

SQL Server 安全数据加密

数据加密

自从 SQL Server 2005 开始支持数据加密，之后的版本逐一提升功能。你可以使用多种加密类型，同时有多种密钥将数据转换为不可读的(无意义的)数据，除非用户有解密用的密钥。SQL Server 支持多种加密算法，可以让 SQL Server 来管理密钥和保密。

加密密钥

SQL Server 可以让你使用任何三种加密密钥。加密密钥是数据的小块，当插入算法，将数据转换成乱码，没有正确的解密密钥几乎是不可能的转换回未加密的值。

->非对称密钥：该类型的加密使用公共/私有密钥对。一个密钥加密数据，另一个密钥进行解密。你可以与任何人分享的公共密钥，以便对任何它们加密数据，只有你可以通过使用私钥解密。SQL Server 使用 512、1024、2048 位的 Rivest-Shamir-Adelman(RSA)加密算法。

->对称密钥：在这种类型的加密，加密密钥和解密密钥是一样的。有时被称为一个共享的秘密，因为共享数据的双方必须有相同的密钥。在某些情况下，使用对称密钥是很困难的，因为从一方传递秘密到另一方是一个问题。对称密钥是数据库中使用的理想选择，因为它们永远不会离开数据库。SQL Server 支持 RC4、RC2、DES、AES 算法。

->证书：证书是非对称密钥加密的一部分，用于公钥加密的数字包装器。SQL Server 可以创建供你使用的证书，或者你可以从第三方认证机构获得。

SQL Server 使用加密密钥层次结构，如图 8.1 所示，加密和保护你在数据库中存储的密钥

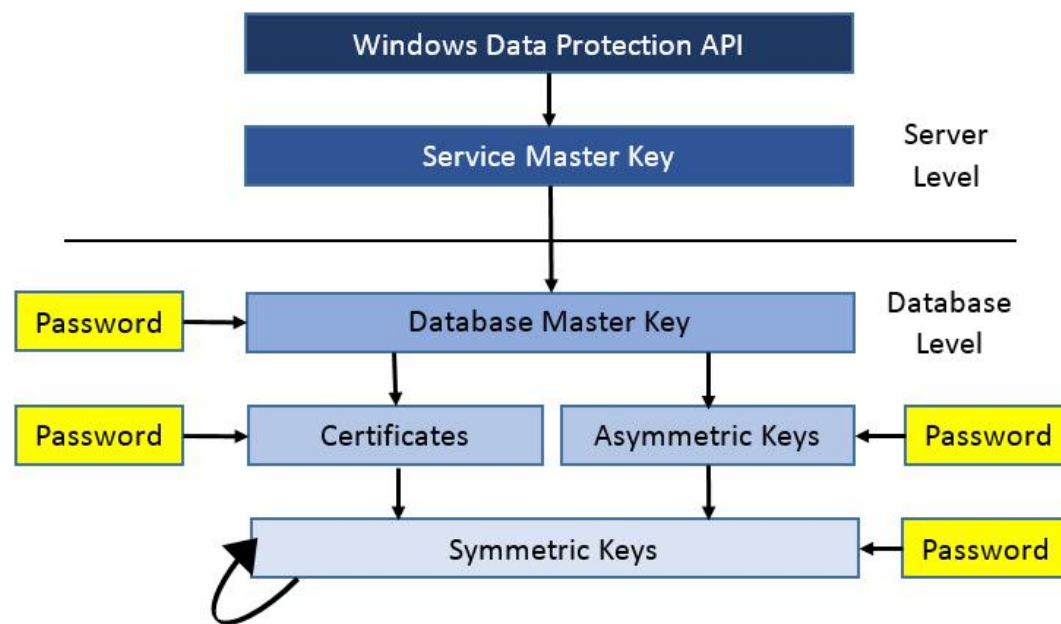


图 8.1 加密密钥层次结构

在服务器级别，每个数据库实例有一个 Service Master Key 用于加密低层次的密钥。这个密钥是在 SQL Server 实例安装时创建。你可以备份和还原它，甚至重新生成。SQL Server 为你管理，而且你从来不会明确使用它。Service Master Key 是存储和保护在 Windows，使用 Windows Data protection API, or DPAPA。

Service Master Key 有一些内部使用，但在这里对于我们而言 SQL Server 使用它来加密和保护你创建的 Database Master Key。在任何你想加密数据的数据库中都需要一个 Database Master Key。它是一个对称密钥用于加密和保护任何你创建密钥。你必须显式地在数据库环境中创建它，如代码 8.1 所示：

```
-- Set up sample encryption database
USE master;

GO

-- Set up a login
```

```

IF SUSER_SID('User1') IS NOT NULL DROP LOGIN User1;

CREATE LOGIN User1 WITH password = '3f@$fWDY3QvP&K0';

GO

IF DB_ID('EncryptionDB') IS NOT NULL DROP DATABASE EncryptionDB;

CREATE DATABASE EncryptionDB;

GO

USE EncryptionDB;

GO

CREATE USER User1 FOR LOGIN User1;

GO

USE EncryptionDB;

GO

-- Databases do not have a master key by default, so you
must create it before you can use it:

CREATE MASTER KEY ENCRYPTION BY PASSWORD = 'gK#3hbQKDFQ
Y0oF';

GO

```

代码 8.1 在 EncryptionDB 库中创建 Database Master Key

Database Master Key 存储两次：它首先被 Service Master Key 加密存储，然后被你提供的密码再次加密。你可以删除两者中的一个，但不能全删除。通常情况下不会操作它们。

作为对称密钥，Database Master Key 在你使用前必须打开它。打开一个密钥加载到内存然后解密，然后就可以使用。因为服务器级的 Service Master Key 加密了 Database Master Key，数据库能自动为你打开密钥，很少需要你显示的去打

开。类似 Service Master Key，你可以备份、还原、修改 Database Master Key。

你将使用图 8.1 中的其他密钥来加密数据。箭头指示使用哪一个密钥来加密和保护其他的密钥。例如，你可以使用 Database Master Key 来加密证书或非对称密钥。证书和非对称密钥只能保护对称密钥。对称密钥可以通过证书、非对称密钥或其他对称密钥来保护。

密钥管理

图 8.1 显示了加密密钥的另一个方面：你可以使用另一个密钥或一个密码来创建密钥。这就是所谓的密钥管理，这是 SQL Server 能为你管理的一个主要服务。历史上充满了各种国家的秘密，当密钥被截取时/泄露。第二次世界大战期间，美国、英国和其他盟国把大量的资源投入到德国和日本使用的加密密钥，以使他们能够拦截和读取高度敏感的信息。在最近的一次，许多高安全的应用程序被攻破，因为攻击者能够找到在应用程序或在计算机上嵌入的密钥。分享秘密是很困难的。

如果选择使用密码你可以自己管理 SQL Server 加密密钥，然后你肩负着保管密钥的责任。大多数人不想这样做，因为它需要高度专业化的技术技能。但是，如果你想承担这个任务，只要你创建一个密钥时使用密码选项。密码基本上是密钥，你必须确保你可以将密钥保存在安全的位置，并在需要时将其安全传输。

但是你不需处理这些细节，因为 SQL Server 将为你照顾密钥管理。它将为你加密新的密钥，使用你所指定的任何方法，并将数据存储在安全地存储任何其他敏感信息。

你有权管理钥匙，但没有一个很好的理由你不应该这样做。

加密数据

现在来看一个 SQL Server 中加密数据的例子。在这个场景 , Customer 表通常具有客户的常规信息。客户名称和所在城市 , 不是敏感数据 , 不需要加密 ; 但是信用卡类型、帐号可能包含个人敏感信息 , 应该进行加密。

你将使用一个对称密钥来加密表中的数据 , 记住一个对称密钥需要一个证书或非对称密钥在数据库中保护。因此 , 首先使用代码 8.2 创建一个非对称密钥来保护对称密钥 :

```
-- Create an asymmetric key to protect the new symmetric key

CREATE ASYMMETRIC KEY User1AsymmetricKey

    AUTHORIZATION User1

    WITH ALGORITHM = RSA_2048;
```

代码 8.2 创建一个非对称密钥

这个非对称密钥叫做 User1AsymmetricKey , 被 User1 用户拥有。这个密钥使用 2048 位的 RSA 加密 , 这是非常强/复杂的加密算法。这类数据肯定是非常非常重要 !

接着 , 使用代码 8.3 创建一个对称密钥 User1SymmetricKey。本例中使用 TRIPLE_DES 算法 , 并且被刚才创建的非对称密钥保护 :

```
-- Create a symmetric key, protected by the asymmetric key

CREATE SYMMETRIC KEY User1SymmetricKey

    WITH ALGORITHM = TRIPLE_DES

    ENCRYPTION BY ASYMMETRIC KEY User1AsymmetricKey;
```

代码 8.3 创建一个对称密钥

如果你想罗列数据库下的对称加密密钥，你可以使用 sys.symmetric_keys 目录视图来查看。代码 8.4 执行结果如图 8.2 所示。注意，因为 Database Master Key 是一个对称密钥，它也会出现在结果中。

```
-- List the symmetric keys in the database  
SELECT * FROM sys.symmetric_keys;
```

代码 8.4 罗列数据库下的对称密钥

	name	principal_id	symmetric_key_id	key_length	key_algorithm	algorithm_desc
1	##MS_DatabaseMasterKey##	1	101	256	A3	AES_256
2	User1SymmetricKey	1	256	128	D3	TRIPLE_DES

图 8.2 sys.symmetric_keys 目录视图返回结果

代码 8.5 是 EncryptionDB 数据库下 Customer 表的结构：

```
USE EncryptionDB;  
GO  
CREATE TABLE Customer (  
    CustId int,  
    Name nvarchar(30),  
    City varchar(20),  
    CreditCardType varbinary(1000),  
    CreditCardNumber varbinary(1000),  
    Notes varbinary(4000));  
GO  
-- Grant access on the table to user  
GRANT SELECT, INSERT ON Customer to User1;
```

代码 8.5 创建 Customer 表

注意，因为最后三列将会保存加密后的 binary 类型数据而不是原始的 string 数据，因此列的类型设置成 varbinary。列的长度依赖数据的大小以及算法。数据库有一个 User1 用户，对 Customer 表有 SELECT 和 INSERT 权限。

是时候加密一些数据并插入到数据库中。第一步使用代码 8.6 打开对称密钥，这一步引起 SQL Server 在内部存储查找密钥，确保用户有权限使用密钥，然后解密密钥到内存以准备使用：

```
OPEN SYMMETRIC KEY User1SymmetricKey  
    DECRYPTION BY ASYMMETRIC KEY User1AsymmetricKey;
```

代码 8.6 打开对称密钥

数据加密使用 T-SQL 语句 EncryptByKey 函数，使用唯一的 GUID 区分密钥。你可以使用 Key_GUID 函数检索 GUID 而不是直接传送数值。否则，代码 8.7 是一个普通的 T-SQL 插入语句：

```
INSERT INTO Customer VALUES (1, 'Sally Roe', 'Chatinika',  
    EncryptByKey(Key_GUID('User1SymmetricKey'), 'Visa'),  
    EncryptByKey(Key_GUID('User1SymmetricKey'), '1234-5678-9009-8765'),  
    EncryptByKey(Key_GUID('User1SymmetricKey'),  
        'One of our best customers. Treat like royalty.'));
```

代码 8.7 插入加密数据

最后一步使用代码 8.8 关闭对称密钥。它会从内存中移除密钥并释放资源。只要你不需要加密了就应该尽快关闭密钥，因为把它留在内存可能会被攻击者利

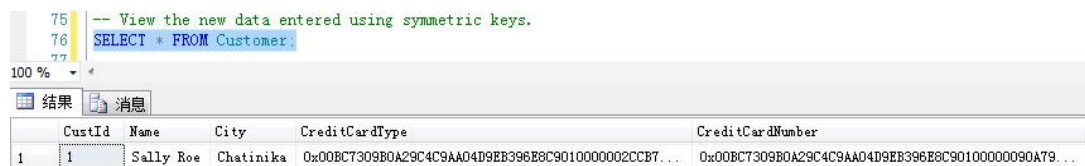
用。

```
CLOSE SYMMETRIC KEY User1SymmetricKey;
```

代码 8.8 关闭对称密钥

提示：如果你要使用密钥在一个单独批处理中加密或解密大量的数据，可以它先打开着。打开和关闭密钥需要处理时间的。但是当你完成的时候，别忘了把它关上！

现在运行一个查询语句查看表中的数据 如图 8.3 所示。你可以看到在 CustId、Name、City 列数据没有加密，但是在加密列文本是随机 binary 数据。你的数据是安全的！



	CustId	Name	City	CreditCardType	CreditCardNumber
1	1	Sally Roe	Chatinika	0x00BC7309B0A29C4C9AA04D9EB396E8C9010000002CCB7...	0x00BC7309B0A29C4C9AA04D9EB396E8C90100000090A79...

图 8.3 表中存储的加密数据

表中的数据是没有价值的，除非有一个方法来检索它。本例中你需要使用一个常规查询语句，并且使用 DecryptByKey 函数来解密数据。这个函数返回 varbinary 数据，因为加密数据可能是任何数据类型。因此要检索原始文本需要你对 DecryptByKey 函数结果进行转换。

```
OPEN SYMMETRIC KEY User1SymmetricKey

    DECRYPTION BY ASYMMETRIC KEY User1AsymmetricKey;

SELECT CustID, Name, City,

    CONVERT (VARCHAR, DecryptByKey (CreditCardType)) AS C
ardType,

    CONVERT (VARCHAR, DecryptByKey (CreditCardNumber)) AS
CardNumber,
```



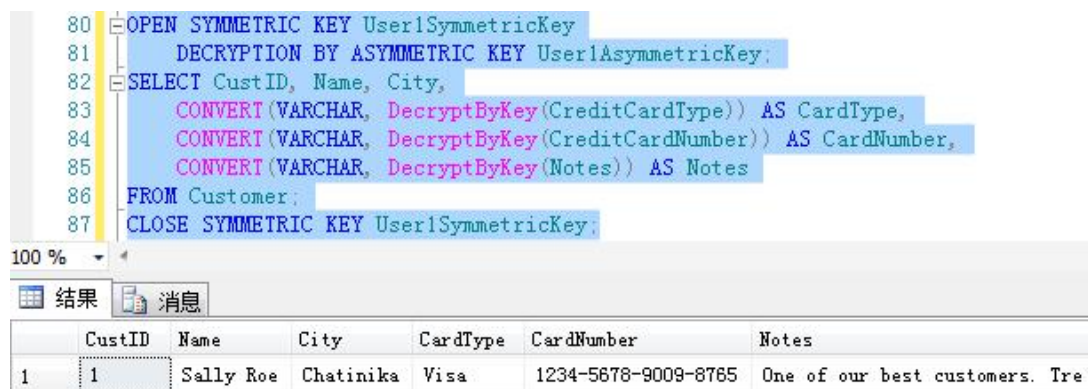
```

        CONVERT(VARCHAR, DecryptByKey(Notes)) AS Notes
FROM Customer;

CLOSE SYMMETRIC KEY User1SymmetricKey;

```

代码 8.9 打开密钥然后执行查询语句



The screenshot shows a SQL Server Enterprise Manager interface. The top pane displays a query script with line numbers 80 through 87. The query opens a symmetric key, decrypts data from the 'Customer' table, and closes the key. The bottom pane shows the results of the query in a table format with columns: CustID, Name, City, CardType, CardNumber, and Notes. The results show one record for a customer named Sally Roe.

```

80 OPEN SYMMETRIC KEY User1SymmetricKey
81     DECRYPTION BY ASYMMETRIC KEY User1AsymmetricKey;
82 SELECT CustID, Name, City,
83     CONVERT(VARCHAR, DecryptByKey(CreditCardType)) AS CardType,
84     CONVERT(VARCHAR, DecryptByKey(CreditCardNumber)) AS CardNumber,
85     CONVERT(VARCHAR, DecryptByKey(Notes)) AS Notes
86 FROM Customer;
87 CLOSE SYMMETRIC KEY User1SymmetricKey;

```

	CustID	Name	City	CardType	CardNumber	Notes
1	1	Sally Roe	Chatinika	Visa	1234-5678-9009-8765	One of our best customers. Tre

图 8.4 解密数据结果