Encrypted Columns Users Guide

Adaptive Server® Enterprise
15.5
# Contents

About This Book ............................................................................................................................ v

## CHAPTER 1 Overview of Encryption ................................................................. 1

## CHAPTER 2 Creating and Managing Encryption Keys ................................. 5

- Creating encryption keys......................................................................................... 5
- Key protection ........................................................................................................... 10
  - Granting access to keys ..................................................................................... 12
  - Key protection using the system-encryption password ........................................ 12
  - Changing the key ................................................................................................. 14
  - Separating keys from data ................................................................................. 14
- Dropping encryption keys......................................................................................... 15

## CHAPTER 3 Encrypting Data............................................................................. 17

- Specifying encryption on new tables ................................................................. 18
  - Specifying encryption on select into ............................................................... 19
- Encrypting data in existing tables ................................................................. 20
- Creating indexes and constraints on encrypted columns ................................. 21
- Creating domain and access rules on encrypted columns ................................. 22
- Decrypt permission ................................................................................................. 22
  - Revoking decryption permission ....................................................................... 24
- Restricting decrypt permission .............................................................................. 24
- Assigning privileges for restricted decrypt permissions ....................................... 24
- Returning default values instead of decrypted data ............................................ 25
  - Defining a decrypt default.................................................................................... 25
  - Permissions and decrypt default ........................................................................ 27
  - Columns with decrypt default values .................................................................. 28
  - Decrypt default columns and query qualifications ............................................. 28
  - decrypt default and implicit grants ..................................................................... 29
  - decrypt default and insert, update, and delete statements .................................. 30
  - Removing decrypt default .................................................................................... 31
- Length of encrypted columns ................................................................................. 32
About This Book

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

How to use this book
- Chapter 3, “Encrypting Data,” – describes what data can be encrypted and the steps to perform 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,” – describes performance implications and resolutions for encrypted columns.

Related documents
The Adaptive Server® Enterprise documentation set consists of:
- 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. To check for critical product or document information that was added after the release of the product CD, use the Sybase® Product Manuals Web site.
- The installation guide for your platform – describes installation, upgrading, and some configuration procedures for all Adaptive Server and related Sybase products.
• **New Feature Summary** – describes the new features in Adaptive Server, the system changes added to support those features, and changes that may affect your existing applications.

• **Active Messaging Users Guide** – describes how to use the Active Messaging feature to capture transactions (data changes) in an Adaptive Server Enterprise database, and deliver them as events to external applications in real time.

• **Component Integration Services Users Guide** – explains how to use Component Integration Services to connect remote Sybase and non-Sybase databases.

• **The Configuration Guide** for your platform – provides instructions for performing specific configuration tasks.

• **Glossary** – defines technical terms used in the Adaptive Server documentation.

• **Historical Server Users Guide** – describes how to use Historical Server to obtain performance information from 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 Users 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).

• **Migration Technology Guide** – describes strategies and tools for migrating to a different version of Adaptive Server.

• **Monitor Client Library Programmers Guide** – describes how to write Monitor Client Library applications that access Adaptive Server performance data.

• **Monitor Server Users Guide** – describes how to use Monitor Server to obtain performance statistics from Adaptive Server.

• **Monitoring Tables Diagram** – illustrates monitor tables and their entity relationships in a poster format. Full-size available only in print version; a compact version is available in PDF format.

• **Performance and Tuning Series** – is a series of books that explain how to tune Adaptive Server for maximum performance:
  
  • **Basics** – contains the basics for understanding and investigating performance questions in Adaptive Server.
• **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.

• **Locking and Concurrency Control** – describes how to use locking schemes to improve performance, and how to select indexes to minimize concurrency.

• **Monitoring Adaptive Server with sp_sysmon** – discusses how to use `sp_sysmon` to monitor performance.

• **Monitoring Tables** – describes how to query Adaptive Server monitoring tables for statistical and diagnostic information.

• **Physical Database Tuning** – describes how to manage physical data placement, space allocated for data, and the temporary databases.

• **Query Processing and Abstract Plans** – explains how the optimizer processes queries, and how to use abstract plans to change some of the optimizer plans.

• **Quick Reference Guide** – provides a comprehensive listing of the names and syntax for commands, functions, system procedures, extended system procedures, datatypes, and utilities in a pocket-sized book (regular size when viewed in PDF format).

• **Reference Manual** – is a series of books that contains detailed Transact-SQL® information:
  • **Building Blocks** – discusses datatypes, functions, global variables, expressions, identifiers and wildcards, and reserved words.
  • **Commands** – documents commands.
  • **Procedures** – describes system procedures, catalog stored procedures, system extended stored procedures, and `dbcc` stored procedures.
  • **Tables** – discusses system tables, monitor 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 instructions for diagnosing system problems. The second part of **Volume 1** is an in-depth discussion about 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 **Volume 2** 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 Users Guide** – documents Transact-SQL, the Sybase-enhanced version of the relational database language. This guide serves as a textbook for beginning users of the database management system, and also contains detailed descriptions of the pubs2 and pubs3 sample databases.

• **Troubleshooting: Error Messages Advanced Resolutions** – contains troubleshooting procedures for problems you may encounter. The problems discussed here are the ones the Sybase Technical Support staff hear about most often.

• **Encrypted Columns Users Guide** – describes how to configure and use encrypted columns with Adaptive Server.

• **In-Memory Database Users Guide** – describes how to configure and use in-memory databases.

• **Using Adaptive Server Distributed Transaction Management Features** – explains how to configure, use, and troubleshoot Adaptive Server DTM features in distributed transaction processing environments.

• **Using Backup Server with IBM® Tivoli® Storage Manager** – describes how to set up and use the IBM Tivoli Storage Manager to create Adaptive Server backups.

• **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 Users 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 are available in 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
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
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.
2. Click MySybase and create a MySybase profile.

Finding the latest information on EBFs and software maintenance
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 2 shows the conventions for syntax statements that appear in this manual:

<table>
<thead>
<tr>
<th>Element</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command names, procedure names, utility names, and other keywords display in sans serif font.</td>
<td>select, spConfigure</td>
</tr>
<tr>
<td>Database names and datatypes are in sans serif font.</td>
<td>master database</td>
</tr>
<tr>
<td>Book names, file names, variables, and path names are in italics.</td>
<td>System Administration Guide, sql.ini file, column_name, $SYBASE/ASE directory</td>
</tr>
<tr>
<td>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.</td>
<td>select column_name, from table_name, where search_conditions</td>
</tr>
<tr>
<td>Type parentheses as part of the command.</td>
<td>compute row_aggregate (column_name)</td>
</tr>
<tr>
<td>Double colon, equals sign indicates that the syntax is written in BNF notation. Do not type this symbol. Indicates “is defined as”.</td>
<td>::=</td>
</tr>
<tr>
<td>Curly braces mean that you must choose at least one of the enclosed options. Do not type the braces.</td>
<td>{cash, check, credit}</td>
</tr>
<tr>
<td>Brackets mean that to choose one or more of the enclosed options is optional. Do not type the brackets.</td>
<td>[cash</td>
</tr>
<tr>
<td>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.</td>
<td>cash, check, credit</td>
</tr>
</tbody>
</table>
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.

- 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
  ```

- An ellipsis (...) means that you can repeat the last unit as many times as you like.

- 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.
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 2 shows the conventions for syntax statements that appear in this manual:

<table>
<thead>
<tr>
<th>Element</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Command names, procedure names, utility names, and other keywords display in sans serif font.</td>
<td>select sp_configure</td>
</tr>
<tr>
<td>Database names and datatypes are in sans serif font.</td>
<td>master database</td>
</tr>
<tr>
<td>Book names, file names, variables, and path names are in italics.</td>
<td>System Administration Guide sql.ini file column_name $SYBASE/ASE directory</td>
</tr>
<tr>
<td>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.</td>
<td>select column_name from table_name where search_conditions</td>
</tr>
<tr>
<td>Type parentheses as part of the command.</td>
<td>compute row_aggregate (column_name)</td>
</tr>
<tr>
<td>Double colon, equals sign indicates that the syntax is written in BNF notation. Do not type this symbol. Indicates “is defined as”.</td>
<td>::=</td>
</tr>
<tr>
<td>Curly braces mean that you must choose at least one of the enclosed options. Do not type the braces.</td>
<td>{cash, check, credit}</td>
</tr>
<tr>
<td>Brackets mean that to choose one or more of the enclosed options is optional. Do not type the brackets.</td>
<td>[cash</td>
</tr>
<tr>
<td>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.</td>
<td>cash, check, credit</td>
</tr>
<tr>
<td>The pipe or vertical bar (</td>
<td>) means you may select only one of the options shown.</td>
</tr>
</tbody>
</table>
### Element Example

An ellipsis (...) means that you can repeat the last unit as many times as you like.

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

  ```sql
  sp_dropdevice [device_name]
  ```

  For a command with more options:

  ```sql
  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:

  ```sql
  select * from publishers
  ```

- Examples of output from the computer appear as follows:

<table>
<thead>
<tr>
<th>pub_id</th>
<th>pub_name</th>
<th>city</th>
<th>state</th>
</tr>
</thead>
<tbody>
<tr>
<td>0736</td>
<td>New Age Books</td>
<td>Boston</td>
<td>MA</td>
</tr>
<tr>
<td>0877</td>
<td>Binnet &amp; Hardley</td>
<td>Washington</td>
<td>DC</td>
</tr>
<tr>
<td>1389</td>
<td>Algodata Infosystems</td>
<td>Berkeley</td>
<td>CA</td>
</tr>
</tbody>
</table>

  (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 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*.
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.

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 Technology (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 32. 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.
CHAPTER 1    Overview of Encryption

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 18 and “Encrypting data in existing tables” on page 20.

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 36.

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.
Creating and Managing Encryption Keys

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 39.
- 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.

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

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.

**Note** A Security and Directory Services license is required to enable this parameter. If the parameter is not enabled, OpenSSL security provider performs login password encryption.

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`. 
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:
Creating encryption keys

- **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.

- **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 42.

- **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:

```sql
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:

```sql
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:

```sql
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 PASSWORD '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.

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  

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  

Encrypted CEK saved in sysencryptkeys  

Decrypted CEK used for DML
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 24.
• 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 42.

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:

```plaintext
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.
Key protection

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.

```sql
create encryption key cc_key_new for AES

alter table customer modify creditcard encrypt with cc_key_new
```

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.
Dropping encryption keys
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 32 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.

```sql
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:

```sql
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:

```sql
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:

```sql
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:

```sql
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]]
```

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

See the Reference Manual: Commands.

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`:

```sql
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 random padding 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:

```sql
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)
```
Creating domain and access rules on encrypted columns

You can create domain rules, check constraints, or access rules on encrypted columns. However, decrypt permission is required on the encrypted column when the encrypted column is used in target list, where clause, and so on. This example creates the rule_creditcard rule on the creditcard column, which has a domain rule defined:

```
create encryption key cc_key
    with init_vector null

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

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

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

Decrypt permission

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

The table owner 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 24.

Users with only select permission on an encrypted column can still process encrypted columns as cipher text through the bulk copy command. Additionally, if an encrypted column specifies a decrypt default value, the column can be named in a select target list or in a where clause by users who do not have permission to decrypt data. See "Returning default values instead of decrypted data" on page 25.
Revoking decryption permission

You can revoke a user’s decryption permission using:

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

For example:

\[ \text{revoke decrypt on customer from public} \]

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:

```
Decrypted 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.

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.
Returning default values instead of decrypted data

- **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:

```sql
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:

```sql
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:

```sql
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:

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

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

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

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

```sql
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:

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

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

```sql
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.

```sql
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:

```sql
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.

Decryption 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.

   ```sql
   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:
CHAPTER 3 Encrypting Data

```sql
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:

```sql
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:

```sql
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:

```sql
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.

Encrypted Columns Users Guide  29
Returning default values instead of decrypted data

exec p_emp

<table>
<thead>
<tr>
<th>name</th>
<th>ssn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tinna Salt</td>
<td>123-45-6789</td>
</tr>
<tr>
<td>Joe Cool</td>
<td>321-54-9879</td>
</tr>
</tbody>
</table>

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:

<table>
<thead>
<tr>
<th>name</th>
<th>ssn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tinna Salt</td>
<td>000-00-0000</td>
</tr>
<tr>
<td>Joe Cool</td>
<td>000-00-0000</td>
</tr>
</tbody>
</table>

**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. Allows a user to update any of the data.
3. Applies the changed row back to the same or a different table

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

```sql
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:

```
<table>
<thead>
<tr>
<th>name</th>
<th>cc_num</th>
</tr>
</thead>
<tbody>
<tr>
<td>Paul Jones</td>
<td>-1</td>
</tr>
<tr>
<td>Mick Watts</td>
<td>-1</td>
</tr>
</tbody>
</table>
```

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

<table>
<thead>
<tr>
<th>User-specified column type</th>
<th>Input data length</th>
<th>Encrypted column type</th>
<th>Maximum encrypted data length (no init vector)</th>
<th>Actual encrypted data length (no init vector)</th>
<th>Maximum encrypted data length (with init vector)</th>
<th>Actual encrypted data length (with init vector)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bigint</td>
<td>8</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>unsigned bigint</td>
<td>8</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>tinyint, smallint, or int (signed or unsigned)</td>
<td>1, 2, or 4</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>tinyint, smallint, or int (signed or unsigned)</td>
<td>0 (null)</td>
<td>varbinary</td>
<td>17</td>
<td>0</td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>float, float(4), real</td>
<td>4</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>float, float(4), real</td>
<td>0 (null)</td>
<td>varbinary</td>
<td>17</td>
<td>0</td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>float(8), double</td>
<td>8</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>float(8), double</td>
<td>0 (null)</td>
<td>varbinary</td>
<td>17</td>
<td>0</td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>numeric(10,2)</td>
<td>3</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>numeric (38,2)</td>
<td>18</td>
<td>varbinary</td>
<td>33</td>
<td>33</td>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>numeric (38,2)</td>
<td>0 (null)</td>
<td>varbinary</td>
<td>33</td>
<td>0</td>
<td>49</td>
<td>0</td>
</tr>
<tr>
<td>char, varchar (100)</td>
<td>1</td>
<td>varbinary</td>
<td>113</td>
<td>17</td>
<td>129</td>
<td>33</td>
</tr>
<tr>
<td>char, varchar(100)</td>
<td>14</td>
<td>varbinary</td>
<td>113</td>
<td>17</td>
<td>129</td>
<td>33</td>
</tr>
<tr>
<td>char, varchar(100)</td>
<td>15</td>
<td>varbinary</td>
<td>113</td>
<td>33</td>
<td>129</td>
<td>49</td>
</tr>
<tr>
<td>char, varchar(100)</td>
<td>31</td>
<td>varbinary</td>
<td>113</td>
<td>49</td>
<td>129</td>
<td>65</td>
</tr>
<tr>
<td>char, varchar(100)</td>
<td>0 (null)</td>
<td>varbinary</td>
<td>113</td>
<td>0</td>
<td>129</td>
<td>0</td>
</tr>
<tr>
<td>binary, varbinary(100)</td>
<td>1</td>
<td>varbinary</td>
<td>113</td>
<td>17</td>
<td>129</td>
<td>33</td>
</tr>
<tr>
<td>binary, varbinary(100)</td>
<td>14</td>
<td>varbinary</td>
<td>113</td>
<td>17</td>
<td>129</td>
<td>33</td>
</tr>
<tr>
<td>binary, varbinary(100)</td>
<td>15</td>
<td>varbinary</td>
<td>113</td>
<td>33</td>
<td>129</td>
<td>49</td>
</tr>
<tr>
<td>binary, varbinary(100)</td>
<td>31</td>
<td>varbinary</td>
<td>113</td>
<td>49</td>
<td>129</td>
<td>65</td>
</tr>
<tr>
<td>binary, varbinary(100)</td>
<td>0 (null)</td>
<td>varbinary</td>
<td>113</td>
<td>0</td>
<td>65</td>
<td>0</td>
</tr>
<tr>
<td>unichar(10)</td>
<td>2 (1 unichar character)</td>
<td>varbinary</td>
<td>33</td>
<td>17</td>
<td>49</td>
<td>33</td>
</tr>
<tr>
<td>unichar(10)</td>
<td>20 (10 unichar characters)</td>
<td>varbinary</td>
<td>33</td>
<td>33</td>
<td>49</td>
<td>49</td>
</tr>
<tr>
<td>univarchar(20)</td>
<td>20 (10 unichar characters)</td>
<td>varbinary</td>
<td>49</td>
<td>33</td>
<td>65</td>
<td>49</td>
</tr>
</tbody>
</table>
**Length of encrypted columns**

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

**Table 3-2: datatype length for encrypted columns**

<table>
<thead>
<tr>
<th>Datatype</th>
<th>Input data length</th>
<th>Encrypted column type</th>
<th>Max encrypted data length (no init_vector)</th>
<th>Actual encrypted data length</th>
<th>Max encrypted data length with init_vector</th>
<th>Actual encrypted data length with init_vector</th>
</tr>
</thead>
<tbody>
<tr>
<td>date</td>
<td>4</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>time</td>
<td>4</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>time null</td>
<td></td>
<td>varbinary</td>
<td>17</td>
<td>0</td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>smalldatetime</td>
<td>4</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>datetime</td>
<td>8</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>smallmoney</td>
<td>4</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>money</td>
<td>8</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
<tr>
<td>money null</td>
<td></td>
<td>varbinary</td>
<td>17</td>
<td>0</td>
<td>33</td>
<td>0</td>
</tr>
<tr>
<td>bit</td>
<td>1</td>
<td>varbinary</td>
<td>17</td>
<td>17</td>
<td>33</td>
<td>33</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing encrypted columns</td>
<td>35</td>
</tr>
<tr>
<td>Permissions for decryption</td>
<td>36</td>
</tr>
<tr>
<td>Dropping encryption</td>
<td>37</td>
</tr>
</tbody>
</table>

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 47
  - 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.
Permissions for decryption

- 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 47. 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 28.
  - 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 63.

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.
Dropping encryption
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.

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role of the key custodian</td>
<td>39</td>
</tr>
<tr>
<td>Key protection using user-specified passwords</td>
<td>42</td>
</tr>
</tbody>
</table>
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 responsibility. 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>
<thead>
<tr>
<th>Role</th>
<th>Can create encryption key?</th>
<th>Can use key in a schema definition?</th>
<th>Can decrypt encrypted data?</th>
</tr>
</thead>
<tbody>
<tr>
<td>sso_role</td>
<td>Yes</td>
<td>No, requires create table permission</td>
<td>No. User with role may have knowledge of password, but requires select permission on table (SSO has implicit decrypt permission).</td>
</tr>
<tr>
<td>sa_role</td>
<td>No, requires create encryption key permission</td>
<td>Yes, but must be granted select permission on the key</td>
<td>No, requires knowledge of password</td>
</tr>
<tr>
<td>keycustodian_role</td>
<td>Yes</td>
<td>No, requires create table permission</td>
<td>No. User with role may have knowledge of password, but requires decrypt and select permission on table or column.</td>
</tr>
</tbody>
</table>
Key protection using user-specified passwords

<table>
<thead>
<tr>
<th>Role</th>
<th>Can create encryption key?</th>
<th>Can use key in a schema definition?</th>
<th>Can decrypt encrypted data?</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBO or schema owner</td>
<td>No, requires create encryption key permission</td>
<td>Yes, but must be granted select permission on the key</td>
<td>No, requires knowledge of password.</td>
</tr>
<tr>
<td>User</td>
<td>No</td>
<td>No</td>
<td>Yes, but must be granted decrypt or select permission and have knowledge of key's password.</td>
</tr>
</tbody>
</table>

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 the key 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:

```plaintext
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.
Key protection using user-specified passwords

- 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.

See “Application transparency using login passwords on key copies” on page 50 for alternatives to encrypting key copies with a user’s login password that do not require the key copy assignee to execute alter encryption key.

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 up to 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.
Key protection using user-specified passwords

- 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:

   ```sql
   create encryption key key1 with passwd 'WorldsBiggestSecret'
   ```

2. Razi grants `select` permission on `key1` to DBO for schema creation:

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

3. DBO creates schema and grants table and column-level access to Bill:

   ```sql
   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.

   ```sql
   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:

   ```sql
   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:

  ```sql
  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.

You cannot create key copies for user “guest” for login association.

Encrypting a key copy with a login password requires two-steps.

### 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 50). 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
  ```
Key protection using user-specified passwords

- **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:

```sql
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:

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

<table>
<thead>
<tr>
<th>ename</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priscilla Kramnik</td>
</tr>
</tbody>
</table>

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):

```sql
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:

```sql
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.ssn
```

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
```
Key protection using user-specified passwords

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:

```sql
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.
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.

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

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

   ```sql
   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:

   ```sql
   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:

```plaintext
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 56.

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
```
Key protection using user-specified passwords

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

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss of password on key copy</td>
<td>55</td>
</tr>
<tr>
<td>Loss of login password</td>
<td>56</td>
</tr>
<tr>
<td>Loss of password on base key</td>
<td>56</td>
</tr>
<tr>
<td>Key recovery commands</td>
<td>57</td>
</tr>
<tr>
<td>Changing ownership of encryption keys</td>
<td>59</td>
</tr>
</tbody>
</table>

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
   ```

Encrypted Columns Users Guide  55
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:

   ```sql
   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 56 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:

   ```sql
   select emp_salary from emp
   where name like 'Priscilla%'
   ```

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 recoverer 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:
Key recovery commands

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 59.

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.
Changing ownership of encryption keys
CHAPTER 7

Auditing Encrypted Columns

<table>
<thead>
<tr>
<th>Topic</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Auditing options</td>
<td>61</td>
</tr>
<tr>
<td>Audit values</td>
<td>61</td>
</tr>
<tr>
<td>Event names and numbers</td>
<td>61</td>
</tr>
<tr>
<td>Masking passwords in command text auditing</td>
<td>62</td>
</tr>
<tr>
<td>Auditing actions of the key custodian</td>
<td>62</td>
</tr>
</tbody>
</table>

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"
```
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.

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, in a case-insensitive search, the string abc matches all strings in the following range: 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:

```sql
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:

```sql
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:

```sql
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.

```sql
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 65 apply.

For example:

```sql
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:

```sql
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.
Movement of encrypted data as cipher text
Index

Symbols

10, 61
::= (BNF notation)
in SQL statements xi, xiii
, (comma)
in SQL statements xi, xiii, xiv
{} (curly braces)
in SQL statements xi, xiii, xiv
() (parentheses)
in SQL statements xi, xiii
[] (square brackets)
in SQL statements xi, xiii, xiv

A
accessing encrypted data 35
  syntax 47
adding decrypt default 25
alter encryption key 10
alter encryption key command 42
alter table, to create encryption 14
application transparency 50
as default 8
auditing
  actions of key custodian 62
  encrypted columns 61
masking passwords in command text 62
options 61
values 61

B
Backus Naur Form (BNF) notation xi, xiii
  base key 43
  loss of password 56
BNF notation in SQL statements xi, xiii
  brackets. See square brackets [ ]

C
capabilities of encryption column support 1
case sensitivity
  in SQL xi, xv
ce_key
  using for building index 21
ce_key_new
  used to create encryption key 14
CEK, column-encryption key 10
changing a key’s password 43
cipher text
  encoded form for data 2
  increases length of encrypted column 2
  movement of encrypted data as 67
  sentinel byte appended to 32
columns
  encrypting, syntax 20
  encryption 32
  processing encrypted 35
  with decrypt default values 28
  with query qualifications 28
comma (,)
in SQL statements xi, xiii, xiv
command
  21
  alter encryption key 42
  create encryption key 42
  exec 22
  select 36
  select into, requires column-level permissions 19
timestamp 34
command text auditing, masking passwords in command text 62
commands
  for removing decrypt defaults 31
  key recovery 57
  syntax for key recovery 57
  syntax for sharing the password 57
text 34
computed column
Index

cannot encrypt 18
encrypted column cannot appear in definition 18
conventions
See also syntax
Transact-SQL syntax xi, xiii
used in the Reference Manual xi, xiii
copies
changing passwords on key 46
creating key 45
key, with login password change 53
create
index on encrypted column 21
create encryption key
examples 9
permissions 10
create encryption key 7, 10, 14
create encryption key command 42
create encryption key syntax 42
create index 21
create table partial syntax for encryption 18
creating
encryption keys 5
key copies 45
password, instructions for 13
curly braces ({}) in SQL statements xi, xiii, xiv

D
data access
users and roles 41
data, encrypted, movement as cipher text 67
database
different, encrypting key from 14
encrypting key from 14
datatypes not supported
test 34
datatypes, encryptable 17
decrypt default
adding and removing 25
defining 25
implicit grants 29
insert and delete 30
permissions 27
removing 31
decrypt default columns
query qualifications 28
decrypt default values, columns with 28
decrypt permission
grant decrypt 22
decrypt permission 1
decrypt_default parameter 25
decrypted data, returning default values instead of 25
decryption
permissions 36
default encryption key
create 14
default values, returning 25
dropping
encryption 15, 37
key copy 53

E
ectrypt data
syntax for 20
encryptable datatypes 17
encrypted column
create index 21
included in a where clause 36
maximum internal length 32
to increase length 2
encrypted columns
auditing 61
indexes 63
joins on 65
processing 35
restrictions on modifying 20
search arguments 66
sort orders 64
steps to use 3
encrypted data
accessing 35
accessing with user password 47
movement as cipher text 67
encryption
changing the key 14
columns 32
create system encryption password 12
default key 14
dropping 15, 37

70

Adaptive Server Enterprise
dropping keys 15
granting permission on keys 12
new tables 18
on existing tables 20
select into 19
encryption keys
changing ownership 59
changing ownership syntax 59
creating 5
creating and managing, chapter 5
creating, considerations before creating 5
from a different database 14
password 1
stored encrypted 1
to encrypt 1
Encryption, encrypted columns 1
event names 61
syntax 61
event numbers 61
exec command 22
existing tables
encrypt data 20

F
floating point data, forms for encryption 2
for algorithm 8

G
grant all command, does not grant decrypt permission.  Command
grant all 23
grant decrypt on, syntax 23
grants, implicit 29

I
image 34
implicit grants and decrypt default 29, 30
indexes on encrypted columns 63
indexing encrypted columns 21
init_vector 8
initialization vector 32
insert 2
int_vector 8
integer data, forms for encryption 2
internal length of encrypted column, maximum 32
issuing statements on encrypted column, requirements 35, 36

J
joins, on encrypted columns 65

K
KEK, key-encryption key 10
key
creating copies 45
key copies 43
changing passwords on 46
with login change 53
key copy
dropping 53
key custodian 43
auditing actions 62
custodian, key, activities of 40
role of 39
key protection 10
key recovery commands 57
key_length num_bits 8
keycustodian_role 39
key-encryption key (KEK) 10
keylength 8
keys
changing 14
creating encryption 5
dropping encrypting 15
granting permissions 12
recovering from lost passwords 55
separating from data 14
using passwords 43
Index

L
length
maximum, of encrypted column 32
of plain text data 32
login password
loss of 56
login password change 53
lost
login password 56
password on encryption key 55
passwords, recovering keys from 55

N
names, event 61
null 8
numbers, event 61

O
options
auditing 61
ownership of encryption keys, changing syntax 59

P
pad 8
parameter 8
parameters
key_length 8
keyname 8
decrypt default 25
null 8
password_phrase 8
parameters for create encryption key
keylength num_bits 8
keylength num_bits 8
keyname 8
parentheses ()
in SQL statements xi, xiii
partial clause, variable 18
password
accessing data with user password 47

alter encryption key, changing, syntax
alter encryption key 43
changing on key copies 46
login change 53
loss of 56
loss of on base key 56
lost for encryption key 55
masking in command text auditing 62
recovering keys from lost 55
system-encryption, key protection 12
user-specified 42
using on keys 43
password, variable, length of 12
password_phrase 8
performance considerations 63
permissions
assigning privileges for restricted decrypt 24
decrypt default 27
decryption 36
restricting decrypt 24
revoking decrypt 24
plain text
data, length of 32
for unencrypted data 2
platforms
encryption forms for all platforms 2
privileges, assigning 24

R
random 8
recovery, of key commands 57
referential integrity searches 66
removing decrypt defaults 25, 31
commands 31
requirements
for issuing 36
for issuing insert 35
for issuing select 36
for issuing update 35
restricted decrypt permissions
assigning privileges for 24
restricting decrypt permissions 24
restrictions on modifying encrypted columns 20
returning default values instead of decrypted data 25

Adaptive Server Enterprise
revoke decryption permission 24
roles
data access 41

S
search arguments, on encrypted columns 66
searches
referenceal integrity 66
select command 36
select into 19
crudption 19
requires decrypt 19
sentinel byte, appended to cipher text 32
set encryption passwd
do not place inside trigger or procedure 50
sort orders on encrypted columns 64
source table, requiring column-level permissions 19
sp_encryption 12
sp_encryption, syntax of 12
sp_help 34
square brackets [ ]
in SQL statements xi, xiii, xiv
steps, administrative, to use encrypted columns 3
symbols
in SQL statements xi, xiii
symmetric encryption algorithm 2
syntax
alter encryption key 42
commands for sharing password with key recovery
user 57
dropping encryption key 15
event names and numbers 61
for encrypting columns 20
for encryption keys, changing ownership 59
for key copy recovery 57
grant decrypt on 23
partial, for encryption 18
set encryption passwd 47
syntax conventions, Transact-SQL xi, xiii
sysencryptkeys 43
storage for column encryption key (CEK) 10
system encryption password 12
instructions for creating 13
system-encryption password for key protection 12

T
tables
encryption on new tables 18
timestamp command not encrypted 34
transparency
application 50
transparent encryption 2

U
untext 34
update, encrypts transparently 2
user password
accessing encrypted data 47
users
data access 41
user-specified passwords 42
using passwords on keys 43

V
values
auditing 61
default 28
variable
partial clause 18
vector, initialization 32

W
where clause, issuing commands on data from encrypted column 36