Database Encryption

SAP® Adaptive Server® Enterprise 16.0

DOCUMENT ID: DC88886-01-1600-01
LAST REVISED: May 2014

Contents

CHAPTER 1 Overview of Encryption

SAP® Adaptive Server® Enterprise (SAP®ASE) authentication and access control mechanisms ensure that only properly identified and authorized users can access data. Data encryption further protects sensitive data against theft and security breaches.

Encrypt entire databases, or only columns, depending on your needs.

While both encrypted columns and fully encrypted databases allow you to comply with security and privacy requirements, the different usages may make one feature easier to deploy than the other. Consider using:

- Encrypt columns when you can easily identify which columns contain sensitive data.
- Encrypt databases when you must perform range searches over sensitive data columns, and when you lack the knowledge of the data model and cannot identify sensitive data columns (for example, in packaged applications that include thousands of tables). In addition, the definition of sensitive data (such as personal information) differs among different locations (such as states or countries); encrypting an entire database can allow you to satisfy these various data security requirements.

The SAP ASE encryption feature enables you to encrypt data that is at rest, without changing your applications. This native support provides the following capabilities:

- Fully encrypt databases
- Column-level granularity
- Use of a symmetric, National Institute of Standards and Technology (NIST)-approved algorithm: Advanced Encryption Standard (AES)
- Performance optimization
- Enforced separation of duties
- Fully integrated and automatic key management
- Application transparency: no application changes are needed
- Data privacy protection from the power of the system administrator

Data encryption and decryption is automatic and transparent. If you have insert or update permission on a table, any data you insert or modify is automatically encrypted prior to storage. Daily tasks are not interrupted.

Selecting decrypted data requires **decrypt** permission in addition to **select** permission. **decrypt** permission can be granted to specific database users, groups, or roles. SAP gives you more control by providing you with granular access capability to sensitive data. SAP also automatically decrypts selected data for users with **decrypt** permission.

Encryption keys are stored in the database in encrypted form. You can encrypt an encryption key using a key encryption key (KEK) derived from a

- System-level user-supplied password
- KEK derived from a user-supplied password (which can be the user's login password)
- Separately created database-level KEK (master key or dual master key)

The password you select reflects your ability to preserve data privacy, even from system administrators. You may choose to protect your column encryption key using dual-control mode to increase the security.

When data is encrypted, it is stored in an encoded form called "cipher text." Cipher text increases the length of the encrypted column from a few bytes to 32 extra bytes. Unencrypted data is stored as plain text.

Column and database encryption uses a symmetric encryption algorithm, which means that the same key is used for encryption and decryption. SAP ASE tracks the key that encrypts the data.

Generally, using data encryption requires these steps:

1. Install the license option ASE_ENCRYPTION. See the *SAP ASE Installation Guide*.
2. The system security officer (SSO) enables encryption in SAP ASE:
   ```
   sp_configure 'enable encrypted columns', 1
   ```
3. Depending on the method you chose to protect encryption keys, create a database-level master key or set the system encryption password.
4. Create one or more named encryption keys. Consider using passwords to protect data even from the database administrator.
5. Specify the data for encryption.
6. Grant **decrypt** permission to users who must see the data. You may choose to specify a default plain text value known as a "decrypt default." The SAP ASE returns this default, instead of the protected data, to users who do not have decrypt permission.

Once you perform these steps, you can run your existing applications against your existing databases, tables and columns, but now the data is securely protected against theft and misuse. SAP ASE utilities and other SAP products can process data in encrypted form, protecting your data throughout the enterprise. For example, you can:

- Use SAP® Control Center or SAP Central SAP ASE Plug-in to manage encrypted data using a graphical interface. See the online help.
- Use the bulk copy utility (**bcp**) to securely copy encrypted data in and out of the server. See the *Utility Guide*.
- Use the SAP ASE migration tool **sybmigrate** to securely migrate data from one server to another. See the *SAP ASE System Administration Guide*.
- Use SAP Replication Server to securely distribute encryption keys and data across servers and platforms. See the *Replication Server Administration Guide* for information on encryption when replicating.

Full Database Encryption

As of version 16.0, you can fully encrypt entire databases, providing protection for an entire database.

When you fully encrypt a database, all of its data, indexes, and transaction logs become encrypted. This encryption is transparent, so that users can perform operations on tables, indexes, and so on, as usual, without noticing any differences.

Column Encryption

Encrypting columns in SAP ASE is more straightforward than using encryption in the middle tier, or in the client application. Use SQL statements to create encryption keys and to specify columns for encryption; existing applications continue to run without change.

When you **insert** or **update** data in an encrypted column, SAP ASE transparently encrypts the data immediately before writing the row. When you **select** from an encrypted column, SAP ASE decrypts the data after reading it from the row. Integer and floating point data are encrypted in the following form for all platforms:

- Most significant bit format for integer data
- Institute of Electrical and Electronics Engineers (IEEE) floating point standard with MSB format for floating point data

You can encrypt data on one platform and decrypt it on a different platform, provided that both platforms use the same character set.

CHAPTER 2 Protect Data with Encryption Keys

SAP ASE uses two types of encryption keys and keeps keys encrypted when they are not in use.

Types of encryption keys:

- Database encryption key (DEK) – the DEK is created in the master database and used to encrypt a database.
- Column encryption key (CEK) – users must have access to the CEK before they can access encrypted data, but it must be encrypted before you store it on disk or in memory. SAP ASE encrypts the CEK using a key encryption key (KEK) and stores it in encrypted form in `sysencryptkeys`. The KEK also decrypts the CEK, allowing you to access decrypted data.

Key management includes creating, dropping, and modifying column encryption keys, distributing passwords, creating key copies, and providing for key recovery in the event of a lost password.

Creating the Database Encryption Key

The database encryption key is a 256-bit symmetric key that is created in the master database and used to encrypt a database.

Prerequisites

Before you can create a database encryption key (DEK):

- Verify that you have a valid SAP ASE encryption feature license (ASE_ENCRYPTION)
- Set the **enable encrypted columns** configuration parameter
- Create a master key and optionally, a dual master key in the `master` database; these protect the database encryption key.
- Ensure that you have the appropriate privileges:
  - If granular permission is enabled, a system permission called `manage database encryption key` is required to create the key.
  - If granular permission is disabled, you must have sso_role, keycustodian_role, or **create encryption key** permission.

Task

Use the **create encryption key** command in the `master` database to create a database encryption key. The syntax is:

```
create encryption key keyname
    [for algorithm]
    for database encryption
    [with
        {[master key]
        [key_length 256]
        [init_vector random]
        [[no] dual_control]}
```

where:

- *keyname* – must be unique in the user's table, view, and procedure name space in the `master` database.
- `for algorithm` – specifies the algorithm. Currently, the only supported algorithm is Advanced Encryption Standard (AES).
- `for database encryption` – explicitly specifies that you are creating an encryption key to encrypt an entire database, rather than a column.
- `master key` – is required for full database encryption. SAP ASE returns an error if the master key does not already exist.
- `key_length 256` – is the size, in bits, of the key you are creating. The only valid length for a database encryption key is 256; SAP ASE returns an error message if you use any other size.
- `init_vector random` – is required for full database encryption. If you specify `init_vector null`, as you can for creating a column encryption key, SAP ASE returns an error.
- `[no] dual control` – indicates whether the database encryption key must be encrypted using dual controls. By default, dual control is not configured.

This example creates a database encryption key that is protected by the master key:

```
sp_configure 'enable encrypted columns', 1
create encryption key master with passwd "testpassword"
set encryption passwd 'testpassword' for key master
create encryption key dbkey for database encryption
```

Dropping a Database Encryption Key

To drop the database encryption key, use the **drop encryption key** command. This command deletes the database encryption key from the `sysencryptkeys` table in the `master` database.

The syntax is:

```
drop encryption key key_name
```

Note: This command fails if the database encryption key you are dropping is still being used to encrypt a database.

Changing a Database Encryption Key

To change the manner in which a database encryption key is protected, as well as who its owner is, use the **alter encryption key** command.

You cannot regenerate a database encryption key for a database.

- To change a database encryption key:
  1. Decrypt the database protected by the database encryption key.
  2. Drop, and re-create the database encryption key.

  Note: You cannot convert a column encryption key into a database encryption key. SAP ASE displays an error message if you alter a different type of encryption key into a database encryption key using the **for database encryption** option.
- To simply change the way a database encryption key is protected, rather than change the database encryption key altogether, use this syntax:

```
alter encryption key key_name
for database encryption
modify encryption with {[master key]
    [[no] dual_control}
```

- To change the owner of a database encryption key:

```
alter encryption key [[database.][owner].]dek_name
    modify owner user_name
```

  The permission to run this option is the same as the permission for **alter encryption key**.

Creating Column Encryption Keys

A column encryption key must exist before a table owner can mark a column for encryption on a new or existing table.

When you set up keys for the first time, consider:

- Key owner or custodian assignment – the system security officer (SSO) must grant **create encryption key** permission to create keys. By default, the **sso_role** and the **keycustodian_role** have **create encryption key** permission.
- Whether keys should be created in a separate key database – SAP recommends that you use a separate database for keys, especially if keys are encrypted by the system encryption password.
- The number of keys needed – you can create a separate key for each encrypted column, or you can use the same key to encrypt columns across multiple tables. From a performance standpoint, encrypted columns that join with equivalent columns in other tables should share the same key. For security purposes, unrelated columns should use different keys.

Column encryption in SAP ASE uses the Advanced Encryption Standard (AES) symmetric key encryption algorithm, with available key sizes of 128, 192, and 256 bits. Random-key generation and cryptographic functionality is provided by the FIPS 140-2 compliant modules.

To securely protect key values, SAP ASE uses a 256-bit key-encrypting key (KEK), which may be a master key, or an internal key derived from either the system encryption password or a user-specified password.

SAP ASE encrypts the new key (the column encryption key) and stores the result in `sysencryptkeys`.

By default, SAP ASE creates 256-bit key-encryption keys. For compatibility with versions earlier than 15.7, it uses a 128-bit key if the KEK is derived from the system encryption password.

The syntax is:

```
create encryption key [[database.][owner].]keyname
    [as default] [for algorithm]
    [with
        {[key_length num_bits]
        [{passwd 'passwd_phrase' | passwd system_encr_passwd |
            master key}]
        [init_vector {null | random}]
        [pad {null | random}]
        [[no] dual_control]
    }]
```

where:

- *keyname* – must be unique in the user's table, view, and procedure name space in the current database. Specify the database name if the key is in another database, and specify the owner's name if more than one key of that name exists in the database. The default value for owner is the current user, and the default value for database is the current database. Only the system security officer can create keys for other users.

  Note: You cannot create temporary key names that start with "#".

- **as default** – allows the system security officer or key custodian to create a database default key for encryption. This enables the table creator to specify encryption without using a keyname on **create table**, **alter table**, and **select into**. SAP ASE uses the default key from the same database. The default key may be changed.
- *for algorithm* – Advanced Encryption Standard (AES) is the only algorithm supported. AES supports key sizes of 128, 192, and 256 bits, and a block size of 16 bytes. The block size is the number of bytes in an encryption unit. Large data is subdivided for encryption.
- **keylength *num_bits*** – the size, in bits, of the key to be created. For AES, valid key lengths are 128, 192, and 256 bits. The default **keylength** is 128 bits.
- **passwd *password_phrase*** – indicates to **ASE** to protect the CEK using the user password *password_phrase*, which can be a quoted alphanumeric string up to 255 bytes in length.
- **passwd *system_encr_passwd*** – indicates to **ASE** to protect the CEK using the system encryption password.
- **master key** – indicates to **ASE** to protect the CEK using the master key. By default, SAP ASE uses the master key (if it exists) to protect column encryption keys.

- **init_vector**
  - **random** – specifies use of an initialization vector during encryption. When an initialization vector is used by the encryption algorithm, the cipher text of two identical pieces of plain text are different, which prevents detection of data patterns. Using an initialization vector can add to the security of your data.

    Use of an initialization vector implies using a cipher-block chaining (CBC) mode of encryption, where each block of data is combined with the previous block before encryption, with the first block being combined with the initialization vector.

    However, initialization vectors have some performance implications. You can create indexes and optimize joins and searches only on columns where the encryption key does not specify an initialization vector.
  - **null** – omits the use of an initialization vector when encrypting. This makes the column suitable for supporting an index.

    The default is to use an initialization vector, that is, **init_vector random**.

    Setting **init_vector null** implies the electronic codebook (ECB) mode, where each block of data is encrypted independently.

    To encrypt one column using an initialization vector and another column without using an initialization vector, create two separate keys—one that specifies use of an initialization vector and another that specifies no initialization vector.
- **pad**
  - **null** – the default, omits random padding of data.

    You cannot use padding if the column must support an index.
  - **random** – data is automatically padded with random bytes before encryption. You can use padding instead of an initialization vector to randomize the cipher text. Padding is suitable only for columns whose plain text length is less than half the block length. For the AES algorithm the block length is 16 bytes.
- **dual control** – indicates whether the new key must be encrypted using dual control. By default, dual control is not configured.

Examples

These examples use various encryption attributes when creating a column encryption key, and many assume you have already created the master key or set the system encryption password.

- Example 1 – specifies a 256-bit key called “safe_key” as the database default key. Because the key does not specify a password, SAP ASE uses the database-level master key as the KEK for safe_key. If there is no master key, SAP ASE uses the system encryption password:

  ```
  create encryption key safe_key as default for AES
      with keylength 256
  ```

  Only the system security officer or a user with the **keycustodian_role** can create a default key.
- Example 2 – creates a 128-bit key called “salary_key” for encrypting columns using random padding:

```
create encryption key salary_key for AES with
     init_vector null pad random
```

- Example 3 – creates a 192-bit key named “mykey” for encrypting columns using an initialization vector:

```
create encryption key mykey for AES
     with keylength 192 init_vector random
```

- Example 4 – creates a key protected by a user-specified password:

```
create encryption key key1
     with passwd 'Worlds1Biggest6Secret'
```

  If a key is protected by a user-specified password, that password must be entered a column encrypted by the key can be accessed.

- Example 5 – creates a key protected by dual-control:

```
create encryption key dualprotectedkey
with passwd "Pass4Tomorrow"
dual_control
```

  Key “dualprotectedkey” is protected by the master key and a user password (in dual control). To access the key, you must enter both the user password for the key and the password for the master key.

Permissions

The **sso_role** and **keycustodian_role** implicitly have permission to create encryption keys. The system security officer or the key custodian uses this syntax to grant **create encryption key** permissions to others:

```
grant create encryption key
    to user_name | role_name | group_name
```

For example:

```
grant create encryption key to key_admin_role
```

To revoke key creation permission, use:

```
revoke create encryption key
    {to | from} user_name | role_name | group_name
```

Note: grant all does not grant **create encryption key** permission to the user. It must be explicitly granted by the system security officer.

See also

- *Chapter 8, Role of the Key Custodian* on page 73
- *Chapter 4, Database-Level Master and Dual Master Keys* on page 21
- *Key Protection* on page 12
- *Performance Considerations* on page 68
- *Dropping Column Encryption Keys* on page 11

Dropping Column Encryption Keys

Column encryption key owners can drop their own keys. The system security officer can drop any key.

Prerequisites

A key can be dropped only if there are no encrypted columns in any database that use the key.

Task

To drop an encryption key, use:

```
drop encryption key [[database.][owner].]keyname
```

For example, this drops an encryption key named `cc_key`:

```
drop encryption key cust.dbo.cc_key
```

When executing **drop encryption key,** SAP ASE does not check for encrypted columns in databases that are suspect, archived, offline, not recovered, or currently being loaded. In any of these cases, the command issues a warning message that names the unavailable database, but does not fail. When the database is brought online, any tables with columns that were encrypted with the dropped key are unusable. To restore the key, the system administrator must load a dump of the dropped key's database that precedes when the key was dropped.

The system security officer can use **sp_encryption** to identify all the columns encrypted with a given key.

See also

- *Creating Column Encryption Keys* on page 7
- *Chapter 8, Role of the Key Custodian* on page 73
- *Chapter 4, Database-Level Master and Dual Master Keys* on page 21
- *Key Protection* on page 12
- *Performance Considerations* on page 68

Changing the Column Encryption Key

Periodically change the keys used to encrypt columns and databases.

Create a new key using **create encryption key**, then use **alter table...modify** to encrypt the column with the new key.

In the following example, assume that the "creditcard" column is already encrypted. The **alter table** command decrypts and reencrypts the credit card value for every row of customer using `cc_key_new`.

```
create encryption key cc_key_new for AES
```

```
alter table customer modify creditcard encrypt with
       cc_key_new
```

Key Protection

The key administrator must decide where keys are stored, when they should be renewed, and which owners can use a given key to encrypt data.

See also

- *Creating Column Encryption Keys* on page 7
- *Dropping Column Encryption Keys* on page 11

Grant Access to Keys

The key owner or a user with the **sso_role** must grant **select** permission on a key before another user can specify the key in the **create table**, **alter table**, and **select into** statements.

The key owner can be the system security officer, the key custodian or, for non-default keys, any user with **create encryption key** permission. Key owners should grant **select** permission on keys as needed.

This example allows users with **db_admin_role** to use the encryption key named "safe_key" when specifying encryption on **create table**, **alter table**, and **select into** statements:

```
grant select on safe_key to db_admin_role
```

Note: Users who process encrypted columns through **insert**, **update**, **delete**, and **select** do not need **select** permission on the encryption key.

Separate Keys from Data

When you specify a data for encryption, you can use a named key from the same database or from a different database. Encrypting with a key from a different database provides a security advantage because, in the event of the theft of a database dump, it protects against access to both keys and encrypted data.

Administrators can also protect each database dump with a different password, making unauthorized access even more difficult.

Encrypting with a key from a different database needs special care to avoid data and key integrity problems in distributed systems. Carefully coordinate database dumps and loads. If you use a named key from a different database, SAP recommends that, when you dump a database that contains:

- Encrypted columns, you also dump the database where the key was created. You must do this if new keys have been added since the last dump.
- An encryption key, dump all databases containing columns encrypted with that key. This keeps encrypted data in sync with the available keys.

If you do not specify a named key, the data is automatically encrypted with the default key from the same database. The system security officer or the key custodian can use **sp_encryption** to identify the columns encrypted with a given key.

CHAPTER 3 Key Encryption

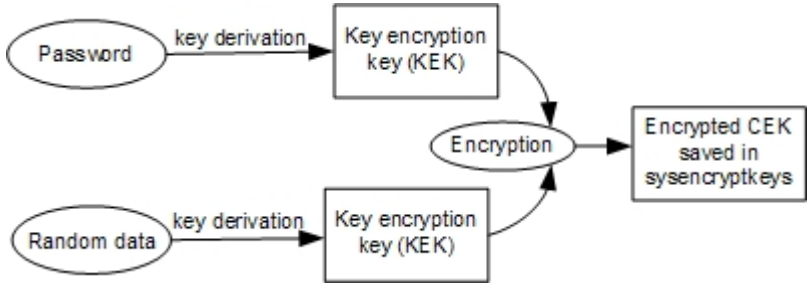
There are two keys between the user and the data: the database-encryption key (DEK) or column-encryption key (CEK) and the key-encryption key (KEK). The DEK and CEK encrypts data and users must have access to it before they can access encrypted data.

It cannot be stored on disk in an unencrypted form. Instead, SAP ASE uses a KEK, or 2 KEKs in dual control, to encrypt the DEK or CEK when you create or alter an encryption key. The KEK also decrypts the DEK or CEK before you can access decrypted data. DEKs and CEKs are stored in encrypted form in `sysencryptkeys`.

The KEK is a master key, created separately by the system security officer or key custodian, is an internally derived key from the system encryption password, a user-specified password, or a login password, depending on how you specify the key's encryption with the **create** and **alter encryption key** statements. Both the system encryption password and the master key are stored in encrypted form.

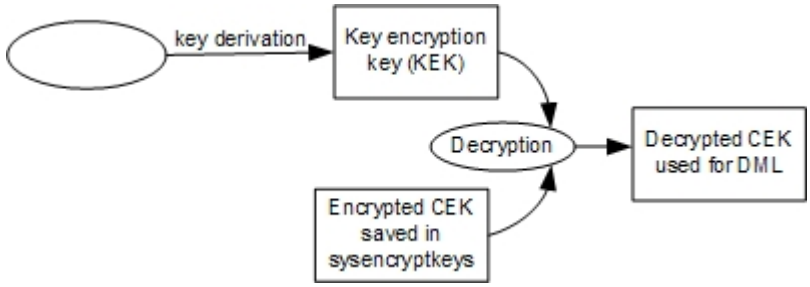
The following figure describes how to create and store a column encryption key for a **create encryption key** statement. The KEK is derived from a password and the KEK and the raw CEK are fed into the encryption function to produce an encrypted CEK.

Figure 1: Create an Encryption Key



The following figure describes how the KEK is used during a DML operation to decrypt the CEK. The raw CEK is then used to encrypt or decrypt data.

Figure 2: Accessing a CEK to Encrypt or Decrypt on DML Statement



Protect Encryption Keys with the Master Key

The master key is a database-level key that is created by a user with the **sso_role** or **keycustodian_role**, and is used as a KEK for user-created encryption keys. Once created, the master key replaces the system encryption password as the default KEK for user-created keys.

Although SAP ASE supports using the system encryption password, for compatibility with versions earlier than 15.7, SAP recommends that you use the master key.

You can use the master key with the dual master key to create a composite key that provides dual control and split knowledge for all user-created keys. You can also create a composite key by using the master key with a DEK's or CEK's explicit password.

Using a master key simplifies the administration of encrypted data because:

- Managing passwords for keys is restricted to setting the password for the master key.
- You need not specify passwords on **create** and **alter encryption key** statements.
- Allows for password distribution and recovery from lost column encryption key passwords.
- Access control over encrypted data is enforced through decrypt permission on the data.
- You need not make any changes to the application.

The syntax for creating a master key is:

```
create encryption key master
    [for AES] with passwd char_literal
```

See the *Reference Manual: Commands*.

See also

- *Restrict Decrypt Permission* on page 56

Protect Encryption Keys with the System-Encryption Password

The system encryption password is a database-specific password, and is the secondary default encryption method for the DEK or CEK.

SAP ASE uses the system encryption password to encrypt keys created in a specified database without an explicit password clause. Once the system security officer or key custodian has set a system encryption password, you need not specify this password to process encrypted columns. SAP ASE internally accesses the system encryption password when it needs to encrypt or decrypt column encryption keys.

The system security officer or key custodian uses **sp_encryption** to set the system encryption password. The system password is specific to the database using **sp_encryption**.

```
sp_encryption system_encr_passwd, password
```

password can be as many as 255 bytes in length.

Set a system encryption password only in the database where encryption keys are created.

The system encryption password protects your encryption keys. Choose long and complex system encryption passwords. Longer passwords are harder to guess or crack by brute force. Include uppercase and lowercase letters, numbers, and special characters in the system encryption password. SAP recommends that the system encryption password be at least 16 bytes in length.

SAP ASE enforces compliance of the system encryption password with the **minimum password length** and **check password for digit** configuration parameters.

Change the system password by using **sp_encryption** and supplying the old password:

```
sp_encryption system_encr_passwd, password [ , old_password]
```

Periodically change the system encryption password, especially when an administrator who knows the system encryption password leaves the company. When the system password is changed, SAP ASE automatically reencrypts all keys in the database with the new password. Encrypted data is unaffected when the system password is changed, in other words, it is not decrypted and reencrypted.

You can u-set the system encryption password by supplying “null” as the argument for *password* and supplying the value for *old_password*. Unset the system password only if you have dropped all the encryption keys in that database that were encrypted by the system encryption password.

The encrypted password value is stored in the `sysattributes` system table in that database. Additionally, the encrypted database feature introduces 43, a new `systtributes`

class that signifies full database encryption. For every storage allocation of the database that undergoes encryption, SAP ASE inserts a row in `sysattributes` with these values:

Column Name	Value
class	43
object	*dbid* (database ID)
object_info1	Starting logical page ID
object_info2	Ending logical page ID
int_value	Last encrypted logical page ID on one storage allocation

This row is removed when SAP ASE finishes encrypting the database.

Protect Keys with User-Specified Passwords

You can limit the power of the system administrator or database owner to access private data when you specify passwords on keys using **create encryption key** or **alter encryption key**.

If keys have explicit passwords, users must have, before they can decrypt data:

- **decrypt** permission on the data
- The encryption key's password

Users must also know the password to run DML commands that encrypt data.

See also

- *Chapter 9, Key Protection Using User-Specified Passwords* on page 77

Protect Encryption Keys with Dual Control

You can secure encryption keys with dual control using the **create encryption key** command.

If you specify **create encryption key with dual_control**, but do not specify a user password, the encryption key is protected by the master key and the dual master key.

If you specify **with dual_control** and include a user-specific password, the encryption key is protected by the master key and the user password.

- Example 1 – protects CEK "Reallysecret" with both the master and dual master keys and fails, unless both keys exist in the database:

```
create encryption key Reallysecret
     with init_vector random dual_control
```

- Example 2 – encrypts CEK "k3" with both the master key and user password "Whybother":

```
create encryption key k3
     with passwd 'Whybother'
     dual_control
```

See also

- *Change a Key's Protection Method* on page 78

CHAPTER 4 Database-Level Master and Dual Master Keys

SAP ASE allows users to create database-level encryption keys called the master key and the dual master key. These keys both act as key encryption keys, and are used to protect other keys, such as column and database encryption keys, and service keys.

The master key and the dual master key must have different owners. You can provide passwords for the master keys using either **isql**, or through a server-private file that is accessible only by the SAP ASE. The passwords to these keys are not stored in the database.

See also

- *Creating Column Encryption Keys* on page 7
- *Dropping Column Encryption Keys* on page 11

Creating the Master and Dual Master Keys

Once created, master keys become the default protection method for encryption keys. A dual master key is required only for dual control of column and database encryption keys.

Prerequisites

Only users with **sso_role** or **keycustodian_role** can create the master key and dual master key. There can only be one master and one dual master key for a database.

Task

To create the master and dual master keys use:

```
create encryption key [dual] master
    [for AES] with passwd char_literal
```

where:

- **master** and **dual** master refer to database-level keys used to encrypt other keys within the database in which they are defined. These keys are not used to encrypt data. The master key is named internally as `sybencrmasterkey` in `sysobjects`, and the dual master key is named internally as `sybencrdualmasterkey` in `sysobjects`.
- **with passwd** must be followed by a character string password that adheres to **sp_passwordpolicy**.

See the *Reference Manual: Commands*.

- Example 1 – creates master key in database `tdb1`:

```
use database tdb1
create encryption key master with passwd
'unforgetablethatswhatyouare'
```

- Example 2 – creates a dual master key in database `tdb1`:

```
use database tb1
create encryption key dual master with passwd 'dualunforgettable'
```

- Example 3 – generates an error because you cannot use a master key as a column encryption key:

```
create table t2 (c1 int encrypt with master)
```

To change the password of a master key or dual master key, use:

```
alter encryption key [dual] master
    with passwd <char_literal>
    modify encryption
    with passwd <char_literal>
```

Creating Master Key Copies

Users or master key owners with **sso_role** or **keycustodian_role** can create copies for master keys.

You may need to:

- Provide access to the master key or dual master key for unattended start-up of the SAP ASE. Such a key copy is referred to as the **automatic_startup** copy.
- Support recovery of the master keys should their passwords be lost. Such a key copy is referred to as the recovery copy.
- Allow a user other than the base key owner to set up encryption passwords for the master or dual master key. This key copy is referred to as a regular copy.

To add master key copies in a database, use:

```
alter encryption key [dual] master
    with passwd char_string
    add encryption
    {with passwd char_string
    for user user_name
    [ for recovery ] | [ for automatic_startup ] }
```

where:

- *char_string* – (first reference) specifies the password that currently encrypts the base copy of the master or dual master key.
- *char_string* – (second reference) specifies the password for the regular or recovery copy. It must not be used for **automatic_startup** copies.
- **for user** – indicates the user to whom the regular or recovery copy must be assigned. Do not use this parameter to enter a password for **automatic_startup** copies.

- **for recovery** – indicates that the key copy is to be used to recover the master key in case the password is lost.
- **for automatic_startup** – indicates that the key copy is to be used to access the master or dual master key after the server is restarted with automatic master key access enabled.
- Example 1 – master key owner creates a key copy for Mary:

```
alter encryption key master
     with passwd 'unforgettablethatswhatur'
     add encryption
     with passwd 'just4now'
     for user mary
```

- Example 2 – dual master key owner Smith creates a key copy for **automatic_startup** with:

```
alter encryption key dual master
     with passwd 'Never4Getable'
     add encryption
     for automatic_startup
```

See also

- *Chapter 10, Key Recovery from Lost Passwords* on page 89

Setting Passwords for the Master and Dual Master Keys

The base key owner, or a user who owns a regular key copy, can set the password for the master and dual master keys. Passwords must be set before master keys can be used.

To set passwords for master keys, you can either use the:

- **set encryption passwd** command
- Use the unattended start-up feature
- (Master key only) the **dataserver** command

The **set encryption** command is:

```
set encryption passwd char_literal
    for key [dual] master
```

where:

- *char_literal* – if the user is the key owner, this is the password that currently encrypts the base copy of the master or dual master key. If the user is not the key owner, this is the password that currently encrypts the user's copy of the key.

Example – sets the password "MasterSecret" for the master key in database `tdb1`:

```
use tdb1
set encryption passwd 'MasterSecret' for key master
```

SAP ASE sets the password in the server memory for the database in which the master or dual master key is defined, and also records the identity of the user setting the password. Once set, the password is available for all access to the master key in the database.

Altering Passwords and Key Encryption Keys for Master Key Copies

Users who own master key copies can change the passwords for their key copies.

To change the password for key copies:

```
alter encryption key [dual] master
    with passwd char_string
    modify encryption
    {with passwd char_string [for recovery]
        | for automatic_startup}
```

where:

- *char_string* – (first instance) If the user is the key owner, this is the password that currently encrypts the base copy of the master or dual master key. If the user is not the key owner, this is the password that currently encrypts the user's copy of the key.
- *char_string* – (second reference) specifies the new password for the regular or recovery copy. Do not use this parameter to enter a password for **automatic_startup** copies.
- **for automatic_startup** – generate a new KEK and use it to create a new **automatic_startup** key copy.

If neither **for recovery** nor **for automatic startup** is specified, and the command is issued by the key owner, SAP ASE alters the base key copy password. If the command is not issued by the key owner, SAP ASE alters the password of the base key copy only if the current user has **sso_role** or **keycustodian_role**.

- Example 1 – master key owner "Jones" creates a key copy for "Mary" using:

  ```
  alter encryption key master
       with passwd 'unforgettablethatswhatyouare'
       add encryption
       with passwd 'just4now'
       for user Mary
  ```

- Example 2 – "Mary" changes the password for her copy using:

  ```
  alter encryption key master
       with passwd 'just4now'
       modify encryption
       with passwd 'maryspasswd'
  ```

- Example 3 – master key owner "John" changes the password for the base key using:

  ```
  alter encryption key master
       with passwd 'unforgettablethatswhatyouare'
       modify encryption
       with passwd 'notunforgettable'
  ```

Users with **sso_role** or **keycustodian_role** can modify the **automatic_startup** key copies to change their key encryption keys. For example, such a user with knowledge of the master key password, can change the key encryption key of the **automatic_startup** key copy using:

```
alter encryption key master
     with passwd 'unforgettablethatswhatyouare'
     modify encryption for automatic_startup
```

The SAP ASE:

- Decrypts the base master key with a key encryption key derived from the password.
- Creates a new master key encryption key and replaces the old key in the master key start-up file with this new key.
- Creates a new **automatic_startup** key copy by encrypting the master key using the new master key encryption key, and replacing the old **automatic_startup** key copy in `sysencryptkeys` with this new copy.

Regenerate Master Keys

Periodically change the master and dual master keys. However, each time you change the master and dual master keys, you must also reencrypt all column and database encryption keys using the new master and dual master keys.

To automate this process, SAP ASE uses the **regenerate key** option which replaces the master or dual master key values with the new values, and reencrypts all column and database encryption keys that are currently encrypted by the master or dual master keys being regenerated:

```
alter encryption key [dual] master
    with passwd char_string
    regenerate key
    [with passwd char_string]
```

When **regenerate key** command is executed, SAP ASE:

- Validates that the supplied password decrypts the base master or dual master key.
- Creates a new master or dual master key.
- Decrypts all column and database encryption keys that are encrypted either solely or partially by the master or dual master key. SAP ASE reencrypts them using the new master or dual master key.
- Replaces the base master or dual master key with the new key encrypted by the second password. If the second password is not supplied, SAP ASE uses the currently configured password to encrypt the new key.
- Drops the regular key copies. The master key owner must re-create regular key copies for designated users using **alter encryption key**.
- Drops the key recovery copy. The master key owner must add a new recovery key copy using **alter encryption key**, and inform the recovery key owners of the new password.
- Replaces the **automatic_startup** copy with a new key copy created by encrypting the new master key with a new randomly generated master key encryption key. SAP ASE writes the new master key encryption key into the master key start-up file.

Dropping Master Keys and Key Copies

A user with **sso_role** or **keycustodian_role** can drop a master or dual master key provided that there are no other column or database encryption keys that are currently encrypted using that master or dual master key.

To drop a master or a dual master key, use:

```
drop encryption key [dual] master
```

When a master or dual master key is dropped, SAP ASE:

- Drops the master or dual master key, and its key copies. All regular key copies, the **automatic_startup** key copy, and recovery key copies are deleted from the database.
- Deletes the master key encryption keys from the master keystart-upfile, if an **automatic_startup** key copy exists.

To delete only the regular key copy, use:

```
alter encryption key [dual] master
    drop encryption for user username
```

To delete only the recovery key copy, use:

```
alter encryption key [dual] master
    drop encryption for recovery
```

To delete only the **automatic_startup** key copy, use:

```
alter encryption key [dual] master
    drop encryption for automatic_startup
```

Recovering the Master Key and Dual Master Key

A user with **sso_role** or **keycustodian_role** can recover the master or dual master key.

To recover the master or dual master key:

```
alter encryption key [dual] master
    with passwd char_string
    recover encryption
    with passwd char_string
```

where the first reference to **passwd** is the password to the recovery key copy and the second reference to **passwd** is the new password for the base key.

Starting SAP ASE in Unattended Start-Up mode

Use unattended start-up mode to allow access to the master keys when the password holders are unavailable.

1. Enable the **automatic master key access** configuration parameter.
2. (Optional) set the master key start-up file path and name. Otherwise, SAP ASE uses the default file path and name.
3. Add **automatic_startup** copies for the master keys or dual master keys for databases for which you intend to have unattended start-up.

Configure Unattended Start-Up Mode

In unattended start-up mode, SAP ASE accesses master keys, even if their passwords are not present in server memory, by reading and decrypting the master key encryption keys from the master key start-up file.

Users with **sso_role** can configure SAP ASE to use unattended start-up mode by setting:

```
sp_configure 'automatic master key access', 1
```

To use unattended start-up mode, you must also create **automatic_startup** key copies for the **master key** and **dual master key** in the database.

Create the Master Key Start-Up File

When **automatic master key access** is enabled, SAP ASE reads in the key encryption keys from the master key start-up file.

If the master key start-up file does not exist, SAP ASE creates a master key start-up file, but does not write the key encryption key values to the file until **automatic_startup** key copies either of the master or dual master keys are created

When **automatic master key access** is disabled, SAP ASE drops the key encryption keys for master and dual master keys from the server memory. SAP ASE does not erase the key encryption key values from the master key start-up file.

A user with the **sso_role** can specify the master key start-up file path and name using:

```
sp_encryption mkey_startup_file
        [, {new_path | default_location | null}]
        [, {sync_with_mem | sync_with_qrm}]
```

where:

- *new_path* – specifies the location and name of the master key start-up file. *new_path* is not supported in standalone SAP ASE Cluster Edition installations.

- **default_location** – sets the master key start-up file to the default path and name: `$SYBASE_ASE/security/ase_encrcols_mk_<servername>.dat`. **default_location** is not supported in standalone SAP ASE Cluster Edition installations.
- **null** – displays the current master key start-up file path and name.
- **sync_with_mem** – writes the master key encryption keys existing in server memory to the master key start-up file, if configuration option automatic master key access is enabled. **sync_with_mem** is not supported in standalone SAP ASE Cluster Edition installations.
- **sync_with_qrm** – (Available only with standalone Cluster Edition installations) updates the key copy in the local master key start-up file with the copy on the quorum device.

How SAP ASE Uses the Master Key Start-Up File

SAP ASE reads the master and dual master key encryption keys from the master key start-up file into the server memory.

If:

- The server is started with **automatic master key access** enabled, or
- **automatic master key access** is enabled while the server is running.

If:

- An **automatic_startup** key copy of the master or dual master key is created, SAP ASE writes the master or dual master key encryption keys to the file.
- The key encryption key of the **automatic_startup** key copy of the master or dual master key is altered, SAP ASE writes the new master or dual master key encryption keys to the file.
- An **automatic_startup** key copy is dropped, SAP ASE deletes the corresponding record in the file.
- A database is dropped, SAP ASE deletes all records belonging to the dropped database.
- A master or dual master key is dropped, SAP ASE deletes the corresponding record.
- A new master key start-up file is specified using **sp_encryption mkey_startup_file**, SAP ASE synchronizes the server memory with the contents of the new file.

Once a master key encryption key is in memory, the master key can be accessed through the **automatic_startup** copy even if the master key password is not set.

CHAPTER 5 Secure External Passwords and Hidden Text

SAP ASE provides strong encryption for external login passwords and hidden text, using the AES-256 symmetric encryption algorithm.

You may choose strong encryption for external passwords to:

- Replication Agents – replicated databases.
- CIS – remote descriptors and logins.
- Job Scheduler – Job Scheduler Agent.
- RTMS – real-time messaging.
- Secure Sockets Layer (SSL) and Lightweight Directory Access Protocol (LDAP) – SSL and LDAP access accounts. Passwords are administered using stored procedures `sp_ldapadmin` and `sp_ssladmin` can be secured.

Objects that have SQL text stored in `syscomments`, such as stored procedures, user-defined functions and computed columns can be optionally encrypted with strong encryption using **sp_hidetext**.

Note: Encrypting external passwords and hidden text requires the ASE_ENCRYPTION license.

Service Keys

Service keys are 256-bit, persistent encryption keys used to strongly encrypt external login passwords and hidden text, and are stored in `sysencryptkeys`.

Encrypt service keys using either:

- A static key – is the default key encryption key for service keys, and can be used if no master key has been created in the current database. With this method, SAP ASE can use service keys after an unattended start-up.
- The master key – provides stronger protection than a static key. SAP ASE requires the password to decrypt the database-specific master key.

The database objects that describe these service keys include:

- **syb_extpasswdkey** – identifies service key for encryption of external login passwords in `sysattributes`. Only one **syb_extpasswdkey** exists for any database. When the **syb_extpasswdkey** is changed, all data encrypted using the key is reencrypted using the new key.

Although external login passwords are generally stored in the master database, RepAgent stores this information in replicate databases.

- **syb_syscommkey_dddddd** – identifies service key for encryption of hidden text in `syscomments`, where "dddddd" is a global identifier generated by SAP ASE to uniquely identify the key. The global identifier is included with the name to distinguish names when there are many `syb_syscommkey` keys associated with the same object. The global identifier distinguishes the key, on both the local database and in the replicate database.
  Strong encryption of hidden text requires a service key in each database where `sp_hidetext` is executed to hide SQL text. When a new service key is created, any existing service key in the database persists until explicitly dropped, and any hidden text is not reencrypted until you reissue **sp_hidetext**.

Note: The system encryption password does not encrypt service keys.

During an upgrade to version 15.7 or later, procedural objects are recompiled from source. Connected users are restricted in what they can do until the master key password is entered for databases where strong encryption of hidden text is enabled, and service key is protected by master key.

An authorized user must set the master key password on such databases using:

```
use mydb
go
set encryption passwd password for key master
go
```

Creating Service Keys

A user with **sso_role** or **keycustodian_role** can create a service key and becomes the owner of the key.

Prerequisites

To create service keys:

- An ASE_ENCRYPTION license is required.
- The **enable encrypted columns** configuration parameter must be set.
- The user creating the service key must have **sso_role** or **keycustodian_role**.
- The master key must be created before the service key, if you are protecting service keys with the master key.

Task

Use:

```
create encryption key [syb_extpasswdkey | syb_syscommkey]
    [ with { static key | master key }]
```

By default, the static key encrypts the keys. To use the master key, use the **with master key** parameter.

When a **syb_extpasswdkey** is created, all external passwords in `sysattributes` are reencrypted with the new key using strong encryption.

When a **syb_syscommkey** is created, any subsequent execution of `sp_hidetext` uses the new key with strong encryption. `sp_hidetext` must be executed on an existing database object for the object to be encrypted with the new key. Because reencrypting hidden text may involve very large amounts of data, database administrators should defer executing `sp_hidetext` to times when there is low system demand.

Note: You cannot use dual control with service keys.

Dropping Service Keys

drop encryption key ensures that there are no remaining references to the encryption key, and then deletes it. You cannot drop a nonexistent **syb_extpasswdkey** or **syb_syscommkey_dddddd**. To ensure that you delete all hidden text keys, use **sp_encryption** to identify all existing keys.

Prerequisites

Users must have a **keycustodian_role** or **sso_role** to delete an unused service key.

Task

Note: If your ASE_ENCRYPTION license has expired, encrypted data is no longer available, and you cannot execute the **drop encryption key** command. Contact SAP Technical Support to obtain a temporary license.

To delete an unused service key for external logins, use:

```
drop encryption key syb_extpasswdkey
    with password encryption downgrade
```

When **with password encryption downgrade** is specified, SAP ASE resets external login passwords with the algorithm used in versions earlier then 15.7. The Replication Agent password, and the CIS and RTMS external login passwords are reset to an invalid value. The administrator must manually reenter the passwords, after the key is dropped, to resume usage of the corresponding services.

To delete an unused single service key for hidden text, use:

```
drop encryption key syb_syscommkey_dddddd
```

SAP ASE checks if there are any references to the specified key **_dddddd**, and drops the key if no references are found.

Because **syb_syscommkey_dddddd** indicates a single key, you cannot specify **syb_syscommkey_dddddd** with the **with text encryption downgrade** parameter.

To delete multiple keys:

```
drop encryption key syb_syscommkey with text encryption downgrade
```

- If you specify **with text encryption downgrade**, you cannot specify a single service key with **syb_syscommkey_dddddd**, only with **syb_syscommkey**.
- Without the "dddddd" suffix for the **syb_syscommkey**, SAP ASE reencrypts all the hidden text in `syscomments` with the algorithm used in versions earlier than 15.7, and drops all **syb_syscommkey_dddddd** keys.

Modify Service Keys

You can regenerate **syb_extpasswdkey** or change its protection encryption from master key to static key, or vice versa. You cannot regenerate **syb_syscommkey**.

Changing the syb_extpasswdkey

You can change **syb_extpasswdkey** from static to dynamic.

Change the **syb_extpasswdkey** using:

```
alter encryption key syb_extpasswdkey
    [ with { static key | master key} ]
        { regenerate key [ with { static key | master key } ]
        | modify encryption [ with { static key | master key } ] }
```

where:

- The first instance of **with {static key | master key}** is optional and represents how the **syb_extpasswdkey** is currently encrypted.
- The second instance of **with {static key | master key}** allows the administrator to change the encryption on the regenerated key from static to dynamic, or vice versa. If you omit this parameter, the regenerated key remains encrypted as it was before issuing this command.
- The third instance of **with {static key | master key}** changes the protection on the existing key to use the static key or the master key as specified. If you omit this parameter, by default, the static key is used.

1. Creates a new service key for the external login passwords.
2. Reencrypts the passwords in `sysattributes` using the new key.
3. Drops the old key.

For example:

- Create a service key for external login passwords and encrypt all external login passwords with the service key protected by the static key:

  ```
  create encryption key syb_extpasswdkey
  ```
- Regenerate the service key for external login passwords, leaving the new service key protected by the static key and reencrypting all external passwords encrypted by the old service key:

  ```
  alter encryption key syb_extpasswdkey
         regenerate key
  ```
- Change the protection of the service key to be encrypted by the master key. The service key does not change, and external login passwords are not reencrypted:

```
alter encryption key syb_extpasswdkey
      modify encryption with master key
```

Note: Before issuing this command, ensure that the master key password has already been entered by the master key owner.

Changing the syb_syscommkey

To change the **syb_syscommkey**, create a new key and use `sp_hidetext` to reencrypt with the new key.

For example:

- Example 1 – Create a new hidden text encryption key and encrypt all SQL text objects in the `syscomments` table with the newly created key:

  ```
  create encryption key syb_syscommkey
  go
  sp_hidetext
  go
  ```

  Note: When a new **syb_syscommkey** is created, it becomes the default key used by `sp_hidetext` in that database.

- Example 2 – Create a new hidden text encryption key, encrypt the text of a specific stored procedure in `syscomments` with the newly created key, and protect the key with the master key:

  ```
  create encryption key syb_syscommkey
         with master_key
  go
  sp_hidetext sp_mysproc
  go
  ```

  In this example, all other hidden text rows in `syscomments` remain encrypted with the previous encryption key.

Service Keys with External Passwords

Service keys decrypt the private-key password for network listeners using SSL. The private-key password initializes the SSL certificate.

SSL Passwords

How SSL listeners start depends on if the service keys are encrypted by master key and whether the master key is available.

If the service keys are encrypted by the master key and the master key is unavailable:

- When only SSL listeners are specified in the interfaces file, no user can log in to enter the master key or dual master key password. The SAP ASE shuts down because it cannot start any listeners.
- When both SSL and non-SSL listeners are specified in the interfaces file, the non-SSL listener can accept login requests. The SSL listeners are blocked until the master key

password is entered manually by an authorized user after connecting to the SAP ASE on a non-SSL listener port using:

```
use master
go
set encryption passwd password for key master
go
```

When the master key password is correctly entered, SAP ASE wakes the SSL listener processes and they begin to accept incoming login requests.

LDAP Passwords

Service keys are required to decrypt the password for LDAP administration accounts when SAP ASE authenticates users during the LDAP user authentication process. Until authentication is complete, users cannot log in using LDAP.

An authorized user that can authenticate locally using SAP ASE authentication can manually enter the master key password using:

```
use master
go
set encryption passwd password for key master
go
```

See the *Security Administration Guide*.

Replication Agent Passwords

Service keys decrypt passwords that initiate connections by Replication Agents on user databases. Agents that are configured to start automatically are blocked until an authorized user enters the master key password manually, if the service key is encrypted by a master key.

If a service key is in a user database that is replicated, the service key is also available on the replicate database because the `sysencryptkeys` table that stores the encryption keys is also replicated. The master key is also stored in the `sysencryptkeys` table that is replicated, and also available on the replicate database. Because they are encrypted, service keys remain protected during the replication process.

After the SAP ASE has been started, an authorized user can connect and set the master key password for each database using:

```
use mydb
go
set encryption passwd password for key master
go
```

A Replication Agent that is waiting for the master key password can be identified by the status value “passwd sleep”:

```
sp_who
go
```

```
fid spid status loginame orignamehostname blk_spid dbname tempdbname
cmd block_xloid
```

```
--- ---- ----------- ------ -------- -------- ------
------ ---------- --------- -----------
0   38  passwd sleep NULL   NULL      NULL     0
tdb4   tempdb     REP AGENT  0
```

Service Keys Encrypted with the Master Key

If your service keys are encrypted with the master key, the master key's password must be entered into SAP ASE, either automatically or manually, depending on how you specify the master key.

If you do not use automatic master key access, you typically enter the master key's password with **set encryption passwd**. However, if a service key is required to decrypt the private key password for network listeners during start-up, you can supply the master key at the command line, or through a command line prompt.

Use the **dataserver . . . -- master_key_password** parameter to prompt for a master key password during SAP ASE start-up. The user issuing the **-- master_key_password** parameter must know the master key password for the master database and have physical access to the console and keyboard to enter the password.

If you do not include a password, **-- master_key_password** prompts for password at the command line. For example:

```
dataserver --master_key_passwd  -dd_master  -eerrorlog
```

```
master_key_passwd:_
```

The password characters do not appear, and the password is not validated until later in the SAP ASE start-up sequence.

If you include the password with the **-- master_key_password** parameter:

```
dataserver --master_key_passwd=mysecret  -dd_master  -eerrorlog
```

The password, `mysecret`, is blanked out in memory after it is read and used. However, the clear password is visible until the memory is blanked out.

If you enter the incorrect password, attempts to use service keys fail, and SAP ASE services that require the service keys remain unavailable. After the server has started, an authorized user can connect and set the master key password in the master database with:

```
use master
go
set encryption passwd password for key master
go
```

If you have configured only SSL listeners and you enter the wrong password, SAP ASE shuts down because it cannot start any listeners.

SAP recommends that you do not use passwords at the command line because the passwords are visible:

- In memory that can be seen with the UNIX **ps** command
- In memory, on an unattended terminal screen, or on disk in command history buffers and files
- On the screen

SAP encourages customer sites to prompt for passwords to avoid these vulnerabilities when using attended start-up.

CHAPTER 6 Database Encryption

Encrypt databases when you must perform range searches over sensitive data columns, and when you lack the knowledge of the data model and cannot identify sensitive data columns.

Create an Encrypted Database

To create a fully encrypted database, use the **create database** command.

Specify whether to encrypt a database when you create it, and data inserted into the database is automatically encrypted. The size of the database does not change when it is encrypted, and all storage access functions work identically whether a database is encrypted or not. The types of databases that support encryption are:

- Normal user database
- Temporary database
- Archive database

You cannot encrypt an in-memory database.

To create an encrypted database, use:

```
create [temporary] database database_name
    encrypt with key_name
```

Where:

- *database_name* is the name of the encrypted database you are creating.
- *key_name* is the name of the database encryption key.

To create an encrypted archive database, use:

```
create archive database database_name
    encrypt with key_name
```

Where:

- *database_name* is the name of the archive database you are creating
- *key_name* is the same key that you used to encrypt the database that was backed up. SAP ASE verifies that *key_name* matches during the database load. If it does not match, the restoration fails.

Example

Creates an encrypted database called `demodb` with data on device `demodev` and log on device `demologdev`, using an encryption key called `dbkey`:

```
create database demodb on demodev log on demologdev encrypt with
dbkey
```

Usage

There is no special permission to use the `encrypt with` option of the **create database** command. Users however, need **select** permission on the database encryption key to be able to reference it as the *key_name*.

Encrypt an Existing Database

You can encrypt an unencrypted database using the **alter database** command.

Depending on the size of the database, encryption might can take a while. For this reason, the command returns as soon as the database is marked for encryption. Encryption occurs in the background and the process is transparent to users. To check on the status and progress of database encryption, run the **sp_helpdb** system procedure, the **dbencryption_status()** built-in function, or the SAP Control Center user interface. Keep in mind:

- Database encryption occurs while the database is online. This means the database is accessible by other users while it is being encrypted, and does not require you to put it into single-user mode.
- The encryption process does not interrupt any user queries, updates, or insert operations on the database.
- You can suspend and resume database encryption, so that you can resume encrypting the database after restarting SAP ASE.
- The encryption operation is executed page by page.
- You cannot alter archive databases for encryption and decryption.
- SAP ASE records the encryption progress of a database and provides utilities to report its status.

Restrictions:

- You cannot encrypt the `master`, `model`, `dbccdb`, and `dbccalt` databases.
- You cannot decrypt a database that is in the process of being encrypted, or encrypt a database that is being decrypted.
- You cannot unmount a database while it is in the process of being encrypted.
- You cannot load another database on top of a database that is being encrypted.
- Do not execute commands that shrink database size when the database is being encrypted.

The syntax is:

```
alter database database_name
{encrypt with key_name [parallel degree_of_parallelism]
| resume encryption [parallel degree_of_parallelism]
| suspend encryption
}
```

where:

- `encrypt with` *`key_name`* instructs SAP ASE to encrypt the database using *key_name*.

  Specifically, the command retrieves the corresponding key ID from the `sysencryptkeys` system table in the `master` database and set the `encrkeyid` column in its related `sysdatabases` row.

  SAP ASE fails to run **alter database** and displays an error message if the database is already:

  - Encrypted with another key.
  - Being encrypted.

  If you run this command on a partially encrypted database that is not currently being encrypted, SAP ASE treats the command as if you specified the `resume encryption` option, as long as the key name is the same as the previously specified key.
- `parallel` *`degree_of_parallelism`* determines how many worker threads to initiate for the task.

  Create a thread for each database storage virtual device, as long as the number is equal to or fewer than "`number of worker processes`" configuration. The *degree_of_parallelism* number should be no larger than the number of database devices because additional worker threads do not improve encryption performance. If you do not specify *degree_of_parallelism*, SAP ASE internally defines the value based on the number of online engines, as well as how the database is distributed across various devices.
- `resume encryption` resumes the encryption process from the page where encryption was previously suspended.

  The command fails if:

  - There is an encryption process already running in SAP ASE.
  - Encryption was never started on the database.
  - The encryption process already completed.

  You can use `parallel` *`degree_of_parallelism`* with `resume encrypt`.
- `suspend encryption` terminates all encryption worker threads that are encrypting data. SAP ASE records the progress of encryption so that `resume encryption` can restart encryption where the previous encryption process stopped. SAP ASE ignores this command if there is no encryption in progress.

This example alters an existing database called `existdb` for encryption using an encryption key called `dbkey`:

```
alter database existdb encrypt with dbkey
```

The example does not specify the parallel degree, leaving it up to SAP ASE to determine how many worker threads should be initiated to encrypt `existdb` in parallel.

In addition to the parallel degree, another major factor that affects database encryption performance is the buffer pool size. A sufficient buffer cache and appropriate size of buffer pool enable SAP ASE to load a large chunk of pages into memory for every disk read, perform encryption, and write them back.

The following example shows the steps you can take to configure both the buffer cache and buffer pool size for a database called `demodb` that will be encrypted:

1. Create a specific data cache for `demodb`:

   ```
   sp_cacheconfig demodb_cache, '10M'
   ```

   This creates a named buffer cache called `demodb_cache` with 10MB of space for database pages.
2. Create the specific size of buffer pool . The buffer pool size should be 8 times of database page size. For example, the database page size is 2K by default, therefore the buffer pools size should be 8 x 2 = 16K:

   ```
   sp_poolconfig demodb_cache, '10M' , '16k'
   ```

   This creates a 10MB buffer pool of buffers with a size that is 16K in the named cache called `demodb_cache`.
3. Bind the database to the buffer cache:

   ```
   sp_bindcache demodb_cache, demo_db
   ```

Encryption Status and Progress

To obtain information on whether a database is encrypted or not, as well as how far along the encryption process has gone on a database being encrypted, use the **sp_helpdb** system procedure or the **dbencryption_status** built-in function.

- **sp_helpdb** – the syntax is the following, where *database_name* is the name of the database:

  ```
  sp_helpdb database_name
  ```
- **dbencryption_status** – use status to get information on whether a database is encrypted, and progress to find out how far along the encryption process has gone:
  - `select dbencryption_status("status", db_id("existdb"))`
  - `select dbencryption_status("progress", db_id("existdb"))`

Performance Considerations

When an existing database is being encrypted, it is still kept online. Take performance issues into consideration to mitigate the impact on user access to the database, as well as general SAP ASE response time.

Factors to take into account for good database encryption performance include:

- The number of SAP ASE engines on a multiprocessor machine
- The number of disks the database is stored across
- The buffer pool size associated with the database

Specifying the parallel degree value in **alter database** for encryption or, decryption, essentially tells SAP ASE how many worker threads to initiate when executing the operation.

Since worker threads run concurrently, it is better when they are distributed across multiple CPUs. At the same time, it is better to avoid overwhelming CPU resources, since this could reduce the general response time from SAP ASE. For this reason, take the number of SAP ASE engines into consideration when deciding on the parallel degree value.

Device I/O is a major bottleneck during database encryption. SAP ASE can tackle this from two angles:

- If every separate device is assigned a worker thread, device I/O can be carried out independently and concurrently for best throughput. Therefore parallel degree should consider the number of disks the database is stored across.
- Performance will benefit if a big chunk of pages can be processed for every device read/write. The database must be online while the encryption/decryption is in progress. For this reason, instead of allocating a proprietary buffer, existing buffer manager mechanism has to be leveraged to solve synchronization problem. In this respect, you can create sufficient buffer cache and large I/O size of buffer pool to help SAP ASE improve its encryption performance.

This example shows how to configure both the buffer cache and pool size to fully encrypt a database called `demodb`, which has its data and log distributed across 11 devices:

```
> select dbid, segmap, lstart, size, vstart, vdevno from sysusages
where dbid=db_id('demodb')

dbid   segmap lstart     size       vstart      vdevno
------ ------ ---------- ---------- ----------- -----------
   4        3          0      92160           0           1
   4        4      92160      30720           0           2
   4        3     122880     184320       92160           1
   4        4     307200      61440       30720           2
   4        3     368640     419840      276480           1
   4        4     788480      61440       92160           2
   4        3     849920     122880      696320           1
   4        4     972800     153600      153600           2
   4        3    1126400     819200      819200           1
   4        3    1945600    1638400           0           3
   4        3    3584000    1638400           0           4
   4        3    5222400    1638400           0           5
   4        3    6860800    1638400           0           6
   4        3    8499200    1638400           0           7
   4        3   10137600    1638400           0           8
   4        3   11776000    1638400           0           9
   4        3   13414400    1638400           0          10
   4        3   15052800    1638400           0          11
   4        4   16691200     204800      307200           2
```

1. Configure buffer cache and buffer pool size:
   a. Create a specific data cache for `demodb`:

      ```
      sp_cacheconfig demodb_cache, '100M'
      ```

This creates a buffer cache named `demodb_cache` that has 100MB of space for database pages.

b. Create the specific size of buffer pool, where the buffer pool size is 8 times the size of the database page size:

```
sp_poolconfig demodb_cache, '100M' , '16k'
```

Since the default database page size is 2K, the buffer pool size should be 8 X 2 = 16KB.

This creates a 100MB buffer pool with 16K buffers in the named cache `demodb_cache`.

c. Bind the database to the buffer cache:

```
sp_bindcache demodb_cache, demo_db
```

This binds the database `demo_db` to the created buffer cache `demodb_cache`.

2. Determine what parallel degree to use. In this example, there are 8 SAP ASE engines configured:

```
[Thread Pool:syb_default_pool]
```

Number of threads = 8

The maximum number of worker thread should not exceed 8.

In the meantime, with SAP ASE using 11 database devices, it needs, at most, 11 worker threads to perform device I/O in parallel. Since 11 worker threads would strain the eight engines, the parallel degree should be set to 8. However to allow SAP ASE to maintain its response time and perform other operations, avoid occupying all of its CPU resources by selecting a parallel degree of 6.

a. Make sure sufficient worker threads are configured:

```
sp_configure 'number of worker processes', 6
```

b. Alter database `demodb` for encryption:

```
alter database demodb encrypt with dbkey parallel degree 6
```

sp_who shows 6 worker threads:

```
>sp_who
fid    spid    status    loginame    origname
    hostname    blk_spid    dbname
  tempdbname    cmd
        block_xloid        threadpool
------ -------- ---------------- ---------
……
  0     16    sleeping   NULL       NULL     NULL             0
master
   master DB    ENCRYPTION CONTROLLER        0
NULL
  16    1     sleeping   NULL       NULL     NULL             0
master
   master      WORKER PROCESS             0                NULL
  16    17    sleeping   NULL       NULL     NULL             0
master
```

```
   master      WORKER PROCESS              0                  NULL
……
```

sp_helpdb can report the encryption progress and status:

```
1> sp_helpdb demodb
2> go
name     db_size    owner  dbid  created       durability
lobcomplvl inrowlen status
-------- ---------- ------ ----- ------------- ----------
---------- -------- ----------------
demodb   33000.0 MB sa      4   Sept 27, 2013 full             0
NULL     encryption in progress: 18%
```

You can also use the **dbencryption_status** function to get encryption status and progress:

```
1> select dbencryption_status("status", db_id('demodb'))
2> go
-----------
2
1> select dbencryption_status("progress", db_id('demodb'))
2> go
-----------
21
```

This shows that 21 percent of database pages has been encrypted.

You can also use **dbencryption_status** to find the progress on a specific fragment:

```
1> select dbencryption_status("progress", db_id('demodb'),
92160)
2> go
-----------
83
```

This shows that 83 percent of pages in the fragment with a logical page start of 92160 has been encrypted.

Encrypted databases consume more buffers for encryption and decryption than nonencrypted databases. If clean buffers are unavailable because of encryption and decryption:

- Increase the buffer pool size and the buffer pool wash
- Configure **housekeeper free write percent** to a value that allows the housekeeper task to wash buffers more frequently

Suspend the Encryption Process

To stop encrypting a database in the process of being encrypted, use the `suspend encrypt` option of the **alter database** command.

```
alter database database_name
    suspend encryption
```

The quiesce database Command and Fully Encrypted Databases

When you run the **quiesce database** command on a database that is being encrypted, SAP ASE puts the encryption process on hold.

You need not run the `suspend encryption` option of **alter database** after you run **quiesce database**; **quiesce database** automatically suspends the I/O operation on the database.

After the quiesce mode is released, the task of encrypting (or decrypting) resumes automatically; you need not run the **resume encryption** option in the **alter database**.

Resume the Encryption Process

To resume encrypting a database that had its encryption process interrupted or suspended, use the `resume encryption` option of the **alter database** command.

```
alter database database_name
    resume encryption [parallel degree_of_parallelism]
```

Decrypt an Encrypted Database

To decrypt a fully encrypted database, use the **alter database** command.

The syntax is:

```
alter database database_name
  {decrypt [with key_name] [parallel degree_of_parallelism]
| resume decryption [parallel degree_of_parallelism]
| suspend decryption}
```

where:

- *database_name* – is the name of the fully encrypted database you want to decrypt.
- *key_name* – (optional) is the same database encryption key you used to encrypt the database. If you specify a different key name, the command fails and SAP ASE displays an error message.
- `resume decryption` – resumes the decryption process for the database in which an earlier decryption process has been suspended. SAP ASE ignores this command if the *database_name* is already completely decrypted.
- `parallel` *degree_of_parallelism* – specifies how many worker threads to initiate for the task.
- `suspend decryption` – terminates the decryption process. SAP ASE records where the process was stopped, so that `resume decrypt` can restart the decryption process at the correct place in the database.

You must have **select** permission on the database's *key_name* to use this command.

Recover Fully Encrypted Databases

If SAP ASE cannot retrieve the database encryption key during start-up because the master or dual master key is unavailable, SAP ASE ignores the encrypted database.

SAP ASE needs access to the database encryption key to know what to do with fully encrypted databases. The database encryption key itself is also encrypted, and is decrypted by the master key.

To connect to the server after you restart SAP ASE, the password holder for the master or dual master key can set the encryption password:

```
set encryption passwd for key [dual] master
```

This allows the master key to decrypt the database encryption key, at which point the database encryption key can bring the fully encrypted database online:

```
online database encrypted_database_name
```

Database recovery then occurs as the server comes back online.

You can also set up an automatic recovery; see *Starting Adaptive Server in Unattended Start-Up Mode* in the *Encrypted Columns Users Guide*.

Back Up (Dump) a Fully Encrypted Database

Backing up a fully encrypted database is the same as for normal, unencrypted databases, since the encryption process is performed transparently.

To load a back-up dump of an encrypted database, it must use the same encryption key that was used to encrypt the dump.

The database encryption key is stored in the `master` database, outside of the database you are backing up. For this reason, the backup process is not automatically applied to the database encryption key when you execute the **dump database** command; you must independently back up the database encryption key and the master key separately from the database backup.

To back up the key values, either:

- Use the **ddlgen** utility to generate a DDL statement, or;
- Back them up directly.

Back Up the Database Encryption Key

To resume recoverability, you must back up the database encryption key, the master or dual master key, and the encrypted database.

This example uses the **ddlgen** utility to generate SQL statements on database encryption keys:

```
ddlgen -Usa -P -Sserver -TEK -Nmaster.owner.dek_name
```

The syntax is similar when generating SQL statements for the [dual] master key.

Restore (Load) Backups of Fully Encrypted Databases

Restore a fully encrypted database as you would a normal, unencrypted database.

Before you can load an encrypted database dump:

1. Restore the master key and database encryption key.
2. Create the target database for encryption using the same database encryption key you used for the database you are loading.

Use this command to restore your encrypted database, where *database_name* is the name of the encrypted database you are restoring:

```
load database database_name
```

Note: You cannot use the verification option (`load database database_name with verify only = full`) with encrypted databases. When you specify this option, Backup Server reads all rows and checks that the row formats are valid. Since Backup Server cannot understand encrypted text, the command fails and Backup Server displays an error message.

When you perform **load database** to restore an encrypted database, SAP ASE verifies that the taget database:

- Is an encrypted database. If it is not, SAP ASE displays an error message and the **load database** command fails.
- Has the correct database encryption key. If the database encryption key does not match, SAP ASE displays an error message.

Loading Behavior of Encrypted Databases

Loading behavior differs, depending on the encryption status of both the target database and the database or transaction log being restored.

Loading Behavior	Unencrypted Target Database	Encrypted Target Database	Partially Encrypted Target Database	Partially Decrypted Target Database
Unencrypted Database Dump	Allowed.	Allowed only if using the `with override` clause. Dump security status is reflected in target database.	Allowed only if using the `with override` clause. Dump status is reflected in target database.	Allowed only if using the `with override` clause. Dump security status is reflected.
Unencrypted Transaction Dump	Allowed.	Allowed. Marks the target database as partially encrypted.	Allowed. The target database retains its status as partially encrypted.	Allowed. The target database retains its status as partially encrypted.
Encrypted Database Dump	Not allowed.	Allowed.	Allowed. Dump security status is reflected in the target database.	Allowed only if using the `with override` clause. Dump security status is reflected in the target database.
Encrypted Transaction Dump	Not allowed.	Allowed.	Allowed. The target database retains its status as partially encrypted.	Not allowed.
Partially Encrypted Database Dump	Not allowed.	Allowed. Dump security status is reflected in the target database.	Allowed. The target database retains its status as partially encrypted.	Allowed only if using the `with override` clause. Dump status is reflected in the target database.
Partially Encrypted Transaction Dump	Not allowed.	Allowed. Dump security status is reflected in the target database.	Allowed. The target database retains its status as partially encrypted.	Not allowed.

Loading Behavior	Unencrypted Target Database	Encrypted Target Database	Partially Encrypted Target Database	Partially Decrypted Target Database
Partially Decrypted Database Dump	Not allowed.	Allowed only if using the `with override` clause. Dump status is reflected in the target database.	Allowed only if using the `with override` clause. Dump security status is reflected in the target database.	Allowed. The target database retains its status as partially decrypted.
Partially Decrypted Transaction Dump	Not allowed.	Not allowed.	Not allowed.	Allowed. The target database retains its status as partially decrypted.

Dropping a Database That is Being Encrypted

When you execute the **drop database** command on a database that is being encrypted or decrypted, **drop database** terminates the encryption/decryption process, searches the `sysattributes` system table, cleans up all the progress information, and then drops the database.

Mounting and Unmounting a Fully Encrypted Database

You cannot mount a database that is being encrypted or decrypted; you also cannot mount an encrypted database.

Do not use the **umount database** command on an encrypted database; the command fails and SAP ASE displays a message similar to:

```
Could not unmount encrypted database 'mydatabase'.
```

To unmount an encrypted database, decrypt it first.

Archive Databases and Full Encryption

Archive databases are read-only. The encryption syntax indicates that an archive database can load an encrypted database dump.

As with database backups and loads, restore the master key and database encryption key, and associate the DEK with the archive database.

To dump or load a fully encrypted archive database, perform the same steps as with normal databases.

To create an archive database, use:

```
create archive database database_name
    encrypt with key_name
```

where:

- *database_name* is the name of the archive database you are creating
- *key_name* is the same key that you used to encrypt the database that was backed up (dumped). SAP ASE verifies that *key_name* matches during the database dump. If it does not match, the restoration fails.

Unlike normal databases, an archive database provides a modified page section that stores page modification or allocation information due to redos/undos and transaction loading operations. When you encrypt an archive database, encrypt the data in the modified page section as well, using the database encryption key from the archive database.

There is no special permission to use the `encrypt with` option of the **create archive database** command. Users however, need **select** permission on the database encryption key to reference it as the *key_name*.

CHAPTER 7 Column Encryption

Certain datatypes can be encrypted.

You can encrypt:

- `int, smallint, tinyint`
- `unsigned int`, `unsigned smallint`, `unsigned tinyint`
- `bigint`, `unsigned bigint`
- `decimal` and `numeric`
- `float4` and `float8`
- `money`, `smallmoney`
- `date`, `time`, `smalldatetime`, `datetime`
- `char` and `varchar`
- `unichar`, `univarchar`
- `binary` and `varbinary`
- `bit`

Encrypting Columns on New Tables

To encrypt columns in a new table, use the **encrypt column** qualifier on the **create table** statement.

The following partial syntax for **create table** includes only clauses that are specific to encryption. See the *Reference Manual: Commands* for the complete syntax.

```
create table table_name
(column_name
. . .

[constraint_specification]
[encrypt [with [database.[owner].]keyname]]
[, next_column_specification . . .]
)
```

- *keyname* – identifies a key created using **create encryption key**. The creator of the table must have **select** permission on *keyname*. If *keyname* is not supplied, SAP ASE looks for a default key created using the **as default** clause on the **create encryption key**.

Note: You cannot encrypt a computed column, and an encrypted column cannot appear in an expression that defines a computed column. You cannot specify an encrypted column in the *partition_clause* of a table.

The following example creates two keys: a database default key, and another key (`cc_key`) which you must name in the **create table** command. Both keys use default values for length

and an initialization vector. The `ssn` column in the employee table is encrypted using the default key, and the `creditcard` column in the customer table is encrypted with `cc_key`:

```
create encryption key new_key as default for AES
create encryption key cc_key

create table employee_table (ssn char(15) encrypt,
   ename char(50), ...))

create table customer (creditcard char(20)
   encrypt with cc_key, cc_name char(50), ...)
```

This example creates key `k1`, which uses nondefault values for the initialization vector and random pad. The `employee` esalary column is padded with random data before encryption:

```
create encryption key k1 init_vector null pad random
create table employee (eid int, esalary money encrypt with k1, ...)
```

Specifying Encryption on select into

By default, **select into** creates a target table without encryption, even if the source table has one or more encrypted columns.

To encrypt any column in the target table, you must qualify the target column with the **encrypt** clause, as shown:

```
select [all|distinct] column_list
    into table_name
    [(colname encrypt [with [[database.][owner].]keyname]
            [, colname encrypt
            [with[[database.][owner].]keyname]])]
        from table_name | view_name
```

You can encrypt a specific column in the target table even if the data was not encrypted in the source table. If the column in the source table is encrypted with the same key specified for the target column, SAP ASE optimizes processing by bypassing the decryption step on the source table and the encryption step on the target table.

The rules for specifying encryption on a target table are the same as those for encryption specified on **create table** in regard to:

- Allowable datatypes on the columns to be encrypted
- The use of the database default key when the *keyname* is omitted
- The requirement for **select** permission on the key used to encrypt the target columns.

The following example selects the encrypted column `creditcard` from the `daily_xacts` table and stores it in encrypted form in the `#bigspenders` temporary table:

```
select creditcard, custid, sum(amount) into      #bigspenders
     (creditcard encrypt with cust.dbo.new_cc_key)
     from daily_xacts group by creditcard
     having sum(amount) > $5000
```

Note: select into requires column-level permissions, including **decrypt**, on the source table.

Encrypting Columns in Existing Tables

To encrypt columns in existing tables, use the **modify column** option on the **alter table** statement with the **encrypt** clause.

The syntax is:

```
alter table table_name modify column_name
    [encrypt [with [[database.][owner].]keyname]]
```

where *keyname* identifies a key created using **create encryption key**. The creator of the table must have **select** permission on *keyname*. If *keyname* is not supplied, SAP ASE looks for a default key created using the **as default** clause on the **create encryption key**.

See the *Reference Manual: Commands*.

There are restrictions for modifying encrypted columns:

- You cannot modify a column for encryption or decryption on which you have created a trigger. You must:
  1. Drop the trigger.
  2. Encrypt or decrypt the column.
  3. Re-create the trigger.
- You cannot change an existing encrypted column, modify a column for encryption or decryption on a table, or modify the type of an encrypted column if that column is a key in a clustered or placement index. You must:
  1. Drop the index.
  2. Alter the table/modify the type of column.
  3. Re-create the index.

You can alter the encryption property on a column at the same time you alter other attributes. You can also add an encrypted column using **alter table**.

For example:

```
alter table customer modify custid null encrypt with cc_key
alter table customer add address varchar(50) encrypt with cc_key
```

Index Creation and Constraints on Encrypted Columns

You can create an index on an encrypted column if the encryption key has been specified without any initialization vector or random padding.

An error occurs if you execute **create index** on an encrypted column that has an initialization vector or random padding. Indexes on encrypted columns are generally useful for equality and

nonequality matches. However, indexes are not useful for matching case-insensitive data, or for range searches of any data.

Note: You cannot use an encrypted column in an expression for a functional index.

In the following example, `cc_key` specifies encryption without using an initialization vector or padding. This allows an index to be built on any column encrypted with `cc_key`:

```
create encryption key cc_key
    with init_vector null

create table customer(custid int,
    creditcard varchar(16) encrypt with cc_key)

create index cust_idx on customer(creditcard)
```

You can encrypt a column that is declared as a primary or unique key.

You can define referential integrity constraints on encrypted columns when:

- Both referencing and referenced columns are encrypted with the same key.
- The key used to encrypt the columns specifies **init_vector** null and **pad random** has not been specified.

Referential integrity checks are efficient because they are performed on cipher text values.

In this example, `ssn_key` encrypts the `ssn` column in both the primary and foreign tables:

```
create encryption key ssn_key for AES
    with init_vector null
create table user_info (ssn char(9) primary key encrypt
    with ssn_key, uname char(50), uaddr char(100))
create table tax_detail (ssn char(9) references user_info encrypt
    with ssn_key, return_info text)
```

Domain Creation and Access Rules on Encrypted Columns

You can create domain rules, check constraints, or access rules on encrypted columns. However, decrypt permission is required on an encrypted column when it is used in target list, **where** clause, and so on.

This example creates the `rule_creditcard` rule on the `creditcard` column, which has a domain rule defined:

```
create encryption key cc_key
    with init_vector null

create table customer(custid int,
        creditcard varchar(16) encrypt with cc_key)

create rule rule_creditcard
as @value like '%[0-9]'
sp_bindrule rule_creditcard, creditcard
```

bcp in -C bypasses the domain rule or check constraint for encrypted columns because SAP ASE uses fast bcp with **bcp in -C**. **bcp out -C** generates error number 2929 if an access rule exists on the encrypted column. SAP ASE bypasses the rule or constraint for **insert** and **update** statements when you replicate encrypted columns with domain rules or check constraints. SAP ASE also generates error number 2929 when you replicate encrypted columns with access rules for **update**, **delete**, or **select** statements.

Decrypt Permission

Users must have decrypt permission to select plain text data from an encrypted column, or to search or join on an encrypted column.

The table owner or a user with the `sso_role` uses **grant decrypt** to grant explicit permission to decrypt one or more columns in a table to other users, groups, and roles. Decrypt permission may be implicitly granted when a procedure or view owner grants:

- **exec** permission on a stored procedure or user-defined function that selects from an encrypted column where the owner of the procedure or function also owns the table containing the encrypted column
- **decrypt** permission on a view column that selects from an encrypted column where the owner of the view also owns the table

In both cases, decrypt permission need not be granted on the encrypted column in the base table.

The syntax is:

```
grant decrypt on [owner.] table
    [( column[{,column}])]
    to user| group | role
    [with grant option]
```

Granting decrypt permission at the table or view level grants decrypt permission on all encrypted columns in the table.

To grant decrypt permission on all encrypted columns in the `customer` table, enter:

```
grant decrypt on customer to accounts_role
```

The following example shows the implicit decrypt permission of `user2` on the `ssn` column of the base table “employee”. `user1` sets up the employee table and the `employee_view` as follows:

```
create table employee (ssn varchar(12)encrypt,
    dept_id int, start_date date, salary money)

create view emp_salary as select
    ssn, salary from employee

grant select, decrypt on emp_salary to user2
```

`user2` has access to decrypted Social Security Numbers when selecting from the `emp_salary` view:

```
select * from emp_salary
```

Note: grant all on a table or view does not grant decrypt permission. Decrypt permission must be granted separately.

Users with only select permission on an encrypted column can still process encrypted columns as cipher text through the **bulk copy** command. Additionally, if an encrypted column specifies a decrypt default value, the column can be named in a select target list or in a where clause by users who do not have permission to decrypt data.

See also

- *Restrict Decrypt Permission* on page 56
- *Default Values Returned Instead of Decrypted Data* on page 57

Revoking Decryption Permission

revoke decrypt on revokes a user's decryption permission.

The syntax is:

```
revoke decrypt on [ owner] table[( column[ {,column}])] from user
    | group | role
```

For example:

```
revoke decrypt on customer from public
```

Restrict Decrypt Permission

Restrict access to private data from the database owner by setting the **restricted decrypt permission** configuration parameter.

SAP ASE protects data privacy from the powers of the administrator even if you use the master key or system encryption password for key protection. If you prefer to avoid password management and use the master key or the system encryption password to protect encryption keys, you can restrict access to private data from the database owner by setting the **restricted decrypt permission** configuration parameter. System security officers (SSOs) can use this parameter to control which users have decrypt permission. Once **restricted decrypt permission** is enabled, the SSO is the only user who receives implicit decrypt permission and who has implicit privilege to grant that permission to others. The SSO determines which users receive decrypt permission, or delegates this job to another user by granting decrypt permission with the **with grant** option. Table owners do not automatically have decrypt permission on their tables.

Users with execute permission on stored procedures or user-defined functions do not have implicit permission to decrypt data selected by the procedure or function. Users with decrypt

permission on a view column do not have implicit permission to decrypt data selected by the view.

Note: Users with aliases continue to inherit all decrypt permissions of the user to whom they are aliased. **set proxy**/**set user** statements continue to allow the administrator or database owner the decrypt permissions of the user whose identity is assumed by this command.

If you are using restricted decrypt permission, you can assign the privileges for creating the task's schema and managing keys as follows:

- System security officer – configures restricted decrypt permission, creates encryption keys, grants **select** permission on keys to the database owner, and grants decrypt permission to the end user.
- Database owner – creates the schema and loads data.

See also

- *Protect Encryption Keys with the Master Key* on page 16
- *Decrypt Permission* on page 55

Default Values Returned Instead of Decrypted Data

When users who are not permitted to see confidential data run queries against encrypted columns, they see the decrypt defaults instead of the decrypted data. Decrypt defaults allow legacy applications and reports to run without error, even for users who are not permitted to see confidential data.

See also

- *Decrypt Permission* on page 55

Defining Decrypt Defaults

The **decrypt_default** parameter for **create table** and **alter table** allows an encrypted column to return a user-defined value when a user without decrypt permission attempts to select information from the encrypted column.

Doing so avoids error message 10330:

```
Decrypt permission denied on object <table_name>,
     database <database name>, owner <owner name>
```

Using decrypt defaults on encrypted columns allows existing reports to run to completion without error, and allows users to continue seeing the information that is not encrypted. For example, if the `customer` table contains the encrypted column `creditcard`, you can design the table schema so that:

```
select * from customer
```

Returns the value "**********" instead of returning the credit card data to users who lack decrypt permission.

Add a decrypt default on a new column with **create table**. The partial syntax for **create table** is:

```
create table table_name (column_name datatype
        [[encrypt [with keyname]] [decrypt_default value]], ....)
```

- **decrypt_default** – specifies that this column returns a default value on a **select** statement for users who do not have decrypt permissions.
- *value* – is the value SAP ASE returns on select statements instead of the decrypted value. A constant-valued expression cannot reference a database column but it can include a user-defined function which itself references tables and columns. The value can be NULL on nullable columns only, and the value must be convertible into the column's datatype.

For example, the `ssnum` column for table `t2` returns "??????????" when a user without decrypt permissions selects it:

```
create table t2 (ssnum char(11)
     encrypt decrypt_default '???????????', ...)
```

To add encryption and a decrypt default value to an existing column not previously encrypted, use:

```
alter table table_name modify column_name [type]
     [[encrypt [with keyname]] [decrypt_default value]], …
```

This example modifies the `emp` table to encrypt the `ssn` column and specifies decrypt default:

```
alter table emp modify ssn encrypt
     with key1 decrypt_default '000-00-0000'
```

To add a decrypt default to an existing encrypted column or change the decrypt default value on a column that already has a decrypt default, use:

```
alter table table_name replace column_name decrypt_default value
```

This example adds a decrypt default to the `salary` column, which is already encrypted:

```
alter table employee replace salary
     decrypt_default $0.00
```

This example replaces the previous `decrypt_default` value with a new value and uses a user-defined function (UDF) to generate the default value:

```
alter table employee replace salary
     decrypt_default dbo.mask_salary()
```

To remove a decrypt default from an encrypted column without removing the encryption property, use:

```
alter table table_name replace column_name drop decrypt_default
```

This example removes the decrypt default for `salary` without removing the encryption property:

```
alter table employee replace salary
     drop decrypt_default
```

Permissions and Decrypt Default

You must grant decrypt permission on encrypted columns before users or roles can select or search on encrypted data in those columns. If an encrypted column has a decrypt default attribute, users without decrypt permission can run queries that select or search on these columns, but the plain text data is not displayed and is not used for searching.

In this example, the owner of table `emp` allows users with the **hr_role** to view `emp.ssn`. Because the `ssn` column has a decrypt default, users who have only **select** permission on `emp` and who do not have the **hr_role** see the *decrypt_default* value only and not the actual decrypted data.

```
create table emp (name char(50), ssn (char(11) encrypt
decrypt_default '000-00-000', ...)
grant select permission on table emp to public
grant decrypt on emp(ssn) to hr_role
```

If you have the **hr_role** and select from this table, you see the values for `ssn`:

```
select name, ssn from emp
```

```
name                            ssn
------------------------------  ------------
Joe Cool                        123-45-6789
Tinna Salt                      321-54-9879
```

If you do not have the **hr_role** and **select** from the table, you see the decrypt default:

```
select name, ssn from emp
```

```
name                               ssn
------------------------------  -----------
Joe Cool                        000-00-0000
Tinna Salt                      000-00-0000
```

order by clauses have no effect on the result set if you do not have the **hr_role** for this table.

Columns with Decrypt Default Values

There are no restrictions on how you use columns with **decrypt default** in a query. You can use them in a target list expression, **where** clause, **order by**, **group by**, or subquery.

Although expressions on the decrypt default constant value may not have a practical use, placing a decrypt default on a column does not impose any syntactic restrictions on use of the column in a Transact-SQL statement.

This example uses a **select** statement on a column with a decrypt default value in the target list:

```
create table emp_benefits (col1 name char(30),
     salary float encrypt decrypt_default -99.99)
```

```
select salary/12 as monthly_salary from emp_benefits
     where name = 'Bill Smith'
```

When you perform the **select** statement against this table, but do not have decrypt permission, you see:

```
monthly_salary
--------------------
8.332500
```

When SAP ASE returns a column's decrypt default value on a **select into** command, this decrypt default value is inserted into the target table. However, the target column does not inherit the decrypt default property. You must use **alter table** to specify a decrypt default on the target table.

Use **sp_checksource** to view decrypt default source text defined on encrypted columns.

Decrypt Default Columns and Query Qualifications

If you use a column with the decrypt default property in a **where** clause, the qualification evaluates to false if you do not have **decrypt** permission.

These examples use the `emp` table described above. Only users with the **hr_role** have decrypt permission on `ssn`.

- If you have the **hr_role** and issue the following query, SAP ASE returns one row.

  ```
  select name from emp where ssn = '123-456-7890'
  ```

  ```
  name
  ------------------------------
  Joe Cool
  ```

- If you do not have the **hr_role**, SAP ASE returns no rows:

  ```
  select name from emp where ssn = '123-456-7890'
  ```

  ```
  name
  ------------------------------
  (0 rows affected)
  ```

- If you have the **hr_role** and include an **or** statement on a nonencrypted column, SAP ASE returns the appropriate rows:

  ```
  select name from emp where ssn = '123-456-7890' or
  name like 'Tinna%'
  ```

  ```
  name
  ------------------------------
  Joe Cool
  Tinna Salt
  ```

- If you do not have the **hr_role** and issue the same command, SAP ASE returns only one row:

  ```
  select name from emp where ssn = '123-456-7890' or name like
  'Tinna%'
  ```

```
name
------------------------------
Tinna Salt
```

In this case, the qualification against the encrypted column with the decrypt default property evaluates to false, but the qualification against the nonencrypted column succeeds.

If you do not have **decrypt** permission on an encrypted column, and you issue a **group by** statement on this column with a decrypt default, SAP ASE groups by the decrypt default constant value.

See also

- *Encrypted Columns Process* on page 71

decrypt default and Implicit Grants

If you do not have explicit or implicit permission on a table, SAP ASE returns the **decrypt default** value.

In this example (using the `emp` table), the database owner creates the `p_emp` procedure which selects from the `emp` table that he or she owns:

```
create procedure p_emp as
     select name, ssn from emp
grant exec on p_emp to corp_role
```

Because you have the **corp_role**, you have implicit **select** and **decrypt** permission on `emp`

```
exec p_emp
```

```
name                           ssn
-----------------------------  ------------
Tinna Salt                     123-45-6789
Joe Cool                       321-54-9879
```

If the `emp` table and `p_emp` stored procedure have been created by different users, you must have **select** permission on `emp` to avoid permissions errors. If you have **select** permission but not **decrypt** permission, SAP ASE returns the **decrypt default** value of `emp.ssn`.

In this next example, "joe," who does not own the database, creates the `v_emp` view, which selects from the `emp` table. Any permissions granted on the view are not implicitly applied to the base table.

```
create view v_emp as
     select name, ssn from emp
grant select on v_emp to emp_role
grant select on emp to emp_role
grant decrypt on v_emp to emp_role
```

Although you have the **emp_role**, when you issue:

```
select * from joe.v_emp
```

SAP ASE returns the following because **decrypt** permission on `dbo.emp.ssn` has not been granted to the **emp_role**, and there is no implicit **grant** to **emp_role** on `dbo.emp.ssn`:

```
name                        ssn
--------------------------  ---------------
Tinna Salt                  000-00-0000
Joe Cool                    000-00-0000
```

decrypt default and insert, update, and delete Statements

The **decrypt default** parameter does not affect target lists of **insert** and **update** statements. If you use a column with a decrypt default value in the **where** clause of an **update** or **delete** statement, SAP ASE may not update or delete any rows.

For example, when using the `emp` table and permissions from the previous examples, if you do not have the **hr_role** and issue the following query, SAP ASE does not delete the user's name:

```
delete emp where ssn = '123-45-6789'
(0 rows affected)
```

Decrypt default attributes may indirectly affect inserting and updating data into an application, particularly one with a graphical user interface (GUI) process:

1. Selects data.
2. Allow a user to update any of the data.
3. Applies the changed row back to the same or a different table.

If the user does not have **decrypt** permission on the encrypted columns, the application retrieves the decrypt default value and may automatically write the unchanged decrypt default value back to the table. To avoid overwriting valid data with decrypt default values, use a check constraint to prevent these values from being automatically applied. For example:

```
create table customer (name char(30)),
cc_num int check (cc_num != -1)
encrypt decrypt_default -1
```

If the user does not have **decrypt** permission on `cc_num` and selects data from the `customer` table, this data appears:

```
name                               cc_num
---------------------              ------------
Paul Jones                         -1
Mick Watts                         -1
```

However, if the user changes a name and updates the database, and the application attempts to update all fields from the values displayed, the default value for `cc_num` causes SAP ASE to issue error 548:

```
"Check constraint violation occurred, dbname =
<dbname>, table name = <table_name>, constraint name =
<internal_constraint _name>"
```

Setting a check constraint protects the integrity of the data. For a better solution, you can filter these updates when you write the application's logic.

Removing Decrypt Defaults

Multiple commands allow you to remove decrypt defaults.

Remove the decrypt default using any of these commands:

- **drop table**
- **alter table .. modify .. drop col**
- **alter table .. modify .. decrypt**
- **alter table .. replace .. drop decrypt_default**

For example, to remove the decrypt default attribute from the `ssn` column, enter:

```
alter table emp replace ssn drop decrypt_default
```

If you do not have the **hr_role** and select from the **emp** table after the table owner removed the decrypt default, SAP ASE returns error message 10330.

Length of Encrypted Columns

During **create table**, **alter table**, and **select into** operations, SAP ASE calculates the maximum internal length of the encrypted column. To make decisions on schema arrangements and page sizes, the database owner must know the maximum length of the encrypted columns.

AES is a block-cipher algorithm. The length of encrypted data for block-cipher algorithms is a multiple of the block size of the encryption algorithm. For AES, the block size is 128 bits, or 16 bytes. Therefore, encrypted columns occupy a minimum of 16 bytes with additional space for:

- The initialization vector. If used, the initialization vector adds 16 bytes to each encrypted column. By default, the encryption process uses an initialization vector. Specify **init_vector null** on **create encryption key** to omit the initialization vector.
- The length of the plain text data. If the column type is `char, varchar, binary`, or `varbinary`, the data is prefixed with 2 bytes before encryption. These 2 bytes denote the length of the plain text data. No extra space is used by the encrypted column unless the additional 2 bytes result in the cipher text occupying an extra block.
- A sentinel byte, which is a byte appended to the cipher text to safeguard against the database system trimming trailing zeros.

Table 1. Datatype Length for Encrypted Columns

User-specified column type	Input data length	Encrypted column type	Maximum encrypted data length (no init vector)	Actual encrypted data length (no init vector)	Maximum encrypted data length (with init vector)	Actual encrypted data length (with init vector)
`bigint`	8	`varbinary`	17	17	33	33
`unsigned bigint`	8	`varbinary`	17	17	33	33
`tinyint, smallint, or int` (signed or unsigned)	1, 2, or 4	`varbinary`	17	17	33	33
`tinyint, smallint, or int` (signed or unsigned)	0 (null)	`varbinary`	17	0	33	0
`float, float(4), real`	4	`varbinary`	17	17	33	33
`float, float(4), real`	0 (null)	`varbinary`	17	0	33	0
`float(8), double`	8	`varbinary`	17	17	33	33
`float(8), double`	0 (null)	`varbinary`	17	0	33	0
`numeric(10,2)`	3	`varbinary`	17	17	33	33
`numeric (38,2)`	18	`varbinary`	33	33	49	49
`numeric (38,2)`	0 (null)	`varbinary`	33	0	49	0

User-specified column type	Input data length	Encrypted column type	Maximum encrypted data length (no init vector)	Actual encrypted data length (no init vector)	Maximum encrypted data length (with init vector)	Actual encrypted data length (with init vector)
char, varchar (100)	1	varbinary	113	17	129	33
char, varchar(100)	14	varbinary	113	17	129	33
char, varchar(100)	15	varbinary	113	33	129	49
char, varchar(100)	31	varbinary	113	49	129	65
char, varchar(100)	0 (null)	varbinary	113	0	129	0
binary, varbinary(100)	1	varbinary	113	17	129	33
binary, varbinary(100)	14	varbinary	113	17	129	33
binary, varbinary(100)	15	varbinary	113	33	129	49
binary, varbinary(100)	31	varbinary	113	49	129	65
binary, varbinary(100)	0 (null)	varbinary	113	0	65	0
unichar(10)	2 (1 unichar character)	varbinary	33	17	49	33

User-specified column type	Input data length	Encrypted column type	Maximum encrypted data length (no init vector)	Actual encrypted data length (no init vector)	Maximum encrypted data length (with init vector)	Actual encrypted data length (with init vector)
unichar(10)	20 (10 unichar characters)	varbinary	33	33	49	49
univarchar(20)	20 (10 unichar characters)	varbinary	49	33	65	49
date	4	varbinary	17	17	33	33
time	4	varbinary	17	17	33	33
time	null	varbinary	17	0	33	0
smalldatetime	4	varbinary	17	17	33	33
datetime	8	varbinary	17	17	33	33
smallmoney	4	varbinary	17	17	33	33
money	8	varbinary	17	17	33	33
money	null	varbinary	17	0	33	0
bit	1	varbinary	17	17	33	33

Note:

- The `timestamp` datatype is not supported by SAP ASE.
- `char` and `binary` are treated as variable-length datatypes and are stripped of blanks and zero padding before encryption. Any blank or zero padding is applied when the data is decrypted.
- The column length on disk increases for encrypted columns, but the increases are invisible to tools and commands. For example, **sp_help** shows only the original size.

Encrypted Columns Audits

You can perform and manage encrypted column audits.

See *Auditing* in the *Security Administration Guide*.

Event Names and Numbers

You can query the audit trail for specific audit events.

Use **audit_event_name** with *event id* as a parameter.

```
audit_event_name(event_id)
```

See *Auditing* in the *Security Administration Guide* for values that appear in the event column of `sysaudits`.

Passwords Masked in Command Text Auditing

Passwords are masked in audit records.

For example, if the SSO has enabled command text auditing (that is, auditing all actions of a particular user) for user “alan” in database `db1`:

```
sp_audit "cmdtext", "alan", "db1", "on"
```

And “alan” issues this command:

```
create encryption key key1 with passwd "bigsecret"
```

SAP ASE writes the following SQL text to the `extrainfo` column of the `audit` table:

```
"create encryption key key1 with passwd "xxxxxx"
```

Auditing Actions of the Key Custodian

You can audit all actions in which the **keycustodian_role** is active.

The syntax is:

```
sp_audit "all", "keycustodian_role", "all", "on"
```

Performance Considerations

Encryption is a resource-intensive operation that may introduce a performance overhead to your application in terms of CPU usage and the elapsed time of commands that use encrypted columns.

The amount of overhead depends on the number of CPUs and SAP ASE engines, the load on the system, the number of concurrent sessions accessing the encrypted data, and the number of encrypted columns referenced in a query. The encryption key size and the length of the encrypted data are also factors. In general, the larger the key size and the wider the data, the higher the CPU usage in the encryption operation.

The elapsed time depends on whether the SAP ASE optimizer can make use of an encrypted column.

See also

- *Creating Column Encryption Keys* on page 7
- *Dropping Column Encryption Keys* on page 11
- *Encrypted Columns Process* on page 71

Indexes on Encrypted Columns

You can create an index on an encrypted column if the column's encryption key does not specify the use of an initialization vector or random padding.

Using an initialization vector or random padding results in identical data being encrypting to different patterns of cipher text, which prevents an index from enforcing uniqueness and from performing equality matching of data in cipher text form.

Indexes on encrypted data are useful for equality and nonequality matching of data but not for data ordering, range searches, or finding minimum and maximum values. If SAP ASE is performing an order-dependent search on an encrypted column, it cannot execute an indexed lookup on encrypted data. Instead, the encrypted column in each row must be decrypted and then searched. This slows data processing.

Sort Orders and Encrypted Columns

If you use a case-insensitive sort order, SAP ASE cannot use an index on an encrypted `char` or `varchar` column when performing a join with another column or a search based on a constant value. This is also true of an accent-insensitive sort order.

For example, For example, in a case-insensitive search, the string `abc` matches all strings in the following range: `abc`, `Abc`, `ABc`, `ABC`, `AbC`, `aBC`, `aBc`, `abC`. SAP ASE must compare `abc` against this range of values. By contrast, a case-sensitive comparison of the string `abc` to the column data matches only identical column values, that is, columns containing `abc`. The main difference between case-insensitive and case-sensitive column lookups is that case-

insensitive matching requires SAP ASE to perform a range search whereas case-sensitive matching requires an equality search.

An index on a nonencrypted character column orders the data according to the defined sort order. For encrypted columns, the index orders the data according to the cipher text values, which bears no relationship to the ordering of plain text values. Therefore, an index on an encrypted column is useful only for equality and non-equality matching and not for searching a range of values. `abc` and `Abc` encrypt to different cipher text values and are not stored adjacently in an index.

When SAP ASE uses an index on an encrypted column, it compares column data in cipher text form. For case sensitive data, you do not want `abc` to match `Abc`, and the cipher text join or search based on equality matching works well. SAP ASE can join columns based on cipher text values and can efficiently match **where** clause values. In this example, the `maidenname` column is encrypted:

```
select account_id from customer
     where cname = 'Peter Jones'
     and maidenname = 'McCarthy'
```

Providing that `maidenname` has been encrypted without use of an initialization vector or random padding, SAP ASE encrypts `McCarthy` and performs a cipher text search of `maidenname`. If there is an index on `maidenname`, the search uses of the index.

Joins on Encrypted Columns

SAP ASE optimizes the joining of two encrypted columns by performing cipher text comparisons under certain circumstances.

- The joining columns have the same datatype. For cipher text comparisons, `char` and `varchar` are considered to be the same datatypes, as are `binary` and `varbinary`.
- For `int` and `float` types, the columns have the same length. For `numeric` and `decimal` types, the columns must have the same precision and scale.
- The joining columns are encrypted with the same key.
- The joining columns are not part of an expression. For example, you cannot perform a cipher text join on a join where **t.encr_col1 = s.encr_col1 +1**.
- The encryption key was created with **init_vector** and **pad** set to NULL.
- The join operator is '=' or '<>'.
- The data uses the default sort order.

This example sets a schema to join on cipher text:

```
create encryption key new_cc_key for AES
        with init_vector NULL
create table customer
        (custid int,
        creditcard char(16) encrypt with new_cc_key)
create table daily_xacts
       (cust_id int, creditcard char(16) encrypt with
        new_cc_key, amount money........)
```

You can also set up indexes on the joining columns:

```
create index cust_cc on customer(creditcard)create index daily_cc on
daily_xacts(creditcard)
```

SAP ASE executes the following **select** statement to total a customer's daily charges on a credit card without decrypting the `creditcard` column in either the `customer` or the `daily_xacts` table.

```
select sum(d.amount) from daily_xacts d, customer c
     where d.creditcard = c.creditcard and
     c.custid = 17936
```

Search Arguments and Encrypted Columns

For equality and nonequality comparison of an encrypted column to a constant value, SAP ASE optimizes the column scan by encrypting the constant value once, rather than decrypting the encrypted column for each row of the table.

For example:

```
select sum(d.amount) from daily_xacts d
     where creditcard = '123-456-7890'
```

SAP ASE cannot use an index to perform a range search on an encrypted column; it must decrypt each row before performing data comparisons. If a query contains other predicates, SAP ASE selects the most efficient join order, which often leaves searches against encrypted columns until last, on the smallest data set.

If your query has more than one range search without a useful index, write the query so that the range search against the encrypted column is last. This example which searches for the Social Security Numbers of taxpayers earning more than $100,000 in Rhode Island positions the `zipcode` column before the range search of the encrypted adjusted gross income column:

```
select ss_num from taxpayers
       where zipcode like '02%' and
       agi_enc > 100000
```

Referential Integrity Searches

Referential integrity probes match at the cipher text level if both the following are true:

- The datatypes of the primary key and foreign key match according to the rules described above.
- The encryption of the primary and foreign keys meets the key requirements for joining columns.

Movement of Encrypted Data as Cipher Text

As much as possible, SAP ASE optimizes the copying of encrypted data by copying cipher text instead of decrypting and reencrypting data. This applies to **select into** commands, bulk copying, and replication.

Access Encrypted Data

SAP ASE automatically performs encryption and decryption when you process data in encrypted columns. SAP ASE encrypts data when you update or insert data into an encrypted column, and decrypts data when you select it or use it in a **where** clause.

Encrypted Columns Process

When you issue a **select**, **insert**, **update**, or **delete** command against an encrypted column, SAP ASE automatically encrypts or decrypts the data using the encryption key associated with the encrypted column.

- When you issue an **insert** or **update** on an encrypted column:
  - If you do not have **insert** or **update** permission on the encrypted column, the command fails.
  - If the column is encrypted by a key with a user-specified password, SAP ASE expects the password to be available. If the user-specified password has not been set, the command fails.
  - SAP ASE decrypts the encryption key.
  - SAP ASE encrypts the data using the column's encryption key.
  - SAP ASE inserts the `varbinary` cipher text data into the table.
  - After the insert or update, SAP ASE clears the memory holding the plain text. At the end of the statement, it clears the memory holding the raw encryption keys.
- When you issue a **select** command on data from an encrypted column:
  - The command fails if you do not have **select** permission on the encrypted column.
  - If the encryption key is associated with a column encrypted with a user-specified password, SAP ASE expects the password to be available. If the user-specified password has not been set, the **select** statement fails. Otherwise, SAP ASE decrypts the encryption key.
  - The decryption of the selected data succeeds if you have **decrypt** permission on the column, and SAP ASE returns plain text data to the user.
  - If a decrypt default has been declared on the encrypted column and if you do not have **decrypt** permission on the column, SAP ASE returns the decrypt default value.
- When you include encrypted columns in a **where** clause:
  - If you do not have **decrypt** permission on the column, and the column includes a decrypt default, the **where** clause predicate evaluates to false.
  - When possible, SAP ASE makes the comparison without decrypting the data if:
    - The **where** clause joins an encrypted column with another column encrypted by the same key without use of an initialization vector or random pad
    - The column data is being matched with an equality or an inequality condition to a constant value

See also

- *Access Encrypted Data with a User Password* on page 82
- *Decrypt Default Columns and Query Qualifications* on page 60
- *Performance Considerations* on page 68

Permissions for Decryption

To see or process decrypted data, users must have certain permissions.

User must have:

- **select** and **decrypt** permissions on the column used in the target list and in **where**, **having**, **order by**, **group by**, and other such clauses
- A password used to encrypt the key if you use the **passwd *password_phrase*** clause with the **create** or **alter encryption key** commands.

Configuring SAP ASE for **restricted decrypt permission** restricts implicit decrypt permissions. You must explicitly grant table owners **decrypt** permission to enable them to select from an encrypted column on tables that they own. **execute** permission on a stored procedure or **select** permission on a view does not implicitly grant users **decrypt** permission on the underlying encrypted data through an ownership chain. The user must also have explicit **decrypt** permission on the base table.

Drop Encryption

If you are a table owner, you can use **alter table** with the **decrypt** option to drop encryption on a column.

For example, to drop encryption on the `creditcard` column in the `customer` table, enter:

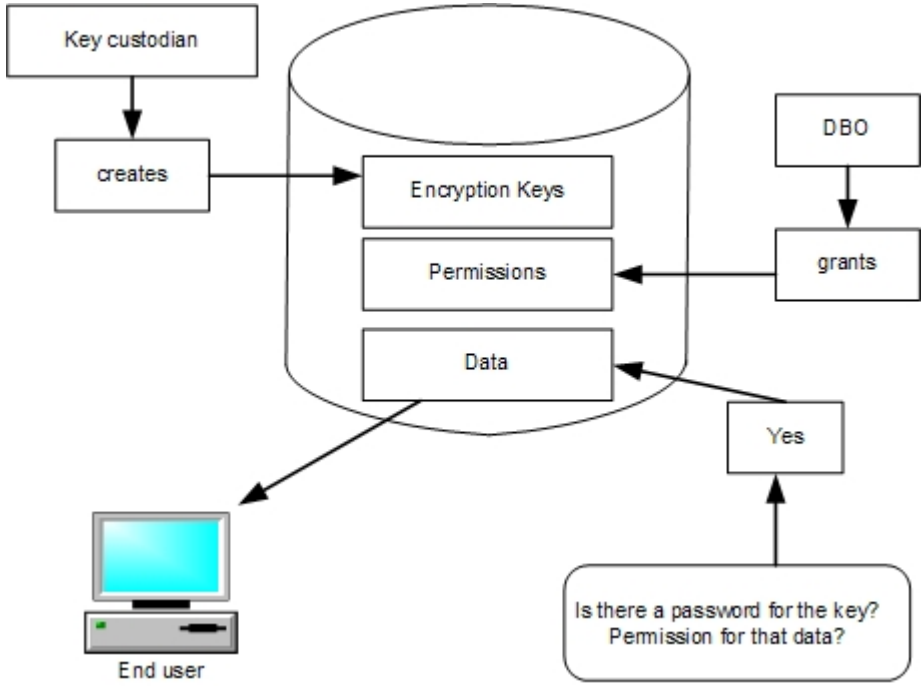
```
alter table customer modify creditcard decrypt
```

If the `creditcard` column was encrypted by a key with an explicit user password, you would need to set that password first.

CHAPTER 8 Role of the Key Custodian

The key custodian, who must be assigned the **keycustodian_role**, maintains encryption keys. Using the **keycustodian_role** role allows you to separate the duties for administering confidential data by ensuring that no administrator has implicit access to data.

This figure illustrates that the database owner, as the schema owner, controls permissions for accessing the data, but has no access without knowledge of the key's password. The key custodian, however, administers keys and their passwords, but has no permissions on the data. Only the qualified end user, with permissions on the data and knowledge of the encryption key's password, can access the data.



The system administrator and database owner do not have implicit key management responsibilities. SAP ASE provides the system role **keycustodian_role** so that the SSO need not assume all encryption responsibility. The key custodian owns the encryption keys, but should have no explicit or implicit permissions on the data. The database owner grants users access to data through column permissions, and the key custodian allows users access to the key's password. **keycustodian_role** is automatically granted to **sso_role** and can be granted by a user with the **sso_role**.

The key custodian can:

- Create and alter encryption keys.
- Assign as the database default key a key he or she owns, as long as he or she also owns the current default key, if one exists.
- Set up key copies for designated users, allowing each user access to the key through a chosen password or a login password.
- Share key encryption passwords with end users.
- Grant schema owners select access to encryption keys on keys owned by the key custodian.
- Create the master key or set the system encryption password.
- Recover encryption keys.
- Drop his or her own encryption keys.
- Change ownership of keys he or she owns.

You can have multiple key custodians, who each own a set of keys. The key custodian grants the schema owner permission to use the keys on **create table**, **alter table**, and **select into**, and may disclose the key password to privileged users or allow users to associate key copies with a personal password or a login password. The key custodian can work with a "key recoverer" to recover keys in the event of a lost password or disaster. If the key custodian leaves the company, the SSO can use the **alter encryption key** command to change key ownership to a new key custodian.

See also

- *Creating Column Encryption Keys* on page 7
- *Dropping Column Encryption Keys* on page 11

Users, Roles, and Data Access

User-specified passwords on encryption keys ensure that data privacy is protected from the system administrator.

- The key custodian can own the keys, but not see the data.
- The database owner can own the schema, but not the data.
- A user can see and process the data because of:
  - Key access, granted by the key custodian
  - Data access, granted by the table owner

Role	Can Create Encryption Key?	Can Use Key in a Schema Definition?	Can Decrypt Encrypted Data?
sso_role	Yes	No, requires **create table** permission	No. User with role may have knowledge of password, but requires select permission on table (SSO has implicit decrypt permission).
sa_role	No, requires **create encryption key** permission	Yes, but must be granted **select** permission on the key	No, requires knowledge of password
keycustodian_role	Yes	No, requires **create table** permission	No. User with role may have knowledge of password, but requires **decrypt** and **select** permission.
database owner or schema owner	No, requires **create encryption key** permission	Yes, but must be granted **select** permission on the key	No, requires knowledge of password.
User	No	No	Yes, but must be granted **decrypt** or **select** permission and have knowledge of key's password.

CHAPTER 9 Key Protection Using User-Specified Passwords

Use **create encryption key** to associate a password with a key.

The syntax is:

```
create encryption key [[db.][owner].]keyname  [as default]
    [for algorithm_name]
    [with {[keylength num_bits]
    [passwd 'password_phrase']
    [init_vector {NULL | random}]
    [pad {NULL | random}]}]
```

where *password_phrase* is a quoted alphanumeric string of up to 255 bytes in length that SAP ASE uses to generate thekey encryption key (KEK).

SAP ASE does not save the user-specified password. It saves a string of validating bytes known as the "salt" in `sysencryptkeys.eksalt`, which allows SAP ASE to recognize whether a password used on a subsequent encryption or decryption operation is legitimate for a key. You must supply the password to SAP ASE before you can access any column encrypted by **keyname**.

When you create an encryption key, its entry in the `sysencryptkeys` table is known as the base key. For some users and applications, the *base key*, encrypted by either the master key, the system encryption password, or an explicit password, is sufficient. Any explicit password is shared among users requiring access to the key. Additionally, you can create *key copies* for different users and applications. Each key copy can be encrypted by an individual password and is stored as a separate row in `sysencryptkeys`. An encryption key is always represented by one base key and zero or more key copies.

This example shows how to use passwords on keys, and the key custodian's function in setting up encryption. The password on the key is shared among all users who have a business need to process encrypted data.

1. Key custodian "razi" creates an encryption key:
   ```
   create encryption key key1
        with passwd 'Worlds1Biggest6Secret'
   ```
2. "razi" distributes the password to all users who need access to encrypted data.
3. Each user enters the password before processing tables with encrypted columns:
   ```
   set encryption passwd 'Worlds1Biggest6Secret'
        for key razi.key1
   ```
4. If the key is compromised because an unauthorized user gained access to the password, "razi" alters the key to change the password.

See also

- *Protect Keys with User-Specified Passwords* on page 18

Change a Key's Protection Method

You can use the **alter encryption key** command to change the protection method for an encryption key.

The syntax is:

```
alter encryption key [[database.database][owner].] keyname
    [with {passwd {'old_passwd' | system_encr_passwd
        | login_passwd} | master key}]
    modify encryption
    [with [{passwd {'old_passwd' | system_encr_passwd |
login_passwd}
        | master key}] [[no] dual_control]]
```

where:

- *keyname* – identifies a column encryption key.
- **with passwd '*old_password*'** – specifies the user-defined password previously specified to encrypt the base key or the key copy with a **create encryption key** or **alter encryption key** statement. The password can be up to 255 bytes long. If you do not specify **with passwd** on the base key, the default is the master key or the system encryption password.
- **with passwd '*new_password*'** – specifies the new password SAP ASE uses to encrypt the column encryption key or key copy. The password can be up to 255 bytes long. If you do not specify **with passwd** and you are encrypting the base key, the default is **system_encr_passwd**.
- **system_encr_passwd** – is the default encryption password. You cannot modify the base key to be encrypted with the system encryption password if one or more key copies already exist. This restriction prevents the key custodian from inadvertently exposing an encryption key to access by an administrator after the key custodian has set up the key for restricted use by individual users. You cannot modify key copies to encrypt using the system encryption password.
- *login_passwd* – is the login password of the current session. You cannot modify the base key to use *login_password* for encryption. A user can modify his own key copy to encrypt with his login password.
- **master key** – in the first instance indicates that the current encryption uses the master key. In the second instance, it indicates that the KEK or CEK must be re-encrypted with the master key.

Example 1: In this example, the key custodian alters the base key because the password was compromised or a user who knew the password left the company.

1. Key custodian "razi" creates an encryption key:

```
create encryption key key1
     with passwd 'MotherOfSecrets'
```

2. “razi” shares the password on the base key with “joe” and “bill”, who need to process the encrypted data (no key copies are involved).
3. “joe” leaves the company.
4. “razi” alters the password on the encryption key and then shares it with “bill”, and “pete”, who replaceses “joe.” The data does not need to be reencrypted because the underlying key has not changed, just the way the key is protected. The following statement decrypts `key1` using the old password and reencrypts it with the new password:

```
alter encryption key key1
    with passwd 'MotherOfSecrets'
    modify encryption
    with passwd 'FatherOfSecrets'
```

Example 2: Use the master key to encrypt an existing CEK “k2”:

```
alter encryption key k2
        with passwd 'goodbye'
        modify encryption
        with master key
```

Example 3: Re-encrypt an existing CEK “k3” that is currently encrypted by the master key, to use dual control:

```
alter encryption key k3
        modify encryption
        with master key
        dual_control
```

Note: You can omit **with master key** in this example to achieve the same encryption.

Example 4: Re-encrypt an existing CEK “k4” that is currently encrypted by the master key and password “k4_password”, to remove dual control. The CEK and all its key copies are controlled by a single key derived from “k4_new_password”:

```
alter encryption key k4
        with passwd 'k4_password'
        modify encryption
        with passwd 'k4_new_password'
        no dual_control
```

Example 5: Encrypt an existing CEK “k5” that is currently encrypted by the master key, for dual control encrypted by the master key and password “k5_password”:

```
alter encryption key k5
        modify encryption
        with passwd 'k5_password'
        dual_control
```

Example 6: Encrypt a CEK for dual control by the master key and password “k6_password”:

```
create encryption key k6
        with passwd 'k6_password'
        dual_control
```

For user “ned”, encrypt his existing key copy of CEK “k6” that is currently encrypted with dual control by the master key and password “k6_password”, for dual control by the master key and password “k6_ned_password”:

```
alter encryption key k6
        with passwd 'k6_password'
        add encryption
        with passwd 'k6_ned_password'
        for user ned
```

Note: User “ned” cannot change the dual control property of his key copy.

Example 7: Encrypt a CEK “k7” currently encrypted by the master and dual master key, to use the system encryption password:

```
alter encryption key k7
        modify encryption
        with passwd system_encr_passwd
        no dual control
```

See also

- *Protect Encryption Keys with Dual Control* on page 18

Create Key Copies

The key custodian may need to make a copy of the key temporarily available to an administrator or an operator who must load data into encrypted columns or databases. Because this operator does not otherwise have permission to access encrypted data, he or she should not have permanent access to a key.

You can make key copies available to individual users as follows:

- The key custodian uses **create encryption key** to create a key with a user-defined password. This key is known as the base key.
- The key custodian uses **alter encryption key** to assign a copy of the base key to an individual user with an individual password.

This syntax shows how to add a key encrypted using an explicit password for a designated user:

```
alter encryption key [database.[ owner ].]key
        with passwd 'base_key_password'
        add encryption with passwd 'key_copy_password'
        for user_name ''
```

where:

- *base_key_password* – is the password used to encrypt the base key, and may be known only by the key custodian. The password can be upto 255 bytes in length. SAP ASE uses the first password to decrypt the base column-encryption key.

- *key_copy_password* – the password used to encrypt the key copy. The password cannot be longer than 255 bytes. SAP ASE makes a copy of the decrypted base key, encrypts it with a key encryption key derived from the *key_copy_password*, and saves the encrypted base key copy as a new row in `sysencryptkeys`.
- *user_name* – identifies the user for whom the key copy is made. For a given key, `sysencryptkeys` includes a row for each user who has a copy of the key, identified by their user ID (`uid`).
- The key custodian adds as many key copies as there are users who require access through a private password.
- Users can alter their copy of the encryption key to encrypt it with a different password.

The following example illustrates how to set up and use key copies with an encrypted column:

1. Key custodian "razi" creates the base encryption key with a user-specified password:
   ```
   create encryption key key1 with passwd 'WorldsBiggestSecret'
   ```
2. "razi" grants **select** permission on `key1` to database owner for schema creation:
   ```
   grant select on key key1 to dbo
   ```
3. database owner creates schema and grants table and column-level access to "bill":
   ```
   create table employee (empname char(50), emp_salary money encrypt
   with      razi.key1, emp_address varchar(200))
        grant select on employee to bill
        grant decrypt on employee(emp_salary) to bill
   ```
4. Key custodian creates a key copy for "bill" and gives "bill" the password to his key copy. Only the key custodian and "bill" know this password.
   ```
   alter encryption key key1 with passwd 'WorldsBiggestSecret'
        add encryption with passwd 'justforBill'
        for user 'bill'
   ```
5. When "bill" accesses `employee.emp_salary`, he first supplies his password:
   ```
   set encryption passwd 'justforBill' for key razi.key1
        select empname, emp_salary from dbo.employee
   ```

When SAP ASE accesses the key for the user, it looks up that user's key copy. If no copy exists for a given user, SAP ASE assumes the user intends to access the base key.

Change Passwords on Key Copies

Once a user has been assigned a key copy, he or she can use **alter encryption key** to modify the key copy's password.

This example shows how a user assigned a key copy alters the copy to access data through his or her personal password:

- Key custodian "razi" sets up a key copy on an existing key for "bill" and encrypts it with a temporary password:

```
alter encryption key key1 with passwd 'MotherOfSecrets'
     add encryption with passwd 'just4bill' for user bill
```

- "razi" sends "bill" his password for access to data through `key1`.
- "bill" assigns a private password to his key copy:

```
alter encryption key razi.key1 with passwd 'just4bill'
     modify encryption with passwd 'billswifesname'
```

Only "bill" can change the password on his key copy. When "bill" enters the command above, SAP ASE verifies that a key copy exists for "bill". If no key copy exists for "bill", SAP ASE assumes the user is attempting to modify the password on the base key and issues an error message:

```
Only the owner of object '<keyname>' or a user with
     sso_role can run this command.
```

You cannot create key copies for user "guest" for login association.

Access Encrypted Data with a User Password

You must supply the encryption key's password to encrypt or decrypt data on an **insert**, **update**, **delete**, **select**, **alter table**, or **select into** statement.

If the system encryption password protects the encryption key, you need not supply the system encryption password because SAP ASE can already access it. Similarly, if your key copy is encrypted with your login password, SAP ASE can access this password while you remain logged in to the server. For keys encrypted with an explicit password, you must set the password in your session before executing any command that encrypts or decrypts an encrypted column with this syntax:

```
set encryption passwd 'password_phrase'
    for {key | column} {keyname | column_name}
```

where:

- *password_phrase* – is the explicit password specified with the **create encryption key** or **alter encryption key** command to protect the key.
- **key** – indicates that SAP ASE uses this password to decrypt the key when accessing any column encrypted by the named key
- *keyname* – may be supplied as a fully qualified name. For example:

  ```
  [[database.][owner].]keyname
  ```
- **column** – specifies that SAP ASE use this password only in the context of encrypting or decrypting the named column. End users do not necessarily know the name of the key that encrypts a given column.
- *column_name* – name of the column on which you are setting an encryption password. Supply *column_name* as:

  ```
  [[ database.][ owner ]. ]table_name.column_name
  ```

Each user who requires access to a key encrypted by an explicit password must supply the password. SAP ASE saves the password in encrypted form in the user session's internal

context. SAP ASE removes the key from memory at the end of the session by overwriting the memory with zeros.

This example illustrates how SAP ASE determines the password when it must encrypt or decrypt data. It assumes that the **ssn** column in the **employee** and **payroll** tables is encrypted with `key1`, as shown in these simplified schema creation statements:

```
create encryption key key1 with passwd "Ynot387"
create table employee (ssn char (11) encrypt with key1, ename
char(50))
create table payroll (ssn char(11) encrypt with key1, base_salary
float)
```

1. The key custodian shares the password required to access `employee.ssn` with "susan". He does not need to disclose the name of the key to do this.
2. If "susan" has **select** and **decrypt** permission on `employee`, she can select employee data using the password given to her for `employee.ssn`:

   ```
   set encryption passwd "Ynot387" for column employee.ssn
        select ename from employee where ssn = '111-22-3456'
   ```

   ```
   ename
   ----------------------
   Priscilla Kramnik
   ```
3. If "susan" attempts to select data from `payroll` without specifying the password for `payroll.ssn`, the following **select** fails (even if "susan" has **select** and **decrypt** permission on `payroll`):

   ```
   select base_salary from payroll where ssn = '111-22-3456'
   ```

   ```
   You cannot execute 'SELECT' command because the user encryption
   password
   has not been set.
   ```

   To avoid this error, "susan" must first enter:

   ```
   set encryption passwd "Ynot387" for column payroll.ssn
   ```

The key custodian may choose to share passwords on a column-name basis and not on a key-name basis to avoid users hard-coding key names in application code, which can make it difficult for the database owner to change the keys used to encrypt the data. However, if one key is used to encrypt several columns, it may be convenient to enter the password once. For example:

```
set encryption passwd "Ynot387" for key key1
select base_salary from payroll p, employee e
     where p.ssn = e.ssn
        and e.ename = "Priscilla Kramnik"
```

If one key is used to encrypt several columns and the user is setting a password for the column, the user needs to set password for all the columns they want to process. For example:

```
set encryption passwd 'Ynot387' for column payroll.ssn
set encryption passwd 'Ynot387' for column employee.ssn
select base_salary from payroll  p, employee e
```

```
    where p.ssn = e.ssn
    and e.ename = 'Priscilla Kramnik'
```

If a password is set for a column and then set at the key level for the key that encrypts the column, SAP ASE discards the password associated with the column and retains the password at the key level. If two successive entries for the same key or column are entered, SAP ASE retains only the latest. For example:

1. If a user mistypes the password for the column `employee.ssn` as "Unot387" instead of the correct "Ynot387":

   ```
   set encryption passwd "Unot387"
        for column employee.snn
   ```

2. And then the user reenters the correct password, SAP ASE retains only the second entry:

   ```
   set encryption passwd "Ynot387"
        for column employee.ssn
   ```

3. If the user now enters the same password at the key level, SAP ASE retains only this last entry:

   ```
   set encryption passwd "Ynot387" for key key1
   ```

4. If the user now enters the same password at the column level, SAP ASE discards this entry because it already has this password at the key level:

   ```
   set encryption passwd "Ynot387"
        for column payroll.ssn
   ```

If a stored procedure or a trigger references data encrypted by a user specified password, you must set the encryption password before executing the procedure or the statement that fires the trigger.

Note: SAP recommends that you do not place the **set encryption passwd** statement inside a trigger or procedure; this could lead to unintentional exposure of the password through **sp_helptext**. Additionally, hard-coded passwords require you to change the procedure or trigger when a password is changed.

See also

- *Encrypted Columns Process* on page 71

Application Transparency Using Login Passwords on Key Copies

The key custodian can set up key copies for encryption using a user's login password, and thereby providing ease of use, better security, lower overhead, and application transparency.

- Ease of use – users whose login password is associated with a key can access encrypted data without supplying a password.

- Better security – users have fewer passwords to track, and are less likely to write them down.
- Lower administrative overhead for key custodian – the key custodian need not manually distribute temporary passwords to each user who requires key access through a private password.
- Application transparency – applications need not prompt for a password to process encrypted data. Existing applications can take advantage of the measures to protect data privacy from the power of the administrator.

To encrypt a key copy with a user's login password, use:

```
alter encryption key [[database.][owner].]keyname
    with passwd 'base_key_password'
    add encryption for user 'user_name' for login_association
```

where **login_association** tells SAP ASE to create a key copy for the named user, which it later encrypts with the user's login password. Encrypting a key copy with a login password requires:

1. Using **alter encryption key**, the key custodian creates a key copy for each user who requires key access via a login password. SAP ASE attaches information to the key copy to securely associate the key copy with a given user. The identifying information and key are temporarily encrypted using a key derived from the master key or—if no master key exists—the system encryption password. The key copy is saved in `sysencryptkeys`.
2. When a user processes data requiring a key lookup, SAP ASE notes that a copy of the encryption key identified for this user is ready for login password association. Using the master key or the system encryption password to decrypt the information in the key copy, SAP ASE validates the user information associated with the key copy against the user's login credentials, and encrypts the key copy with a KEK derived from the user's login password, which has been supplied to the session.

When adding a key copy with **alter encryption key** key for **login_association**, the master key or the system encryption password must be available for encryption of the key copy. The system encryption password must still be available for SAP ASE to decrypt the key copy when the user logs in. After SAP ASE has reencrypted the key copy with the user's login password, the system encryption password is no longer required.

Note: You must use the default SAP ASE authentication method with `syslogins` to access key copies using a login password. User authentication through external services such as LDAP or Kerberos results in an error accessing the key if the user's key copies were added for login association.

The following example encrypts a user's copy of the encryption key, `key1`, with the user's login password:

1. Key custodian "razi" creates an encryption key:

   ```
   create encryption key key1 for AES
        with passwd 'MotherofSecrets'
   ```

2. "razi" creates a copy of **key1** for user "bill", initially encrypted with the master key or the system encryption password, but eventually to be encrypted by "bill"'s login password:

```
alter encryption key key1 with
     passwd 'MotherofSecrets'
     add encryption
     for user 'bill'
     for login_association
```

3. SAP ASE uses the master key or the system encryption password to encrypt a combination of the key and information identifying the key copy for "bill", and stores the result in `sysencryptkeys`.
4. "bill" logs in to SAP ASE and processes data, requiring the use of `key1`. For example, if `emp.ssn` is encrypted by `key1`:

```
select * from emp
```

   SAP ASE recognizes that it must encrypt "bill"'s copy of `key1` with his login password. SAP ASE uses the master key or the system encryption password to decrypt the key value data saved in step 4. It validates the information against the current login credentials, then encrypts `key1`'s key value with a KEK generated from "bill"'s login password.
5. During future logins when "bill" processes columns encrypted by `key1`, SAP ASE accesses `key1` directly by decrypting it with "bill"'s login password, which is available to SAP ASE through "bill"'s internal session context.

   Users who are aliased to "bill" cannot access the data encrypted by `key1` because their own login passwords cannot decrypt `key1`.
6. When "bill" loses authority to process confidential data, the key custodian drops "bill"'s access to the key:

```
alter encryption key key1
     drop encryption
     for user 'bill'
```

A user can encrypt a key copy directly with a login password with **alter encryption key** using the **with passwd login_passwd** clause. However, the disadvantages of using this method over the login association are:

- The key custodian must communicate the key copy's first assigned password to the user.
- The user must issue **alter encryption key** to reencrypt the key copy with a login password.

For example:

- "razi" adds a key copy for user "bill" encrypted by an explicit password:

```
alter encryption key key1
     with passwd 'MotherofSecrets'
add encryption with passwd 'just4bill'
     for user bill
```

- "razi" shares the key copy's password with "bill".
- "bill" decides to encrypt his key copy with his login password for his own convenience:

```
alter encryption key key1 with passwd "just4bill" modify
encryption with passwd login_passwd
```

- Now, when "bill" processes encrypted columns, SAP ASE accesses "bill"'s key copy through his login password.

Login Password Change and Key Copies

If you hold a key copy encrypted by a login password on one or more keys, you need not modify the key copies after you change your login password. **alter login** decrypts your key copies with your old login password and reencrypts them using the new login password.

If the SSO uses **alter login** to change your password, **alter login** drops your key copies. This prevents an administrator from gaining access to a key through a known password. After a mandatory password change of this kind, the key custodian must use **alter encryption key** to add a key copy for **login_association** for the user whose password is changed. **alter login** ignores offline databases and, for keys stored in offline databases, the key custodian follows the steps for recovering a lost key copy password when the database comes back online.

The key custodian may also need to perform these steps when a user's password is changed after the server is started using the `-p` flag. If the SSO, who uses the `-p` flag also has access to keys through key copies encrypted with his or her login password, then the key custodian must drop and re-create the SSO's key copies.

See also

- *Loss of Login Password* on page 89

Dropping a Key Copy

When a user changes jobs or leaves the company, the key custodian should drop the user's key copy.

The syntax is:

```
alter encryption key keyname
    drop encryption for user user_name
```

For example, if user "bill" leaves the company, the key owner can prevent "bill"'s access to `key1` by dropping his key copy:

```
alter encryption key key1
    drop encryption for user bill
```

SAP ASE does not require a password for this command because no key decryption is required.

drop encryption key drops the base key and all its copies.

CHAPTER 10 Key Recovery from Lost Passwords

Key custodians can recover keys and lost passwords, and manage the ownership of encryption keys.

See also

- *Creating Master Key Copies* on page 22

Loss of Password on Key Copy

If a user loses a password for the encryption key, the key custodian must drop the user's copy of the encryption key and issue to the user another copy of the encryption key with a new password.

In this example, the key custodian assigned a copy of `key1` to "bill", and "bill" changed his password on `key1` to a password known only to him. After losing his password, "bill" requests a new key copy from the key custodian.

1. The key custodian deletes Bill's copy of the key:

```
alter encryption key key1
     drop encryption for user bill
```

2. The key custodian makes a new copy of `key1` for user "bill" and gives "bill" the password:

```
alter encryption key key1
     with passwd 'MotherofSecrets'
     add encryption with passwd 'over2bill'
     for user bill
```

3. "bill" automatically has permission to alter his own copy of `key1`:

```
alter encryption key key1
     with passwd 'over2bill'
     modify encryption
     with passwd 'billsnupasswd'
```

Loss of Login Password

If a user who has key copies encrypted by his or her login password loses that password, the key custodian can recover access for the user.

For example, if the user "bill", who has key copies encrypted by his login password, loses his login password, you can recover his access to encryption keys with these steps:

1. The SSO uses **alter login** to issue “bill” a new login password. SAP ASE drops any key copies assigned to “bill” for login association or key copies already encrypted by “bill”’s login password.
2. The key custodian follows the regular procedure for setting up key encryption by login association. He verifies that the master key or the system encryption password was set, and creates a key copy for “bill":

   ```
   alter encryption key k1
        with passwd 'masterofsecrets'
        add encryption for bill
        for login_association
   ```

   This step assumes the key custodian still knows the base key’s password. If the key’s encryption password is unknown, the key custodian must first follow the key recovery procedure.
3. The next time “bill” accesses data encrypted by `k1`, SAP ASE reencrypts the key copy for "bill" using the new login password for “bill”. For example, if `emp_salary` is encrypted by key `k1`, the following statement automatically reencrypts the key copy for “bill” with his login password:

   ```
   select emp_salary from emp
        where name like 'Prisicilla%'
   ```

See also

- *Login Password Change and Key Copies* on page 87

Loss of Password on Base Key

Key custodians can use key recovery if the base key password is lost. Key recovery is vital because, without the password, the key custodian cannot change the key’s password or add key copies.

If all users share access to data through the base key and a user forgets the password, he or she can get the password from another user or the key custodian. If no one remembers the password, all access to the data is lost. Because of this, SAP ASE recommends that you back up keys by creating a copy of the base key that you can use for recovery. This copy is called the key recovery copy.

The key custodian should:

- Appoint one user as the key recoverer. The key recoverer's responsibility is to remember the password to the key recovery copy.
- Make a copy of the base key for the key recoverer. Every key that requires recovery after a disaster must have a key recovery copy.

Key Recovery Commands

SAP ASE does not allow access to data through the recovery key copy. A key recovery copy exists only to provide a backup for accessing the base key.

Set up a recovery key copy using:

```
alter encryption key keyname with passwd base_key_passwd
add encryption with passwd recovery_passwd
for user key_recovery_user for recovery
```

where:

- *base_key_passwd* – is the password the key custodian assigned to the base key.
- *recovery_passwd* – is the password used to protect the key recovery copy.
- *key_recovery_user* – user assigned the responsibility for remembering a password for key recovery.

After setting the key recovery copy, the key custodian shares the password with the key recovery user, who can alter the password using:

```
alter encryption key keyname with passwd old_recovery_passwd
    modify encryption with passwd new_recovery_passwd for recovery
```

During key recovery, the key recovery user tells the key custodian the password of the key recovery copy. The key custodian restores access to the base key using:

```
alter encryption key keyname with passwd recovery_key_passwd
    recover encryption with passwd new_base_key_passwd
```

where:

- *recovery_key_passwd* – is the password associated with the key recovery copy, shared with the key custodian by the recovery key user. SAP ASE uses the *recovery_key_passwd* to decrypt the key recovery copy to access the raw key.
- *new_base_key_passwd* – is the password used to encrypt the raw key. SAP ASE updates the base key row in `sysencryptkeys` with the result.

This example shows how to set up the recovery key copy and use it for key recovery after losing a password:

1. The key custodian creates a new encryption key protected by a password.

   ```
   create encryption key key1 for AES
        passwd 'loseitl8ter'
   ```

2. The key custodian adds a encryption key recovery copy for `key1` for “charlie”.

   ```
   alter encryption key key1 with passwd 'loseitl8ter'
        add encryption
        with passwd 'temppasswd'
   ```

```
     for user charlie
     for recovery
```

3. "charlie" assigns a different password to the recovery copy and saves this password in a locked drawer:

```
alter encryption key key1
     with passwd 'temppasswd'
     modify encryption
     with passwd 'finditl8ter'
     for recovery
```

4. If the key custodian loses the password for base key, he can obtain the password from "charlie" and recover the base key from the recovery copy using:

```
alter encryption key key1
     with passwd 'finditl8ter'
     recover encryption
     with passwd 'newpasswd'
```

The key custodian now shares access to **key1** with other users by sharing the base key's password, or by dropping and adding key copies where changes in personnel have occurred.

Ownership Change of Encryption Keys

The SSO can transfer key ownership to a named user. Changing ownership may occur in the normal course of business, or as part of key recovery.

This command, when executed by the SSO, transfers key ownership to a named user:

```
alter encryption key  [[database.][owner].]keyname
     modify owner user_name
```

where *user_name* is the name of the new key owner. This user must already be a user in the database where the key was created.

For example, if "razi" is the key custodian, and owns the key `encr_key`, but is being replaced by a new key custodian named "tinnap", change the key ownership using:

```
alter encryption key encr_key modify owner tinnap
```

Only the SSO or the key owner can run this command. If the new owner already has a copy of the key, you see:

```
A copy of key encr_key already exists for user tinnap
```

A user who already has a regular key copy or a recovery key copy cannot become the new owner of the key. SAP ASE does not allow a key copy to be made for a key owner.

If the previous key owner had granted any permissions on the key, the granter user ID in `sysprotects` system table is changed to the user ID of the new owner of the key. The ownership change is effective immediately; the new owner need not log in again for the change to take effect.