WHERE`), třídicích kritériích (podmínky `ORDER BY`) nebo seskupování (podmínka `GROUP BY`).

Avšak někdy nemusíte mít na výběr – můžete potřebovat zašifrovat sloupec, který je součástí podmínky `WHERE` nebo dalšího kritéria dotazu. Jedna věc, kterou můžete udělat pro zefektivnění, je zúžit výběr tím, že použijete další podmínky, které budou obsahovat nezašifrované sloupce.

Můžete vytvořit „pseudoindex" vašich dat tím, že přidáte další sloupec do vaší tabulky s hodnotou jednosměrného otisku vašeho prostého textu a vytvoříte index pro tento sloupec. Je možné použít funkci `HashBytes` zabudovanou v systému SQL Server 2008 pro generování jednosměrných hodnot otisků vašeho prostého textu algoritmem MD5, SHA-1 nebo jiným a uložit je v novém sloupci. Indexování této nové hodnoty prostého textu otisku může značně zefektivnit hledání rovnosti (operátor = v jazyce T-SQL). Avšak hledání v intervalu (operátory jako `<`, `>`, `BETWEEN` a tak dále) není možné použít na otiscích nebo zašifrovaných datech.

Jedním z důsledků pseudoindexování s hodnotou otisku je to, že opět otevírá dveře pro útoky na principu statistické analýzy za použití hodnot otisku jako vodítka. Použití hodnot otisku jako indexu také umožňuje slovníkové útoky na tyto hodnoty. Slovníkový útok je takový útok, při kterém hacker používá dlouhý seznam hodnot prostého textu a pokusí se uhádnout prostý text ze zašifrované hodnoty nebo hodnoty otisku pomocí hrubé síly.

Další metoda pseudoindexování zašifrovaných dat je variace na předchozí metodu s tím rozdílem, že používá hashovaný autentizační kód zprávy (HMAC – hashed message authentication code) místo hodnoty otisku. Podstata algoritmu HMAC je v použití „tajné" hodnoty, kterou zkombinuje s prostým textem a vygeneruje hodnotu otisku založenou na těchto datech. Přestože metoda algoritmu HMAC poskytuje ochranu proti slovníkovým útokům, neposkytuje žádnou další ochranu proti statistické analýze.

TIP

Člen týmu databázového stroje systému SQL Server Raul Garcia poskytuje na svém blogu (*http://blogs.msdn.com/raulga*) výborný příklad použití kódu s algoritmem HMAC pro vytvoření „indexu" na vašich zašifrovaných datech.

Hlavní věc, kterou berte v úvahu při používání možností zašifrování dat systému SQL Server, je to, že zašifrování a účinnost prohledávání jsou opačné cíle. Účelem zašifrování je zabezpečení dat, často na úkor účinnosti hledání. Zatímco můžete použít metody navrhované zde ke zvýšení účinnosti dotazů SQL na zašifrovaných datech, metody indexů hodnot otisků a algoritmu HMAC vyžadují více úložného prostoru a ve skutečnosti mohou obejít ochranu systému SQL Server proti statistické analýze (pomocí náhodného inicializačního vektoru).

Shrnutí

Systém SQL Server 2008 staví na šifrovacích funkcích uvedených v systému SQL Server 2005, a doplňuje podporu pro transparentní zašifrování dat a rozšiřitelnou správu klíčů, která nebyla dostupná v dřívějších vydáních. Společně se stále rostoucími požadavky na ochranu osobních a podnikových dat uložených v databázích SQL šifrování systému SQL Server poskytuje mechanismy pro splnění regulačních požadavků a ochranu vašich cenných dat.

V této kapitole jsme se zabývali zabudovanými nástroji systému SQL Server 2008 společně s úvodem do tří vrstev šifrovacích klíčů, které poskytuje systém SQL Server: hlavní klíče služby, hlavní klíče databáze a šifrovací klíče, společně s nově uvedenými pojetími certifikátů

serveru a šifrovacích klíčů databází. Také jsme mluvili o dostupných typech klíčů a certifikátů pro zabezpečení vašich dat, včetně certifikátů, párů asymetrických klíčů, symetrických klíčů a heslových frází.

Systém SQL Server nabízí několik symetrických a asymetrických šifrovacích algoritmů pro zabezpečení šifrovacích klíčů a dat, stejně jako generování hodnot otisku a podporu podpisu dat pro ověření obsahu. Systém SQL Server 2008 přidává podporu pro transparentní šifrování dat pro zabezpečení celé databáze najednou, rozšiřitelnou správu klíčů, která vám umožňuje dále zabezpečit vaše data pomocí hardwarových a softwarových nástrojů třetích stran.

Systém SQL Server 2008 obsahuje nové příkazy a funkce související s šifrováním a katalogové pohledy zabezpečení a pohledy správy dat souvisejících se zabezpečením, které poskytují možnosti celkové správy zabezpečení.