SYBASE®

User Guide for Encrypted Columns

Adaptive Server® Enterprise

15.0.2

DOCUMENT ID: DC00968-01-1502-01

LAST REVISED: November 2008

Sybase, Inc., One Sybase Drive, Dublin, CA 94568.

Contents

About This Book

Audience

This book is intended for system administrators configuring Adaptive Server® Enterprise for encrypted columns.

How to use this book

- Chapter 1, "Overview of Encryption," – describes the Adaptive Server encrypted column feature.
- Chapter 2, "Creating and Managing Encryption Keys," – describes commands for creating, altering, and droping encryption keys.
- Chapter 3, "Encrypting Data," – describes what data can be encrypted and the steps to preform for encryption.
- Chapter 4, "Accessing Encrypted Data," – describes how to access encrypted data.
- Chapter 5, "Protecting Data Privacy from the Administrator," – describes how to protect your encrypted data from the system administrator.
- Chapter 6, "Recovering Keys from Lost Passwords," – describes what to do if you or a user loses an encryption key or a password.
- Chapter 7, "Auditing Encrypted Columns," – describes how to audit encrypted data.
- Chapter 8, "Performance Considerations," – describess performance implications and resolutions for encrypted columns.
- Chapter 9, "System Information for Encrypted Columns," – describes changes made to system tables, commands, system procedures, utilities, CIS, and replication for encrypted columns.

Related documents

The Adaptive Server Enterprise documentation set consists of the following:

- The release bulletin for your platform – contains last-minute information that was too late to be included in the books.

  A more recent version of the release bulletin may be available on the World Wide Web. To check for critical product or document information that was added after the release of the product CD, use the Sybase Technical Library.

- The *Installation Guide* for your platform – describes installation, upgrade, and configuration procedures for all Adaptive Server and related Sybase products.
- *What's New in Adaptive Server Enterprise?* – describes the new features in Adaptive Server version 15.0, the system changes added to support those features, and changes that may affect your existing applications.
- *ASE Replicator User's Guide* – describes how to use the Adaptive Server Replicator feature of Adaptive Server to implement basic replication from a primary server to one or more remote Adaptive Servers.
- *Component Integration Services User's Guide* – explains how to use the Adaptive Server Component Integration Services feature to connect remote Sybase and non-Sybase databases.
- The *Configuration Guide* for your platform – provides instructions for performing specific configuration tasks for Adaptive Server.
- *Enhanced Full-Text Search Specialty Data Store User's Guide* – describes how to use the Full-Text Search feature with Verity to search Adaptive Server Enterprise data.
- *Glossary* – defines technical terms used in the Adaptive Server documentation.
- *Historical Server User's Guide* – describes how to use Historical Server to obtain performance information for SQL Server® and Adaptive Server.
- *Java in Adaptive Server Enterprise* – describes how to install and use Java classes as datatypes, functions, and stored procedures in the Adaptive Server database.
- *Job Scheduler User's Guide* – provides instructions on how to install and configure, and create and schedule jobs on a local or remote Adaptive Server using the command line or a graphical user interface (GUI).
- *Messaging Service User's Guide* – describes how to useReal Time Messaging Services to integrate TIBCO Java Message Service and IBM WebSphere MQ messaging services with all Adaptive Server database applications.
- *Monitor Client Library Programmer's Guide* – describes how to write Monitor Client Library applications that access Adaptive Server performance data.
- *Monitor Server User's Guide* – describes how to use Monitor Server to obtain performance statistics from SQL Server and Adaptive Server.

- *Performance and Tuning Series* – a series of books that explain how to tune Adaptive Server for maximum performance:
    - *Basics* – the basics for understanding and investigating performance questions in Adaptive Server.
    - *Locking and Concurrency Control* – describes how the various locking schemas can be used for improving performance in Adaptive Server, and how to select indexes to minimize concurrency.
    - *Query Processing and Abstract Plans* – describes how the optimizer processes queries and how abstract plans can be used to change some of the optimizer plans.
    - *Physical Database Tuning* – describes how to manage physical data placement, space allocated for data, and the temporary databases.
    - *Monitoring Adaptive Server with sp_sysmon* – describes how to monitor Adaptive Server's performance with sp_sysmon.
    - *Improving Performance with Statistical Analysis* – describes how Adaptive Server stores and displays statistics, and how to use the set statistics command to analyze server statistics.
    - *Using the Monitoring Tables* – describes how to query Adaptive Server's monitoring tables for statistical and diagnostic information.
- *Quick Reference Guide* – provides a comprehensive listing of the names and syntax for commands, functions, system procedures, extended system procedures, data types, and utilities in a pocket-sized book (regular size when viewed in PDF format).
- *Reference Manual* – is a series of four books that contains the following detailed Transact-SQL information:
    - *Building Blocks* – Transact-SQL datatypes, functions, global variables, expressions, identifiers and wildcards, and reserved words.
    - *Commands* – Transact-SQL commands.
    - *Procedures* – Transact-SQL system procedures, catalog stored procedures, system extended stored procedures, and dbcc stored procedures.
    - *Tables* – Transact-SQL system tables and dbcc tables.
- *System Administration Guide* –

- *Volume 1* – provides an introduction to the basics of system administration, including a description of configuration parameters, resource issues, character sets, sort orders, and diagnosing system problems. The second part of this book is an in-depth description of security administration.
- *Volume 2* – includes instructions and guidelines for managing physical resources, mirroring devices, configuring memory and data caches, managing multiprocessor servers and user databases, mounting and unmounting databases, creating and using segments, using the reorg command, and checking database consistency. The second half of this book describes how to back up and restore system and user databases.

- *System Tables Diagram* – illustrates system tables and their entity relationships in a poster format. Full-size available only in print version; a compact version is available in PDF format.
- *Transact-SQL User's Guide* – documents Transact-SQL, the Sybase enhanced version of the relational database language. This manual serves as a textbook for beginning users of the database management system. This manual also contains descriptions of the pubs2 and pubs3 sample databases.
- *Troubleshooting Series* (for release 15.0) –
  - *Troubleshooting: Error Messages Advanced Resolutions* – contains troubleshooting procedures for problems that you may encounter when using Sybase® Adaptive Server® Enterprise. The problems addressed here are those which the Sybase Technical Support staff hear about most often
  - *Troubleshooting and Error Messages Guide* – contains detailed instructions on how to resolve the most frequently occurring Adaptive Server error messages. Most of the messages presented here contain error numbers (from the master..sysmessages table), but some error messages do not have error numbers, and occur only in Adaptive Server's error log.
- *User Guide for Encrypted Columns* – describes how configure and use encrypted columns with Adaptive Server
- *Using Adaptive Server Distributed Transaction Management Features* – explains how to configure, use, and troubleshoot Adaptive Server DTM features in distributed transaction processing environments.

- *Using Sybase Failover in a High Availability System* – provides instructions for using Sybase Failover to configure an Adaptive Server as a companion server in a high availability system.
- *Unified Agent and Agent Management Console* – describes the Unified Agent, which provides runtime services to manage, monitor and control distributed Sybase resources.
- *Utility Guide* – documents the Adaptive Server utility programs, such as isql and bcp, which are executed at the operating system level.
- *Web Services User's Guide* – explains how to configure, use, and troubleshoot Web Services for Adaptive Server.
- *XA Interface Integration Guide for CICS, Encina, and TUXEDO* – provides instructions for using the Sybase DTM XA interface with X/Open XA transaction managers.
- *XML Services in Adaptive Server Enterprise* – describes the Sybase native XML processor and the Sybase Java-based XML support, introduces XML in the database, and documents the query and mapping functions that comprise XML Services.

Other sources of information

Use the Sybase Getting Started CD, the SyBooks CD, and the Sybase Product Manuals Web site to learn more about your product:

- The Getting Started CD contains release bulletins and installation guides in PDF format, and may also contain other documents or updated information not included on the SyBooks CD. It is included with your software. To read or print documents on the Getting Started CD, you need Adobe Acrobat Reader, which you can download at no charge from the Adobe Web site using a link provided on the CD.
- The SyBooks CD contains product manuals and is included with your software. The Eclipse-based SyBooks browser allows you to access the manuals in an easy-to-use, HTML-based format.

  Some documentation may be provided in PDF format, which you can access through the PDF directory on the SyBooks CD. To read or print the PDF files, you need Adobe Acrobat Reader.

  Refer to the *SyBooks Installation Guide* on the Getting Started CD, or the *README.txt* file on the SyBooks CD for instructions on installing and starting SyBooks.

- The Sybase Product Manuals Web site is an online version of the SyBooks CD that you can access using a standard Web browser. In addition to product manuals, you will find links to EBFs/Maintenance, Technical Documents, Case Management, Solved Cases, newsgroups, and the Sybase Developer Network.

  To access the Sybase Product Manuals Web site, go to Product Manuals at http://www.sybase.com/support/manuals/.

Sybase certifications on the Web

Technical documentation at the Sybase Web site is updated frequently.

❖ Finding the latest information on product certifications

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click Certification Report.
3. In the Certification Report filter select a product, platform, and timeframe and then click Go.
4. Click a Certification Report title to display the report.

❖ Finding the latest information on component certifications

1. Point your Web browser to Availability and Certification Reports at http://certification.sybase.com/.
2. Either select the product family and product under Search by Base Product; or select the platform and product under Search by Platform.
3. Select Search to display the availability and certification report for the selection.

❖ Creating a personalized view of the Sybase Web site (including support pages)

Set up a MySybase profile. MySybase is a free service that allows you to create a personalized view of Sybase Web pages.

1. Point your Web browser to Technical Documents at http://www.sybase.com/support/techdocs/.
2. Click MySybase and create a MySybase profile.

Sybase EBFs and software maintenance

❖ Finding the latest information on EBFs and software maintenance

1. Point your Web browser to the Sybase Support Page at http://www.sybase.com/support.
2. Select EBFs/Maintenance. If prompted, enter your MySybase user name and password.
3. Select a product.
4. Specify a time frame and click Go. A list of EBF/Maintenance releases is displayed.

   Padlock icons indicate that you do not have download authorization for certain EBF/Maintenance releases because you are not registered as a Technical Support Contact. If you have not registered, but have valid information provided by your Sybase representative or through your support contract, click Edit Roles to add the “Technical Support Contact” role to your MySybase profile.
5. Click the Info icon to display the EBF/Maintenance report, or click the product description to download the software.

Conventions

The following sections describe conventions used in this manual.

SQL is a free-form language. There are no rules about the number of words you can put on a line or where you must break a line. However, for readability, all examples and most syntax statements in this manual are formatted so that each clause of a statement begins on a new line. Clauses that have more than one part extend to additional lines, which are indented. Complex commands are formatted using modified Backus Naur Form (BNF) notation.

Table 1 shows the conventions for syntax statements that appear in this manual:

Table 1: Font and syntax conventions for this manual

Element	Example
Command names,procedure names, utility names, and other keywords display in sans serif font.	select sp_configure
Database names and datatypes are in sans serif font.	master database
Book names, file names, variables, and path names are in italics.	*System Administration Guide* *sql.ini* file *column_name* *$SYBASE/ASE* directory

Element	Example

Variables—or words that stand for values that you fill in—when they are part of a query or statement, are in italics in Courier font.	select *column_name*
	    from *table_name*
	    where *search_conditions*

Type parentheses as part of the command.	compute *row_aggregate* (*column_name*)
Double colon, equals sign indicates that the syntax is written in BNF notation. Do not type this symbol. Indicates “is defined as”.	::=
Curly braces mean that you must choose at least one of the enclosed options. Do not type the braces.	{cash, check, credit}
Brackets mean that to choose one or more of the enclosed options is optional. Do not type the brackets.	[cash | check | credit]
The comma means you may choose as many of the options shown as you want. Separate your choices with commas as part of the command.	cash, check, credit
The pipe or vertical bar(|) means you may select only one of the options shown.	cash | check | credit
An ellipsis (...) means that you can *repeat* the last unit as many times as you like.	buy thing = price [cash | check | credit] [, thing = price [cash | check | credit]]... You must buy at least one thing and give its price. You may choose a method of payment: one of the items enclosed in square brackets. You may also choose to buy additional things: as many of them as you like. For each thing you buy, give its name, its price, and (optionally) a method of payment.

- Syntax statements (displaying the syntax and all options for a command) appear as follows:

  sp_dropdevice [*device_name*]

  For a command with more options:

  select *column_name*
     from *table_name*
     where *search_conditions*

  In syntax statements, keywords (commands) are in normal font and identifiers are in lowercase. Italic font shows user-supplied words.

- Examples showing the use of Transact-SQL commands are printed like this:

  ```
  select * from publishers
  ```

- Examples of output from the computer appear as follows:

```
pub_id    pub_name                 city         state
-------   ---------------------    -----------  -----
0736      New Age Books            Boston       MA
0877      Binnet & Hardley         Washington   DC
1389      Algodata Infosystems     Berkeley     CA

(3 rows affected)
```

In this manual, most of the examples are in lowercase. However, you can disregard case when typing Transact-SQL keywords. For example, SELECT, Select, and select are the same.

Adaptive Server's sensitivity to the case of database objects, such as table names, depends on the sort order installed on Adaptive Server. You can change case sensitivity for single-byte character sets by reconfiguring the Adaptive Server sort order. For more information, see the *System Administration Guide*.

Accessibility features

This document is available in an HTML version that is specialized for accessibility. You can navigate the HTML with an adaptive technology such as a screen reader, or view it with a screen enlarger.

Adaptive Server HTML documentation has been tested for compliance with U.S. government Section 508 Accessibility requirements. Documents that comply with Section 508 generally also meet non-U.S. accessibility guidelines, such as the World Wide Web Consortium (W3C) guidelines for Web sites.

Note You might need to configure your accessibility tool for optimal use. Some screen readers pronounce text based on its case; for example, they pronounce ALL UPPERCASE TEXT as initials, and MixedCase Text as words. You might find it helpful to configure your tool to announce syntax conventions. Consult the documentation for your tool.

For information about how Sybase supports accessibility, see Sybase Accessibility at http://www.sybase.com/accessibility. The Sybase Accessibility site includes links to information on Section 508 and W3C standards.

If you need help

Each Sybase installation that has purchased a support contract has one or more designated people who are authorized to contact Sybase Technical Support. If you cannot resolve a problem using the manuals or online help, please have the designated person contact Sybase Technical Support or the Sybase subsidiary in your area.

CHAPTER 1 Overview of Encryption

This chapter describes the Adaptive Server™ encrypted column feature.

Adaptive Server 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.

The Adaptive Server encryption column enables you to encrypt column-level data that is at rest, without changing your applications. This native support provides the following capabilities:

- Column-level granularity
- Use of a symmetric, National Institute of Standards and Techology (NIST)-approved algorithm: Advanced Encryption Standard (AES)
- Optimized for performance
- Enforced separation of duties
- Fully integrated and automatic key management
- Application transparency: no application changes are needed
- Protects data privacy 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 from an encrypted column requires decrypt permission in addition to select permission. Decrypt permission can be granted to specific database users, groups, or roles. Sybase gives you more control by providing you with granular access capability to sensitive data. Sybase also automatically decrypts selected data for users with decrypt permission.

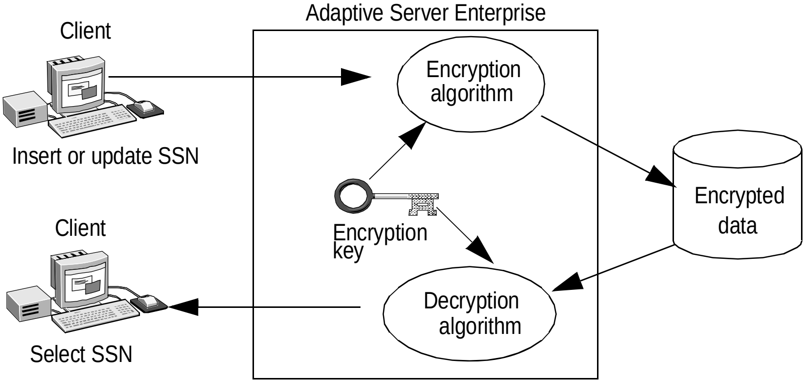
Encryption keys are stored in the database in encrypted form. You can encrypt an encryption key using a system-level or a user-supplied password (which can be the user’s login password). The password you select reflects your ability to preserve data privacy, even from system administrators.

Encrypting columns in Adaptive Server is more straightforward than using encryption in the middle tier, or in the client application. You use SQL statements to create encryption keys and to specify columns for encryption, and existing applications continue to run without change.

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. See “Length of encrypted columns” on page 30. Unencrypted data is stored as plain text.

Figure 1-1 describes encryption and decryption processing in Adaptive Server. In this example, a client is updating and encrypting a Social Security Number (SSN).

Figure 1-1: Encryption and decryption in Adaptive Server



Column encryption uses a symmetric encryption algorithm, which means that the same key is used for encryption and decryption. Adaptive Server tracks the key that is used to encrypt a given column.

When you insert or update data in an encrypted column, Adaptive Server transparently encrypts the data immediately before writing the row. When you select from an encrypted column, Adaptive Server 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.

Generally, using encrypted columns requires these administrative steps:

1. Install the license option ASE_ENCRYPTION. See the *Adaptive Server Enterprise Installation Guide*.
2. The system security officer (SSO) enables encryption in Adaptive Server:

   ```
   sp_configure 'enable encrypted columns', 1
   ```

3. Use sp_encryption to set the system encryption password for a database.
4. Create one or more named encryption keys. See Chapter 2, “Creating and Managing Encryption Keys.” Consider using passwords to protect data even from the database administrator. See Chapter 5, Protecting Data Privacy from the Administrator.
5. Specify the columns for encryption. See “Specifying encryption on new tables” on page 16 and “Encrypting data in existing tables” on page 18.
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 Adaptive Server returns this default, instead of the protected data, to users who do not have decrypt permission. See “Permissions for decryption” on page 34.

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

- Use the Sybase Central Adaptive Server Plug-in to manage encrypted columns using a graphical interface. See the online help for Sybase Central.
- Use the bulk copy utility (bcp) to securely copy encrypted data in and out of the server. See the *Utility Guide*.
- Use the Adaptive Server migration tool sybmigrate to securely migrate data from one server to another. See the *Adaptive Server Enterprise System Administration Guide*.
- Use Sybase 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.

CHAPTER 2

Creating and Managing Encryption Keys

Adaptive Server includes commands to create encryption keys, alter the properties of an encryption key, and drop unused encryption keys. Key owners must grant permission to table owners to use a specific key or keys to configure encryption at the column level.

Creating encryption keys

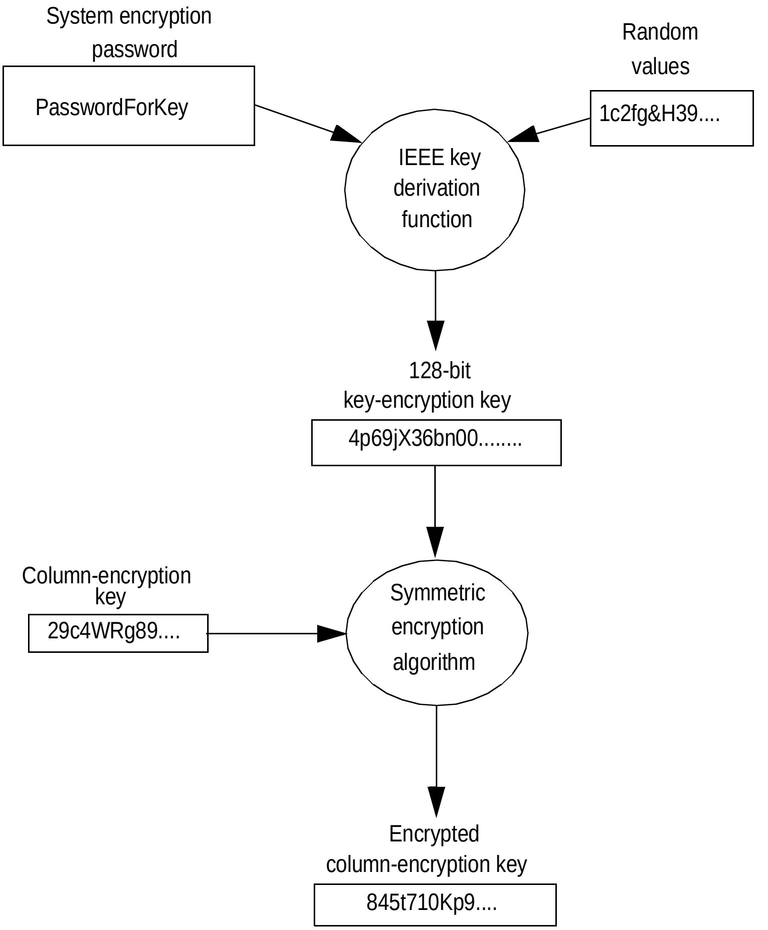
An encryption key must exist before a table owner can mark a column for encryption on a new or existing table. When you set up keys for the first time, consider:

- Key owner or custodian assignment – the system security officer must grant create encryption key permission to create keys. The sso_role and the keycustodian_role have automatic create encryption key permission. See “Role of the key custodian” on page 37.
- Whether keys should be created in a separate key database – Sybase 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 Adaptive Server 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, Adaptive Server uses a 128-bit key-encrypting key, which is derived from either the system encryption password or a user-specified password. Adaptive Server encrypts the new key (the column encryption key) and stores the result in sysencryptkeys.

Figure 2-1: Encrypting user keys


System encryption password
PasswordForKey
Random values
1c2fg&H39....
IEEE key derivation function
128-bit key-encryption key
4p69jX36bn00........
Column-encryption key
29c4WRg89....
Symmetric encryption algorithm
Encrypted column-encryption key
845t710Kp9....

Syntax for create encryption key

The syntax for create encryption key is:

```
create encryption key [[database.][owner].]keyname
  [as default] [for algorithm]
  [with
    {[key_length num_bits]
    [password 'password_phrase']
    [init_vector {null | random}]
    [pad {null | random}]
  }]
```

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 keys with names starting with ‘#’ as the first character.

- 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. Adaptive Server uses the default key from the same database. The default key may be changed. See “alter encryption key” on page 73.
- *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.
- *password_phrase* – a quoted alphanumeric string up to 255 bytes in length that Adaptive Server uses to protect the key. By default, Adaptive Server uses the system encryption password to protect encryption keys. See “Key protection using user-specified passwords” on page 40.
- 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. See Chapter 8, “Performance Considerations.”

  - 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.

create encryption key examples

This example specifies a 256-bit key called “safe_key” as the database default key:

```
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.

This 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
```

This 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
```

This example 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 before accessing a column encrypted by the key. See Chapter 5, Protecting Data Privacy from the Administrator for information about using keys with explicit passwords.

create encryption key permissions

The sso_role and keycustodian_role implicitly have permission to create encryption keys. The system security officer 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
```

Use this syntax to revoke key creation permission:

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.

Key protection

Adaptive Server keeps keys encrypted when not in use. There are actually two keys between the user and the data: the column-encryption key (CEK) and the key-encryption key (KEK). The CEK encrypts data and users must have access to it before they can access the encrypted data, but it cannot be stored on disk in an un-encrypted form. Instead, Adaptive Server uses a KEK to encrypt the CEK when you create or alter an encryption key. The KEK is also used to decrypt the CEK before you can access decrypted data. The KEK is derived internally 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. CEKs are stored in encrypted form in sysencryptkeys.

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

Figure 2-2 describes creating and storing 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 2-2: Steps to create an encryption key

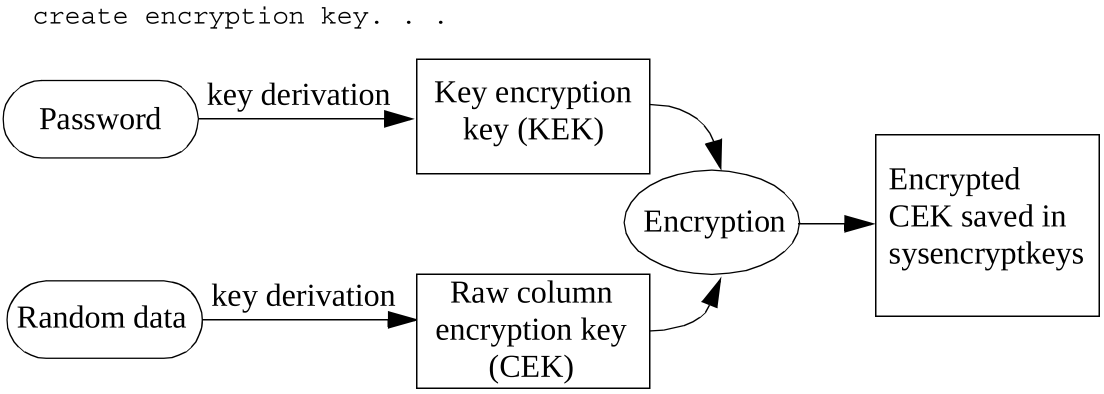
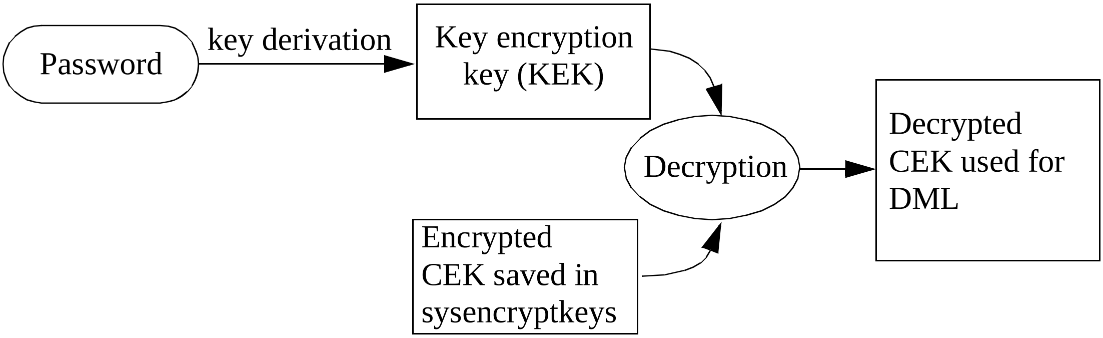

create encryption key. . .
Password
key derivation
Key encryption key (KEK)
Encryption
Encrypted CEK saved in sysencryptkeys
Random data
key derivation
Raw column encryption key (CEK)

Figure 2-3 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 2-3: Accessing a CEK to encrypt or decrypt on DML statement


Password
key derivation
Key encryption key (KEK)
Decryption
Decrypted CEK used for DML
Encrypted CEK saved in sysencryptkeys

Granting access to keys

The key owner must grant select permission on the 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 nondefault keys, any user with create encryption key permission. Key owners should grant select permission on keys as needed.

The following example allows users with db_admin_role to use the encryption key that is 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.

Key protection using the system-encryption password

The system encryption password is a database-specific password. By default, Adaptive Server uses this password to encrypt keys created in a given database. Once the system security officer or key custodian has set a system encryption password, you need not specify this password to process encrypted columns. Adaptive Server internally accesses the system encryption password when it needs to encrypt or decrypt column encryption keys.

The system security officer or key custodian use sp_encryption to set the system encryption password. The system password is specific to the database using sp_encryption, and its encrypted value is stored in the sysattributes system table in that database.

```
sp_encryption system_encr_passwd, password
```

password can be as many as 255 bytes in length, and is the default method Adaptive Server uses to encrypt all keys in the selected database.

Using a system encryption password simplifies the administration of encrypted data because:

- Managing passwords for keys is restricted to setting up and changing the system encryption password.
- You need not specify passwords on create and alter encryption key statements.
- Password distribution and recovery from lost passwords are not required.
- Access control over encrypted data is enforced through decrypt permission on the column. See “Restricting decrypt permission” on page 22.
- You need not make any changes to the application.

Set a system encryption password only in the database where encryption keys are created. If you choose to protect your keys with individual user passwords, you may not need to set the system encryption password. You can create encrypted columns in the same database as the keys or in other databases. See “Key protection using user-specified passwords” on page 40.

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. Sybase recommends that system encryption password be at least 16 bytes in length. In addition, when creating your password:

- Do not use information such as your birthday, street address, or any other word or number that has anything to do with your personal life.
- Do not use names of pets or loved ones.
- Do not use words that appear in the dictionary or words spelled backwards.

Adaptive Server enforces compliance of the system encryption password with the minimum password length and check password for digit configuration parameters.

The system security officer or key custodian can 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 with knowledge of the system encryption password leaves the company. When the system password is changed, Adaptive Server automatically reencrypts all keys in the database with the new password. Encrypted data is not affected when the system password is changed, in other words, data is not decrypted and reencrypted.

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

Changing the key

Periodically change the keys used to encrypt columns. 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
```

See "alter table" on page 68 for more information.

Separating keys from data

When you specify a column for encryption, you can use a named key from the same database or from a different database. If you do not specify a named key, the column is automatically encrypted with the default key from the same 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, Sybase 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.

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

Dropping encryption keys

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
```

Key owners can drop their own keys. The system security officer can drop any key. A key can be dropped only if there are no encrypted columns in any database that use the key.

When executing drop encryption key, Adaptive Server 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 naming 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 not usable. To restore the key, the system administrator must load a dump of the dropped key’s database from a time 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.

CHAPTER 3 Encrypting Data

You can encrypt these datatypes:

- 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

Encrypted data on disk is stored in the varbinary datatype. See "Length of encrypted columns" on page 30 for more information about the length of the varbinary data.

Null values are not encrypted.

Specifying encryption 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* for complete syntax of create table.

```
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, Adaptive Server 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, Adaptive Server 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 data in existing tables

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

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

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, Adaptive Server looks for a default key created using the as default clause on the create encryption key.

See the *Adaptive Server Enterprise Reference Manual* for the complete syntax for alter table.

There are restrictions on modifying encrypted columns:

- You cannot modify a column for encryption or decryption on which you have created a trigger. You must:

  a Drop the trigger.

  b Encrypt or decrypt the column.

  c 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:

  a Drop the index.

  b Alter the table/modify the type of column.

  c 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
```

Creating indexes 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)
```

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 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.

Configure Adaptive Server for restricted decrypt permission to restrict users from implicit decrypt permission. See “Restricting decrypt permission” on page 22.

Users with only select permission on an encrypted column can still process encrypted columns as cipher text through the bulk copy command. See “bulk copy (bcp)” on page 108. 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 “Returning default values instead of decrypted data” on page 23.

Revoking decryption permission

You can revoke a user’s decryption permission using:

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

For example:

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

Restricting decrypt permission

Adaptive Server protects data privacy from the powers of the administrator even if you use the system encryption password for key protection. If you prefer to avoid password management and use 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.

Assigning privileges for restricted decrypt permissions

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 and grants select permission on keys to the DBO, and grants decrypt permission to the end user.
- DBO – creates the schema and loads data.

Returning default values instead of decrypted data

This section describes how to use decrypt defaults with encrypted columns. 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 not permitted to see confidential data.

Defining a decrypt default

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, avoiding 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.

Adding and removing a decrypt default

Specify 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 Adaptive Server 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*

Tthis 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 funcion (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 the decrypt default attribute 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 Adaptive Server 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.

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.

1 If you have the hr_role and issue the following query, Adaptive Server returns one row.

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

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

2 If you do not have the hr_role, Adaptive Server returns no rows:

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

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

3 If you have the hr_role and include an or statement on a nonencrypted column, Adaptive Server returns the appropriate rows:

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

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

4 If you do not have the hr_role and issue the same command, Adaptive Server 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, Adaptive Server groups by the decrypt default constant value.

decrypt default and implicit grants

If you do not have explicit or implicit permission on a table, Adaptive Server returns the decrypt default value.

In this example (using the emp table described above), the DBO creates the p_emp procedure which selects from the DBO-owned emp table:

```
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, Adaptive Server returns the decrypt default value of emp.ssn.

In this next example, “joe,” a non-DBO user, creates the v_emp view, which selects from the DBO-owned emp table. In this case, 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
```

Adaptive Server 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, Adaptive Server 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, Adaptive Server 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 if 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 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 Adaptive Server 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

You can 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, Adaptive Server returns error message 10330.

Length of encrypted columns

During create table, alter table, and select into operations, Adaptive Server 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.

In Table 3-1, the lengths in the Maximum encrypted data length columns reflect the value in sycolumns.encrlen for a column of the specified type and length.

Table 3-1: cipher text lengths

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

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)
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
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
unichar(10)	20 (10 unichar characters)	varbinary	33	33	49	49
univarchar(20)	20 (10 unichar characters)	varbinary	49	33	65	49

Note text, image, timestamp and unitext datatypes are not supported by Adaptive Server.

Table 3-2: datatype length for encrypted columns

Datatype	Input data length	Encrypted column type	Max encrypted data length (no init_vector)	Actual encrypted data length (no init vector)	Max encrypted data length with init_vector	Actual encrypted data length (with init_vector)
date	4	varbinary	17	17	33	33
time	4	varbinary	17	17	33	33

Datatype	Input data length	Encrypted column type	Max encrypted data length (no init_vector)	Actual encrypted data length (no init vector)	Max encrypted data length with init_vector	Actual encrypted data length (with init_vector)
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

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.

Note 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.

CHAPTER 4 Accessing Encrypted Data

Adaptive Server automatically performs encryption and decryption when you process data in encrypted columns. Adaptive Server 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.

Processing encrypted columns

When you issue a select, insert, update, or delete command against an encrypted column, Adaptive Server 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, Adaptive Server expects the password to be available. If the user-specified password has not been set, the command fails. See “Accessing encrypted data with user password” on page 45
  - Adaptive Server decrypts the encryption key.
  - Adaptive Server encrypts the data using the column’s encryption key.
  - Adaptive Server inserts the varbinary cipher text data into the table.

    - After the insert or update, Adaptive Server 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, Adaptive Server expects the password to be available. If the user-specified password has not been set, the select statement fails. See “Accessing encrypted data with user password” on page 45. Otherwise, Adaptive Server decrypts the encryption key.
    - The decryption of the selected data succeeds if you have decrypt permission on the column, and Adaptive Server 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, Adaptive Server 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. See “Decrypt default columns and query qualifications” on page 26.
    - When possible, Adaptive Server 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 “Performance Considerations” on page 61.

Permissions for decryption

To see or process decrypted data, users 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. See Chapter 5, “Protecting Data Privacy from the Administrator,”.

Configuring Adaptive Server 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. Users cannot expect that execute permission on a stored procedure or select permission on a view does not explicitly grant users decrypt permission against the underlying table. The user must also have explicit decrypt permission on the base table.

Dropping 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
```

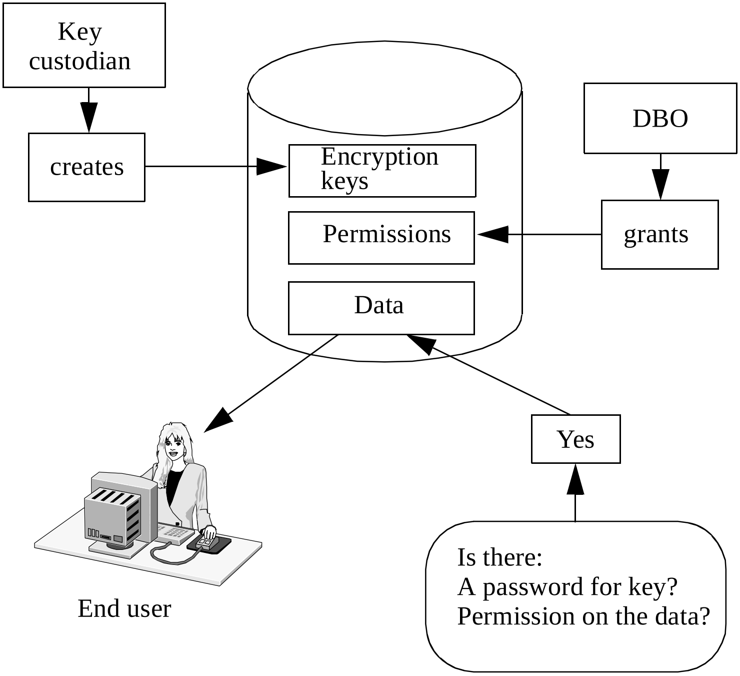
If the creditcard column was encrypted by a key with an explicit user password, you would need to set that password first.

CHAPTER 5 Protecting Data Privacy from the Administrator

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. Figure 5-1 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.

Figure 5-1: Database owner controlling permissions for data



The system administrator and database owner do not have implicit key management responsibilities. Adaptive Server provides the system role keycustodian_role so that the SSO need not assume all encryption responsibity. The key custodian owns the encryption keys, but should have no explicit or implicit permissions on the data. The DBO 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.

- 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.

Users, roles, and data access

User-specified passwords on encryption keys ensure that data privacy is protected from the system administrator. Table 5-1 explains how:

- The key custodian can own the keys, but not see the data.
- The DBO 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

Table 5-1: Permissions for users and roles on encrypted columns

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 on table or column.

Role	Can create encryption key?	Can use key in a schema definition?	Can decrypt encrypted data?
DBO 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.

Key protection using user-specified passwords

You can limit the power of the system administrator or DBO to access private data when you specify passwords on keys using create encryption key or alter encryption key. If keys have explicit passwords, before users can decrypt data, they need:

- decrypt permission on the column
- The encryption key's password

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

Use create encryption key to associate a password with a key:

```
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 Adaptive Server uses to generate thekey encryption key (KEK).

Adaptive Server does not save the user-specified password. It saves a string of validating bytes known as the "salt" in sysencryptkeys.eksalt, which allows Adaptive Server to recognize whether a password used on a subsequent encryption or decryption operation is legitimate for a key. You must supply the password to Adaptive Server 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 system encryption password or by 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.

Changing a key's password

You can use the alter encryption key command to change the current password for an encryption key:

```
alter encryption key [[database.database][owner].] keyname
    [with passwd 'old_password' | system_encr_passwd | login_passwd]
    modify encryption
    [with passwd 'new_password' | system_encr_passwd |
        login_passwd]
```

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 system encryption password.
- with passwd '*new_password*' – specifies the new password Adaptive Server 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.

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 is Joe’s replacement. 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'
   ```

Creating 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. Because this operator does not otherwise have permission to access encrypted data, he 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. Adaptive Server 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. Adaptive Server 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 DBO for schema creation:

```
grant select on key key1 to dbo
```

3 DBO 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 Adaptive Server accesses the key for the user, it looks up that user's key copy. If no copy exists for a given user, Adaptive Server assumes the user intends to access the base key.

Changing 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 (whose UID is "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, Adaptive Server verifies that a key copy exists for Bill. If no key copy exists for Bill, Adaptive Server 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.
```

Accessing encrypted data with 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 Adaptive Server can already access it. Similarly, if your key copy is encrypted with your login password, Adaptive Server can access this password while you remain logged in to the server (see "Application transparency using login passwords on key copies" on page 48). 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 Adaptive Server 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 Adaptive Server 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. Adaptive Server saves the password in encrypted form in the user session's internal context. Adaptive Server removes the key from memory at the end of the session by overwriting the memory with zeros.

This example illustrates how Adaptive Server 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 (user ID "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 DBO 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, Adaptive Server 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, Adaptive Server 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, Adaptive Server 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, Adaptive Server 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, Adaptive Server 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 a column encrypted by a user specified password, you must set the encryption password before executing the procedure or the statement that fires the trigger.

Note Sybase 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.

Application transparency using login passwords on key copies

The key custodian can set up key copies for encryption with a user's login password, and thereby provide:

- 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 Adaptive Server 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 two steps.

1 Using alter encryption key, the key custodian creates a key copy for each user who requires key access via a login password. Adaptive Server 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 system encryption password. The key copy is saved in sysencryptkeys.

2 When a user processes a column requiring a key lookup, Adaptive Server notes that a copy of the encryption key identified for this user is ready for login password association. Using the system encryption password to decrypt the information in the key copy, Adaptive Server 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 system encryption password must be available for encryption of the key copy. The system encryption password must still be available for Adaptive Server to decrypt the key copy when the user logs in. After Adaptive Server has reencrypted the key copy with the user's login password, the system encryption password is no longer required.

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

1 Key custodian Razi (with user ID "razi") creates an encryption key:

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

2 If there is not already a system encryption password, Razi sets one:

```
sp_encryption system_encr_passwd, 'keepitsecret'
```

3 Razi creates a copy of key1 for user Bill (with user ID "bill"), initially encrypted with 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
```

4 Adaptive Server uses 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.

5 Bill logs in to Adaptive Server and processes data, requiring the use of key1. For example, if emp.ssn is encrypted by key1:

```
select * from emp
```

Adaptive Server recognizes that it must encrypt Bill's copy of key1 with his login password. Adaptive Server uses 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.

6 During future logins when Bill processes columns encrypted by key1, Adaptive Server accesses key1 directly by decrypting it with Bill's login password, which is available to Adaptive Server 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.

7 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, Adaptive Server 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 have changed your login password. As part of changing the login password, sp_password decrypts your key copies with your old login password and reencrypts them using the new login password.

If the SSO uses sp_password to change your password without supplying your old password, sp_password 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. sp_password 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. See “Loss of login password” on page 54.

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.

Dropping a key copy

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

```
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
```

Adaptive Server 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 6

Recovering Keys from Lost Passwords

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 issues 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 (who has user ID "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 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 sp_password to issue Bill a new login password. Adaptive Server 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 system encryption password was set, and creates Bill’s key copy:

```
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. See “Loss of password on base key” on page 54 for more information.

3 The next time Bill accesses data encrypted by k1, Adaptive Server reencrypts Bill's key copy using Bill's new login password. For example, if emp_salary is encrypted by key k1, the following statement automatically reencrypts Bill’s key copy with his login password:

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

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, Adaptive Server 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:

1. Appoint one user as the key recoverer. The key recoverer’s responsibility is to remember the password to the key recovery copy.
2. 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

Adaptive Server 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. Adaptive Server 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. Adaptive Server updates the base key row in sysencryptkeys with the result.

You may also need to change ownership of the key to another key custodian. See "Changing ownership of encryption keys" on page 57.

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.

Changing ownership of encryption keys

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 Tina (user ID “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. Adaptive Server 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 grantor uid in sysprotects system table is changed to the uid 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.

CHAPTER 7 Auditing Encrypted Columns

Auditing options

See Chapter 18, “Auditing” in the *System Administration Guide: Volume 1* for encrypted columns auditing information (specifically Table 18-5, which lists the values in the event and extrainfo columns).

Audit values

See Chapter 18, “Auditing” in the *System Administration Guide: Volume 1* for values that appear in the event column of sysaudits (specifically Table 18-2, which lists auditing options, requirements, and examples).

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 Chapter 18, “Auditing” in the *System Administration Guide: Volume 1* for values that appear in the event column of sysaudits (specifically Table 18-6, which lists the audit event values).

Masking passwords 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 (user ID “alan”) in database db1:

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

And Alan issues this command:

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

Adaptive Server 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

To audit all actions in which keycustodian_role is active, use:

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

CHAPTER 8 Performance Considerations

Encryption is a CPU-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 Adaptive Server 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 Adaptive Server optimizer can make use of an encrypted column.

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 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 Adaptive Server 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, Adaptive Server 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`. Adaptive Server 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 Adaptive Server 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 Adaptive Server 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. Adaptive Server 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, Adaptive Server 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

Adaptive Server optimizes the joining of two encrypted columns by performing cipher text comparisons if:

- 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)
```

Adaptive Server 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 non-equality comparison of an encrypted column to a constant value, Adaptive Server optimizes the column scan by encrypting the constant value once, rather than decrypting the encrypted column for each row of the table. The same restrictions listed in “Joins on encrypted columns” on page 63 apply.

For example:

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

Adaptive Server 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, Adaptive Server 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, Adaptive Server 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.

CHAPTER 9 System Information for Encrypted Columns

This chapter provides information about system tables, commands, system procedures, utilities, CIS, and replication that are affected by, and are used by, encrypted columns.

System tables

See the *Reference Manual: Tables* for information about updates to the system tables for encrypted columns.

System commands

set proxy

If a user issues set proxy to assume the privileges, login name, and suid of another user, Adaptive Server checks the proxy user’s access to database objects, rather than the original user’s access. When Adaptive Server accesses a key copy, however, it does so on behalf of the original user and not the proxy user. Because a key copy may be encrypted by a user's login password. Adaptive Server uses the name and password information to check for automatic access to encryption keys using login credentials. Adaptive Server does not have access to the proxy user's password.

For example, if user1 has set his proxy to user2, that means user2 has access to the key through user1’s key copy, which may be encrypted by user1’s login password, or by a user-defined password which user1 must have passed to user2.

set encryption password

See “Accessing encrypted data with user password” on page 45 for information about using set encryption password. See the Reference Manual: Commands for the complete set syntax.

alter table

Use alter table to:

- Encrypt or decrypt existing data
- Add an encrypted column to a table
- Add, drop, or replace a decrypt default

The following partial syntax for alter table includes only clauses specific to encryption. See the *Reference Manual* for the complete syntax.

Syntax

Encrypt a column:

```
alter table tablename  add column_name
     encrypt [with [database.[owner].]keyname]
     [decrypt_default constant expression]
```

Decrypt an existing column:

alter table *tablename* modify *column_name*
 [decrypt [with [*database*.[*owner*].]*keyname*]]

keyname – identifies a key created using create encryption key. The table owner must have select permission on *keyname*. If *keyname* is not supplied, Adaptive Server looks for a default key created using create encryption key or alter encryption key as default.

Examples

Example 1 To create an encryption key and encrypt ssn column in existing employee table, enter.

```
set encryption passwd '4evermore' for key ssn_key
alter table employee modify ssn
         encrypt with ssn_key
```

If ssn in this encrypted by `key1`, alter table would cause Adaptive Server to decrypt ssn using `key1` and reencrypt ssn using 'ssn_key'.

Example 2 This adds an encrypted column to an existing table. Because keyname is omitted, Adaptive Server uses the database default encryption key:

```
alter table sales_mgr
     add bonus money null encrypt
```

Example 3 To decrypt credit card data that is no longer sensitive, enter:

```
alter table stolen_ccards
     decrypt ccard
```

If ccard was encrypted by a key protected by a user-defined password, precede this command with the set encryption key command.

Example 4 To add a decrypt default to an existing encrypted column, enter:

```
alter table employee
replace salary decrypt_default $0.0
```

Example 5 A user-defined password that protects a keyname must be set using set encryption passwd before you can execute alter table. To remove a decrypt default from the encrypted salary column without decrypting the column, enter:

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

Usage

- Use alter table to change an encrypted column. This operation may take a significant amount of time if the table contains a large number of rows.
- Modifying a column for encryption can cause the row size of the table to increase.

- You cannot use alter table to encrypt or decrypt a column belonging to a clustered or placement index. To encrypt or decrypt such a column:
  a. Drop the index.
  b. Alter the column.
  c. Re-create the index.
- You cannot use alter table to encrypt or decrypt a column if the table has a trigger defined. To modify the column:
  a. Drop the trigger.
  b. Alter the column.
  c. Re-create the trigger.
- If the type of the encrypted column which belongs to a clustered or placement index is modified, it results in the index being out of order. alter table displays an error. To modify the type:
  a. Drop the index
  b. Modify the type
  c. Re-create the index
- alter table reports an error if you:
  - Change a computed column to an encrypted column or change an encrypted column to a computed column
  - Enable a column for encryption where the column is referenced in an expression used by a computed column
  - Change a computed column to reference an encrypted column.
  - Encrypt a column that is a member of a functional index
  - Specify an encrypted column as a partition key
  - Encryption-enable a column that is already used as a partition key

create index

To improve performance on both equality and nonequality searches, and on joins, create indexes on encrypted columns. See “Performance Considerations” on page 61 for information on how Adaptive Server makes use of indexes on encrypted columns.

create index reports an error if you create:

- A functional index using an expression that references an encrypted column.
- An index on a column encrypted with initialization vector or random padding.

create table

Use the encrypt qualifier with create table to set up encryption on a table column and optionally specify a decrypt default.

The following partial create table syntax only includes clauses specific to encryption. See the *Reference Manua*l for the complete syntax:

Syntax

```
create table tablename (colname datatype [default_clause]
      [identity_clause][column_constraints]
      [encrypt [with [database.[owner].]keyname]
            [decrypt default constant expression]]
      [, next_colname datatype [optional clauses]]
```

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, Adaptive Server looks for a default key created using the as default clause on create encryption key or alter encryption key.

Note You cannot reference a column in the *partition_clause* of create table that is specified for encryption in the target table.

Example

Example 1 Creates an employee table with a nullable encrypted column. Adaptive Server uses the database default encryption key to encrypt the ssn data:

```
create table employee_table (ssn char(15) null
   encrypt name char(50), deptid int)
```

Example 2 To create a customer table with an encrypted column for credit card data, enter:

```
create table customer (ccard char(16) unique
   encrypt with cc_key
   decrypt_default 'XXXXXXXXXXXXXXXX', name char(30))
```

The ccard column has a unique constraint and uses cc_key for encryption. Because of the decrypt_default specifier, Adaptive Server returns the value 'XXXXXXXXXXXXXXXX' instead of the actual data when a user without decrypt permission selects the ccard column.

Usage

create table displays an error if you:

- Specify a computed column based on an expression that references one or more encrypted columns.
- Use the encrypt and compute parameters on the same column.
- List an encrypted column in the partition clause

select into

select into requires column-level permissions, including decrypt, on the source table. The following partial syntax for select into includes only clauses specific to encryption. See the *Reference Manual* for the complete syntax.

Syntax

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

Example

This example encrypts the creditcard column in the bigspenders table:

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

Usage

- If you use the encrypt clause without specifying a key name, Adaptive Server uses the database default key to encrypt the data in the target column.
- If a column in the source table is encrypted and you do not specify the encrypt clause for the target column, Adaptive Server decrypts the data in the source table and inserts plain text data in the target column.

- If you specify encryption for the target column with the same key used for the source column data, and if the key does not use an initialization vector or random padding, then Adaptive Server copies the data from the source column to the target column as cipher text, without intermediate decryption and reencryption operations.
- If however, you specify encryption for the target column using a different key from that used for the source column, or if the key uses an initialization vector or padding during encryption, the Adaptive Server performs a decryption and encryption operation for each selected row of the encrypted column.

create encryption key

See “Creating encryption keys” on page 5 for create encryption key usage information. See the *Reference Manual: Commands* for the complete syntax.

alter encryption key

See Chapter 5, “Protecting Data Privacy from the Administrator,” for alter encryption key usage information. See the *Reference Manual: Commands* for the complete syntax.

drop encryption key

drop encryption key drops the key copies when you drop the base key. The key owner and the SSO can drop encryption keys. The command fails if any column in any database is encrypted using the key.

The syntax is:

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

grant create encryption key

The SSO grants permission to create encryption keys. Only the SSO and the key custodian have implicit permission to create encryption keys.

The syntax is:

```
grant create encryption key to user | role| group
```

Note grant all in a database does not grant create encryption key permission.

revoke create encryption key

The SSO can revoke the permission to create encryption keys from users, groups, and roles:

The syntax is:

```
revoke create encryption key from user | role | group
```

grant decrypt

The table owner or the SSO can grant decrypt permission on a table or a list of columns in a table if you have not configured 'restricted decrypt permission'. If you have configured restricted decrypt permission, only the SSO can grant decrypt permission.

The syntax is:

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

Note grant all on a table or column does not grant decrypt permission

revoke decrypt

The table owner or the SSO can revoke decrypt permission on a table or a list of columns in a table if you have not configured restricted decrypt permission. If you have configured restricted decrypt permission, only the SSO can revoke decrypt permission.

The syntax is:

```
revoke decrypt on [owner.] tablename[(columnname [{,columnname}])]
from user | group | role
```

unmount database

When columns are encrypted by keys from other databases, unmount all interdependent databases as a set. The interdependency of the databases containing the encrypted columns and the databases containing the keys is similar to the interdependency of databases that use referential integrity.

Use the override option to unmount a database containing columns encrypted by a key in another database.

In this example, the encryption key created in key_db has been used to encrypt columns in col_db. These commands successfully unmount the named databases:

```
unmount database key_db, col_db
unmount database key_db with override
unmount database col_db with override
```

If you include with override, Adaptive Server issues a warning message, but the operation is successful.

If you do not include with override, commands fail with an error message

dump and load database

If the database you are loading contains encryption keys used in other databases, load database does not succeed unless you use with override.

Syntax

```
load database key_db from device_file with override
```

Usage

dump and load work on the cipher text of encrypted columns, ensuring that the data in encrypted columns remains encrypted while on disk. dump and load act on entire databases. Default keys, and keys created in the same database, are dumped and loaded along with the data they protect.

If your keys are in a separate database from the columns they encrypt, Sybase recommends that:

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

- After loading the database containing the encryption keys and the database containing the encrypted columns, bring both databases online at the same time.

If you load the database containing the keys into a different database, errors result when you try to access the encrypted columns in other databases. To change the database name of the keys' database:

- Before dumping the database containing the encrypted columns, use alter table to decrypt the data.
- Dump the databases containing keys and encrypted columns.
- After loading the databases, use alter table to reencrypt the data with the keys in the newly named database.

Warning! The consistency issues between encryption keys and encrypted columns are similar to those for cross-database referential integrity. See "Cross-database constraints and loading databases" in Chapter 12 of the *Adaptive Server Enterprise System Administration Guide: Volume One*.

quiesce database

You can use quiesce database when the database containing encrypted columns also contains the encryption key.

You must use with override to quiesce a database whose columns are encrypted with keys used in other databases.

quiesce database *key_db*, *col_db* is allowed, where *key_db* is the database with the encryption key and *col_db* is the database with a table that has a column encrypted with the key in *key_db*.

For example, the following commands succeed when key_db contains the encryption key used to encrypt columns in col_db:

```
quiesce database key_tag hold key_db for external
         dump to '/tmp/keydb.dat'

quiesce database encr_tag hold col_db for external dump
      to '/tmp/col.dat' with override

quiesce database col_tag hold key_db, col_db for
    external dump to '/tmp/col.dat'
```

drop database

To prevent accidental loss of keys, drop database fails if the database contains keys currently used to encrypt columns in other databases. To drop a database:

1. Decrypt the columns or modify the columns for encryption by a different key using alter table
2. Drop the table or database containing the encrypted columns

In this example, key_db is the database where the encryption key resides, and col_db is the database containing the encrypted columns:

```
drop database key_db, col_db
```

Adaptive Server raises an error and does not drop key_db; however, col_db is dropped. To drop both databases, drop col_db first:

```
drop database col_db, key_db
```

dbcc

For encryption, dbcc checkcatalog includes these consistency checks:

- For each encryption key row in sysobjects, sysencryptkeys is checked for the existence of a row defining that key.
- For each column in syscolumns marked for encryption, the existence of the key is checked in sysobjects and sysencryptkeys.

dbcc checkcatalog ensures that:

- The corresponding base key is present in sysencryptkeys for every key copy in sysencryptkeys. If the base key is not present, Adaptive Server issues an error.
- For every key copy, the corresponding uid is present in sysusers. If the uid is not present, Adaptive Server issues an error.
- For every decrypt default defined on a column, that the corresponding decrypt default is present in sysobjects and sysattributes. If the corresponding decrypt default is not present, Adaptive Server issues an error.

System stored procedures

sp_helprotect

Syntax

sp_helprotect [name [, username [, "grant"
[,"none"|"granted"|"enabled"|role_name [,permission_name]]]]]

The value for *permission_name* can be any of the values from sysprotects.action.

This example executes sp_helprotect using the “Decrypt” action from sysprotects.action:

```
sp_helprotect @permission_name = 'Decrypt'
```

```
grantor   grantee   type   action   object     column   grantable
-------   --------  -----  -------  -------    ------   ---------
sa1        hr_login   Grant  Decrypt  employee   ssn       TRUE
sa1        hr_role    Grant  Decrypt  employee   ssn       FALSE
```

Any user can run sp_helprotect to view his or her permission information. Only the SSO can view permissions for all users.

sp_dropuser

sp_dropuser drops all key copies from sysencryptkeys for the specified user in the current database. sp_dropuser fails if the user owns an encryption key in any database.

sp_help

For tables that include an encrypted column, sp_help *tablename* displays whether a column is encrypted and whether the encrypted column has a decrypt default. For example:

```
create table encr_table(col1 int encrypt decrypt_default 1)
```

The output from this example is similar to:

```
Column_name   Type     Length   Prec   Scale   Nulls   Default_name
Rule_name    Access_Rule_name Computed_Column_object  Identity     Encrypted
Decrypt_Default_name
```

```
-----------    ----    ------    ----   -----   -----   ------------
---------    --------------   --------------------    ---------   ------------
------------
c1             int     4         NULL   NULL    0        NULL
NULL         NULL             NULL                    0          1
encr_table_col1_1036527695
```

sp_configure

enable encrypted columns

enable encrypted columns enables encrypted columns, and is a dynamic configuration option.

You cannot set enable encrypted columns unless you have purchased, installed, and registered the ASE_ENCRYPTION license on your server. Any attempt to set it without such licensing results in Msg. 10834:

```
Configuration parameter 'enable encrypted columns'
cannot be enabled without license 'ASE_ENCRYPTION'
```

Note Using encrypted columns increases the logical memory used by 8198 kilobytes.

restricted decrypt permission

restricted decrypt permission enables or disables restricted decrypt permission in all databases, and is a dynamic configuration option: you need not restart Adaptive Server for it to take affect.

The SSO runs this command to enable or disable restricted decrypt permission in all databases:

```
sp_configure "restricted decrypt permission", [1 | 0]
```

When restricted decrypt permission is set to 0 (off), decrypt permission on encrypted columns acts the same as in versions earlier than 15.0.2:

- The table owner or the SSO explicitly grants decrypt permission. However, with grant option on decrypt permission is supported.

- Decrypt permission is granted implicitly to table owners and the SSO, as well as to any user through a chain of ownership. For example, if user Fred owns the proc1 stored procedure, which selects data from the encrypted column fred.table1.col1, and if Fred grants exec permission on proc1 to Harry, then Harry has implicit decrypt permission on fred.table1.col1
- Decrypt permission is not needed for alter table decryp. because the table owner has implicit decrypt permission on encrypted columns.

When restricted decrypt permission is set to 1 (on):

- Decrypt permission is granted implicitly only to the SSO.
- The SSO can grant decrypt permission using the with grant option parameter. This allows the SSO to decide who can grant decrypt permission in the system. For example, if the SSO wants user1 to be able to grant decrypt permission on user3.user3_tab, the SSO issues:

  ```
  grant decrypt on user3.user3_tab to user1
  with grant option
  ```

  If you use a system encryption password, Sybase recommends that, to protect data privacy, you do not grant decrypt permission to the DBO to. Access to keys through user passwords prevents the DBO and other parties from accessing the data unless they have a key's password; however, you may find it convenient for the DBO to decide which users should see the decrypted data. If you are not protecting keys and data with user-specified passwords, the SSO should retain the sole responsibility to grant decrypt permission.
- Table ownership does not give a user implicit decrypt permission. That is, if you create a table with encrypted columns, you do not have decrypt permission on them unless it is explicitly granted to you.
- No user is implicitly granted decrypt permission through an ownership chain. For example, if Fred owns the proc1 stored procedure, which selects data from the encrypted column fred.table1.col1, and if Fred grants exec permission on proc1 to Harry, then Harry must also have explicit decrypt permission on fred.table1.col1 to see the data.
- Aliased users assume the permissions of the user to whom they are aliased. Similarly, a user with sa_role, who is implicitly aliased to the DBO in any database, inherits any decrypt permissions that have been explicitly granted to the DBO.
- Decrypt permission is required for alter table decrypt statement because the table owner does not have implicit decrypt permission on the table.

If you change restricted decrypt permission from 0 to 1, currently executing statements that use implicit decrypt permission finish; however any subsequent statements that use implicit decrypt permission fail with this error until the SSO grants the user decrypt permission on the necessary columns:

```
Msg 10330 "DECRYPT permission denied on object object_name, database
database_name, owner owner_name."
```

If you change restricted decrypt permission from 1 to 0, the rows that reflect explicit grants remain in the sysprotects system table. However, these rows have no effect on implicitly granted decrypt permissions because Adaptive Server does not check sysprotects to make sure decrypt permission can be implicitly granted. sp_helprotect displays misleading information for only those users who were granted or revoked explicit decrypt permission before you reconfigure the system, and who now have implicit decrypt permission.

Sybase recommends that, to keep the system consistent, you revoke any explicit decrypt permissions granted to users before you switch between enabling or disabling restricted decrypt permission to keep the system consistent.

sp_helpconfig

Any user in the server can query the value of the encrypted columns options using sp_helpconfig. For example, to find out if restricted decrypt permission is active, enter:

```
sp_helpconfig 'restricted decrypt permission'
```

The output is similar to:

```
0 - restricted decrypt permission disabled (default).
1 - restricted decrypt permission enabled

Minimum Value                Maximum Value   Default Value   Current Value
Memory Used       Unit        Type

 ----------------            -----------     ----------      ---------
-------     ------------      -------------
0                            1                0              0
0           switch             dynamic
```

you can also run sp_helpconfig 'enable encrypted columns' to determine if encrypted columns in enabled on the server.

sp_password

When you use sp_password to change your password, Adaptive Server:

- Uses the original password to decrypt all the key copies that were encrypted using your login password then reencrypts them with the new password.
- Updates any key copies assigned to you that are designated for login association. The key copies' password type remains the same.

After a password change, log out of all your Adaptive Server sessions then log in again before accessing any encrypted data. Alternatively, you can use the immediate parameter of sp_password to propagate the password change to all sessions.

When the SSO issues sp_password to reset a user's password, the user's login password-encrypted key copies are dropped by Adaptive Server because the user's old login password is not available. Adaptive Server requires the key custodian to re-create the key copies for the user, if they are needed

sp_audit

The sp_audit 'encryption_key' option manages auditing for encrypted column commands and events. encryption_key audits these commands:

- create encryption key
- sp_encryption
- alter encryption key
- drop encryption key

sp_displayaudit

sp_displayaudit displays current audit settings. sp_displayaudit displays the encryption_key information under the database audit options.

sp_encryption

The SSO or the key custodian uses sp_encryption to set the system encryption password. The system password is specific to the database in which sp_encryption is executed, and its encrypted value is stored in the sysattributes system table in that database.

```
sp_encryption system_encr_passwd, ‘password’
```

The password specified using sp_encryption can be 255 bytes in length, and is used by Adaptive Server to encrypt all keys all keys that do not specify a user-specified password, login password, or login association in that database. Once the system encryption password has been set in a database, Adaptive Server has automatic access to it, not only to encrypt new keys, but also to decrypt keys when a user reads or writes encrypted columns.

The system encryption password must be set in every database where encryption keys are created without the with passwd clause. The system encryption password must be set when key copies are added for login_association, until the key copy assignees have logged in to Adaptive Server and used their key copy.

The SSO or key custodian can change the system password by using:

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

When the system password is changed, Adaptive Server automatically reencrypts all keys encrypted with the system encryption password in the database with the new password.

Syntax

```
sp_encryption help | helpkey
```

```
sp_encryption help | helpkey [, keyname | wildcard]
          [, all_dbs | key_copy | display_cols ]
```

```
sp_encryption [help | helpkey][, system_encr_password]
          [, display_keys | all_dbs]
```

```
sp_encryption helpcol [, table_name | column_name ]
```

```
sp_encryption helpuser [, user_name | wildcard ][, key_copy]
```

- helpkey – lists encryption key properties, including:
  - Whether the database contains encryption keys.

  - When run by a user with sso_role, key custodian, or DBO: key name, key owner, key length, key algorithm, key type, pad, initialization vector, type of password used to encrypt the key, whether key recovery has been enabled and count of key copies.

    The output is sorted on owner.key name. When run by a non-privileged user, this command lists keyname, key owner and key type.

- help – included for backward compatibility. Includes the same output as helpkey.
- *keyname* – name of the key you are investigating. Lists the properties defined for *keyname*. If *keyname* is omitted, lists properties for all keys.
- *wildcard* – lists the properties for keys matching the wildcard pattern in the current database.

  For information on using wildcard characters, see Chapter 4, “Using wildcard characters” in *Reference Manual:Building Blocks*.
- all_dbs – lists information on encryption keys in all available databases. Only the SSO can run all_dbs.
- key_copy – lists all user copies for the specified key in the current database. The output is sorted by *key_owner.keyname* and includes information about:
  - The base key owner.
  - Whether the key copy is a recovery key copy.
  - The user to whom a copy belongs.
  - Whether the copy is encrypted with a user-encryption password, a login password, or the system encryption password for login association (indicated in the output by Login Access).
- display_cols – displays the key name, all keys (or matching wildcard keys) in the current database, and the columns the key encrypts. When SSO includes display_cols, all columns encrypted by the keys across all available databases are included. When a user without the sso_role runs display_cols, only those columns encrypted by the key in the current database appear. Data is sorted by *keyname*, *key_owner*, *database*, *table_owner*, *table_name*, and *column_name*.

- helpcol *column_name* – displays the column name and the key used to encrypt the column. If the SSO includes helpcol, the key name appears in the output, even if the key is not present in the current database. If a user without the SSO includes helpcol, Adaptive Server prints the keyid of the key if it is not present in the current database, omitting the *keyname*. The output includes: *owner*.*table*.*column*, *database*.*owner*.*keyname*. The information is sorted by *owner*.*table*.*column*.
- helpuser – displays the keys owned by or assigned to a user in the current database.
- system_encr_passwd – displays the keys and key copies that are encrypted using the system encryption password in the current database.
- system_encr_passwd, all_dbs – displays the properties of the system encryption password in every database where it has been set. The output is sorted by . Only the SSO can run this command. If the system encryption password has not been set for all databases, you see:

  ```
  The system encryption password has not been set for
  all available databases
  ```

- display_keys – used with *system_encr_passwd* to display the keys and key copies that are encrypted using the system encryption password.

Examples

Example 1

To display the properties of all base encryption keys in the current database when run by the SSO, key custodian, or the DBO, issue:

```
sp_encryption helpkey
```

This is the output:

```
Key Name    Key Owner   Key Length   Key Algorithm
Key Type              Pad         Init Vector    Type of Password
Key Recovery     # of Key Copies
----------  --------    ---------    --------------
----------           -------     ------------   -----------
--------         ------
tinnap_key  tinnap       128          AES
Symmetric key         0           1              System Encr Passwd
0             0
tinnap_key1 tinnap       128          AES
Symmetric default key 0            1             User Passwd
1             3
sample_key1  dbo         192          AES
Symmetric key 1        1                         Login Passwd
```

```
1               2
```

When run by user “tinnap”, this displays the following properties of all base encryption keys in the current database:

```
sp_encryption helpkey
```

This is the output:

```
Key Name           Key Owner   Key Type
--------------      ---------   ------------
tinnap_key          tinnap      Symmetric key
tinnap_key1         tinnap      Symmetric default key
sample_key1          dbo        Symmetric key
```

Example 2

Displays properties of all base encryption keys with names similar to “tinnap%” in the current database when run by SSO, key custodian, or DBO:

```
sp_encryption helpkey, "tinnap%"
Key Name   Key Owner   Key Length   Key Algorithm   Key Type
Pad     Init Vector    Type of Password       Key Recovery
# of Key Copies
---------  --------    ---------     -------------- ---------
-----   -----------   --------------         -----------
--------------
tinnap_key   tinnap      128            AES         Symmetric key
0       1            System Encr Passwd       0
0
tinnap_key1  tinnap      128            AES          Symmetric default key
0       1            User Passwd              1              3
```

When run by user “tinnap”, displays the following properties for the base encryption keys in the current database with names similar to “tinnap%”:

```
sp_encryption helpkey, "tinnap%"
```

This is the output:

```
Key Name            Key Owner  Key Type
------------------  ---------  ------------
tinnap_key           tinnap     Symmetric key
tinnap_key1          tinnap     Symmetric default key
```

Example 3

Displays the properties of base encryption key sample_key1 when run by the SSO, key custodian, or DBO in the current database:

```
sp_encryption helpkey, sample_key1
```

```
Key Name     Key Owner   Key Length  Key Algorithm  Key Type
Pad          Init Vector   Type of Password   Key Recovery
# of Key Copies
-----------  --------    ----------  -------------  ---------  -----
-----------  -----------  ------------      ---------
--------------
sample_key1  dbo         192         AES            Symmetric Key
1             1            Login              1
2
```

When “tinnap” executes the command:

```
sp_encryption helpkey, sample_key1
```

this is the output:

```
Key Name       Key Owner    Key Type
-------------  -----------  ------------
sample_key1    dbo          Symmetric key
```

Example 4

Displays the properties of all base encryption keys in all available databases (only the SSO can run this command):

```
sp_encryption helpkey, NULL, all_dbs
```

This is the output:

```
Db.Owner.Keyname           Key Length  Key Algorithm  Key Type
Pad    Init Vector   Type of Password     Key Recovery    #of Key Copies
------------------------   ----------   -----------  -------------------
-----  ----------   ---------------       ------          --------------
keydb.dbo.cc_key           256           AES       Symmetric default key
1      1            System EncrPasswd     0              0
keydb.dbo.sample_key1      128          AES        Symmetric key
0      0            System Encr Password  1              4
keydb1.tinnap.tinnap_key   128          AES        Symmetric key
0      1            System Encr Passwd    0              0
keydb1.tinnap.tinnap_key1  128          AES        Symmetric default key
0      1            User Password         1              3
keydb1.dbo.sample_key1     192          AES        Symmetric key
1      1             Login Passwd         1              2
```

Example 5

Displays the properties of all base encryption keys similar to %key1 in all available databases (only the SSO can run this command):

```
sp_encryption helpkey, '%key', all_dbs
```

This is the output:

```
Db.Owner.Keyname           Key Length  Key Algorithm  Key Type
Pad    Init Vector   Type of Password     Key Recovery    #of Key Copies
------------------------   ----------   -----------  -------------------
-----  ----------   ---------------       ------          -----
keydb.dbo.cc_key           256           AES       Symmetric default key
1      1            System EncrPasswd     0               0
keydb1.tinnap.tinnap_key   128          AES        Symmetric key
0      1            System Encr Passwd    0               0
```

Example 6

Displays the properties of base encryption key sample_key1 in all available databases (only the SSO can run this command):

```
sp_encryption helpkey, sample_key1, all_dbs
```

This is the output:

```
Db.Owner.Keyname           Key Length  Key Algorithm  Key Type
Pad    Init Vector   Type of Password     Key Recovery   #of Key Copies
------------------------   ----------   -----------  -------------------
-----  ----------   ---------------       ------         -----
keydb.dbo.sample_key1      128          AES        Symmetric key
0      0            System Encr Password  1              4
keydb1.dbo.sample_key1     192          AES        Symmetric key
1      1             Login Passwd          1             2
```

Example 7

Displays all the user access copies of keys when run by the SSO, key custodian, or DBO in the current database:

```
sp_encryption helpkey, Null, “key_copy”
```

This is the output:

```
Owner.Keyname         Assignee    Type of Password           Key Recovery
------------------    ----------   ---------------            ----------
tinnap.tinnap.key1    joesmp       User Passwd                0
tinnap.tinnap.key1    samcool      User Passwd                1
tinnap.tinnap.key1    billyg       User Passwd                0
dbo.sample.key1       tinnap       Login Access               0
dbo.sample.key1        joesmp       Login Passwd              1
```

When user “tinnap” runs this command, it displays the key copies assigned to this user and the key copies for the keys “tinnap” owns in the current database:

```
sp_encryption helpkey, Null, “key_copy”
```

This is the output:

```
Owner.Keyname         Assignee    Type of Password           Key Recovery
```

```
------------------    ----------    ---------------          ----------
tinnap.tinnap.key1    joesmp        User Passwd              0
tinnap.tinnap.key1    samcool       User Passwd              1
tinnap.tinnap.key1    billyg        User Passwd              0
dbo.sample.key1       tinnap        Login Access             0
```

Example 8

Displays all the user access copies of keys with name similar to “sample%” when run by the SSO, key custodian, or DBO:

```
sp_encryption helpkey, "sample%", “key_copy”
```

This is the output:

```
Owner.Keyname        Assignee         Type of Password       Key Recovery
------------------    ----------       ---------------         ----------
dbo.sample_key1        tinnap             Login Access             0
dbo.sample_key1        joesmp              Login Passwd            1
```

When user “tinnap” runs this command, it displays the key copies of keys with names similar to “sample%” assigned to user “tinnap”, and the key copies for keys with names similar to “sample%” for which “tinnap” is the owner in the current database:

```
sp_encryption helpkey, "sample%", “key_copy”
```

This is the output:

```
Owner.Keyname        Assignee         Type of Password       Key Recovery
------------------   ----------       ---------------        ----------
dbo.sample_key1       tinnap           Login Access          0
```

Example 9

When run by the SSO, key custodian, or the DBO, displays all key copies for key tinnap_key1 in the current database:

```
sp_encryption helpkey, tinnap_key1, “key_copy”
```

This is the output:

```
Owner.Keyname         Assignee        Type of Password       Key Recovery
------------------     ---------       ----------------       ----------
tinnap.tinnap_key1     joesmp          User Passwd            0
tinnap.tinnap_key1     samcool         User Passwd            1
tinnap.tinnap_key1     billyg          User Passwd           0
```

When run by user “joesmp”, this displays all encryption key copies assigned to user “joesmp” and also all the key copies for that keyname if the user is the owner of the key in the current database:

```
sp_encryption helpkey, tinnap_key1, “key_copy”
```

This is the output:

```
Owner.Keyname          Assignee        Type of Password       Key Recovery
------------------      ---------       ----------------       ----------
tinnap.tinnap_key1      joesmp          User Passwd            0
```

Example 10

When run by the SSO, displays all encrypted columns in all available databases encrypted by keys in the current database:

```
sp_encryption helpkey, null, display_cols
```

This is the output:

```
Key Name        Key Owner   Database Name    Table Owner      Table Name
Column Name
------------    -------      -----------      ----------       ------------
--------------
tinnap_key      tinnap       testdb1          tinnap           t3
c3
tinnap_key1     tinnap       testdb           tinnap           t4
c4
sample_key1     dbo          coldb            dbo              t1
c1
sample_key1     dbo          coldb            billyg           t2
c2
```

When this statement is run by user “tinnap”, Adaptive Server displays the columns in the current database encrypted by keys in the current database:

```
sp_encryption helpkey, null, display_cols
```

This is the output:

```
Key Name        Key Owner   Database Name    Table Owner      Table Name
Column Name
------------    -------      -----------      ----------       ------------
--------------
tinnap_key      tinnap       testdb1          tinnap           t3
c3
```

Example 11

When run by the SSO, displays all encrypted columns in all available databases encrypted by a key with a name like “%key%” in the current database:

```
sp_encryption helpkey, "%key%", display_cols
```

This is the output:

```
Key Name       Key Owner   Database Name    Table Owner      Table Name
Column Name
------------   -------      -----------      ----------       ------------
--------------
tinnap_key1    tinnap       testdb           tinnap           t4
c4
sample_key1    dbo          coldb            dbo              t1
c1
sample_key1    dbo          coldb            billyg           t2
c2
```

When this statement is run by user “tinnap”, Adaptive Server returns all columns that are encrypted by keys with name matching “%key%” in the current database:

```
sp_encryption helpkey, "%key%", display_cols
```

This is the output:

```
Key Name       Key Owner   Database Name    Table Owner      Table Name
Column Name
------------   -------      -----------      ----------       ------------
--------------
tinnap_key1    tinnap       testdb           tinnap           t4
c4
```

Example 12

This example displays all columns which have been encrypted by key sample_key1 across all available databases:

```
sp_encryption helpkey, sample_key1, display_cols
```

This is the output:

```
Key Name          Key Owner         Database Name         Table Owner
Table Name   Column Name
------------       ----------       ---------------       ----------
-----------  -----------
sample_key1        dbo               coldb                dbo
t1             c1

sample_key1        dbo               coldb                billyg
t2           c2
```

When run by user “tinnap”, displays all columns in the current database that are encrypted by key sample_key1:

```
sp_encryption helpkey, sample_key1, display_cols
```

This is the output:

```
Key Name          Key Owner         Database Name         Table Owner
Table Name   Column Name
------------       ----------       ---------------       ----------
-----------  -----------
sample_key1        dbo               coldb                dbo
t1             c1

sample_key1        dbo               coldb                billyg
t2           c2
```

Example 13

When run by the SSO, key custodian, or DBO, lists keys and key copies that are encrypted with the system encryption password in the current database:

```
sp_encryption helpkey, system_encr_passwd, display_keys
```

This is the output:

```
Owner.Keyname                   Assignee
---------------                  -------------
dbo.cc_key                        NULL
dbo.sample_key1                   NULL
dbo.sample_key1                   tinnap
```

When run by user “tinnap”, this command displays the keys owned by, or key copies assigned to, user “tinnap” that are encrypted with the system encryption password in the current database:

```
sp_encryption helpkey, system_encr_passwd, display_keys
```

This is the output:

```
Owner.Keyname                   Assignee
---------------                  -------------
dbo.sample_key1                 tinnap
```

Example 14

Lists all base keys owned by users in the current database when the SSO, key custodian, or DBO runs this command:

```
sp_encryption helpuser
```

This is the output:

```
Owner.Keyname                 Type of Password
---------------                -------------------
tinnap.tinnap_key             System Encr Passwd
tinnap.tinnap_key1            User Passwd
dbo.sample_key1               Login Passwd
```

If user “tinnap” runs this command, lists all base keys owned by this user in the current database:

```
sp_encryption helpuser
```

This is the output:

```
Owner.Keyname                 Type of Password
---------------                -------------------
tinnap.tinnap_key             System Encr Passwd
tinnap.tinnap_key1            User Passwd
```

Example 15

When run by the SSO, key custodian, or DBO, lists all base encryption keys owned by user “tinnap” in the current database:

```
sp_encryption helpuser, tinnap
```

This is the output:

```
Owner.Keyname                 Type of Password
---------------                -------------------
tinnap.tinnap_key             System Encr Passwd
tinnap.tinnap_key1            User Passwd
```

If run by user “tinnap”, lists all base encryption keys owned by user “tinnap” in the current database:

```
sp_encryption helpuser, tinnap
```

This is the output:

```
Owner.Keyname                 Type of Password
---------------                -------------------
tinnap.tinnap_key             System Encr Passwd
tinnap.tinnap_key1            User Passwd
```

Example 16

When run by the SSO, key custodian, or DBO, lists all key copies assigned to all users in the current database:

```
sp_encryption helpuser, NULL, “key_copy”
```

This is the output:

```
Owner.Keyname            Assignee        Type of Password     Key Recovery
---------------------      -----------      ----------------   ---------
dbo.sample_key1            tinnap          Login Passwd           0
tinnap.tinnap_key1         joesmp          User Passwd            0
dbo.sample_key1            joesmp          Login Passwd           1
tinnap.tinnap_key1         samcool         User Passwd            1
tinnap.tinnap_key1          billyg          User Passwd           0
```

If user “tinnap” runs this statement, it displays the key copies assigned to this user and the key copies for the keys owned by this user in the current database:

```
sp_encryption helpuser, NULL, “key_copy”
```

This is the output:

```
Owner.Keyname            Assignee        Type of Password    Key Recovery
---------------------     -----------      ----------------   ---------
dbo.sample_key1           tinnap           Login Passwd       0
tinnap.tinnap_key1        joesmp           User Passwd        0
tinnap.tinnap_key1        samcool          User Passwd        1
tinnap.tinnap_key1        billyg           User Passwd        0
```

Example 17

When run by the SSO, key custodian, or DBO, lists all the key copies in the current database with assignee names like “%na%”:

```
sp_encryption helpuser, '%na%', “key_copy”
```

This is the output:

```
Owner.Keyname          Assignee         Type of Password      Key Recovery
---------------------   -----------      ------------------    -----------
dbo.sample_key1         tinnap           Login Passwd          0
tinnap.tinnap_key1      joesmp           User Passwd           0
dbo.sample_key1         joesmp           Login Passwd          1
```

When run by user “tinnap”, lists all the key copies in the current database with assignee name like “%na%” and the key copies for keys owned by this user with name like “%na%” only if the user’s name matches the wildcard pattern:

```
sp_encryption helpuser, '%na%', “key_copy”
```

This is the output:

```
Owner.Keyname          Assignee         Type of Password      Key Recovery
---------------------   -----------      ------------------    -----------
dbo.sample_key1         tinnap           Login Passwd          0
```

```
tinnap.tinnap_key1      joesmp          User Passwd          0
tinnap.tinnap_key1      samcool         User Passwd          1
tinnap.tinnap_key1      billyg          User Passwd          1
```

Example 18

When run by the SSO, key custodian, or DBO, lists all encrypted columns in the current database (coldb in this example) and the keys used to encrypt the columns:

```
sp_encryption helpcol
```

This is the output:

```
Owner.Table.Column            Db.Owner.Keyname
-----------------------        ---------------------
dbo.t1.c1                      keydb1.dbo.sample_key1
billyg.t2.c2                   keydb.dbo.sample_key1
tinnap.t3.c3                   coldb.dbo.sample_key2
```

When user "tinnap" runs this statement in the coldb database, Adaptive Server displays values for keyid instead of keyname for those keys not in coldb:

```
sp_encryption helpcol
```

This is the output:

```
Owner.Table.Column            Db.Owner.Keyname
-----------------------        ---------------------
dbo.t1.c1                      keydb1.123456
billyg.t2.c2                   keydb.2345678
tinnap.t3.c3                   coldb.dbo.sample_key3
```

Example 19

When run by the SSO, lists all encrypted columns in table t3 in the current database, and the keys used to encrypt the columns across all available databases:

```
sp_encryption helpcol, t3
```

This is the output:

```
Owner.Table.Column            Db.Owner.Keyname
-----------------------       ---------------------
tinnap.t3.c3                  coldb.dbo.sample_key2
```

When run by user "tinnap", lists all encrypted columns in table t3 in the current database and the keys used to encrypt the columns:

```
sp_encryption helpcol, t3
```

This is the output:

```
Owner.Table.Column          Db.Owner.Keyname
-----------------------     ---------------------
tinnap.t3.c3                coldb.dbo.sample_key3
```

Example 20

When run by the SSO, lists all encrypted columns named c1 in the current database across all available databases, and the keys used to encrypt the columns:

```
sp_encryption helpcol, c1
```

This is the output:

```
Owner.Table.Column          Db.Owner.Keyname
-----------------------     ---------------------
dbo.t1.c1                   keydb1.dbo.sample_key1
```

When run by user “tinnap”, lists all encrypted columns named c1 in the current database and the keyid of the keys used to encrypt the columns if the key is not present in the current database:

```
sp_encryption helpcol, c1
```

This is the output:

```
Owner.Table.Column          Db.Owner.Keyname
-----------------------     ---------------------
dbo.t1.c1                   keydb1.123456
```

Example 21

When run by the SSO, lists all encrypted columns in table dbo.t1 in the current database and the keys used to encrypt the columns across all available databases:

```
sp_encryption helpcol, dbo.t1
```

This is the output:

```
Owner.Table.Column          Db.Owner.Keyname
-----------------------     ---------------------
dbo.t1.c1                   keydb1.dbo.sample_key1
```

When run by user “tinnap”, lists all encrypted columns in table dbo.t1 in the current database and the keyid of the keys used to encrypt the columns if the key is not present in the current database:

```
sp_encryption helpcol, dbo.t1
```

This is the output:

```
Owner.Table.Column              Db.Owner.Keyname
-----------------------         ---------------------
dbo.t1.c1                       keydb1.123456
```

Example 22

When run by the SSO, lists all encrypted columns named c1 in table t1 in the current database and the keys used to encrypt the columns across all available databases:

```
sp_encryption helpcol, t1.c1
```

This is the output:

```
Owner.Table.Column              Db.Owner.Keyname
-----------------------         ---------------------
dbo.t1.c1                       keydb1.dbo.sample_key1
```

When run by user “tinnap”, lists all encrypted columns named c1 in table t1 in the current database and the keyid of the keys used to encrypt the columns if the key is not present in the current database:

```
sp_encryption helpcol, t1.c1
```

This is the output:

```
Owner.Table.Column              Db.Owner.Keyname
-----------------------         ---------------------
dbo.t1.c1                       keydb1.12345678
```

Example 23

When run by the SSO, lists all encrypted columns named c1 in table t1 owned by the DBO in the current database, and the keys used to encrypt the columns across all available databases:

```
sp_encryption helpcol, dbo.t1.c1
```

This is the output:

```
Owner.Table.Column              Db.Owner.Keyname
-----------------------         ---------------------
dbo.t1.c1                       keydb1.dbo.sample_key1
```

When run by user “tinnap”, lists all encrypted columns named c1 in table t1 owned by the DBO, and the keyid of keys used to encrypt the columns if the key is not present in the current database:

```
sp_encryption helpcol, dbo.t1.c1
```

This is the output:

```
Owner.Table.Column           Db.Owner.Keyname
-----------------------      ---------------------
dbo.t1.c1                    keydb1.123456789
```

Example 24

When run by the SSO, lists the properties of the system encryption password in each database:

```
sp_encryption helpkey, system_encr_passwd, all_dbs
```

This is the output:

```
Database  Type of system_encr_passwd  Last modified by
Date
---------- ---------------------------- -------------
---- -------------------
master    persistent    sa    Aug 26 2008 10:05AM
```

Usage

- The privileges granted to the user who runs sp_encryption determines the output. See "Usage restrictions" on page 99 for more information.
- If you run sp_encryption helpkey and no keys are present in the database, you see an informational message.
- You must specify the *key_copy* parameter to get information about key copies. If you do not specify the *key_copy* parameter, sp_encryption returns information only about base keys.
- If *keyname* is NULL in sp_encryption helpkey, keyname, key_copy, lists all the key copies in the current database for a SSO, key custodian, or DBO. If it is run by a user without privileges, it lists all the key copies assigned to the user in the current database and all key copies of the keys owned by the user in the current database.
- For sp_encryption helpcol, *column_name* uses the form *name*.*name*.*name*, where:
  - *name* – if sp_encryption finds no tables of this name, it looks for all columns of that name.
  - *name*.*name* – is *owner.table*. If sp_encryption finds no tables of this name, it looks for a single column named *table.column*.
  - *name*.*name*.*name* – is *owner*.*table*.*name*.

For all columns identified by these rules in the current database, sp_encryption displays column name along with the key used to encrypt the column.

The output for sp_encryption helpcol, *column_name* is *owner*.*table*.*column* and *db*.*owner*.*keyname*. The *keyname* is expressed as *database*.*keyid* when run by non-SSO users, and the key is present in a different database from the encrypted column. The result set is sorted by *owner*.*table*.*column*.

Usage restrictions

- Only an SSO can run sp_encryption helpkey, [,keyname | wildcard], all_dbs to get the properties of keys in all databases. If a user without the sso_role runs this command, they receive an "unauthorized user" error message. If no keys qualify the keyname or wildcard, Adaptive Server returns a message stating `'There are no encryption keys (key copies) like keyname in all databases'`.
- When the SSO runs sp_encryption helpkey, *keyname*, *display_cols*, it lists all columns across all available databases encrypted by *keyname*. If it is run by a user without privileges, it lists the columns in the current database encrypted by *keyname*.

  If the SSO runs sp_encryption helpkey, *keyname*, *display_cols* and the *keyname* value is NULL, it displays all encrypted columns across all available databases. When run by a user without privileges, it displays all encrypted columns in the current database.
- If an SSO, key custodian, or DBO runs sp_encryption helpuser, *user_name*, key_copy without specifying a *user_name* and *key_copy* for the helpuser parameter, it lists all the base keys owned by all users in the current database. If sp_encryption is run by a user without privileges without specifying a *user_name* or *key_copy*, it displays the base keys owned by the current user.

  If any user runs sp_encryption helpuser, *user_name*, it lists all the base keys owned by owner.keyname. If a user without privileges runs the command and owns no base keys, Adaptive Server displays an informational message stating this.

  If an SSO, key custodian, or DBO runs sp_encryption helpuser, *user_name*, *key_copy*, it lists the key copies assigned to *user_name*. If a user without privileges issues this command, its lists the key copies assigned to this user and all the key copies of the keys owned by the user in the current database, with these columns in the result set: Owner.Keyname, Assignee, Type of Password, and Key Recovery. The output is sorted by Assignee.

If *user_name* is NULL for sp_encryption helpuser *user_name*, *key_copy*, it lists all the key copies in the current database for a SSO, key custodian, or DBO. For users without privileges, it lists all the key copies assigned to the user in the current database and the key copies for the keys owned by this user.

- When a SSO, key custodian, or DBO runs sp_encryption helpkey, *keyname*, *key_copy*, it lists the key copies in the current database for *keyname*. If this is run by a user without privileges, it lists the key copies assigned to the user for that *keyname* and the key copies for that *keyname* if the user is the key owner.
- The SSO, key custodian, and DBO can run sp_encryption helpkey, *system_encr_passwd*, display_keys to receive information on all keys and key copies in the current database encrypted by system encryption password. Users without privileges receive information about the base encryption keys or key copies they own or are assigned in the current database. Key copies are encrypted with the system encryption password only when they are created for login association. The output is sorted by *owner*.*keyname*.

Utilities

ddlgen

ddlgen supports generation of DDL statements for encryption keys. The syntax is:

```
ddlgen -Usa -P -Sserver -TEK -Ndb_name.owner.key_name
```

where:

- EK – is the encrypted key type
- *db_name.owner.key_name* – is the fully qualified name for the encrypted key.

The type EK, used for encryption key, generates the DDL to create an encryption key and to grant permissions on it. ddlgen generates encrypted column information and a grant decrypt statement, along with the table definition.

See the *Adaptive Server Enterprise Utility Guide* for the complete ddlgen syntax. See the *Replication Server Administration Guide* for examples of using ddlgen with replicated databases.

Generating DDL for a single encryption key

To generate DDL for an encryption key “ssn_key” in a database called “SampleKeysDB,” the syntax is:

ddlgen -Usa -P -S*server* -TEK -NSampleKeysDB.dbo.ssn_key

If “ssn_key” was created as:

```
create encryption key ssn_key
```

ddlgen generates this output:

```
--------------------------------------------------
-- DDL for EncryptedKey 'ssn_key'
--------------------------------------------------
print 'ssn_key'
go
use SampleKeysDB
go
IF EXISTS (SELECT 1 FROM sysobjects o, sysusers u WHERE
o.uid=u.uid AND o.name = 'ssn_key' AND u.name = 'dbo'
AND o.type = 'EK')
    drop encryption key SampleKeysDB.dbo.ssn_key
IF (@@error != 0)
BEGIN
    PRINT "Error CREATING EncryptedKey 'ssn_key'"
    SELECT syb_quit()
END
go
create encryption key SampleKeysDB.dbo.ssn_key for AES
with keylength 128
init_vector random
go
```

Generating DDL for all encryption keys

This example generates DDL for all encryption keys in a database accounts on a machine named “HARBOR” using port 1955:

```
ddlgen -Uroy -Proy123 -SHARBOR:1955 -TEK
-Naccounts.dbo.%
```

Alternatively, you use the -D option to specify the database name:

This is the output:

```
    ddlgen -Uroy -Proy134 -SHARBOR:1955 -TEK -Ndbo.%
     -Daccounts
------------------------------------------------------
-----------------------
```

```
-- DDL for EncryptedKey 'ssn_key'
------------------------------------------------------
-----------------------
      print 'ssn_key'

create encryption key accounts.dbo.ssn_key
      for AES
      with keylength 128
      init vector random
go


------------------------------------------------------
-----------------------
-- DDL for EncryptedKey 'ek1'
------------------------------------------------------
-----------------------
print 'ek1'

create encryption key accounts.dbo.ek1 as default
     for AES
     with keylength 192
     init vector NULL
go

use accounts
go

grant select on accounts.dbo.ek1 to acctmgr_role
go
```

Generating DDL with -XOD

ddlgen has an option -XOD which generates the create encryption key that specifies the key's encrypted value as represented in sysencryptkeys. Use the -XOD to synchronize encryption keys across servers for data movement.

When a user specifies -XOD, ddlgen generates DDL that includes a system encryption password (if it has been set and DDL is generated for a key encrypted with a system encryption password) and DDL for keys.

For the following syntax and output, the system encryption password has been set in sampleKeysdb, and ek1 has been created with encryption by the system encryption password. The ddlgen command below generates syntax to set the system encryption password using an encrypted version of the original setting in sampleKeysdb. It then creates syntax to create ek1 using the encrypted value of ek1 as stored in sysencryptkeys in sampleKeysdb.

ddlgen -Usa -P -S*server* -TEK -NsampleKeysdb.dbo.ek1 -XOD

The output for the command is:

```
-- System Encryption Password

use SampleKeysDB
go

sp_encryption 'system_encr_passwd',
'0x8e050e3bb607225c60c7cb9f59124e99866ca22e677b2cdc9a4d09775850f4721',
NULL, 2, 0
go

--------------------------------------------------------------------------
-- DDL for EncryptedKey 'ek1'
--------------------------------------------------------------------------

print '<<<<< CREATING EncryptedKey - "ek1" >>>>>'
go

IF EXISTS (SELECT 1 FROM sysobjects o, sysusers u WHERE o.uid=u.uid

    AND o.name = 'ek1' AND u.name = 'dbo'

    AND 'o.type = 'EK')

       drop encryption key sampleKeysdb.dbo.ek1
go
if (@@error != 0)
BEGIN
    PRINT "Error CREATING EncryptedKey 'ek1'"
    SELECT syb_quit()
END
go
create encryption key SampleKeysDB.dbo.ek1 for AES
with keylength 128
passwd 0x0000C7BC28C3020AC21401
init_vector NULL
keyvalue
0xCE74DB1E028FF15D908CD066D380AB4AD3AA88284D6F7742DFFCADCAABE4100D01
keystatus 32
```

```
go
```

Note When migrating keys from a source to a target server using ddlgen, set the system encryption password, if it exists, to NULL in the target server to run the ddlgen output from the source server for encryption keys generated using -XOD. If you do not set the password to NULL, you see errors when you try to execute the ddlgen output against the target server.

Generating DDL without specifying -XOD

If you do not specify the -XOD option, and the key to be migrated has been created in the source database using the with passwd clause, ddlgen generates a create encryption key command with password as its explicit password.This is similar to what ddlgen does for roles and login passwords, and its output looks similar to the following:

```
-----------------------------------------------------------------------
-- DDL for EncryptedKey 'ssn_key'
-----------------------------------------------------------------------
print '<<<< CREATING EncryptedKey - "ssn_key" >>>>>'
go
use SampleKeysDB
go
IF EXISTS (SELECT 1 FROM sysobjects o, sysusers u WHERE o.uid=u.uid
    AND o.name = 'ssn_key' AND u.name = 'dbo' AND o.type = 'EK'
       drop encryption key SampleKeysDB.dbo.ssn_key
IF (@@error !=0)
BEGIN
    PRINT "Error CREATING EncryptedKey 'k1'"
    SELECT syb_quit()
END
go

-- The DDL is generated with a default password – 'password' as
-- a password was specified when this key was created.

create encryption key SampleKeysDB.dbo.ssn_key for AES
with keylength 128
passwd 'password'
init_vector random
go
```

Key-copy support

ddlgen generates DDL for key copies along with the DDL for the base key. For example, the following syntax generates DDL for “ssn_key” and its key copies:

```
ddlgen -Usa -P -Sserver -TEK -NSampleKeysDB.dbo.ssn_key
```

The output from ddlgen looks like:

```
-----------------------------------------------------------------------------
-- DDL for EncryptedKey 'ssn_key'
-----------------------------------------------------------------------------
print '<<<<< CREATING EncryptedKey - "k1" >>>>>'
use SampleKeysDB
go
IF EXISTS (SELECT 1 FROM sysobjects o, sysusers u WHERE  o.uid=u.uid
       AND o.name = 'ssn_key' AND u.name = dbo AND o.type = 'EK)
          drop encryption key SampleKeysDB.dbo.ssn_key
IF (@@error != 0)
BEGIN
       PRINT "Error CREATING EncryptedKey 'ssn_key'"
       SELECT syb_quit()
END
go

-- The DDL is generated with a default password – 'password' as
-- a password was specified when this key was created.

create encryption key SampleKeysDB.dbo.ssn_key for AES
with keylength 128
passwd 'password'
init_vector random
go

alter encryption key SampleKeysDB.dbo.ssn_key
with passwd 'password'
add encryption with passwd 'passwd'
for user 'user1'
go
```

If you include the -XOD flag, the DDL for key copy looks similar to this:

```
alter encryption key SampleKeysDB.dbo.ssn_key add encryption
with keyvalue
0x84A7360AA0B28801D6D4CBF2F8219F634EE641E1082F221A2C58C9BBEC9F49B501
passwd 0x000062DF4B8DA5709E5E01
keystatus 257
for user 'user1'
```

```
go
```

Encryption key copy (EKC) filter

ddlgen supports the EKC (encryption key copy) extended type on the -F filter argument, which suppresses the generation of key copies for encryption keys.

This example uses -FEKC to avoid creating DDL for key copies when generating DDL for the “ssn_key” encryption key:

```
ddlgen -Usa -P -Sserver -TEK -NSampleKeysDB.dbo.ssn_key  -FEKC
```

This is the output from ddlgen:

```
-----------------------------------------------------------------------------
-- DDL for EncryptedKey 'ssn_key'
-----------------------------------------------------------------------------
print '<<<<< CREATING EncryptedKey - "k1" >>>>>'
go
use SampleKeysDB
go
IF EXISTS (SELECT 1 FROM sysobjects o, sysusers u WHERE o.uid=u.uid
       AND o.name = 'ssn_key' AND u.name = 'dbo' AND o.type = 'EK')
          drop encryption key SampleKeysDB.dbo.ssn_key
IF (@@error != 0)BEGIN
       PRINT "Error CREATING EncryptedKey 'ssn_key'"
END
go

-- The DDL is generated with a default password – 'password' as
-- a password was specified when this key was created.

create encryption key SampleKeysDB.dbo.ssn_key for AES
with keylength 128
passwd 'password'
init_vector random
go
```

Create table DDL

ddlgen can generate decrypt_default statements for encrypted columns along with a table’s DDL.

This example issues a ddlgen command on a table called employee, which has an “ssn” column that is encrypted with encryption key “ssn_key,” and a decrypt default value that is set to 100:

```
ddlgen -Usa -P -Sserver -TU -N db1.dbo.employee
```

The DDL output from the command is:

```
create table employee (
  ssn            int       not null  encrypt with ssn_key decrypt_default 100 ,
  last_name      int       not null ,
  first_name     int       not null
)
lock allpages
 on 'default'
go
```

sybmigrate

sybmigrate migrates data from one server to another.

By default, sybmigrate migrates encrypted columns in cipher text format. This avoids the overhead of decrypting data at the source and reencrypting it at the target. In some cases, however, sybmigrate chooses the reencrypt method of migration, which does decrypt data at the source and reencrypts it at the target.

For databases with encrypted columns, sybmigrate:

1. Migrates the system encryption password. If you specify not to migrate the system encryption password, sybmigrate migrates the encrypted columns using the reencrypt method instead of migrating cipher text.
2. Migrates the encryption keys. You may select the keys to migrate. sybmigrate automatically selects keys in the current database used to encrypt columns in the same database. If you have selected migration of the system encryption password, sybmigrate migrates the encryption keys using their actual values. The key values from the sysencryptkeys system table have been encrypted using the system encryption password and these are the values that are migrated. If you have not migrated the system encryption password, sybmigrate migrates the keys by name, to avoid migrating keys that do not decrypt correctly at the target. Migrating the key by name causes the key at the target to be created with a different key value from the key at the source.
3. Migrates the data. By default, the data is transferred in cipher text form. cipher text data can be migrated to a different operating system. Character data requires that the target server uses the same character set as the source.

sybmigrate works on a database as a unit of work. If your database on the source server has data encrypted by a key in another database, migrate the key's database first.

sybmigrate chooses to reencrypt migrated data when:

- Any keys in the current database are specifically not selected for migration, or already exist in the target server. There is no guarantee that the keys at the target are identical to the keys are the source, so the migrating data must be reencrypted.
- The system password was not selected for migration. When the system password at the target differs from that at the source, the keys cannot be migrated by value. In turn, the data cannot be migrated as cipher text.
- The user uses the following flag:

```
sybmigrate -T 'ALWAYS_REENCRYPT'
```

Reencrypting data can slow performance. A message to this effect is written to the migration log file when you perform migration with reencryption mode.

To migrate encrypted columns, you must have both sa_role and sso_role enabled.

bulk copy (*bcp*)

bcp transfers encrypted data in and out of databases in either plain text or cipher text form. By default, bcp copies plain text data, processing them as follows:

- Data is automatically encrypted by Adaptive Server before insertion when executing bcp in. Slow bcp is used. The user must have insert and select permission on all columns.
- Data is automatically decrypted by Adaptive Server when executing bcp out. select permission is required on all columns; in addition, decrypt permission is required on the encrypted columns.

This example copies the customer table out as plain text data in native machine format:

```
bcp uksales.dbo.customer out uk_customers -n -Uroy -Proy123
```

If the data to be copied out as plain text is encrypted by a key that uses an explicit password, you can supply that password to bcp using the --c password or --colpasswd options.

For example, if the salary column in the employee table is encrypted by a key that is protected by an explicit password, you can only copy out the salary data as plain text by providing bcp with the password, as follows:

```
bcp hr.dbo.employee out  -c -Upjones -PX15tgol --
colpasswd hr.dbo.employee.salary '4mIneIsonly'
```

Alternatively, if you know the name of the key that encrypts the salary column, you can use:

```
bcp hr.dbo.employee out  -c -Upjones -PX15tgol --
keypasswd keydb.dbo.hr_key '4mIneIsonly'
```

bcp uses the password to issue a set encryption passwd command before selecting the data.

Use the --keypasswd and --colpasswd options in a similar way on the bcp command line when copying the data back in.

Use the -C option for bcp to copy the data as cipher text. When copying cipher text, you may copy data out and in across different operating systems. If you are copying character data as cipher text, both platforms must support the same character set.

The -C option for bcp allows administrators to run bcp when they lack decrypt permission on the data. When the -C option is used, bcp processes data as follows:

- Data is assumed to be in cipher text format during execution of bcp in, and Adaptive Server performs no encryption. Use the -C option only if the file being copied into Adaptive Server was created using the -C option on bcp out. The cipher text must have been copied from a column with exactly the same column attributes and encrypted by the same key as the column into which the data is being copied. Fast bcp is used. The user must have insert and select permission on the table.
- Data is copied out of Adaptive Server without decryption on bcp out. The cipher text data is in hexadecimal format. The user must have select permission on all columns. For copying cipher text, decrypt is not required on the encrypted columns.
- Encrypted char or varchar data retains the character set used by Adaptive Server at the time of encryption. If the data is copied in cipher text format to another server, the character set used on the target server must match that of the encrypted data copied from the source. The character set associated with the data on the source server when it was encrypted is not stored with the encrypted data and is not known or converted on the target server.

You can also perform bcp without the -C option to avoid the character set issue.

You cannot use the -J option (for character set conversion) with the -C option.

The following example copies the customer table. The cc_card column is copied out as human-readable cipher text. Other columns are copied in character format. User “roy” is not required to have decrypt permission on customer cc_card.

```
 bcp uksales.dbo.customer out uk_customers -C -c -Uroy
-Proy123
```

When copying data as cipher text, ensure that the same keys are available in the database when the data is copied back in. If necessary, use the ddlgen utility to move keys from one database to another.

Component Integration Services (CIS)

By default, encryption and decryption are handled by the remote Adaptive Server. CIS makes a one-time check for encrypted columns on the remote Adaptive Server. If the remote Adaptive Server supports encryption, CIS updates the local syscolumns catalog with the encrypted-column-related metadata as follows:

- create proxy_table automatically updates syscolumns with any encrypted-column information from the remote tables.
- create existing table automatically updates syscolumns with any encrypted-column metadata from the remote tables. The encrypt keyword is not allowed in the *columnlist* for create existing table. CIS automatically marks columns as encrypted if it finds any encrypted columns on the remote table.
- create table at the location with encrypted columns is not allowed.
- alter table is not allowed on encrypted columns for proxy tables.
- select into existing brings the plain text from the source and inserts it into destination table. The local Adaptive Server then encrypts the plain text before insertion into any encrypted columns.

The following columns are updated from the remote server’s syscolumns catalog:

- encrtype – type of data on disk.
- encrlen – length of encrypted data.
- status2 – status bits that indicate that column is encrypted.

Replicating encrypted data

If your site replicates schema changes, the following DDL statements are replicated:

- alter encryption key
- create table and alter table with extensions for encryption
- create encryption key
- grant and revoke create encryption key
- grant and revoke select on the key
- grant and revoke decrypt on the column
- sp_encryption system_encr_passwd
- drop encryption key

The keys are replicated in encrypted form.

If your system does not replicate DDL, manually synchronize encryption keys at the replicate site. ddlgen supports a special form of create encryption key for replicating the key's value. See "ddlgen" on page 100.

For DML replications, the insert and update commands replicate encrypted columns in encrypted form, which safeguards replicated data while Replication Server processes it in stable queues on disk.

Replication Server version 12.6 ESD # 5 and later supports encrypted columns.

See the *Replication Server Administration Guide* for information on using encryption during replication.

Index